

Article

Length Instruction Fine-Tuning with Chain-of-Thought (LIFT-COT): Enhancing Length Control and Reasoning in Edge-Deployed Large Language Models

Pinzhe Chen * and Zhen Li *

Ocean College, Jiangsu University of Science and Technology, Zhenjiang 212003, China

* Correspondence: 18552718929@163.com (P.C.); justlz@just.edu.cn (Z.L.)

Abstract: This paper investigates the effectiveness of Chain-of-Thought (COT) reasoning in addressing length bias when deploying Large Language Models (LLMs) on the device side within wireless network environments. By conducting systematic experiments on two representative datasets, we perform a comparative evaluation of LLM performance under standard prompting versus COT-augmented prompting, using two benchmark tests. The study emphasizes how the COT strategy regulates response length and improves compliance with predefined maximum length (`max_len`) constraints. Experimental results demonstrate that COT reasoning significantly mitigates length bias, enhances output length control, and improves overall performance. These findings contribute to the theoretical foundation of LLM deployment in edge computing scenarios and provide actionable insights for future research on resource-constrained language model applications.

Keywords: wireless network; resource-constrained environment; large language models (LLMs); Length Instruction Fine-Tuning with Chain-of-Thought (LIFT-COT); response quality optimization



Academic Editor: Arkaitz Zubiaga

Received: 26 March 2025

Revised: 16 April 2025

Accepted: 16 April 2025

Published: 19 April 2025

Citation: Chen, P.; Li, Z. Length Instruction Fine-Tuning with Chain-of-Thought (LIFT-COT): Enhancing Length Control and Reasoning in Edge-Deployed Large Language Models. *Electronics* **2025**, *14*, 1662. <https://doi.org/10.3390/electronics14081662>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Large Language Models (LLMs) are increasingly being deployed at the edge of wireless networks to support dynamic reasoning tasks, where response quality and precise length control remain critical. With the advancement of OpenAI's o1 series, understanding and managing the reasoning process of LLMs has become ever more important. While the o1 model showcases strong COT reasoning capabilities, it also presents notable challenges in response quality. For example, when external information is needed but network search is unavailable, the o1-preview model may fabricate examples instead of acknowledging its limitations, thereby generating misleading content. Moreover, it tends to overstate certainty, often failing to express appropriate uncertainty when unsure, which undermines user trust [1].

These challenges have intensified scrutiny of LLM deployment in wireless edge environments. Besides response quality, controlling output length is vital [2]. In wireless networks, where bandwidth and computational resources are constrained, precise length regulation is essential for optimizing transmission efficiency and computation. However, LLMs often exhibit length bias, producing either excessively long or overly brief responses that impair usability [3]. Long responses can delay transmission, while short ones may lack sufficient informational content.

Recent studies indicate that both human annotators and LLMs tend to exhibit a length bias, favoring longer outputs during evaluations [4]. Although certain benchmarks

introduce length penalties during scoring to counteract this bias [5], these efforts have not fully resolved the issue. However, more effective mechanisms remain needed to enforce length constraints in LLM outputs.

As wireless network technology evolves, the necessity of deploying LLMs at the edge has become more pronounced. Inspired by OpenAI's application of Chain-of-Thought (COT) reasoning, this paper proposes a novel integration of COT steps into the prompt design to mitigate length bias. COT encourages LLMs to reason sequentially through sub-tasks before delivering final answers. Theoretical insights suggest that this approach deepens task comprehension [6], improves reasoning accuracy, and enhances length controllability.

To validate this hypothesis, we conduct rigorous experiments using two representative datasets across two benchmark suites. In these experiments, prompts are augmented with COT elements, and we record response word counts and adherence rates to predefined length constraints. Results demonstrate that embedding COT into prompts significantly improves LLMs' ability to control output length, providing a robust new strategy for enhancing the reliability and usability of LLMs in wireless network scenarios.

Existing work has shown that leading instruction-following models without COT integration struggle to comply with length requirements. For example, COT-augmented versions of AlpacaEval 2 [4] and MT-Bench [7] have revealed that state-of-the-art models such as GPT-4 Turbo violate length constraints in nearly half of the evaluated cases, highlighting the limitations of current models in length-sensitive applications [8].

To address this, we propose Length Instruction Fine-Tuning with Chain-of-Thought (LIFT-COT)—a method that enhances existing instruction-following datasets by integrating COT into prompts alongside explicit length instructions. This technique is designed to improve model performance in complex tasks while ensuring compliance with strict output length guidelines. We fine-tune ten widely-used LLMs—including ERNIE Bot 3.5, ERNIE Bot 4.0, and Mistral Large2—using the LIFT-COT approach and evaluate them using our custom length instruction benchmarks.

2. Related Work

2.1. Length Bias and Model Performance

Reinforcement learning (RL) tends to produce longer responses when enhancing instruction-following performance [9]. Studies reveal that both humans and models exhibit a "length bias", favoring longer responses. Yet, longer does not necessarily mean better; even if preferred by annotators [10], they may harbor hidden security risks, particularly in critical domains like electricity, transportation, and healthcare, where length bias can result in erroneous decisions and safety hazards. Furthermore, it compromises accuracy and stability [11], detracting from user experience. In complex tasks, human evaluators may overlook these inaccuracies [12].

In the Table 1, the ERNIE Bot 3.5 model has been fine-tuned with LIFT-COT and tested with different length constraints for the same question. The first set of instructions simply adds length constraints, while the second integrates chain-of-thought alongside them. The final row shows the response generated from the original input (partially displayed due to space), without any length instructions.

Table 1. Examples of Length Instruction Generation.

Description	Instruction	<MAX_LEN>	Generation	Length
Just length instruction added	Answer the following instruction using <MAX_LEN> words or less.	20	Yes, practice regularly with a good pen and paper, and focus on making each letter clear and consistent.	18
	I'm trying to teach myself to have nicer handwriting. Can you help?	40	Yes, I can help. Start by selecting a pen that feels comfortable in your hand. Then, practice writing letters and words in a clear and consistent manner. Use a guide or template if needed to help improve.....	49
		80	Sure, I can help. Start by analyzing your current handwriting and identifying areas for improvement. Use a pen with a smooth flow and identifying areas for improvement. Use a pen with a smooth flow and comfortable grip. Practice writing each letter and word slowly and deliberately.....	79
Length instruction with COT introduced	Answer the following instruction using <MAX_LEN> words or less.	20	Practice slowly, focus on clarity & consistency. Use a guide if needed. Be patient.	14
	I'm trying to teach myself to have nicer handwriting. Can you help? Let's think step by step.	40	Yes, start with basic strokes and letters. Practice regularly, focusing on consistency and neatness. Use a pen that's comfortable to hold. Be patient and don't give up.	27
		80	Certainly! Start by identifying the areas you want to improve, such as letter spacing, size, or shape. Practice writing slowly and deliberately, focusing on the individual strokes and curves of each letter. Use a guide or template to help you maintain consistency. Gradually increase your speed and confidence as you become more comfortable with the new habits. Remember, it takes time and practice to develop a pleasing handwriting style, so be patient and keep at it!	77
Original input without any length guidance	I'm trying to teach myself to have nicer handwriting. Can you help?	no word limitation of answer	Yes, I can definitely help you with improving your handwriting. Here are some tips to get you started: 1. Start with basic strokes: Practice writing straight lines, curves, and circles to improve your control and consistency. 2. Use a guide: A handwriting workbook or guide can help you learn proper letter formation and spacing.....	156

Existing research has introduced length penalty mechanisms to address length bias. Ref. [13] propose an effective baseline method: fine-tuning using the longest data from the training set. This provides a comparative benchmark for optimizing the consistency of large language models (LLMs) in processing varying input lengths. By emphasizing long text training, models can better capture long text information and mitigate length bias. However, these methods have not yet fully resolved the issue.

2.2. Length-Aware Model Training Methods

Historically, various fields have explored learning methods that account for length, yet their deployment in edge-side large language models (LLMs) within wireless networks

remains limited. Currently, research on length-aware training methods for this context is nascent, with few methods available and their efficacy still to be proven. Existing methods inadequately address the unique characteristics of wireless networks and the constraints of edge deployment, hindering effective mitigation of length bias. Therefore, there is an urgent need to devise effective length-aware training solutions specifically for LLMs deployed on the edge in wireless networks, to enhance both model performance and practicality. To this end, this paper introduces the Length Instruction Fine-Tuning with Chain-of-Thought (LIFT-COT) method, which integrates the Chain-of-Thought (COT) approach.

Although some large language models in production attempt to address output length in system prompts, like Claude 3 [14], which advises concise answers for simple questions and comprehensive ones for complex ones, systematic evaluation remains lacking. Meanwhile, the post-training datasets used for fine-tuning are either unreleased [15,16] or have unclear compositions [17,18], and their small size makes it hard to ascertain if they contain length instructions. Furthermore, research indicates that models generally exhibit a bias towards longer responses when not adjusted.

For deploying LLMs on the edge side in wireless networks, we selected two suitable datasets and integrated COT into our experiments, adhering to the maximum length instruction limits outlined in existing research. By setting maximum length standards based on prior findings and embedding COT in the original prompts, we aimed to assess its impact on models' adherence to length instructions and verify its effectiveness in mitigating length bias.

3. AlpacaEval-LI & MT-Bench-LI: New Length-Instructed Benchmarks

Theoretically, powerful instruction-following models should adhere to given length limitations, as these are part of the instructional scope. For instance, prompts could include instructions like "Explain <concept>. Keep your answer to 300 words or fewer. Let's break it down step by step." Depending on various usage scenarios and contexts, users anticipate receiving answers of varying lengths from the same base model. In this context, we initially assess the current models' capacity to follow length instructions. To do so, we developed length instruction (LI) benchmarks akin to AlpacaEval-LI and MT-Bench-LI.

3.1. Enhancing General Instructions with Length Constraints

To assess models' adherence to length instructions, we fortified existing instruction-following tasks by incorporating COT into the instructions, already inclusive of a maximum length limit, as per the template in Figure 1. This endeavor seeks to determine whether models can adhere to the specified length instructions while effectively addressing the given queries.

Answer the following instruction using <MAX_LEN> words or less.

<ORIGINAL_INSTRUCTION> + Let's think step by step.

Figure 1. Length instruction following. We define the above prompt template (LIFT-COT) in order to require models to generate responses within a maximum response length in the context of introducing COT.

3.1.1. Target Length

To evaluate the cutting-edge (SOTA) models comprehensively, we aligned the target instruction length with the datasets from the study "Following Length Constraints in

Instructions” [8]. By analyzing the generation lengths of three prominent SOTA models—GPT-4 Turbo (11/06) [15] by Achiam et al. (2023), Claude 3 Opus (02/29), and Mistral Large (24/02)—For each original prompt, we collect and analyze their generated text lengths to find the minimum. Then we set <MAX_LEN> in the template to this minimum. During adaptation, we consider prompt diversity. Different prompts can trigger different generation behaviors. Some lead to longer responses, others to shorter ones. By basing the target length limit on the minimum generation length per prompt, we ensure a tailored length constraint. It is short enough to make models optimize strategies and not be easily met by all SOTA models, better reflecting their actual capabilities and limitations to create an accurate and challenging evaluation environment.

3.1.2. Baseline for Adhering to Length Instructions

To establish a solid baseline that strictly conforms to length constraints, we adopt the method outlined in the above section, setting it based on the minimum generation length among three models. Rather than depending on a single model, we determine the baseline response length as the shortest among the three for a given prompt, ensuring both high-quality content and compliance with length requirements. For each test model, we will conduct a thorough pairwise comparison of its output against the baseline.

Figure 2: State-of-the-art (SOTA) models can follow length instructions better. A comparison of ERNIE Bot 4.0 and ERNIE Bot 4.0 Turbo on AlpacaEval-Length-Instructed (LI) examples is shown. The x-axis represents the target length, while the y-axis displays the ratio of the actual generated length to the target. Red dots signify violations—cases where the generated length surpasses the target limit—and blue dots indicate compliance with the limit.

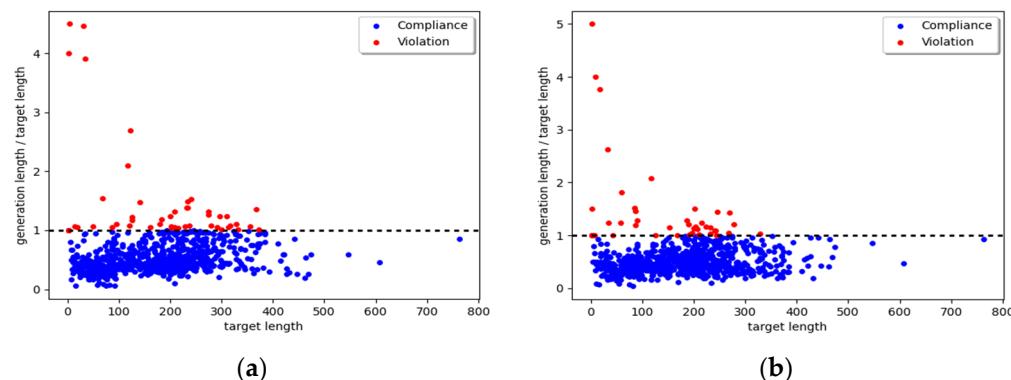


Figure 2. (a,b) respectively show the scatter plot of ERNIE Bot 4.0 and the scatter plot of ERNIE Bot 4.0 Turbo.

3.1.3. Metrics

To comprehensively evaluate the performance of models in following length instructions, we propose the following four key metrics:

Accuracy Rate (Acc%): The proportion of responses that exactly meet the length constraints. This metric comprehensively assesses the models’ ability to generate appropriately-lengthed responses. Specifically, any response strictly adhering to the given length constraint is deemed accurate and factored into the accuracy rate calculation.

Violation Rate (Vlt%): Measures instances where models fail to adhere to length instructions. Calculation Method: Determined by the percentage of responses that exceed the specified length constraints among all responses.

Target Length Deviation (TLD): The gap between predicted and actual target response lengths, showing model accuracy in length estimation. Positive means overestimation,

negative means underestimation, for bias assessment. Calculation Method: Find length differences per response and average them.

Target Length Variance (TLV): Measures dispersion of predicted lengths around the target length, indicating prediction stability. High variance: big differences, unstable model. Low variance: clustered lengths, stable performance. Calculation Method: Square differences between predicted and actual lengths per response, then average squared values.

In addition, we will report other pertinent metrics, including the average response length (measured in words). For word counting, we employ NLTK’s word tokenization function, excluding punctuation. The detailed word counting process is described in detail in Appendix A.

3.2. Length-Instructed AlpacaEval & Length-Instructed MT-Bench

Our evaluation benchmark aligns with the study “Following Length Constraints in Instructions” [8], utilizing AlpacaEval 2 across task domains like creativity, brainstorming, writing, QA, math, and reasoning. By introducing length instructions embedded with COT, as outlined in the previous part, we crafted the AlpacaEval Length Instruction (LI) benchmark. Furthermore, we expanded the MT-Bench evaluation by incorporating COT-embedded length instructions, enabling us to test models in various prompt scenarios. This dataset encompasses 80 challenging multi-turn questions spanning eight prompt categories: writing, role-playing, extraction, reasoning, math, coding, STEM, and humanities/social sciences. Following what has been described previously, we set three length constraints for each prompt (considering only the first turn), which resulted in 240 MT-Bench-LI prompts. In the subsequent experiments, the two benchmarks, AlpacaEval-LI and MT-Bench-LI, will be employed.

Figure 2 displays the ratio of generated length to target instruction length across varying target lengths. ERNIE Bot 4.0 exceeded the target length (red dots) about 5.7% of the time, particularly for targets exceeding 200 words. The scatter plot revealed a similar pattern for ERNIE Bot 4.0 Turbo. Results for other large language models are presented at the end of the text.

4. The Dynamic Reasoning Method with COT

4.1. Introduction and Principle of COT

COT is vital in natural language processing, enhancing language models’ reasoning abilities through targeted guidance. Its essence involves using examples or guiding statements in prompts to prompt models to tackle problems logically, breaking down the overall task T into subtasks [10] $T = T_1 + T_2 + \dots + T_n$ (as detailed in “The Prompt Report: A Systematic Survey of Prompting Techniques”), processing each to obtain results R_1, R_2, \dots, R_n , and ultimately deriving the final answer A (denoted as $A = f(R_1, R_2, \dots, R_n)$). Additionally, it harnesses the learning capabilities of large language models (LLMs) to refine answer quality.

The COT technique demands models to showcase their reasoning process prior to giving the final answer. By integrating COT prompts into the training dataset and imposing output length limits during fine-tuning, the model learns to reason incrementally. Additionally, COT prompt templates train models to self-verify, streamline reasoning steps, and optimize output length through iterative fine-tuning. This enables precise control over outputs, bolstering reasoning abilities, accuracy, and overall efficiency.

However, exploring why the LIFT-COT combination works synergistically rather than conflicting is crucial. On the one hand, the chain of thought propels the model’s reasoning forward: as the reasoning steps increase, the output length grows correspondingly to present a complete logic, like when solving complex math problems; the model must detail

each step's operation basis and intermediate results. On the other hand, the output length limitation in training plays a key regulatory role, defining a reasonable output range, and iterative fine-tuning optimizes it based on the model's validation set performance. Through such coordination, the model balances reasoning presentation and practical needs, precisely controls output, and boosts reasoning, accuracy, and overall efficiency.

4.2. Design of the Dynamic Reasoning Framework

4.2.1. Components of the Framework

Input Interface: This interface receives user inputs and prompt info in a wireless setting, preprocessing and formatting them to ensure that the reasoning process can comprehend the data effectively.

COT Core Module: As the framework's heart, this module introduces COT reasoning. Based on inputs and length instructions, it precisely inserts reasoning chains, aiding the model in grasping problem logic and managing output length.

Length Control Unit: This unit monitors output length, comparing it to preset limits. If exceeded, it collaborates with the COT Core Module to tweak COT info or restart the reasoning engine, ensuring length compliance.

Model Reasoning Engine: Leveraging a large language model, this engine crafts answers through reasoning, using input prompts and COT info. It works seamlessly with other modules to refine the reasoning process, enhancing answer accuracy and quality.

Output Interface: This interface organizes and formats answers, transmitting them wirelessly to users. Based on feedback and evaluation metrics, it fine-tunes the framework's performance, optimizing service delivery.

4.2.2. Work Flow

In a wireless network, user queries are received by the input interface, preprocessed, and then sent to the COT (Chain-of-Thought) core module. This module inserts reasoning chains into prompts, optimizing model understanding and output length control, based on input and length instructions. The refined prompts are passed to the large language model reasoning engine, which generates preliminary answers. The length control unit monitors this in real-time, adjusting or restarting the reasoning process with the COT core if the output exceeds limits. Finally, the output interface organizes and formats the answers, feeding them back to users via the wireless network. Based on user feedback and evaluations, the framework is continually optimized for efficient and accurate service quality improvement.

5. Optimization of Problem Design and Algorithms

5.1. Optimization of Problem Design

In the Figure 3, this is a flowchart of the process where the user's input through the mobile phone terminal undergoes base station resource limitation processing, as well as the application of the LIFT-COT (Length Instruction Fine-Tuning with Chain-of-Thought) method and the self-iterative algorithm.

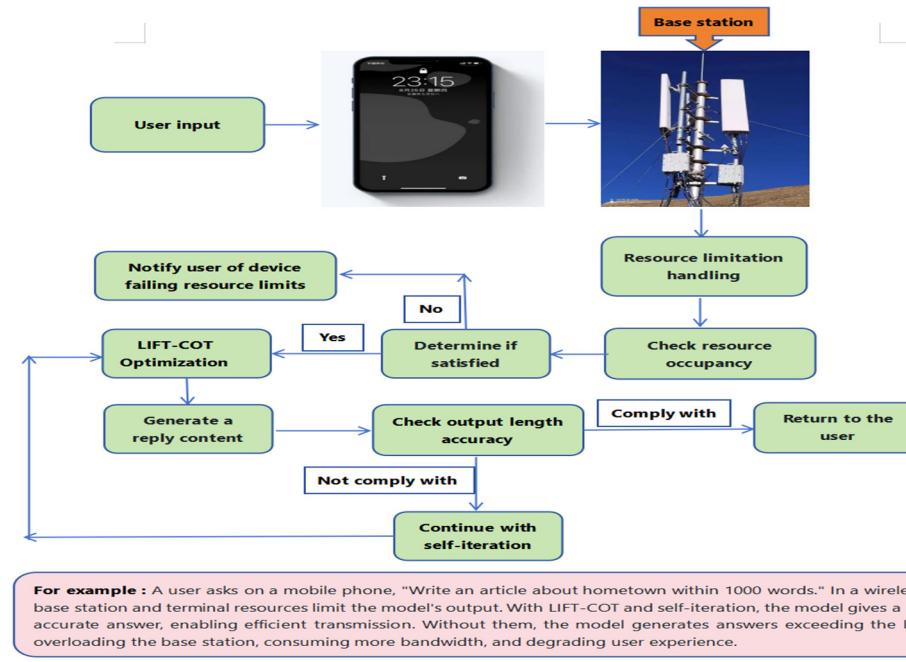


Figure 3. The flowchart of the Optimization of Problem Design.

5.1.1. Optimization Objectives

When deploying LLMs on wireless network edges, length deviation of model outputs significantly impacts performance. In mobile intelligent customer service, for instance, users ask questions like "Why does my phone charge slowly?" via their phones. Excessive length prolongs waiting times and consumes more data, while insufficient length fails to provide adequate solutions. Here, our LIFT-COT method is crucial. The phone sends questions with a preset length limit (e.g., ≤ 100 chars) to the cloud/edge model. The model uses LIFT-COT to analyze questions via COT steps, generating length-compliant, semantically accurate, and logically coherent responses like "Check the cable, clear debris, or try another charger", ensuring precise solutions without compromising user experience [19] or task efficiency. Our optimization goal is to maximize the number of tasks completed N , with answers that are length-aligned to the maximum length L^{\max} , semantically accurate, and logically coherent.

5.1.2. Constraint Conditions

Constraints on CPU usage and memory occupancy:

Suppose the maximum CPU usage of the device is U_{CPU}^{\max} , and the device has the maximum available memory M^{\max} ; then, when the model is running, it needs to satisfy the following simultaneously:

$$U_{CPU} \leq U_{CPU}^{\max}, \quad (1)$$

$$M \leq M^{\max}. \quad (2)$$

Constraints on bandwidth occupancy and data transmission delay:

Suppose the maximum bandwidth occupied by model data transmission is B^{\max} , and the maximum acceptable data transmission delay is D^{\max} ; that is, when the model is running, it needs to satisfy the following simultaneously:

$$B \leq B^{\max}, \quad (3)$$

$$D \leq D^{\max}. \quad (4)$$

Accuracy Constraint Based on the Reference Text:

Let the model's answer be A , and the correct reference answer be R . Define the accuracy calculation function as $\text{Accuracy}(A, R)$, and set the minimum acceptable accuracy as Accuracy^{\min} . The model is required to satisfy:

$$\text{Accuracy}(A, R) \geq \text{Accuracy}^{\min}. \quad (5)$$

This constraint guarantees the fundamental accuracy of the model's output answers, preventing the dissemination of misleading information to users. However, in practical applications, models often suffer from the "hallucination" phenomenon [20], where significant discrepancies arise between the model's output and the expected result. These discrepancies can lead to excessively long outputs, burdening data transmission and causing information redundancy, or overly concise outputs that fail to fully address the question. Furthermore, the content may lack logical coherence, contain falsehoods, or exhibit irrational reasoning, for instance, answering "Does the sun rise in the west?" with "Yes" (a factually incorrect statement) or responding to the reasoning question, "If A implies B , and B implies C , what is the relationship between A and C " with the answer " A and C are unrelated" (irrational reasoning). In the context of deploying large language models (LLMs) on the wireless network side, "hallucination" significantly impairs the model's performance and application effectiveness [21].

Length Instruction Adhering to the Accuracy Constraint:

To address the problems brought about by hallucination, especially those adverse effects in terms of length, we have set the length instruction adhering to the accuracy constraint. The length L of the model's generated answer needs to comply with the preset maximum length limit L^{\max} . We define the length instruction adhering to the accuracy calculation function as $\text{Accuracy}_{\text{length}}(L, L^{\max})$, and set the minimum acceptable accuracy as $\text{Accuracy}_{\text{length}}^{\min}$. Then, when the model is running, it is required to satisfy

$$\text{Accuracy}_{\text{length}}(L, L^{\max}) \geq \text{Accuracy}_{\text{length}}^{\min}. \quad (6)$$

This enables precise control over the model's output length, ensuring that it meets expectations, boosts communication efficiency, and enhances output quality. For instance, in a mobile news summary app, an overly long model-generated summary hinders quick browsing and key information absorption, while a too-short one may fail to satisfy users' information needs.

To accurately measure the similarity between the model's output and the reference text, metrics like BLEU, ROUGE, and CIDEr [22] can be adopted. BLEU assesses accuracy by calculating the n -gram overlap between the generated and reference texts. The formula is

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^N w_n \log p_n\right), \quad (7)$$

where BP is the brevity penalty factor, p_n is the n -gram precision with a length of n , w_n is the corresponding weight and is usually evenly distributed so that $\sum_{n=1}^N w_n = 1$, and N is generally set to 4. BLEU can be used to measure the model's ability to generate accurate text and assist in determining whether the length instruction adhering to the accuracy constraint is satisfied. ROUGE mainly focuses on the longest common subsequence (LCS)

between the generated text and the reference text and evaluates it by calculating the recall rate, precision rate, and F-value. The precision rate can be estimated as

$$P_L = \frac{LCS(X, Y)}{|Y|}, \quad (8)$$

where $LCS(X, Y)$ is the length of the longest common subsequence between the reference text X and the generated text Y , and $|Y|$ is the length of the generated text. The recall rate can be calculated as

$$R_L = \frac{LCS(X, Y)}{|X|}, \quad (9)$$

where $LCS(X, Y)$ is the length of the longest common subsequence between the reference text X and the generated text Y , $|X|$ is the length of the reference text, and the F-value can be calculated as

$$F_L = \frac{(1 + \beta^2) \cdot P_L \cdot R_L}{R_L + \beta^2 \cdot P_L}, \quad (10)$$

β determines the weight of the recall rate relative to the precision. When $\beta = 1$, it results in the F1 score. ROUGE evaluates similarity by assessing content coverage and logical coherence, checking for logical flaws in the model's output and ensuring that it meets the accuracy constraint of the question.

Initially designed for image captioning, CIDEr also holds significance in evaluating model-generated text quality and semantic consistency. It emphasizes the consensus and commonality of n-grams and is defined as

$$CLDER = \frac{1}{|C|} \sum_{i=1}^{|C|} CLDER_i, \quad (11)$$

where $CLDER_i$ is the CIDEr score for the i -th n -gram, and $|C|$ denotes the total number of n-grams; using multiple references allows CIDEr to evaluate the semantic and informational match between model-generated answers and reference answers. This assessment evaluates the quality of the output, focuses on logical relevance, ensures accuracy, and reduces hallucinations. Constraint conditions can be set:

$$BIEU(A, R) \geq BIEU^{\min}, \quad (12)$$

$$ROUGE(A, R) \geq ROUGE^{\min}, \quad (13)$$

$$CIDEr(A, R) \geq CIDEr^{\min}. \quad (14)$$

Among them, $BIEU^{\min}$, $ROUGE^{\min}$, and $CIDEr^{\min}$ are the preset minimum acceptable values, respectively, which prompt the text generated by the model to meet certain standards in terms of similarity and semantic consistency with the reference text, improve accuracy and reliability, and reduce the occurrence of hallucination.

Constraint on the Number of Generated Tokens:

Let T be the number of tokens in the model's generated answer. To prevent unreasonable outputs and control resource consumption, as well as reduce the interference of hallucination on the token generation mechanism, a maximum allowable number of generated tokens T^{\max} is set, that is,

$$T \leq T^{\max}. \quad (15)$$

During the model processing, we define an encoding function $E(a)$ that transforms input answer a into token IDs. This function preprocesses user questions or model answers into token sequences, facilitating subsequent computational analyses. By capping the token count in conjunction with the encoding function, we effectively manage the model's input and output scale. This boosts processing efficiency and accuracy, especially in resource-limited environments, and minimizes abnormal token counts due to hallucinations. During training or evaluation, various calculations and operations can be performed based on these token IDs. Furthermore, we define a reward function $R(E(a))$ that assesses rewards based on the encoded input. It awards higher values for meeting expectations and possibly lower or negative values otherwise, guiding the model to produce outputs that better align with requirements and reducing hallucinations. Additionally, a loss function can also be defined:

$$L(R) = -\frac{1}{N} \sum_{i=1}^N R_i. \quad (16)$$

Here, N represents the token count. This function can measure the degree of difference between the model's output and the expected output. During the optimization process, the model parameters are adjusted to minimize the loss function's value.

To better regulate the impact of both the reward and loss functions [23] on model optimization, constraint conditions are imposed. For instance, to cap the fluctuation range of the reward function, we set

$$\frac{1}{M} \sum_{m=1}^M (R(E(a_m)) - \overline{R(E(a)))})^2 \leq \sigma_R^2, \quad (17)$$

where M is the number of samples, a_m is the m -th input answer, $\overline{R(E(a)))}$ is the average reward, and σ_R^2 is the upper limit of the variance of the reward function. Meanwhile, by preventing excessive fluctuations, the model can be made more stable. For the loss function, the requirement is

$$|L(R) - L^{\text{target}}| \leq \varepsilon_L, \quad (18)$$

where L^{target} is the expected loss target and ε_L is the deviation threshold of the loss function, ensuring that the loss function converges towards the target value and avoiding excessive deviations.

Constraint on the Strength of Chain-of-Thought Guidance:

The strength of chain-of-thought guidance in the LIFT-COT method has a significant impact on the model's performance. Let the parameter related to the strength of chain-of-thought guidance be s , with its minimum value being s^{\min} and its maximum value being s^{\max} . It is required to satisfy

$$s^{\min} \leq s \leq s^{\max}. \quad (19)$$

Adequate guidance intensity ensures effective utilization of chain-of-thought reasoning by the model, preventing dependency imbalances. Insufficient guidance hampers its role, compromising inference and output quality. Conversely, overly strong guidance restricts model flexibility and innovation, potentially leading to unreasonable inferences.

Constraint on the Effectiveness of the Length Control Strategy:

Let the evaluation indicator for the effectiveness of the length control strategy be E_{length} , and the minimum acceptable effectiveness value be E_{length}^{\min} . It is required to satisfy

$$E_{\text{length}} \geq E_{\text{length}}^{\min}. \quad (20)$$

This constraint can ensure that the length of the model's output meets the requirements and avoids problems such as resource waste and incomplete or inaccurate information transmission caused by uncontrolled length.

5.1.3. Decision Variables

In the optimization process, the parameter adjustment vector x in the LIFT-COT method is a crucial decision variable. It includes parameters s , related to the guidance intensity of chain-of-thought reasoning, and those pertaining to output length control strategies. Adjusting these parameters directly influences the model's ability to complete tasks under multiple constraints and the quality of its output. For example, adjusting parameter s can alter the model's reliance on and utilization of chain-of-thought reasoning, optimizing reasoning effectiveness. Conversely, the output length control parameters allow for precise adjustment of the model's output length, ensuring that it meets predefined standards.

5.1.4. Formulation of the Optimization Problem

By adjusting the decision variables x , the number of completed tasks N is maximized while satisfying all constraint conditions, as illustrated in the preceding formulas. In practice, it is crucial to set the parameter values in the constraint conditions based on the specific attributes of the model architecture and wireless network environment.

5.2. Optimization Algorithm Design

5.2.1. Algorithm Overview

This optimization algorithm, rooted in the concept of self-iteration, is inspired by the cyclic structure and convergence mechanism of self-reflection methods [24]. Unlike traditional algorithms, it comprehensively optimizes the model through multi-dimensional evaluations and iterative cycles, adapting to complex wireless network edge environments. It reduces hallucinations and enhances both the number of tasks completed and output quality. During evaluation, it considers not just traditional accuracy rates but also factors like length instruction accuracy, resource consumption, and model inference stability. This dynamic and comprehensive optimization approach enables the model to better adapt to constrained wireless network edge environments, gradually converging to an optimal state. It effectively minimizes "hallucination" phenomena in outputs, adapts to resource-limited conditions, and boosts both task completion rates and output quality.

Meanwhile, this algorithm incurs no extra computational overhead [25]. Compared with the traditional approach of using penalty terms in the loss function, the self-iterative algorithm holds distinct advantages. It iterates based on feedback to adeptly handle complex, dynamic constraints like tokens, precision, and resources. For scenarios hard to formulate mathematically and embed in the loss function, it precisely tunes algorithm parameters and processes via iteration feedback, progressively fulfilling multiple constraint requirements and reaching the optimization goal. In contrast, when dealing with complex constraints, the penalty term method struggles to devise well-fitting penalty terms and faces issues with weight setting, causing the model to either overly focus on constraints and deviate from the core learning task or fail to restrain itself effectively, exhibiting a significant disparity from the self-iterative algorithm.

5.2.2. Algorithm Flow

1. Initialization Phase:

Initialize the number of completed tasks N to 0. Determine the values of relevant constraint parameters according to device performance and application requirements.

Initialize the decision variables x , including the COT guidance intensity parameter s and other parameters related to output length control strategies. They can be initialized randomly or based on empirical values. For example, s can be set as $s_0 = \frac{s^{\min} + s^{\max}}{2}$. Set the number of iterations as $t = 0$. Load the pre-trained Language Model with Large-scale Parameters (LLM) model as well as carefully prepared training data and test data.

2. Iterative Loop Phase:

(a) Generation Stage:

Using current parameters and input prompts, the model crafts the output text. The COT prompt explicitly outlines length limitations, such as, “Analyze logically and provide an answer within L^{\max} ”, ensuring the model adheres to length constraint”. This reduces content hallucinations and enhances output quality and reliability.

(b) Evaluation Stage:

Assess the iteration by counting completed tasks N_{new} and those successfully meeting all constraints, a vital metric for model performance. For each output o_i , compute multi-dimensional evaluation metrics.

- Semantic Understanding Accuracy:

Utilize a semantic similarity method. For instance, employ a pre-trained semantic vector model (like BERT [26]) to convert the model-generated answer o_i and the reference correct answer r_i into semantic vectors. Then, calculate the cosine similarity between these vectors as the accuracy indicator. The formula is

$$\text{Semantic Accuracy}(o_i, r_i) = \cos_s \text{imilarity}(\text{BERT}(o_i), \text{BERT}(r_i)). \quad (21)$$

- Accuracy of Precise Length Instruction Following:

Introduce the concept of relative error, and the calculation formula is

$$\text{PreciseAccuracy}_{length}(L_i, L^{\max}) = \begin{cases} 1, & L_i \leq L^{\max} \\ 1 - \frac{|L_i - L^{\max}|}{L^{\max}}, & L_i > L^{\max}. \end{cases} \quad (22)$$

where L_i represents the length of the answer generated by the model, and L^{\max} represents the preset maximum length. To reflect the closeness of the model’s output length to the preset maximum length limitation.

(c) Comprehensive Resource Consumption Indicators:

CPU Usage Fluctuation Evaluation:

Evaluate the stability by calculating the standard deviation of CPU usage within a certain period of time. The formula is

$$\sigma_{U_{CPU}} = \sqrt{\frac{1}{m} \sum_{j=1}^m (U_{CPU,j} - \bar{U}_{CPU})^2}, \quad (23)$$

where m is the number of monitoring times, m is the CPU usage at the j -th monitoring, and \bar{U}_{CPU} is the average CPU usage.

Peak Memory Occupancy Evaluation:

Document the peak M_{peak} and average M_{avg} memory occupancy to assess memory usage.

Dynamic Bandwidth Occupancy Evaluation:

Measure the rate of change in bandwidth occupancy:

$$\rho_B = \frac{B_{t+1} - B_t}{B_t}. \quad (24)$$

Among them, B_t and B_{t+1} are the bandwidth occupancies at adjacent time points. They can be used to evaluate the dynamic changes in bandwidth occupancy.

Data Transmission Delay Jitter Evaluation:

Calculate the variance of data transmission delay:

$$\sigma_D^2 = \frac{1}{n} \sum_{i=1}^n (D_i - \bar{D})^2, \quad (25)$$

where n is the number of transmissions, n is the transmission delay at the i -th transmission, and \bar{D} is the average transmission delay. This can be used to measure the stability of data transmission delay.

Model Inference Stability Evaluation:

Let $o_{i,k}$ be the output of the model's k -th inference for input i . Evaluate the stability of the model's inference by calculating the average of the edit distances between $o_{i,k}$ (such as the Levenshtein distance):

$$\overline{\text{EditDistance}} = \frac{1}{K(K-1)} \sum_{k=1}^K \sum_{j \neq k}^K \text{EditDistance}(o_{i,k}, o_{i,j}), \quad (26)$$

where k is the number of inferences. Meanwhile, calculate the average response length (in words, using tools like NLTK for word tokenization and excluding punctuation marks) to evaluate the distribution of the model's output lengths.

3. Feedback Adjustment Phase:

If $|P - P_{\text{prev}}| < \epsilon$ (where P_{prev} denotes the comprehensive performance indicator from the previous iteration) or $t = T_{\text{max}}$, it is deemed that the model has either met the optimization goal or cannot be further optimized. Subsequently, halt the iteration and output the current model parameters x alongside the optimization results (encompassing the task completion count, various accuracy metrics, etc.). If not, proceed with adjusting the model parameters to enhance performance and mitigate hallucination. Employ the backpropagation algorithm to compute the gradient of the comprehensive performance indicator P relative to the model parameters x . Using the gradient $\nabla_x P$ descent algorithm, update the model parameters according to the formula $x_{\text{new}} = x - \alpha \nabla_x P$, where α is the learning rate, dynamically tuned based on the model's convergence status. For instance, a strategy of $\alpha_t = \frac{\alpha_0}{1+\beta t}$ (where α_0 is the initial learning rate and β is the decay rate) can be employed. As iterations accumulate, progressively reduce the learning rate to prevent overshooting the optimal solution and guarantee convergence. Increment the iteration count $t = t + 1$ and proceed to the next iteration round, repeating the processes of generation, evaluation, and adjustment.

5.2.3. Length Control Strategies

(a) In COT prompts, clearly establish output length limits and emphasize the relevance and realism of the content, preventing irrelevance or errors due to excessive length.

(b) If the model's output exceeds the length limit L^{max} , adjust the reasoning steps or guidance in COT prompts to guide the model towards generating a concise and accurate response. A simplification coefficient γ ($0 < \gamma < 1$) can be set to adjust the reasoning

steps to the original number of steps multiplied by γ to simplify the reasoning process. Alternatively, guide the model to focus on the key points of the question and eliminate redundancy, such as by saying, “Please revisit the question and streamline your answer to ensure an accurate response within the specified length”.

(c) During the evaluation phase, outputs that exceed the length limit are penalized by lowering their scores, encouraging the model to prioritize length control in subsequent generations and reduce hallucinations. A length violation penalty term $\lambda(L_i - L^{\max})^+$ is added to the comprehensive performance indicator (where λ is the penalty coefficient, $(L_i - L^{\max})^+ = \max(0, L_i - L^{\max})$). When the model’s output length exceeds the limit, this penalty term will decrease the comprehensive performance indicator, prompting the model to adjust its generation strategy to meet the length requirement.

5.2.4. Algorithm Termination Conditions

(a) If the change in the value of the comprehensive performance indicator P in consecutive k iterations is less than the convergence threshold ϵ , the model is deemed converged, and iteration ceases. The threshold k , typically ranging from 5 to 10, is determined by the model’s stability and optimization speed. This indicates that, under current parameters, the model has attained a stable optimization state in accuracy (minimizing hallucination), resource utilization, and task completion, with further iterations unlikely to yield significant performance gains.

(b) When the iteration count reaches its maximum T_{\max} , iteration should halt [27], regardless of convergence status, to prevent infinite loops or overtraining. This ensures optimization within limited time and resources, avoiding new “hallucination” issues or overfitting due to excessive iteration. In practice, T_{\max} can be set according to specific needs and resource availability, balancing optimization effectiveness and computational cost.

Through the aforementioned optimization schemes, the LIFT-COT method excels in deploying LLMs on the wireless network edge. It optimizes performance under resource constraints, boosts task completion rates, enhances output quality, and significantly improves compliance with maximum length limitations. Additionally, it effectively manages resource consumption and mitigates the “hallucination” phenomenon.

6. Experimental Setup and Results

6.1. Experimental Datasets and Benchmarks

Our training data and benchmarks align with those used in the “Instructions Following Length Constraints” study. Utilizing the length instruction template in Figure 1, we introduced COT quantitatively after the maximum length limit instructions, resulting in two new enhanced datasets for instruction fine-tuning, named LIFT-COT. Notably, the AlpacaEval-LI baseline model, without COT, has a violation rate of 6.2% and the average response length (Words) of 180 words.

Comparing the performance of models with COT from Table 2 to the AlpacaEval-LI baseline, we notice a decline in violation rates for most large language models. This suggests that LIFT-COT enhances problem understanding and output accuracy. Furthermore, the average response length of ten SOTA models dropped significantly, hinting that LIFT-COT is an effective length bias mitigation strategy, precisely controlling model output length.

As can be seen from Tables 2 and 3, Compared with the AlpacaEval-LI baseline model, many state-of-the-art large language models (LLMs) show a decreasing trend in the violation rate (Vlt (%)), and their average response lengths (Words) also decrease significantly.

Table 2. Results of SOTA models following length instructions in the AlpacaEval-LI benchmark tests.

Standard Models	AlpacaEval-LI		
	Vlt (%)	Acc (%)	Words
ERNIE Bot 3.5 + (LIFT-COT)	35.7	64.3	163
ERNIE Bot 4.0 + (LIFT-COT)	5.7	94.3	105
ERNIE Bot 4.0 Turbo + (LIFT-COT)	4.7	95.3	100
Doubao + (LIFT-COT)	0.9	99.1	64
Mistral Large2 + (LIFT-COT)	15.6	84.4	168
Codestratal + (LIFT-COT)	5.9	94.1	104
Mistral Nemo + (LIFT-COT)	2.9	97.1	66
Kimi + (LIFT-COT)	9.6	90.4	137
Tongyi Qianwen 2.5 + (LIFT-COT)	5.7	94.3	131
iFlytek Spark + (LIFT-COT)	10.3	89.7	137
AlpacaEval-LI Baseline Model (without COT)	6.2	93.8	180

Table 3. Results of SOTA models following length instructions in the MT-Bench-LI benchmark tests.

Standard Models	MT-Bench-LI		
	Vlt (%)	Acc (%)	Words
ERNIE Bot 3.5 + (LIFT-COT)	76.2	23.8	265
ERNIE Bot 4.0 + (LIFT-COT)	15.4	84.6	131
ERNIE Bot 4.0 Turbo + (LIFT-COT)	64.6	35.4	217
Doubao + (LIFT-COT)	7.9	92.1	79
Mistral Large2 + (LIFT-COT)	59.2	40.8	234
Codestratal + (LIFT-COT)	49.6	50.4	193
Mistral Nemo + (LIFT-COT)	34.2	65.8	176
Kimi + (LIFT-COT)	66.2	33.8	207
Tongyi Qianwen 2.5 + (LIFT-COT)	30.4	69.6	160
iFlytek Spark + (LIFT-COT)	23.7	76.3	145

6.2. Experimental Results and Analysis

Figure 4 shows large language models fine-tuned with length instruction (LIFT-COT) excel at adhering to length guidelines. Shown is the Mistral Large model's performance on AlpacaEval Length-Instructed (LI) examples. The two scatter plots for the same model depict scenarios where only length limit instructions are added and where COT is introduced alongside length limits, respectively. The x-axis represents the target length, while the y-axis shows the ratio of actual generated length to target length. Red dots signify violations—generated length exceeding the target—while blue dots indicate compliance.

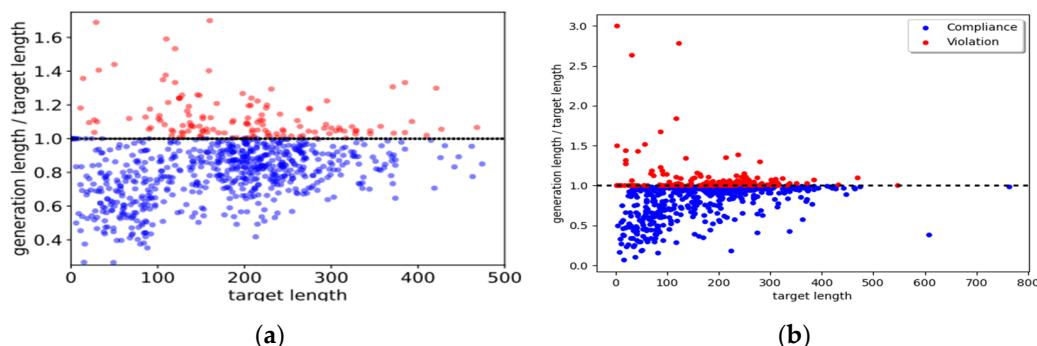


Figure 4. (a,b) respectively show “Just added length instruction to the Mistral Large” and “Mistral Large with LIFT-COT”.

It shows the performance of the Mistral Large model before and after introducing COT under the AlpacaEval benchmark. After comparison, it can be found that the violation rate shows a decreasing trend through the Length Instruction Fine-Tuning with Chain-of-Thought (LIFT-COT) method. (The scatter plot of adding only length limit instructions is from the literature “Following Length Constraints in Instructions” [8]).

We reported the AlpacaEval-Li accuracy rate (Acc%) and violation rate (Vlt%) of existing SOTA LLMs in Table 2. We can find that current state-of-the-art models, like ERNIE Bot 4.0 Turbo, struggle with adhering to length instructions. On AlpacaEval-LI, it violates instructions at a rate of 4.7%, and on MT-Bench-LI, this rate spikes to 64.6%. In contrast, the Doubao instruction model series boasts much lower violation rates: just 0.9% on AlpacaEval-LI and 7.9% on MT-Bench-LI. Notably, the performance differences of models also play a crucial role in task completion [28], ERNIE Bot 3.5 exhibits the highest violation rates on both benchmarks and has an average response length far exceeding other models. This indicates a relatively weak adherence to length instructions, potentially due to inadequate instruction understanding and execution [29], which could hinder its practical application performance. Clearly, ERNIE Bot 3.5 lags behind other models in terms of adhering to length instructions.

Figure 5: Bar charts illustrate the accuracy rate (Acc%) and average response length (Words) of state-of-the-art large language models (SOTA LLMS) across two benchmarks. Models are vertically ranked by accuracy rate (Acc%), with purple bars representing accuracy and blue bars representing average response length (Words). It is evident that, in most instances, higher accuracy rates correspond to shorter average response lengths.

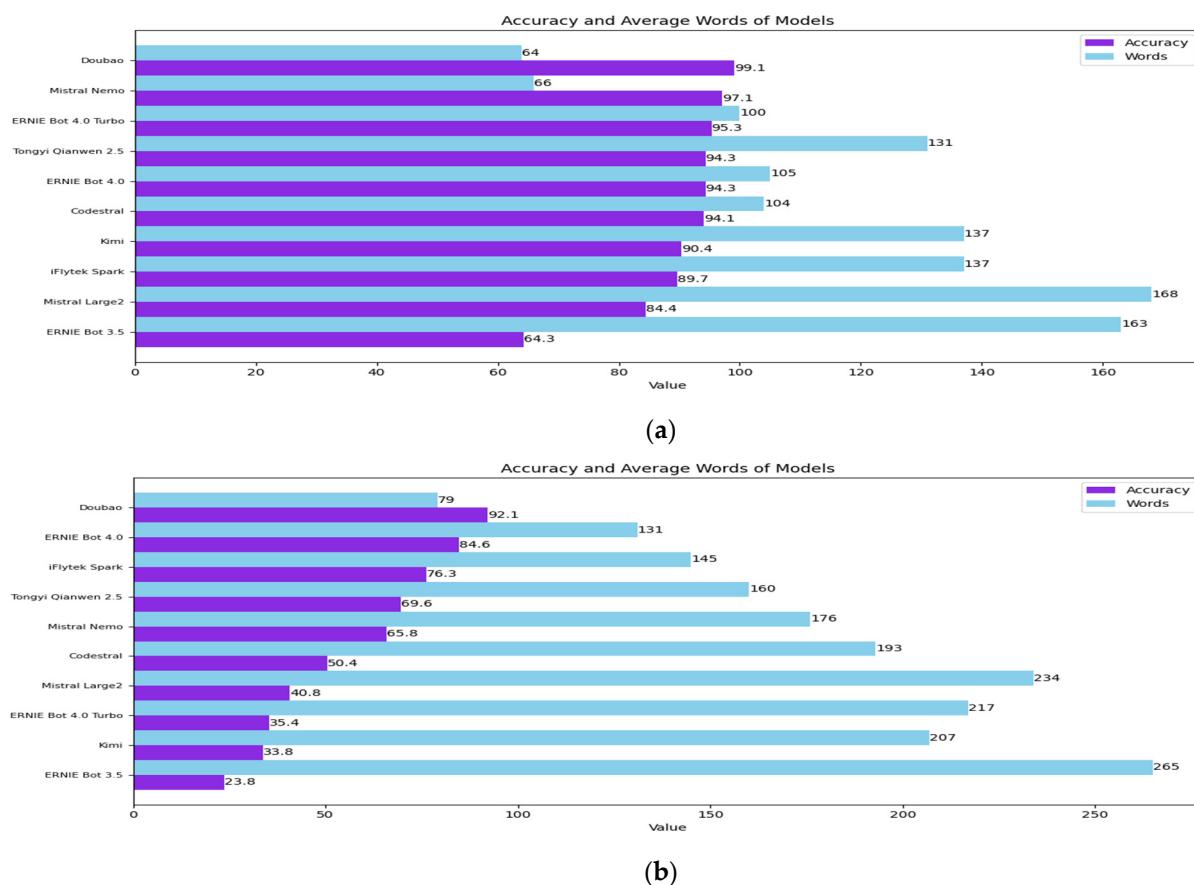


Figure 5. (a,b) respectively show the Accuracy and Average Words of models on AlpacaEval-LI and MT-Bench-LI.

Experimental results confirm that incorporating COT is an effective strategy to address the length bias issue. By introducing and analyzing COT, we gain deeper insights into model reasoning, enhancing output accuracy and enabling finer control over output length. This aligns with OpenAI's documentation on COT's impact on response quality, further underscoring its significance and potential in device-side LLM dynamic reasoning within wireless networks. In mobile information retrieval on wireless terminals such as mobile phones, LIFT-COT's advantage is crucial. When users search on mobile browsers with length restrictions, like "nearby restaurants in 80 words", the optimized model quickly provides accurate, length-aligned answers, saving data and boosting efficiency. During lunch rushes, its value shines: for "nearby lunch spots in 60 words", it offers targeted responses like, "XX Restaurant nearby serves lunch, featuring XX dish, on XX Street, XX. High cost-performance." These responses meet user needs, boost satisfaction, showcasing LIFT-COT's practical value.

Building on COT's pivotal role, our novel optimization algorithm substantially outperforms traditional methods. Inspired by self-iteration and leveraging self-reflection mechanisms, this algorithm integrates multi-dimensional metrics during evaluation, including semantic accuracy, length instruction adherence, and resource consumption. This boosts the model's adaptability to edge environments in wireless networks and reduces hallucinations. During feedback adjustment, the algorithm dynamically tunes parameters based on overall performance, updates them through backpropagation and gradient descent, and incorporates length control, model compression, dynamic resource allocation, and inference optimization. This not only corrects length deviations but also boosts task completion rates, output quality (by minimizing hallucinations and enhancing logic), and resource control efficiency.

6.3. Analysis of Comprehensive Performance Indicators

The comprehensive performance metric p , devised in the preceding algorithm, macroscopically assesses a model's overall effectiveness on the edge of wireless networks, considering various constraints. It aligns with factors like task completion, accuracy, and resource consumption, collectively guiding the optimization of model parameters to meet the complex, diverse demands of the edge side in wireless networks.

Based on this foundation, this section delves into the specific performance dimension of a model's adherence to length instructions. While metrics like accuracy (denoted as $Acc\%$) and violation rate (denoted as $Vlt\%$) offer insights, a comprehensive view necessitates considering additional factors. Hence, we introduce the average response length (denoted as L) and construct a comprehensive performance metric formula. Let ω_1 , ω_2 and ω_3 be the weight coefficients reflecting their importance. The comprehensive performance metric p is defined as $p = \omega_1 \times Acc\% - \omega_2 \times Vlt\% - \omega_3 \times L$, offering a refined perspective and crucial supplement in the context of length instruction adherence. Specifically, a higher ω_1 emphasizes the model's accuracy contribution; ω_2 controls the negative impact of the violation rate; and ω_3 accounts for the influence of average response length. This formula enables a multi-dimensional assessment of models' adherence to length instructions, overcoming the limitations of single metrics and facilitating a holistic comparison of different models, thus providing a scientific basis for optimal model selection.

7. Conclusions

7.1. Limitations

Large language models exhibit differences in their ability to adhere to length instructions, and an inappropriate selection can lead to data errors. Therefore, the LIFT-COT method has a strong dependence on the model's ability to follow conditional instructions.

This paper sets length limits by word count, adjustable by characters or other metrics. Future generalization could include diverse wording for length instructions, like “100 characters max”. Other types of instructions, like “at least 100 words”, are unexplored. While addressing length bias, it may stem from human preference for detailed answers, enhancing model–human alignment. This research supports this alignment. Longer answers may also result from higher computational load, warranting future analysis.

7.2. Conclusions and Future Work

To address length deviation in general instruction following, we integrate COT into existing length instructions and evaluate model response within specified limits. Using MT-Bench-LI and AlpacaEval-LI benchmarks, we find that top models with only length instructions fail to follow them. Thus, we propose the LIFT-COT method, introducing COT based on different length limit instructions to improve output length control. Additionally, adding COT to existing length-limited general instructions gives users higher controllability, security, and stability in practical cases, enhancing the user experience. In wireless networks with limited bandwidth and resources, precise control of model output length ensures more efficient communication and computation.

Looking ahead, given mobile phones’ performance differences as wireless network terminals, optimizing the LIFT-COT method for various phone types is crucial. Hardware variances impact its efficiency and accuracy, so a dynamic adaptation mechanism based on real-time phone performance to adjust thought chain complexity and length control is key to stable operation. Moreover, considering edge computing and lightweight needs, more edge-trained lightweight models should be introduced in the future, along with empirical evaluations of system performance in real edge environments. Future research should also focus on combining LIFT-COT with the RLHF mechanism for multimodal (text + image) generation. Using RLHF’s precise feedback to enhance length control is expected to expand functional boundaries, enabling more possibilities for content creation and interaction in complex scenarios.

Author Contributions: P.C.: Conceptualization, Methodology, Data Curation, Analysis, Writing—Original Draft. Z.L.: Review, Supervision, Validation, Guidance. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Technical Foundation of the Equipment Development Department under Grant No. JZX7J202104ZL002600.

Data Availability Statement: Publicly available datasets were analyzed in this study. AlpacaEval-LI and MT-Bench-LI datasets can be found here: [https://github.com/facebookresearch/RAM/tree/main/projects/length_instruct] (accessed on 14 September 2024).

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A

Algorithm A1. A Word Count Function We Use

Start

Input: text (a string)

Output: word_count (an integer representing the number of words excluding punctuation)

Procedure CountWords

BEGIN

Algorithm A1. *Cont.*

```

// Initialize an empty list to store words
words ← []
// Retrieve a set of all punctuation characters
punctuation ← GetPunctuationSet()
// Tokenize the input text into individual words
tokenized_words ← WordTokenize(text)
// Iterate over the list of tokenized words
FOR each word IN tokenized_words DO
    // If the word is not a punctuation character, add it to the words list
    IF word NOT IN punctuation THEN
        words.APPEND(word)
    END IF
END FOR
// Calculate the number of words in the words list
word_count ← LENGTH(words)
// Return the count of words
RETURN word_count
END
End Algorithm

```

Algorithm A2. Optimize_LLM_Deployment

```

Start
// Initialize parameters and load data
Initialize parameters:
    - U_CPU_max (maximum CPU usage)
    - M_max (maximum memory occupation)
    - B_max (maximum bandwidth occupation)
    - D_max (maximum data transmission delay)
    - Accuracy_min (minimum acceptable accuracy)
    - epsilon (convergence threshold)
    - T_max (maximum number of iterations)
    - x (decision variables including COT guidance intensity parameter s and other length
control parameters)
    - P_prev (previous performance metric, initialized to 0)
Load pre-trained LLM model:
    - model ← load_pretrained_LLM() // Load the pre-trained Large Language Model
Load training and testing datasets:
    - train_data, test_data ← load_data() // Load training and testing datasets
// Main loop starts
While True:
    // Generation Phase: Generate outputs using the current model parameters
    generated_outputs ← model.generate(test_data, x) // Generate outputs based on
current parameters
    // Evaluation Phase: Evaluate the generated outputs against constraints
    N_new ← evaluate_outputs(generated_outputs, test_data, U_CPU_max, M_max,
B_max, D_max) // Count valid outputs
    // Performance Metric Calculation: Calculate the performance metric P

```

Algorithm A2. Cont.

```

P ← calculate_performance_metric(N_new, generated_outputs) // Calculate
overall performance metric
// Check Termination Conditions: Break the loop if convergence or max iterations
reached
If check_termination(P, P_prev, epsilon, t, T_max) Then
    Break
// Feedback and Adjustment Phase: Update model parameters based on
performance
gradient ← calculate_gradient(P, x) // Compute gradient for parameter update
x ← update_parameters(x, gradient, t) // Update parameters using gradient
descent
t ← t + 1 // Increment iteration count
P_prev ← P // Update previous performance metric
// Output Optimization Results: Return the optimized parameters and performance
Return x, N_new, P

```

Appendix B

Scatter plots of the remaining response word counts of large language models in AlpacaEval-LI & MT-Bench-LI.

- Scatter plots of AlpacaEval-LI.

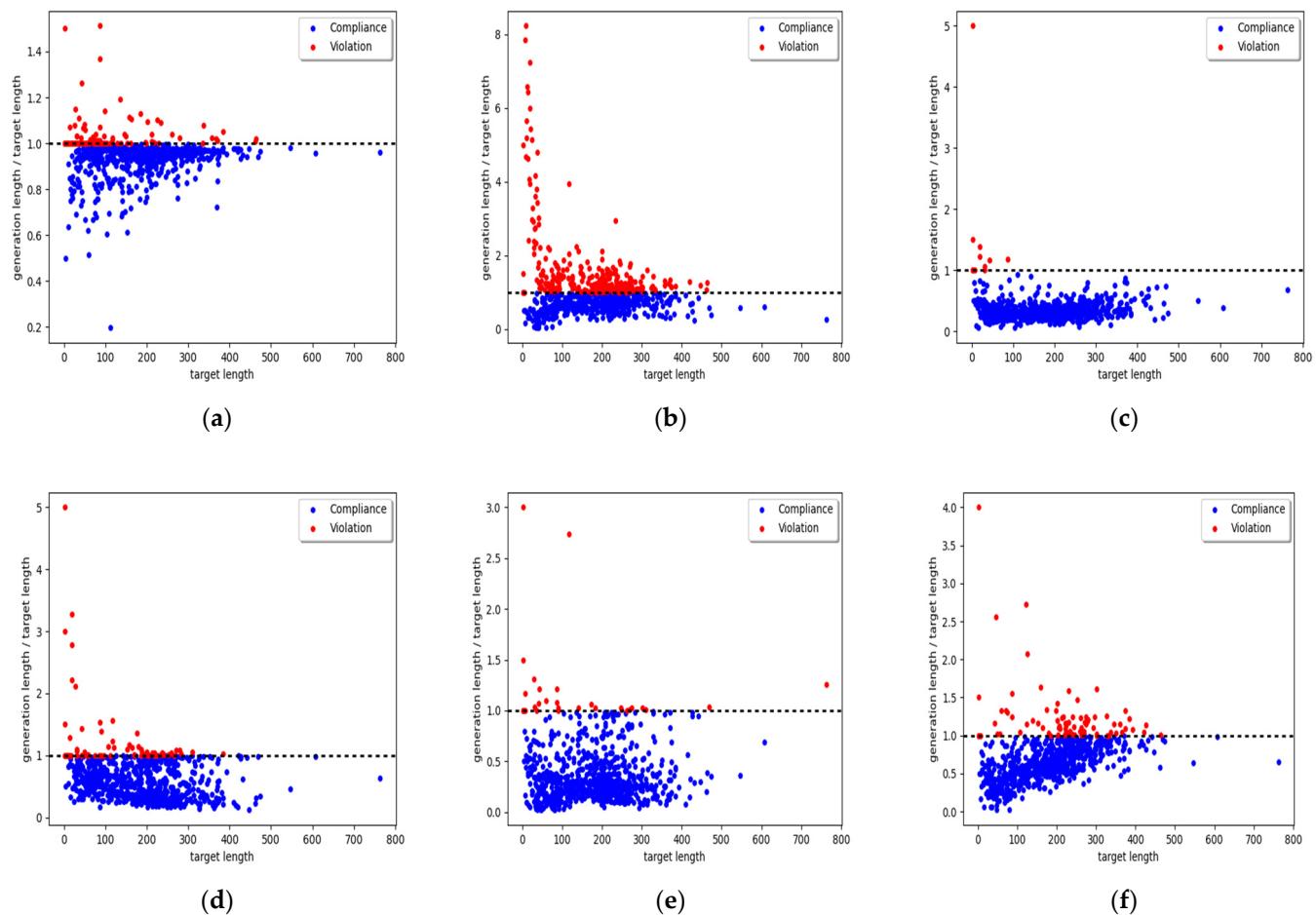
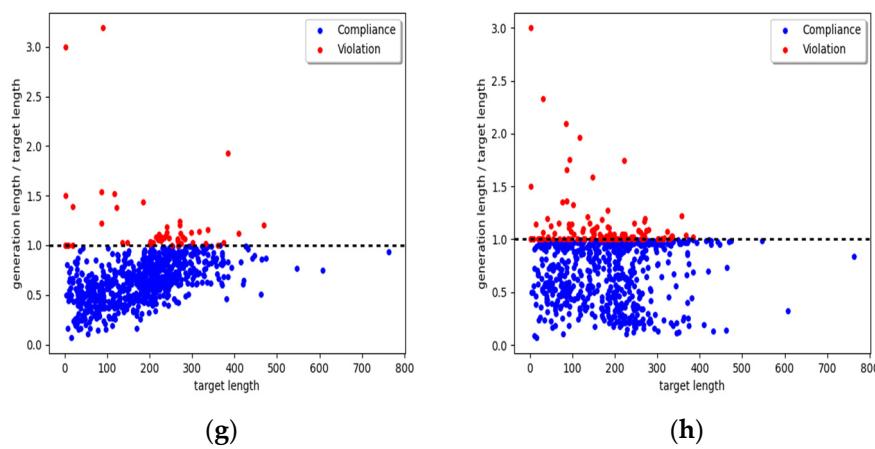


Figure A1. Cont.

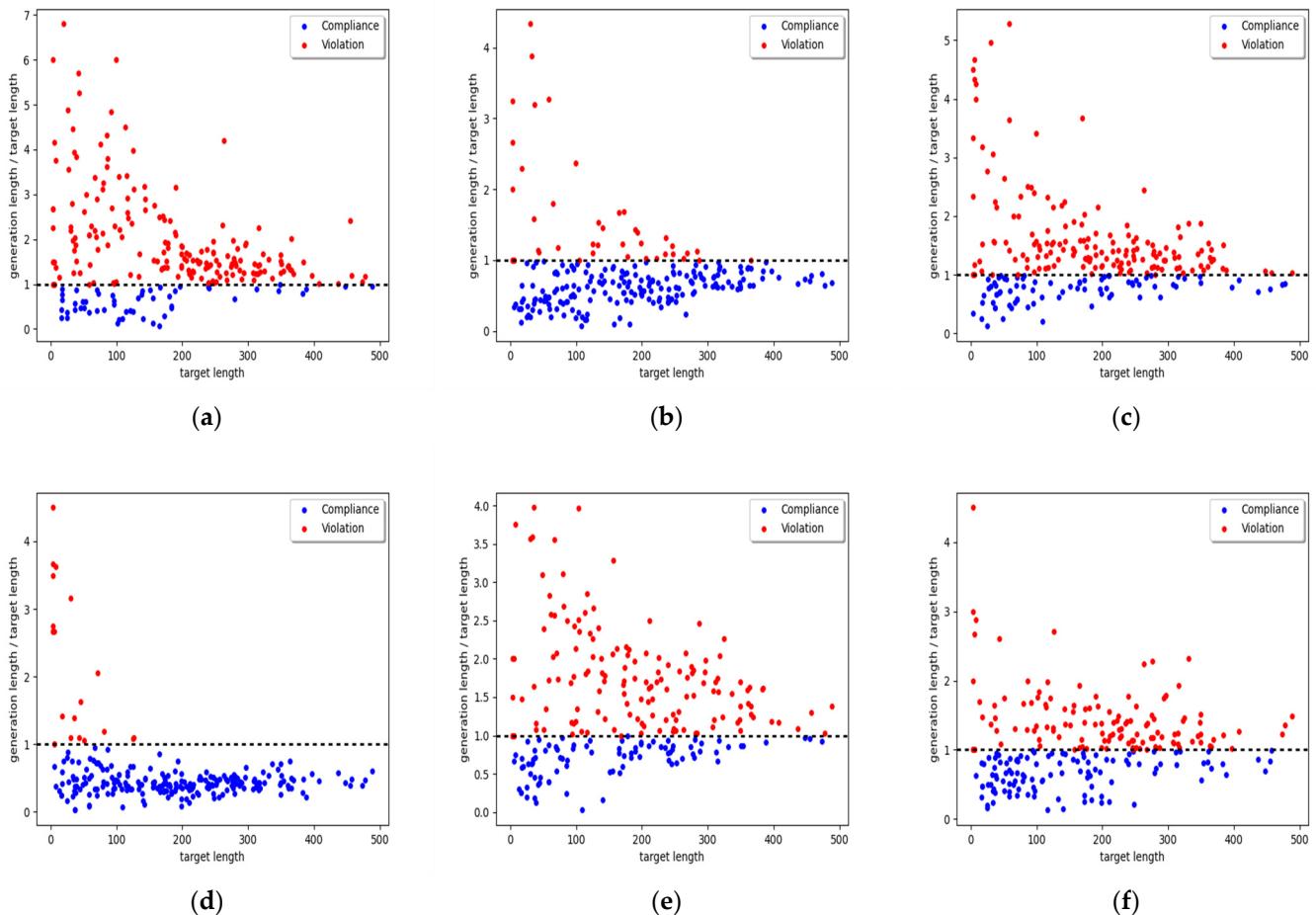


(g)

(h)

Figure A1. Figure (a–h) represent, respectively, the scatter plots of the baseline model (with just length instruction added), ERNIE Bot 3.5, Doubao, Codestral, Mistral Nemo, Kimi, Tongyi Qianwen 2.5, and iFlytek Spark.

- Scatter plots of MT-Bench-LI



(a)

(b)

(c)

(d)

(e)

(f)

Figure A2. Cont.

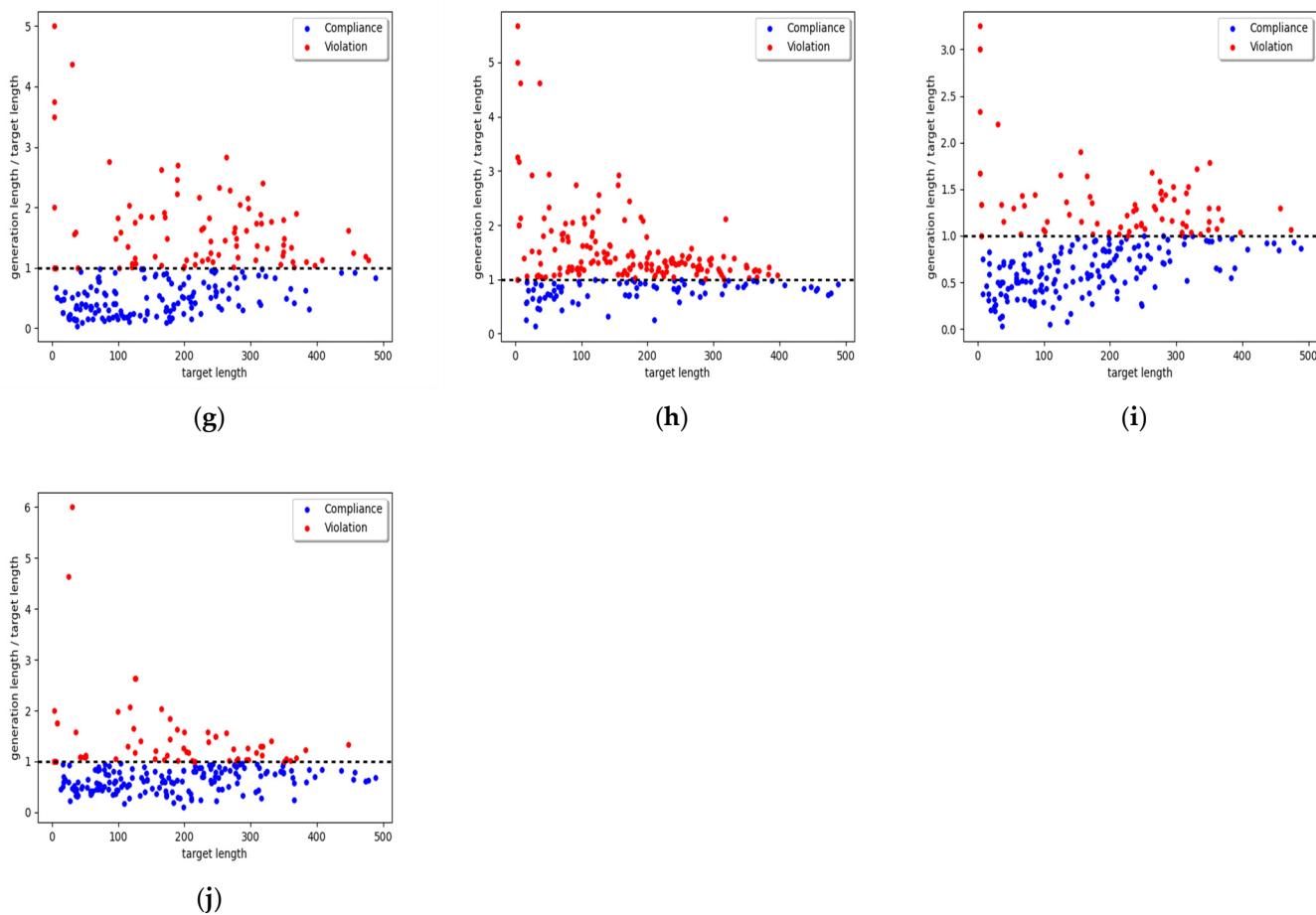


Figure A2. Figure (a–j) represent, respectively, the scatter plots of ERNIE Bot 3.5, ERNIE Bot 4.0, ERNIE Bot 4.0 Turbo, Doubao, Mistral Large2, Codestral, Mistral Nemo, Kimi, Tongyi Qianwen 2.5, and iFlytek Spark.

References

- OpenAI. OpenAI o1 System Card. 12 September 2024. Available online: <https://www.studocu.com/in/document/university-of-mumbai/bachelor-of-engineering/o1-system-card-amammsnske/121416702> (accessed on 12 September 2024).
- Wahidur, R.S.M.; Tashdeed, I.; Kaur, M.; Lee, H.N. Enhancing Zero-Shot Crypto Sentiment With Fine-Tuned Language Model and Prompt Engineering. *IEEE Access* **2024**, *12*, 10146–10159. [CrossRef]
- Naganawa, H.; Hirata, E. Enhancing Policy Generation with GraphRAG and YouTube Data: A Logistics Case Study. *Electronics* **2025**, *14*, 1241. [CrossRef]
- Dubois, Y.; Li, C.X.; Taori, R.; Zhang, T.; Gulrajani, I.; Ba, J.; Guestrin, C.; Liang, P.S.; Hashimoto, T.B. Alpacafarm: A Simulation Framework for Methods that Learn from Human Feedback. *Adv. Neural Inf. Process. Syst.* **2023**, *36*, 30039–30069.
- Dubois, Y.; Galambosi, B.; Liang, P.; Hashimoto, T.B. Length-Controlled Alpacaeval: A Simple Way to Debias Automatic Evaluators. *arXiv* **2024**, arXiv:2404.04475.
- Ge, S.; Sun, Y.; Cui, Y.; Wei, D. An Innovative Solution to Design Problems: Applying the Chain-of-Thought Technique to Integrate LLM-based Agents with Concept Generation Methods. *IEEE Access* **2024**, *13*, 10499–10512. [CrossRef]
- Zheng, L.; Chiang, W.L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E.; et al. Judging LLM-as-a-Judge with MT-bench and Chatbot Arena. In Proceedings of the 37th Conference Neural Information Processing Systems Datasets and Benchmarks Track, New Orleans, LA, USA, 10 December 2023.
- Yuan, W.; Kulikov, I.; Yu, P.; Cho, K.; Sukhbaatar, S.; Weston, J.; Xu, J. Following Length Constraints in Instructions. *arXiv* **2024**, arXiv:2406.17744.
- Singhal, P.; Goyal, T.; Xu, J.; Durrett, G. A Long Way to Go: Investigating Length Correlations in RLHF. *arXiv* **2023**, arXiv:2310.03716.
- Park, R.; Rafailov, R.; Ermon, S.; Finn, C. Disentangling Length from Quality in Direct Preference Optimization. *arXiv* **2024**, arXiv:2403.19159.

11. Schulhoff, S.; Ilie, M.; Balepur, N.; Kahadze, K.; Liu, A.; Si, C.; Li, Y.; Gupta, A.; Han, H.; Schulhoff, S.; et al. The Prompt Report: A Systematic Survey of Prompting Techniques. *arXiv* **2024**, arXiv:2406.06608.
12. Casper, S.; Davies, X.; Shi, C.; Gilbert, T.K.; Scheurer, J.; Rando, J.; Freedman, R.; Korbak, T.; Lindner, D.; Freire, P.; et al. Open Problems and Fundamental Limitations of Reinforcement Learning from Human Feedback. *arXiv* **2023**, arXiv:2307.15217.
13. Zhao, H.; Andriushchenko, M.; Croce, F.; Flammarion, N. Long is More for Alignment: A Simple but Tough-to-Beat Baseline for Instruction Fine-Tuning. *arXiv* **2024**, arXiv:2402.04833.
14. Anthropic. Claude 3. 2024. Available online: <https://www.anthropic.com/news/clause-3-family> (accessed on 12 December 2024).
15. Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F.L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. Gpt-4 Technical Report. *arXiv* **2023**, arXiv:2303.08774.
16. Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv* **2023**, arXiv:2307.09288.
17. Bai, Y.; Jones, A.; Ndousse, K.; Askell, A.; Chen, A.; DasSarma, N.; Drain, D.; Fort, S.; Ganguli, D.; Henighan, T.; et al. Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback. *arXiv* **2022**, arXiv:2204.05862.
18. Köpf, A.; Kilcher, Y.; von Rütte, D.; Anagnostidis, S.; Rui Tam, Z.; Stevens, K.; Barhoum, A.; Nguyen, D.; Stanley, O.; Nagyfi, R.; et al. Openassistant Conversations-Democratizing Large Language Model Alignment. *Adv. Neural Inf. Process. Syst.* **2023**, *36*, 47669–47681.
19. Fragakis, N.; Trichopoulos, G.; Caridakis, G. Empowering Education with Intelligent Systems: Exploring Large Language Models and the NAO Robot for Information Retrieval. *Electronics* **2025**, *14*, 1210. [CrossRef]
20. Chakraborty, N.; Ornik, M.; Driggs-Campbell, K. Hallucination Detection in Foundation Models for Decision-Making: A Flexible Definition and Review of the State of the Art. *arXiv* **2024**, arXiv:2403.16527. [CrossRef]
21. Düring, B.; Georgiou, N.; Merino-Aceituno, S.; Scalas, E. Continuum and Thermodynamic Limits for a Simple Random-Exchange Model. *Stoch. Process. Their Appl.* **2022**, *149*, 248–277. [CrossRef]
22. Wang, J. Hallucination Reduction and Optimization for Large Language Model-Based Autonomous Driving. *Symmetry* **2024**, *16*, 1196. [CrossRef]
23. Tan, J.; Chen, G.; Huang, Z.; Liu, H.; Ang Jr, M.H. E-GAIL: Efficient GAIL through including negative corruption and long-term rewards for robotic manipulations. *Appl. Intell.* **2025**, *55*, 633. [CrossRef]
24. Ji, Z.; Yu, T.; Xu, Y.; Lee, N.; Ishii, E.; Fung, P. Towards Mitigating Hallucination in Large Language Models via Self-Reflection. *arXiv* **2023**, arXiv:2310.06271.
25. Cauteruccio, F.; Marchetti, M.; Traini, D.; Ursino, D.; Virgili, L. Adaptive patch selection to improve Vision Transformers through Reinforcement Learning. *Appl. Intell.* **2025**, *55*, 607. [CrossRef]
26. Dimitriadis, D.; Tsoumacas, G. Enhancing Yes/No Questions Through Weak Supervision by Extracting Question Answering. *Appl. Intell.* **2023**, *53*, 27560–27570. [CrossRef]
27. Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; et al. Self-Refine: Iterative Refinement with Self-Feedback. *Adv. Neural Inf. Process. Syst.* **2023**, *36*, 46534–46594.
28. Yashwanth, Y.S.; Shettar, R. Zero and Few Short Learning Using Large Language Models for De-Identification of Medical Records. *IEEE Access* **2024**, *12*, 110385–110393. [CrossRef]
29. Sen, S.; Gonzalez, V.; Husom, E.J.; Tverdal, S.; Tokas, S.; Tjøsvoll, S.O. ERG-AI: Enhancing occupational ergonomics with uncertainty-aware ML and LLM feedback. *Appl. Intell.* **2024**, *54*, 12128–12155. [CrossRef]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.