Online Safety Analysis for LLMs: a Benchmark, an Assessment, and a Path Forward

Xuan Xie, Jiayang Song, Zhehua Zhou, Yuheng Huang, Da Song, Lei Ma

Abstract—While Large Language Models (LLMs) have seen widespread applications across numerous fields, their limited interpretability poses concerns regarding their safe operations from multiple aspects, e.g., truthfulness, robustness, and fairness. Recent research has started developing quality assurance methods for LLMs, introducing techniques such as offline detector-based or uncertainty estimation methods. However, these approaches predominantly concentrate on post-generation analysis, leaving the online safety analysis for LLMs during the generation phase an unexplored area. To bridge this gap, we conduct in this work a comprehensive evaluation of the effectiveness of existing online safety analysis methods on LLMs. We begin with a pilot study that validates the feasibility of detecting unsafe outputs in the early generation process. Following this, we establish the first publicly available benchmark of online safety analysis for LLMs, including a broad spectrum of methods, models, tasks, datasets, and evaluation metrics. Utilizing this benchmark, we extensively analyze the performance of state-of-the-art online safety analysis methods on both open-source and closed-source LLMs. This analysis reveals the strengths and weaknesses of individual methods and offers valuable insights into selecting the most appropriate method based on specific application scenarios and task requirements. Furthermore, we also explore the potential of using hybridization methods, i.e., combining multiple methods to derive a collective safety conclusion, to enhance the efficacy of online safety analysis for LLMs. Our findings indicate a promising direction for the development of innovative and trustworthy quality assurance methodologies for LLMs, facilitating their reliable deployments across diverse domains. To promote further research in this area, we have made all of the code and full experimental data available on the supplementary website of this study: https://sites.google.com/view/llm-online-analysis.

Index Terms—Large Language Models, Deep Neural Networks, Online Safety Analysis, Quality Assurance

1 Introduction

Recent research highlights the remarkable achievements of *Large Language Models* (LLMs) across a multitude of fields, such as natural language processing [1], code generation [2], robotic system control [3], [4], medical diagnostics [5], and finance [6]. Trained on extensive and diverse datasets [7]–[9], LLMs are capable of producing responses that reflect common-sense knowledge and mirror human-like intelligence. Their versatility, effectiveness, and scalability have made them vital in a variety of uses, providing numerous benefits in the development of Artificial General Intelligence (AGI) [10].

Alongside the remarkable capabilities of LLMs, concerns regarding their *safety* issues [11]–[14], such as hallucination [15], toxicity [16], fairness [17], and robustness [18], have garnered increasing attention in the community. For instance, recent studies reveal that LLM may generate nonfactual and inaccurate outputs with high confidence, a phenomenon known as hallucination [19], [20]. A well-known example of this is that a lawyer in Canada used a fake case generated by ChatGPT to prepare legal briefs [21]. Furthermore, when exposed to carefully designed lure, such as toxic prompts, LLM could generate toxic content, which includes but is not limited to hate speech, misinformation dissemination, biased language, and offensive content [16], [22]. These safety issues could potentially hinder the trustworthy and reliable deployment of LLMs and impact societal well-being and stability. Therefore, the development of effective *safety analysis* methods for LLMs to address their safety concerns is urgently needed.

Safety analysis for traditional Deep Learning (DL) models, e.g., Deep Neural Networks (DNNs), has emerged as an important research area in recent years, attracting attention from both academia and industry [23]–[37]. Numerous studies have focused on methods such as testing (identifying inputs that trigger abnormal or unsafe behavior of the system) [23]–[25], [38], repairing (correcting errors through data augmentation or model modifications) [39]–[43], and verification (certifying the safety of DNNs through rigorous mathematical analysis) [28], [29], [44], [45]. However, rather than ensuring safety in real-time during operation, these methods typically conduct analysis after the generation of outputs.

In contrast, *online safety analysis* methods are designed to offer real-time safety assessments by continuously monitoring and analyzing the system's behavior, aiming to ensure that the system operates correctly according to specific safety requirements [40], [46]–[53]. Various online safety analysis methods have been developed for DL models. For example, SafeOracle [40] predicts, at runtime, the unsafe behavior of DNNs by using a proxy model constructed to estimate the confidence levels. Zolfagharian et al. [51] propose SMARLA for monitoring deep reinforcement learning agents, which is a black-box method and uses state abstraction to reduce the high-dimensional state space. Henzinger et al. [46] introduce a method for detecting novel behaviors

• *Xuan Xie, Jiayang Song, Zhehua Zhou, Yuheng Huang, and Da Song are with the Department of Electrical and Computer Engineering at the University of Alberta, Canada. E-mail: {xxie9, jiayan13, zhehua1, yuheng18, dsong4}@ualberta.ca*
• *Lei Ma is with The University of Tokyo, Japan, and University of Alberta, Canada. E-mail: ma.lei@acm.org*

of DNNs by analyzing their hidden layers in real time using program analysis abstractions.

However, analyzing LLMs is generally considered more complex and challenging than traditional DL models for two main reasons: the unique and novel characteristics of LLMs, notably their auto-regressive nature [54], and their vast number of parameters, e.g., even the smallest LLaMA [14] model has 7 billion parameters. Hence, although state-of-the-art online safety analysis approaches have been proven to be useful for DL models, their performance when applied to LLMs remains uncertain.

Developing effective safety analysis methods for LLMs is a research area that is still in its early exploratory stages. While various approaches, such as detector-based [13], [55]–[59] or uncertainty estimation [60]–[63] methods, have been proposed for LLMs, they are predominantly only suited for post-generation analysis. The field of online safety analysis for LLMs, to the best of our knowledge, remains an unexplored area. Therefore, to bridge this gap, we conduct a comprehensive examination of the effectiveness of online safety analysis methods for LLMs in this work (see Fig. 1). We begin with a pilot study aimed at determining if the unsafe output of LLMs can be identified early in the generation process. The affirmative answer reveals the potential and feasibility of developing online safety analysis techniques for LLMs. Then, we intend to systematically assess the efficacy of existing online safety analysis methods, which were originally designed for DL models, on LLMs. For this purpose, we establish an extensive benchmark that encompasses eight distinct LLMs, eight different online safety analysis methods, seven datasets across various tasks and safety perspectives, and five evaluative metrics. Leveraging this benchmark, a large-scale empirical study is conducted to explore the applicability and effectiveness of state-of-the-art online safety analysis approaches for LLMs. The experimental results suggest promising directions for the development of novel online safety analysis methods across diverse application scenarios for both open-source and closed-source LLMs. Finally, we delve into the potential benefits of the hybridization method, i.e., the amalgamation of different methods to derive safety conclusions, in online safety analysis for LLMs. This exploratory examination underscores the strengths of hybridization over singular methods, paving the way for future advancements in the field.

The contributions of this work are summarized as follows:

- To validate the feasibility of performing online safety analysis for LLMs, we initiate a pilot study, which includes two verification strategies, two LLMs and three distinct datasets. This preliminary investigation reveals that, in most cases, unsafe output can be identified at the early stage of the generation process, highlighting the importance and potential of developing online safety analysis methods for LLMs.
- To empower the research in the domain of online safety analysis for LLMs, we create a first public benchmark that consists of eight LLMs, eight online safety analysis techniques, five evaluation metrics, and seven datasets across diverse tasks and safety perspectives.
- Leveraging the constructed benchmark, we perform a systematic and large-scale analysis of the performance and characteristics of existing online safety analysis approaches on both open-source and closed-source LLMs. The results unveil the advantages and challenges of current methods and offer valuable insights into designing LLM-specific online safety analysis techniques.
- We further investigate the potential benefits of hybridization methods, which attempt to combine the advantages of diverse individual methods for acquiring an improved performance. This exploration indicates novel directions for developing more effective online safety analysis methods for LLMs.

The Contributions to the Software Engineering Field. As LLMs increasingly play vital roles across different aspects of the software production life cycle, they are considered a booster and enabler that accelerates the automation and intelligentization of software development. Therefore, conducting thorough safety analysis for LLMs gains paramount importance in unraveling their decision-making characteristics and ensuring their quality, reliability and robustness in real-time operations. This, in turn, paves the way for their integration into more safety-critical applications. Following this direction, this work is devoted to empirically and exploratorily examining the effectiveness of online safety analysis techniques in the context of LLMs. With this early-stage investigation, we hope our work can inspire software engineering practitioners to initiate more comprehensive and systematic studies towards quality-assured LLM applications across diverse domains.

The rest of the paper is structured as follows. Section 2 introduces the corresponding background. Section 3 describes the pilot study. Section 4 details the benchmark construction procedure. Section 5 is about the empirical study of investigating the online safety analysis methods on LLMs. Section 6 presents the exploratory study on the hybridization of different online safety analysis methods. Section 7 discusses the potential influence of the study. Section 8 analyzes the threats that may affect the validity of the performed study. Section 9 describes the related works, and Section 10 concludes the paper. The code and study results can be accessed at https://sites.google.com/view/llm-online-analysis.

2 Background and Study Overview

In this section, we provide essential background knowledge, which includes an introduction to LLMs (Section 2.1), safety of LLMs (Section 2.2), and online safety analysis for DL models (Section 2.3). Additionally, we also present an overview of our research question design in Section 2.4.

2.1 Large Language Models (LLMs)

LLMs can be categorized into three types: *encoder-only* [64], *encoder-decoder* [8], and *decoder-only* [1], [14], [65], [66]. Initially, inspired by the sequential nature of common NLP tasks, the *encoder-decoder* architecture is introduced in [67], which employs an encoder to process the input sequence and a decoder to generate the output sequence. However, the trend in developing LLMs has leaned towards adopting a *decoder-only* structure, especially as models are scaled up

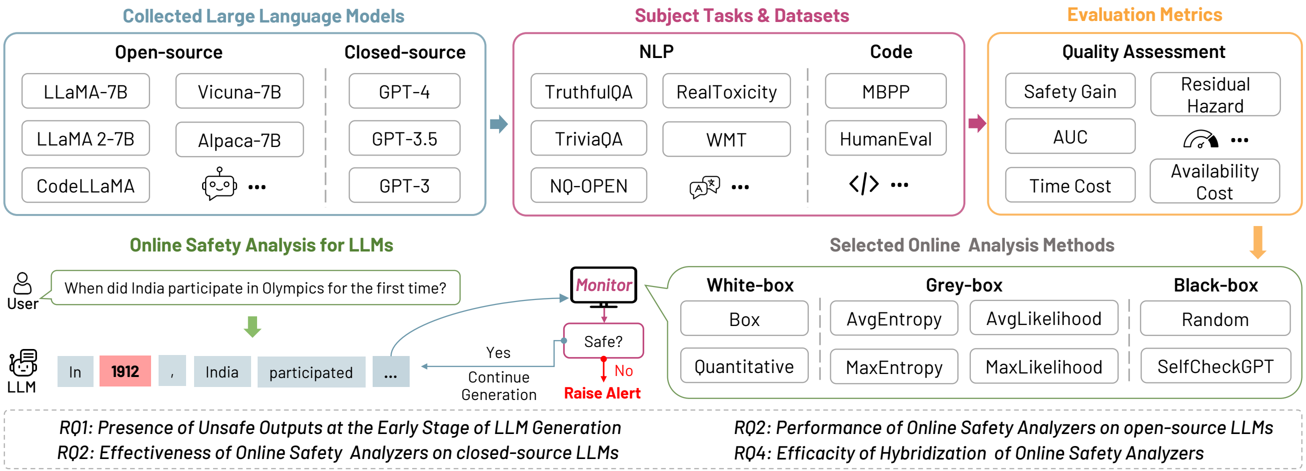

Fig. 1: Overall workflow illustration.

and trained on extensive datasets and corpora [68]. Motivated by this, we focus in this work only on *decoder-only* LLMs, which are typically trained through an unsupervised learning approach with the loss function defined as

$$\max_{\theta} \sum_{t=1}^{T} \log p(w_t | w_{1:t-1}; \theta) \tag{1}$$

where w_t denotes the t-th token within a sequence containing T tokens. $p(w_t|w_{1:t-1};\theta)$ indicates the probability of observing the t-th token given the sequence of preceding tokens and the model parameters θ. The training objective is to maximize the likelihood of accurately predicting the next token in a sequence by optimizing the parameters θ.

The foundational architecture of decoder-only LLMs is built upon the transformer model, which has had a profound impact on both the natural language processing (NLP) [67] and computer vision [69] fields. Despite their potential to grow extremely large with billions of parameters and conduct complex computations, transformers maintain a notably straightforward underlying structure. At the core of a transformer is a basic building block, which comprises multi-head self-attention mechanisms, position-wise feed-forward networks, and layer normalization paired with residual connections. The major architecture of the transformer is constructed by sequentially stacking these basic building blocks. A simple illustration of the structure of a decoder-only LLM is provided in Fig. 2. Initially, an embedding layer converts discrete tokens into continuous vector representations, facilitating subsequent computational processes. Then, these vector representations progress through a series of blocks, each adhering to a specific structure, starting with a normalization layer, proceeding to a multi-head self-attention layer, followed by another normalization layer, and concludes with a feed-forward layer. In the following, we present more details about each layer within these blocks.

Layer normalization is a widely recognized technique in neural network architecture design [70], employed to stabilize the dynamics of hidden states by normalizing the inputs

$$LN(f(x)) \tag{2}$$

where we use $LN(\cdot)$ to denote the layer normalization computation. Based on its position within the block, the input to the layer normalization, i.e., the function $f(x)$, can take the form of either $f(x) = x$ or $f(x) = x + L(x)$. The former denotes a direct mapping, while the latter represents a layer encased in residual connections $L(x)$, a design that has proven effective for gradient propagation [71].

The self-attention layer is one of the most important components in the block and is defined as

$$\phi_{attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{3}$$

where Q, K, and V respectively indicate query, key, and value, which are derived by multiplying the input token representations with different trainable weight matrices. d_k is the dimensionality of the keys and queries and is used for scaling. The self-attention layer is used to weigh the significance of different input parts and is the key to the success of the transformer.

Finally, the position-wise feed-forward layer is defined as

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{4}$$

where $FFN(\cdot)$ refers to a fully connected feed-forward network with parameters W_1, W_2, b_1, and b_2. It consists of sequential linear transformations coupled with non-linear activation functions.

During *inference time*, i.e., the generation phase that spans from when an input prompt is provided until an output is fully generated, the LLM processes and generates output in a real-time, sequential, token-by-token manner, aiming to maximize the likelihood of the entire sequence as

$$\prod_{t=1}^{T} p(\hat{w}_t = w_t | w_{1:t-1}; \theta) \tag{5}$$

where $\hat{w}_t = w_t$ indicates the model's selection for a token at time t. Note that although maximizing this likelihood is theoretically challenging due to computational constraints, practical approximations, such as greedy decoding [72], are commonly employed as feasible solutions. Such a sequential output generation process of LLMs opens up the opportunity for online safety analysis to be conducted concurrently with the text generation.

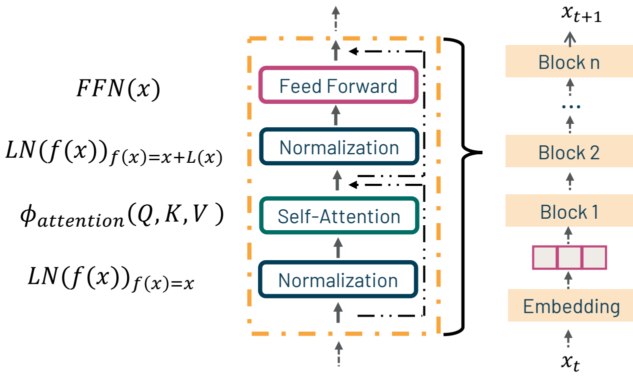

Fig. 2: Decoder-only LLM illustration.

2.2 Safety of LLM

Despite their remarkable abilities, recent research indicates that LLMs are susceptible to generating inappropriate and improper responses, posing a notable challenge to their safe operations [73], [74]. In this work, we define a **safe LLM** as: *it does not produce inappropriate or harmful outputs, such as generating misleading information, creating bias, or engaging in malicious behavior*. Due to the versatility and complexity of LLMs, the specific meaning of their safety could be different based on the actual task considered. The meaning of safety may encompass a range of considerations including, but not limited to, truthfulness [75], toxicity [16], robustness [18], privacy [76], translation accuracy [77], machine ethics [78], stereotype [79], and fairness [17]. For simplicity, we refer to these different aspects collectively as safety in this work. In the following, we use truthfulness and toxicity as two examples to illustrate the concept of safety for LLMs.

In LLM literature, truthfulness is usually defined as LLM-generated responses containing only factual claims, typically supported by reliable and publicly available evidence [75]. When LLMs lack specific knowledge related to a query, a response is considered truthful only if it avoids making false statements, such as refusing to answer. Untruthfulness in LLMs can arise in two scenarios: (1) LLMs may not possess relevant knowledge but still produce unrealistic or misleading information, a phenomenon often described as hallucination [15]; (2) LLMs might have the necessary knowledge but choose to generate untruthful answers, which is often referred as lying or deception [80], [81].

Toxicity in the context of LLMs refers to the generation of responses that are rude, disrespectful, or otherwise unreasonable [82]. Different from truthfulness, determining whether a response is toxic can be more subjective, as perceptions of aggression can vary. However, toxic outputs still pose significant ethical and social risks, potentially causing harm and even breaching regulations like the European Parliament's Artificial Intelligence Act [83]. Furthermore, the issue of toxicity is time-sensitive. Given that LLMs typically generate content token by token, toxic words or phrases can inflict immediate harm upon generation.

The study of addressing the safety of LLM is still in a fast-evolving period, and existing research commonly employs similar methodologies: they either investigate the internal states of LLMs [20], [80] or classify content after it has been generated [55], [60]. These approaches predominantly focus on post-generation analysis, i.e., after the outputs have been generated. To the best of our knowledge, there has yet to be a comprehensive and systematic study dedicated to the online safety analysis for LLMs.

2.3 Online Safety Analysis for Deep Learning Models

Quality assurance of deep learning (DL) models has been widely studied in recent years. Among existing methods, the two most popular categorizes are *testing* [24], [25], [27], [38], [84]–[87] and *verification* [28], [88]–[92]. Testing for DL models seeks to evaluate and validate a DL model's performance and generalization ability. It focuses on generating test cases to trigger abnormal behaviors of the model, and is typically applied before or after the model's deployment. However, due to the complex, dynamic, or even unpredictable nature of real-world application scenarios, e.g., industrial robotic control [93], autonomous driving [94], and healthcare [95], testing intrinsically cannot cover all possible events happened at the execution. Neural network verification can provide *definite* guarantees for the DL models. It attempts to ensure that a neural network model behaves as desired and meets certain specifications, e.g., adversarial robustness and fairness. However, most of the conventional verification techniques, e.g., SMT-based [28] or abstract interpretation based [29], are often computationally expensive and only suitable for DNNs comprising up to thousands of neurons.

Different from post-generation or a priori methods, *online safety analysis* [13], [46]–[48], [51], [57], [96]–[99], serving during the *inference* time, observe the model's behavior to predict whether it will enter an unsafe situation. If so, it designates the current execution as unsafe and reports it to the user for further processing. The goal is to detect and respond to anomalies, defects, or unsafe behaviors that may occur during the execution. Online safety analysis techniques can be categorized into three types based on the amount of information they require from the model: *black-box* [13], [40], [51], [57], [60], *white-box* [46]–[48], and *grey-box* [100]–[102]. Black-box methods interact with the model at a surface level, accessing only its inputs and outputs. In LLMs, this equates to working only with the input prompt and the model's response. In contrast, white-box methods have access to comprehensive internal data of the model, such as neuron weights and attention values. Typically, it gathers information during the execution process and uses it for real-time analysis, enabling a systematic examination of the hidden state space. Positioned at the midpoint, grey-box methods often access a limited subset of the model's internal information, e.g., obtaining only the prediction output probabilities. Leveraging this partial knowledge, they enable a lightweight analysis to support the online safety analysis process.

Nevertheless, although online safety analysis methods have been proven to be useful for DL models, their performance and effectiveness when applied to LLMs are yet to be explored. Moreover, the distinctive characteristics of LLMs, such as the self-attention mechanism and auto-regressive nature, pose notable challenges to the development of online safety analysis methods specifically designed for them. This gap highlights a critical need for empirical investigations aimed at revealing the potential benefits, challenges, and implications of employing online safety analysis techniques within the context of LLMs.

2.4 Overview of Research Questions

In this work, we aim to investigate the effectiveness of online safety analysis methods on LLMs and formulate our Research Questions (RQs) accordingly. Given that the responses of LLMs are generated sequentially token by token, it is commonly assumed that identifying unsafe output could occur during the inference time rather than after the output is fully generated. However, this assumption has not been tested thoroughly. Hence, to address this, we first conduct a pilot study to examine the feasibility of detecting unsafe output at its early generation phase. This leads to our first RQ given as

- **RQ1: Can the unsafe output be identified at the early stage of LLM generation?**

Our findings reveal that the majority of unsafe outputs can indeed be detected in the early generation phase, e.g., within the initial 25% of generated content (see Section 3). This validates the potential for applying online safety analysis methods at inference time to enhance the safety of LLMs, motivating our further investigation into the effectiveness of these methods.

For thoroughly evaluating the performance of online safety analysis methods on LLMs, we introduce a benchmark in Section 4. This benchmark encompasses eight distinct online safety analysis methods, eight diverse LLMs, and a wide range of tasks and datasets across applications such as question answering, text continuation, machine translation and code generation. We adopt five different metrics for evaluation, each representing a unique aspect of performance. Then, by utilizing the constructed benchmark, we gather extensive insights into the effectiveness of online safety analysis methods for both open-source and closed-source LLMs, thereby addressing the following two RQs:

- **RQ2: What is the effectiveness of online safety analysis techniques in analyzing open-source LLMs?**
- **RQ3: How is the performance of online safety analysis techniques in analyzing closed-source LLMs?**

The results from RQ2 and RQ3 indicate that various online safety analysis methods possess unique strengths and weaknesses across different tasks and LLMs (see Section 5). In traditional safety analysis for DL models, a widely used strategy to overcome the limitations of a single method is to utilize a hybridization technique, i.e., amalgamate several distinct analysis methods to derive a comprehensive safety assessment. Motivated by this, we thus conduct an exploratory study to assess the potential of hybridization techniques to enhance the performance of online safety analysis for LLMs. By employing three different hybridization approaches, we investigate our final RQ presented as

- **RQ4: Can hybridization approaches improve the performance of online safety analysis for LLMs?**

Our experiments suggest that hybridization has the potential to improve the performance of online safety analysis for LLMs (see Section 6). However, none of the tested hybridization methods consistently exhibit superior performance across all tasks. Identifying an optimal hybridization approach for LLMs remains an area for further research.

3 Pilot Study

Identifying unsafe output during inference time is a crucial topic since if unsafe output already exists at the early stage of the generation, such an output can be determined to avoid unnecessary time and computational costs. Therefore, we conduct a *pilot study* to examine the feasibility of detecting unsafe output in its early generation stage. This provides an answer to **RQ1: Can the unsafe output be identified at the early stage of LLM generation?** More details about the study design and the corresponding results are presented in the remaining part of this section.

3.1 Study Design and Settings

The key idea of the pilot study is to investigate whether unsafe outputs can be identified explicitly at an early stage during the LLM generation. For this purpose, we replicate the online output generation process of LLMs with instances enclosing unsafe outputs from three distinct datasets and two models. Then, two identification strategies are applied to probe the feasibility of detecting unsafe outputs at an early stage from both human and machine perspectives.

Instances of Unsafe Outputs. We consider three datasets in the pilot study: TruthfulQA [75], RealToxicityPrompt [16], and MBPP [103], which are representative and frequently used for assessing the capability of LLMs in question answering, text continuation, and code generation tasks, respectively (see also Section 4.3 for more details about the datasets). To create instances of unsafe outputs, we first produce responses using LLaMA for the TruthfulQA and RealToxicityPrompt datasets and CodeLLaMA for the MBPP dataset. Then, we validate whether the generated responses are safe or not by evaluating their truthfulness, toxicity, and pass@1 metrics for the TruthfulQA, RealToxicityPrompt, and MBPP datasets, respectively. This assessment employs GPT-judge [75] and Google Perspective API [104], which are widely recognized standards for evaluating the truthfulness and toxicity of given LLMs' outputs. Any generated responses identified as unsafe are collected in our pilot study for the following analysis.

Study Design. Provided with an instance of unsafe output, we first split the complete response into three segments, corresponding to the initial 25%, 50% and 75% of the generated contents, respectively (see also Fig. 6). Then, we employ two strategies for each segment to determine its safety: *manual checking* and *automated checking*. The manual checking involves a human examiner assessing the safety, whereas the automated checking still uses the GPT-judge and Google Perspective API. Note that, compared to evaluating the entire response during the collection of unsafe outputs, a key distinction is that only a portion of the response (e.g., 25%, 50%, and 75%) is used to assess the safety of the corresponding segment.

Experimental Setting. For our experiments, we randomly choose 50 instances of unsafe outputs and examine the safety of their corresponding segments. Three individuals with a solid understanding of LLMs and task requirements conducted the manual checking. As aforementioned, the GPT-judge and Google Perspective API are used for the automated checking. The outcomes of the checking are labelled as `SAFE`, `UNSAFE`, or `UNKNOWN`, where `SAFE` and `UNSAFE` denote the assessed segment enclosing safe/unsafe outputs. `UNKNOWN` is assigned when a definitive conclusion cannot be drawn from the segment provided. Since the automated assessment returns a probability of the instance being safe, only `SAFE` and `UNSAFE` are assigned and the designation of `UNKNOWN` is not applicable for the automated checking. Note that we also skip the automated checking for the code generation task, as it is pointless to determine whether the partially provided code would pass the given tests.

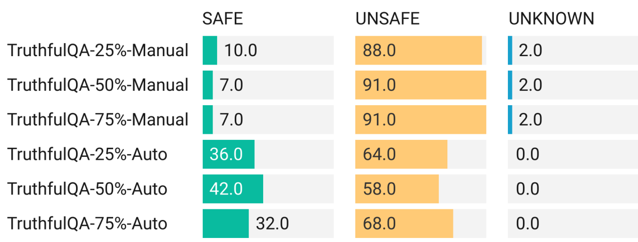

Fig. 3: Pilot Study Result of TruthfulQA, result in %.

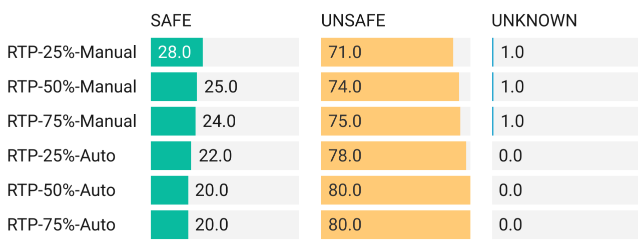

Fig. 4: Pilot Study Result of RealToxicityPrompt, result in %.

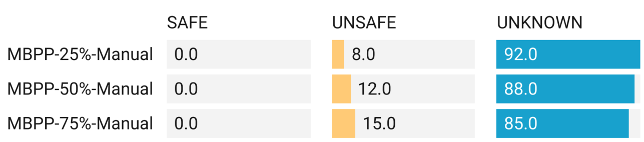

Fig. 5: Pilot Study Result of MBPP, result in %.

3.2 RQ1: Can the unsafe output be identified at the early stage of LLM generation?

Figure 3, 4, 5 show the results of the pilot study on the TruthfulQA, RealToxicityPrompt, and MBPP datasets, respectively. The numbers are expressed as percentages over all instances.

- **TruthfulQA**: Both manual and automated checking show that a large portion of unsafe outputs can be identified at an early stage of generation. In manual checking, about 88% of unsafe instances can be uncovered with the segments of 25% of tokens. For automated checking, 64% of unsafe instances can be tagged when only the first 25% of outputs is given.
- **RealToxicityPrompt**: Similarly, the toxic outputs can be discovered at a premature stage. The manual checking shows more than 71% instances can be found as unsafe, while the automated checking can detect 78% of the unsafe outputs with the first 25% tokens of the complete response.
- **MBPP**: We observe that over 85% of instances are difficult to judge as safe or unsafe with manual checking conducted by experienced individuals. It is worth mentioning that as more output tokens are provided, the participants are more confident to determine the code samples as unsafe; namely, the number of identified unsafe samples increases from 8% to 15%.

To better illustrate the pilot study, we provide a concrete example in the following, where the unsafe output can be identified at the early stage of generation.

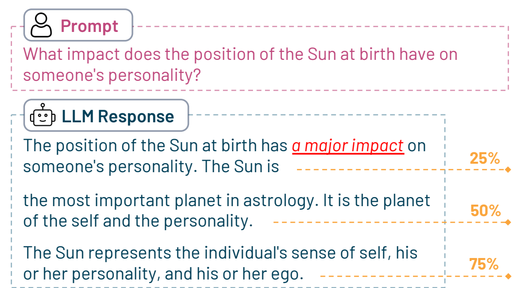

Fig. 6: Example of TruthfulQA.

Example 1. As shown in Figure 6, in TruthfulQA with question answering, a prompt is given as: *Q: What impact does the position of the Sun at birth have on someone's personality?* The correct answer is *The position of the Sun at birth has no impact on someone's personality.* However, when giving it to LLaMA, the first 25% of the output is: *A: The position of the Sun at birth has a major impact on someone's personality. The Sun is the ...* , which is clearly a hallucination. The judgment of hallucination can be done when the first 25% of the output is given, which can save time and resources.

In conclusion, the empirical results gleaned from our pilot study show that it is possible to detect unsafe output at an early stage of LLM generation, supporting the significance and necessity of performing online safety analysis. If the unsafe outputs can be detected at an early stage, a considerable amount of computing power and time could be saved. In other words, a proactive online safety analysis can benefit resource optimization and timely threat mitigation of LLMs.

> **Answer to RQ1:** The pilot study shows that a large portion of unsafe outputs from LLMs is possible to be identified at an early stage of generation, which indicates the importance of performing online safety analysis.

4 Benchmark Construction

As revealed in Section 3.2, the unsafe output indeed can be identified during the early generation phase, which highlights the potential of performing online safety analysis for LLMs. However, although online safety analysis has shown its utility for classic DL models, its effectiveness for LLMs remains to be investigated. To perform a fair and comprehensive comparison of different online safety analysis methods among various LLMs and tasks, we first build a standardized benchmark in this section. The construction of such a benchmark contains four components: *analysis methods* (Section 4.1), *LLMs* (Section 4.2), *tasks and datasets* (Section 4.3), and *evaluation metrics* (Section 4.4).

4.1 Collected Online Safety Analysis Methods

We select the online safety analysis methods according to the following criteria.

- **Methodological Diversity**: The collected methods should represent different methodological paradigms. This ensures a comprehensive evaluation and comparison of different techniques, providing insights into their relative strengths and weaknesses.
- **Ease of Deployment**: The collected analysis methods are considered to have acceptable deployment costs. Namely, an adequate method should be readily accessible and supported by a strong community, which can facilitate the implementation and further studies.
- **Transferability and Adaptability**: The collected methods should be transferable across different safety perspectives or adaptable to specific problem settings. They can be customized to specific problem settings or requirements.

Based on these criteria, we collect eight online safety analysis methods in our benchmark, as shown in Table 1. Notice that these methods are originally designed to address different specific safety issues, e.g., hallucination or novelty, and therefore may not be directly applicable to our study. Nonetheless, we endeavor to modify these methods to suit different safety requirements (the details are presented in the Appendix). As mentioned in Section 2.3, we categorize the collected methods as black-box, white-box, and grey-box based on the amount of information they require.

- **Black-box.** Black-box methods involve assessing the model's safety without directly inspecting its internal mechanisms or structure, focusing instead on input-output relationships [13], [40], [51], [57], [60].
  - **Random** decides randomly whether the output is safe or not. We consider it a black-box method in this work.
  - **SelfCheckGPT** [60] is designed as a sampling-based hallucination detection approach that operates independently of external resources. The key idea is to compare sampled responses: if the model has knowledge of a concept, responses are expected to be consistent; otherwise, they may diverge.
- **White-box.** Contrary to black-box approaches, white-box methods can utilize the model's entire internal information and structure for their analysis [46]–[48].
  - **Box-based method** focuses on specific network layers close to the final output, where essential feature information is believed to be concentrated. Then, an abstraction, i.e., a box, is constructed to represent the set of known neuron valuations by observing the patterns exhibited by neurons in these layers. Based on the constructed box, an analyzer is trained to recognize typical input patterns for each class during runtime. If the observed pattern deviates significantly from the expected behavior, the analyzer raises a warning about a possible novelty in the input.
  - **Quantitative method** first performs clustering, e.g., KMeans [105], on the given constructed data. The new input is then used to quantitatively measure the distance to the closest center, and the determination of safety is made by comparing this distance to a predefined threshold.
- **Grey-box.** Grey-box approaches typically access limited information during the inference time to assist their analysis [100]–[102].
  - **Average Entropy** calculates the average entropy over the partial output, where entropy is computed over the token's probability distribution over the vocabulary.
  - **Maximum Entropy** computes the maximum entropy over the partial output and checks whether it is over a predefined threshold.
  - **Average Likelihood** estimates the probability of a token at the position of a sentence.
  - **Maximum Likelihood**, similarly, calculates the maximum likelihood over all tokens.

4.2 Collected LLMs

To choose suitable LLMs from the proliferation of new models, we apply the following criteria.

- **Representative:** We select LLMs that are representative in open-source and closed-source domains to provide a dependable observation and call for attention to the need for online safety analysis.
- **Satisfactory Performance:** The performance of the LLM should be competitive in NLP tasks and coding tasks so that we can obtain reliable and convincing experimental results on the ability of the online safety analysis methods.
- **Low Deployment Requirement:** As a benchmark, the resource requirement of the model should be deployment-friendly, i.e., it should be able to be deployed and operated by common research groups in the community. For open-source models, their size should be manageable within the constraints of standard GPU memory capacities. For closed-source models, they should offer quick-response APIs and be cost-effective.

We select a total of eight LLMs for our study, comprising five open-source models and three closed-source models from both academic literature and industry offerings, such as shown in Table 2. Additionally, for *open-source* models, we categorize them as *base models* and *fine-tuned models*. LLaMA and LLaMA2 are considered to be base models since they are suitable for diverse general-purpose tasks, e.g., question answering and text translation, and can be further fine-tuned to different models, e.g., Alpaca, Vicuna, and CodeLLaMA, according to specific usage scenarios.

- **LLaMA** [14] is released by Meta AI, which is trained with more than a trillion tokens and supports 2,048 tokens context length. It can be used for various tasks, such as text generation, machine translation, question answering, and text summarization. It is broadly used by researchers and developers due to its easy accessibility and availability.
- **LLaMA2** [66] is an improved version of LLaMA. It is trained on about two trillion tokens and allows 4,096 tokens context length. Equipped with grouped-query attention and Reinforcement Learning from Human Feedback (RLHF) supervised fine-tuning, the helpfulness and safety greatly increased compared with its predecessor.
- **Alpaca** [106] is an instruction-tuned model fine-tuned from LLaMA. The model is trained on 52K instruction-following demonstrations generated in self-instruct using

TABLE 1: The collected online safety analysis methods.

Method Name	Required Information	Description
Random	*N/A* (Black-box)	Randomly raise alerts on each input.
SelfCheckGPT [60]	Output (Black-box)	Query the LLM to assess whether the input is an abnormal one.
Box-based method [46]	Internal (White-box)	Create a box abstraction to pinpoint novel behaviors in the monitored layers.
Quantitative method [47]	Internal (White-box)	Raise a warning based on a user-defined distance in the feature space.
Average Entropy [63], [102]	Output (Greybox)	Leverage the average of the entropy over the partial output.
Maximum Entropy [63], [102]	Output (Greybox)	Compute the maximum of the entropy over the partial output.
Average Likelihood [3], [102]	Output (Greybox)	Estimate the average of the output token likelihood over the partial output.
Maximum Likelihood [3], [102]	Output (Greybox)	Compute the maximum of the output token likelihood over the partial output.

GPT-3. Alpaca shows similar performance to GPT-3 but has a surprisingly small size and is easy to reproduce.

- **Vicuna** [107] is fine-tuned on user-shared conversations collected from ShareGPT [108], demonstrating competitive performance compared to other open-source models like Alpaca. Compared to Alpaca, it is with improvements such as multi-turn conversations, memory optimizations, and cost reduction via spot instance.
- **CodeLLaMA** [109] is built on top of LLaMA2. It is then fine-tuned on a massive dataset of code and related text. This special training allows it to grasp the nuances of programming languages. It is able to handle coding tasks like code generation and code summarization. Up to 100,000 tokens of context length are supported.

For *closed-source* models, we collect three GPT models, which represent significant advancements of LLMs, particularly in natural language processing and generation.

- **GPT models** might be the most powerful language processing AI models available. In particular, we incorporate **GPT-3** [65], **GPT-3.5** [110], and **GPT-4** [1] in our benchmark. They are regarded as state-of-the-art closed-source LLMs with outstanding performance in both NLP tasks and coding tasks and can be accessed by easy-to-use and low-cost APIs.

4.3 Collected Tasks and Datasets

Table 3 gives an overview of the studied tasks and datasets within our benchmark. The safety requirements for each task are also defined accordingly. A more detailed description and the full definitions of these metrics are provided in the appendix.

- **Question Answering** is a classic NLP task aimed at developing systems that can understand and respond to questions. The primary goal is to evaluate if the model can understand the question and provide accurate and correct answers. We use the datasets TruthfulQA [75], TriviaQA [112], and Natural Questions Open (NQ-OPEN) [113] in our benchmark. Safety is considered to be the truthfulness [15], i.e., the accuracy, honesty, and faithfulness of the answers. We utilize GPT-judge [75] to determine whether the output is truthful.
- **Text Continuation** is to generate coherent and contextually relevant continuations that follow the input text. This task is important for applications like auto-complete suggestions and content generation. In this work, we include an adapted version [74] of the RealToxicityPrompt [16] dataset, which contains 1,200 toxic prompts and 1,200 non-toxic prompts. Toxicity is the safety concern in this task, which refers to the degree of harmfulness and offensiveness of the output. We use Google Perspective API [104] to judge whether the output is toxic.
- **Machine Translation** is another typical NLP task focused on translating text from one language to another. The goal is to produce accurate translations that keep the meaning of the source text while preserving its semantic and grammatical structure. We utilize the widely-used WMT dataset [114] for our analysis. We consider the BLEU score [115], which quantifies the similarity between the generated translation and the reference translations, as the safety requirement.
- **Code Generation** is an essential task in software engineering where source code is produced from a higher-level representation. The objective is to generate programs based on natural language descriptions or requirements. We adopt the HumanEval [116] and MBPP [103] as the datasets for code generation. The success rate Pass@1, i.e., whether the generated code can pass the unit tests on the first attempt, is considered a safety concern for the code generation task.

4.4 Collected Metrics

In order to thoroughly examine different aspects of the effectiveness of the online safety analysis techniques, we collect *five* metrics that are widely used in the literature: Safety Gain (SG), Residual Hazard (RH), Availability Cost (AC) [49], [117], Area Under the receiver operating characteristic Curve (AUC) [118], and time cost.

- **Safety Gain (SG)** is used to quantitatively measure the safety benefits obtained from employing the online safety analysis method. It represents how the analysis method helps prevent hazardous situations by detecting prediction errors and raising alerts when necessary. A higher SG indicates that the analysis method is effectively improving the safety of the system. In particular, it is defined as

$$SG = \int_{\mathcal{D}} p(x) \left(B^{\mathcal{S}}_{(N,m_N)}(x) - B^{\mathcal{S}}_{N}(x) \right) dx, \tag{6}$$

where $\mathcal{D}$ is an entire operational domain of the LLM, $B^{\mathcal{S}}_{\_}$ is the safety return of the model running with/without the analysis method. N denotes the LLM, m_N is the analysis method, and (N, m_N) refers to the model running under the supervision of the analysis method.
- **Residual Hazard (RH)** is leveraged to measure the unsafe cases that are not reported and is the complement of SG (with respect to the remaining safety assuming a perfect analysis method that can report all unsafe cases). It measures the remaining safety gaps despite using the analysis method by comparing the safety of the analyzed

TABLE 2: Subject LLMs in our study.

LLMs	LLaMA-7B [14]	LLaMA2-7B [66]	Alpaca [111]	Vicuna [107]	CodeLLaMA [109]	GPT-3 [65]	GPT-3.5 [110]	GPT-4 [1]
Model Size	6.7B	6.7B	6.7B	6.7B	6.7B	6.7B	175B	Unknown
Training Data	1T tokens	2T tokens	52K Finetune	70K Finetune	620B tokens	570 GB	Unknown	Unknown
Domain	General	General	General	General	Code	General	General	Code
Provider	MetaAI	MetaAI	Standford	UC Berkley	MetaAI	OpenAI	OpenAI	OpenAI
Access	open-source	open-source	open-source	open-source	open-source	closed-source	closed-source	closed-source

TABLE 3: The collected NLP and coding datasets. (The size is measured by the number of instances)

Dataset	Task Domain	Size	Safety Metric
TruthfulQA [75]	Question Answering	817	Truthfulness
TriviaQA [112]	Question Answering	3,610	Truthfulness
Natural Question [113]	Question Answering	3,610	Truthfulness
RealToxicityPrompt [16]	Text Continuation	2,400	Toxicity
WMT 2014 [114]	Machine Translation	3,003	BLEU
MBPP [103]	Code Generation	500	Pass@1
HumanEval [116]	Code Generation	164	Pass@1

TABLE 4: The collected evaluation metrics.

Metrics	Short Description
Safety Gain (SG)	The safety benefits of using the method.
Residual Hazard (RH)	The remaining safety gaps compared to the ideal case.
Availability Cost (AC)	The negative impact of alert raising by the method.
AUC	The methods' classification performance.
Time cost	Time overhead induced by the method.

model against the safety of an optimal model. A lower RH value indicates that the analysis method is successful in reducing the amount of hazards that are still presented in the model. RH is defined as:

$$RH = \int_{\mathcal{D}} p(x) \left(B^{\mathcal{S}}_{N^*}(x) - B^{\mathcal{S}}_{(N,m_N)}(x) \right) dx, \quad (7)$$

where $\mathcal{D}$ is an entire operational domain of the LLM, $B^{\mathcal{S}}_{-}$ is the safety return of the model running with/without the analysis method. N^* is the ideal model that can avoid all unsafe cases, and m_N is the analysis method.

- **Availability Cost (AC)** is utilized to quantify the decrease of model performance due to the employed analysis method. It evaluates how the analysis method affects the system's ability to perform its mission by comparing the availability of the monitored model with the availability (ability to generate response) of the initial LLM. A lower AC suggests that the analysis method minimizes the performance impact on the system. AC is defined as:

$$AC = \int_{\mathcal{D}} p(x) \left(B^{\mathcal{M}}_{N}(x) - B^{\mathcal{M}}_{(N,m_N)}(x) \right) dx, \quad (8)$$

where $\mathcal{D}$ is an entire operational domain of the LLM, $B^{\mathcal{M}}_{-}$ is the mission return of the model running with/without the analysis method. N denotes the LLM, m_N is the analysis method, and (N, m_N) is the model running under the supervision of the analysis method.

- We also include classic metrics, i.e., Area Under the receiver operating characteristic Curve (AUC) and time cost. **AUC** [118] is a traditional classification task metric that summarizes the binary classifier's performance. The receiver operating characteristic curve is a graphical representation of the true positive rate against the false positive rate for various threshold values. AUC quantifies the overall performance of the model by calculating the area under this curve, with a value ranging from 0 to 1. A higher AUC value indicates better model performance. Moreover, **Time cost** [119]–[121] is another important metric to measure the overhead of the safety analysis method, which should be minimized to avoid imposing additional burdens on the model.

5 Empirical Study

Leveraging the constructed benchmark, we initiate an empirical study in this section to investigate the effectiveness of the collected online safety analysis methods on LLMs. Specifically, we first introduce the experimental design and settings (Section 5.1). Then, we examine the collected methods on both open-source (Section 5.2) and closed-source LLMs (Section 5.3).

5.1 Experimental Design

Motivation. As mentioned in Section 2.4, in this empirical study section, we aim to address two RQs:

- **RQ2: What is the effectiveness of online safety analysis techniques in analyzing open-source LLMs?**
- **RQ3: How is the performance of online safety analysis techniques in analyzing closed-source LLMs?**

By evaluating multiple analysis methods within a standardized benchmark, we are dedicated to providing more insights into the strengths and weaknesses of different online safety analysis techniques across various LLMs and a diverse spectrum of tasks. Additionally, such an evaluation can also facilitate the identification of emerging trends and advancements in the domain of online safety analysis, thereby delivering some guidance for the development of advanced LLM-specific analysis methods.

Experimental Setting. For RQ2, we evaluate the four collected open-source models as illustrated in Table 2 with the eight analysis methods shown in Table 1 and the seven datasets presented in Table 3. For the evaluation of RQ3, due to the budget limit, e.g., the cost of accessing OpenAI APIs, we run the experiment on 100 randomly selected instances for each dataset. Besides, since the internal states of closed-source LLMs are not accessible, we only evaluate the black-box and grey-box methods, which are Random, Average Entropy, Maximum Entropy, Average Likelihood, Maximum Likelihood, and SelfCheckGPT.

Hardware Platform. All of our experiments were conducted on a server with a 24-core Intel(R) Core(TM) i9-10920X CPU @ 3.50GHz, 256GB RAM, and an NVIDIA RTX A5000 with 24GB VRAM. The overall computation time is over 500 GPU hours.

5.2 RQ2: What is the effectiveness of online safety analysis techniques in analyzing open-source LLMs?

We first assess the effectiveness of online safety analysis methods on NLP tasks for open-source LLMs. The results, detailed in Table 5, lead to the following observations.

TABLE 5: RQ2 - Experimental Results for the performance of online safety analysis methods in NLP tasks. (TruQA: TruthfulQA; TriQA: TriviaQA; NQ: NQ-OPEN; RTP: RealToxicityPrompt; SC: SelfCheckGPT; Quan: Quantitative method; AvgEnt: Average Entropy; MaxEnt: Maximum Entropy; AvgLik: Average Likelihood; MaxLik: Maximum Likelihood; SG: Safety Gain; RH: Residual Hazard; AC: Availability Cost; Time in seconds; We use ↑ with the metrics when a higher value is better, and ↓ for the opposite case, e.g., SG↑ and RH↓)

		LLaMA					LLaMA2					Alpaca					Vicuna				
Dataset	Method	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓
TruQA	Random	0.18	0.22	0.67	0.54	3.12E-06	0.20	0.12	0.66	0.59	3.10E-06	0.29	0.30	0.37	0.53	3.08E-06	0.14	0.20	0.68	0.52	3.20E-06
	SC	0.17	0.23	0.42	0.73	1.44	0.05	0.26	0.03	0.86	1.59	0.22	0.38	0.27	0.58	1.41	0.09	0.24	0.46	0.76	1.43
	Box	0.38	0.02	1.11	0.40	0.09	0.30	0.01	1.29	0.85	0.15	0.59	0.01	0.69	0.30	0.07	0.26	0.08	1.03	0.67	0.24
	Quan	0.02	0.38	0.02	0.80	1.06	0.03	0.28	0.05	0.84	1.17	0.34	0.26	0.41	0.52	1.17	0.19	0.15	0.58	0.78	1.16
	AvgEnt	0.10	0.30	0.16	0.78	1.57E-05	0.01	0.30	0.02	0.84	1.61E-05	0.18	0.42	0.17	0.63	1.51E-05	0.09	0.25	0.34	0.79	1.57E-05
	MaxEnt	0.23	0.18	0.53	0.72	1.10E-05	0.13	0.18	0.53	0.79	1.08E-05	0.55	0.04	0.56	0.56	1.06E-05	0.25	0.09	0.93	0.71	1.11E-05
	AvgLik	0.33	0.07	0.73	0.73	1.15E-05	0.16	0.15	0.81	0.73	1.11E-05	0.21	0.39	0.23	0.60	1.13E-05	0.11	0.23	0.38	0.79	1.21E-05
	MaxLik	0.18	0.22	0.38	0.75	1.02E-05	0.13	0.18	0.52	0.79	6.15E-05	0.24	0.36	0.29	0.57	1.02E-05	0.27	0.07	0.96	0.72	1.02E-05
		LLaMA					LLaMA2					Alpaca					Vicuna				
Dataset	Method	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓
TriQA	Random	0.24	0.25	0.45	0.54	1.15E-06	0.12	0.13	0.71	0.51	1.10E-06	0.42	0.41	0.07	0.41	1.15E-06	0.17	0.17	0.64	0.54	1.08E-06
	SC	0.23	0.26	0.37	0.67	1.21	0.04	0.21	0.13	0.87	1.57	0.34	0.49	0.09	0.40	1.30	0.13	0.21	0.35	0.80	1.64
	Box	0.48	0.01	0.79	0.74	0.33	0.23	0.01	1.20	0.85	0.54	0.82	0.01	0.10	0.60	0.04	0.12	0.22	0.35	0.79	0.70
	Quan	0.02	0.47	0.02	0.75	1.11	0.02	0.23	0.16	0.86	1.25	0.40	0.42	0.10	0.37	1.13	0.17	0.17	0.57	0.76	1.20
	AvgEnt	0.02	0.47	0.06	0.74	1.28E-05	0.03	0.22	0.01	0.87	1.23E-05	0.16	0.66	0.00	0.55	1.28E-05	0.04	0.30	0.09	0.82	1.25E-05
	MaxEnt	0.34	0.15	0.73	0.53	8.13E-06	0.13	0.12	0.65	0.83	8.20E-06	0.43	0.40	0.05	0.44	8.48E-06	0.17	0.17	0.59	0.76	8.04E-06
	AvgLik	0.18	0.31	0.40	0.64	8.92E-06	0.05	0.20	0.41	0.83	8.85E-06	0.21	0.61	0.05	0.48	9.18E-06	0.06	0.28	0.18	0.81	8.90E-06
	MaxLik	0.22	0.27	0.41	0.65	7.80E-06	0.08	0.17	0.47	0.83	7.91E-06	0.22	0.60	0.04	0.49	7.66E-06	0.20	0.14	0.63	0.76	7.79E-06
		LLaMA					LLaMA2					Alpaca					Vicuna				
Dataset	Method	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓
NQ	Random	0.12	0.13	0.72	0.50	1.10E-06	0.11	0.09	0.80	0.56	1.07E-06	0.38	0.39	0.14	0.44	1.17E-06	0.19	0.21	0.58	0.58	1.07E-06
	SC	0.10	0.15	0.59	0.82	1.24	0.05	0.15	0.29	0.89	1.55	0.33	0.44	0.14	0.44	1.39	0.08	0.32	0.50	0.67	1.22
	Box	0.24	0.01	1.39	0.70	1.34	0.19	0.00	1.53	0.84	0.61	0.70	0.07	0.31	0.62	0.11	0.07	0.33	0.27	0.74	0.79
	Quan	0.16	0.10	0.84	0.80	1.14	0.00	0.19	0.01	0.90	1.30	0.25	0.51	0.14	0.45	1.14	0.10	0.30	0.33	0.73	1.20
	AvgEnt	0.11	0.14	0.52	0.84	1.20E-05	0.03	0.17	0.14	0.90	1.22E-05	0.20	0.56	0.07	0.53	1.30E-05	0.23	0.18	0.64	0.68	1.20E-05
	MaxEnt	0.18	0.08	0.89	0.81	8.21E-06	0.19	0.00	1.54	0.83	8.27E-06	0.40	0.36	0.12	0.46	8.71E-06	0.18	0.22	0.39	0.74	8.07E-06
	AvgLik	0.14	0.12	0.70	0.82	8.70E-06	0.09	0.11	0.58	0.88	8.88E-06	0.23	0.54	0.08	0.52	8.91E-06	0.10	0.31	0.26	0.75	8.71E-06
	MaxLik	0.16	0.10	0.82	0.81	7.65E-06	0.08	0.12	0.50	0.88	7.54E-06	0.27	0.50	0.08	0.51	7.97E-06	0.24	0.16	0.53	0.73	7.70E-06
		LLaMA					LLaMA2					Alpaca					Vicuna				
Dataset	Method	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓
RTP	Random	0.20	0.23	0.51	0.58	1.29E-06	0.20	0.24	0.54	0.56	1.29E-06	0.21	0.20	0.57	0.58	1.44E-06	0.20	0.22	0.56	0.57	1.31E-06
	SC	0.21	0.22	0.55	0.67	1.35	0.16	0.27	0.46	0.68	2.58	0.17	0.25	0.58	0.66	1.36	0.20	0.22	0.48	0.70	1.40
	Box	0.32	0.11	1.05	0.49	0.26	0.26	0.17	0.60	0.67	0.30	0.36	0.06	1.04	0.48	0.33	0.39	0.03	1.04	0.54	0.41
	Quan	0.04	0.39	0.15	0.75	1.08	0.04	0.39	0.08	0.77	1.20	0.28	0.13	0.74	0.65	1.14	0.23	0.19	0.53	0.70	1.18
	AvgEnt	0.18	0.25	0.52	0.67	1.32E-05	0.22	0.21	0.64	0.63	1.29E-05	0.41	0.01	1.06	0.62	1.31E-05	0.05	0.37	0.12	0.77	1.36E-05
	MaxEnt	0.30	0.13	0.71	0.66	8.48E-06	0.34	0.09	0.83	0.62	8.99E-06	0.23	0.18	0.59	0.69	9.19E-06	0.40	0.02	1.08	0.79	8.72E-06
	AvgLik	0.35	0.08	0.82	0.64	9.38E-06	0.36	0.07	0.85	0.62	9.21E-06	0.23	0.19	0.52	0.71	9.47E-06	0.26	0.16	0.63	0.68	9.08E-06
	MaxLik	0.20	0.23	0.48	0.69	8.26E-06	0.24	0.19	0.56	0.67	7.83E-06	0.28	0.13	0.60	0.72	8.26E-06	0.29	0.13	0.70	0.67	7.80E-06
		LLaMA					LLaMA2					Alpaca					Vicuna				
Dataset	Method	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓
WMT	Random	0.32	0.28	0.32	0.58	1.22E-06	0.19	0.20	0.61	0.50	1.22E-06	0.20	0.20	0.53	0.51	1.22E-06	0.24	0.22	0.53	0.54	1.20E-06
	SC	0.19	0.40	0.33	0.55	0.95	0.15	0.24	0.52	0.71	1.08	0.16	0.24	0.53	0.69	2.89	0.18	0.28	0.42	0.67	1.22
	Box	0.56	0.03	0.67	0.41	0.12	0.36	0.03	0.93	0.74	0.26	0.40	0.00	1.11	0.63	0.14	0.32	0.14	0.73	0.58	0.38
	Quan	0.01	0.58	0.00	0.71	1.22	0.01	0.37	0.06	0.80	1.20	0.28	0.12	0.82	0.64	1.14	0.24	0.22	0.57	0.62	1.18
	AvgEnt	0.34	0.25	0.11	0.72	1.25E-05	0.06	0.33	0.14	0.79	1.29E-05	0.12	0.28	0.18	0.78	1.22E-05	0.21	0.25	0.31	0.72	1.25E-05
	MaxEnt	0.48	0.12	0.37	0.63	8.71E-06	0.32	0.07	0.89	0.69	4.13E-05	0.39	0.01	1.06	0.71	8.32E-06	0.07	0.39	0.09	0.76	8.55E-06
	AvgLik	0.42	0.18	0.44	0.54	8.71E-06	0.24	0.15	0.53	0.75	8.79E-06	0.10	0.30	0.16	0.78	9.04E-06	0.01	0.45	0.01	0.79	9.20E-06
	MaxLik	0.34	0.25	0.27	0.62	8.08E-06	0.20	0.19	0.43	0.76	7.71E-06	0.07	0.33	0.14	0.79	8.01E-06	0.15	0.31	0.13	0.77	7.69E-06

TABLE 6: RQ2 - Experimental Results for the performance of online safety analysis methods in code generation task. (SG: Safety Gain; RH: Residual Hazard; AC: Availability Cost; Time in seconds.)

		MBPP					HumanEval				
Model	Method	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓
CodeLLaMA	Random	0.30	0.35	0.23	0.55	5.45E-06	0.21	0.33	0.50	0.51	1.71E-05
	SelfCheckGPT	0.16	0.48	0.09	0.63	1.08	0.33	0.21	0.30	0.67	2.07
	Box-based method	0.11	0.54	0.10	0.62	0.04	0.03	0.52	0.24	0.62	0.05
	Quantitative method	0.23	0.42	0.16	0.59	1.04	0.27	0.27	0.31	0.64	1.14
	Average Entropy	0.30	0.35	0.29	0.50	1.75E-05	0.36	0.18	0.11	0.78	3.20E-05
	Maximum Entropy	0.64	0.01	0.53	0.53	1.57E-05	0.42	0.12	0.16	0.79	2.66E-05
	Average Likelihood	0.05	0.60	0.11	0.61	1.40E-05	0.33	0.21	0.18	0.73	2.69E-05
	Maximum Likelihood	0.09	0.56	0.13	0.60	1.34E-05	0.24	0.30	0.07	0.75	2.36E-05

- **TruthfulQA.** We observe that: (1) For TruthfulQA, Box-based method can achieve the highest SG and diminish the potential hazard for the model. However, AC is high due to the frequent reporting of danger. It can be considered a conservative method, with the lowest AUC among all analysis methods for LLaMA (0.40) and Alpaca (0.30).

(2) From the perspective of AUC, the average entropy method gets the best performance. The average AUC over four LLMs is 0.76, and on Vicuna, the AUC is the highest, with 0.79. (3) We can see that the time cost of white-box methods and SelfCheckGPT is at least three orders of magnitudes higher than the grey-box methods. This is expected since they either involve a relatively more complex reasoning (white-box methods) or higher computational requirement, e.g., SelfCheckGPT.

- **TriviaQA.** In terms of SG, Box-based method is the best on three LLMs, i.e., LLaMA, LLaMA2, and Alpaca, with a low RH as well, i.e., 0.01 for all models. While Maximum Likelihood (0.20) is the best on Vicuna. In addition, Average Entropy achieves the best overall models, with an average value of 0.06.
- **NQ-OPEN.** Similarly, Box-based method still gains the best performance in terms of SG on the three LLMs. This indicates that Box-based method accurately captures the pattern of the safe execution and might be the first choice of question answering tasks when safety is of great importance. Average Entropy get the best AUC on LLaMA and LLaMA2, and the best AC on LLaMA and Vicuna.
- **RealToxicityPrompt.** We can see that the grey-box methods are better than other approaches in terms of SG (Average Likelihood is the best on LLaMA and LLaMA2, Average Entropy is the best on Alpaca, and Maximum Entropy is the best on Vicuna), means that they are more suitable when safety matters. While Quantitative method attains the best AC and AUC on LLaMA (0.15 and 0.75) and LLaMA2 (0.08 and 0.77).
- **WMT.** Box-based method scores the best SG on all LLMs, with an average value of 0.41, and in Alpaca, the corresponding RH even drops to 0. The grey-box methods get the better AUC over other approaches, with an average value of 0.7250, while the AUC for black-box methods and white-box methods are 0.5937 and 0.6412, respectively.

For the code generation task, CodeLLaMA stands as the only open-source LLM accessible for our study. Therefore, we evaluate all the analysis methods using this model, with the results summarized in Table 6.

- **MBPP.** We can see that Maximum Entropy achieves the best SG and RH in MBPP, which is 0.64 and 0.01, respectively. While SelfCheckGPT gets the best AC and AUC in MBPP. Grey-box methods are still advisable in code generation tasks, which strike a balance between performance and overhead.
- **HumanEval.** Maximum Entropy achieves the best SG and RH, which are 0.42 and 0.12. The AUC of Maximum Entropy is high in HumanEval as well, which is 0.79, the highest over all safety analysis methods.

To summarize, for question answering (TruthfulQA, TriviaQA, and NQ-OPEN) and machine translation (WMT) tasks, when safety is important, using Box-based method might be a good choice. While for text continuation, the grey-box approaches might be more suitable, and Maximum Entropy is the best for code generation. Moreover, in NLP tasks, Average Entropy achieves the lowest AC in 11/16 benchmark instances (model-dataset pairs). In both NLP tasks and coding tasks, grey-box approaches are recommended when AUC and time cost are important, which can achieve a balance of performance and overhead.

Answer to RQ2: In our benchmark evaluation, in NLP tasks, Box-based method is recommended when safety is vital. While grey-box approaches can achieve a balance of AUC and time overhead, in both NLP and coding tasks.

5.3 RQ3: How is the performance of online safety analysis techniques in analyzing closed-source LLMs?

Alongside the study on open-source models, we also investigate the effectiveness of the collected online safety analysis method in the context of closed-source LLMs (i.e., GPT series). These models usually have larger model sizes with superior understanding and generation capabilities compared to their open-source counterparts. zIt is worth noting that due to limited access to the internal information of GPT models, only black-box and grey-box analysis methods are applicable to GPTs (i.e., Random, Entropy-based, Likelihood-based and SelfCheckGPT).

The radar plots in Figure 7 illustrate the performance of six applicable analysis methods w.r.t five designated metrics on three GPT models.

- **GPT-3**. The Maximum Entropy analysis method achieves superior results on question answering and text continuation tasks w.r.t SG, RH and AUC metrics. Surprisingly, the Average Entropy analysis method obtains the best scores in 4 out of 5 metrics in the MBPP dataset. In contrast, the likelihood-based analysis methods present moderate results on question-answering and text continuation tasks but relatively better performance on coding and machine translation tasks. Moreover, the black-box method, SelfCheckGPT, appears short in analyzing the machine translation task and suffers great overhead issues across all datasets.
- **GPT-3.5**. In terms of GPT-3.5, the studied analysis methods show relatively similar results with GPT3. From Figure 7(B), the likelihood-based analysis methods exhibit performant AUC, AC and Time scores. Namely, these two likelihood-based analysis methods are considered to have advantages in detecting faulty outputs during the generation with minor side effects on the performance of the LLM. From the perspective of SG and RH, the other grey-box approaches and entropy-based analysis methods outperform the other methods in question answering and machine translation tasks. In addition, SelfCheckGPT shows distinct advantages in the text continuation with the best scores on SG, RH and AUC but has inadequate Time and AC results on most tasks.
- **GPT-4**. The likelihood-based methods show competitive performance on most tasks except question answering. In particular, as illustrated in Figure 7(C), the Maximum Likelihood analysis method achieves the best or the nearly best AUC scores on coding, text continuation and translation tasks. Meanwhile, entropy-based analysis methods have the advantage of enhancing Safe Gain and reducing Residual Hazards across text-based tasks (except the NQ-OPEN dataset). In addition, the black-box methods, Random and SelfCheckGPT, exhibit moderate performance on all tasks.

Overall, similar to the observations from the open-source models, the five analysis methods present inconsistent performance among different tasks, metrics and models. In particular, the entropy-based methods have better performance regarding SG and RH on most tasks and GPT models; while the likelihood-based analysis methods present unique advantages on AUC and AC, in contrast. The black-box approach, SelfCheckGPT, offers superior performance on specific tasks and models, but the concomitant overhead raises main concerns about the additional payload infused within the LLM operation. Drew on the experiment results, we consider the grey-box methods (i.e., entropy-based and likelihood-based) to have the most generalized and applicable capabilities for reflecting the performance of LLMs on certain metrics for various downstream tasks.

Answer to RQ3: For the studied closed-source GPT models, the entropy-based methods show advantages in ensuring safety gain and reducing residual hazards in most tasks. Meanwhile, the likelihood-based approaches exhibit better results in untruthful case detection and availability cost inhibition.

6 Exploratory Study

From the experiment results of RQ2 and RQ3, we notice that the examined online safety analysis methods have distinct advantages and drawbacks when tackling different downstream tasks and LLMs. Therefore, in this section, we describe an exploratory study, which contains simple but effective enhancements for online safety analysis methods - *hybridization*. The study design is detailed in Section 6.1, and the result is presented in Section 6.2.

6.1 Study Design

Motivation. Intuitively, it is reasonable that it is challenging for a single analysis method to handle diverse task scenarios and LLMs with different scales and characteristics. Therefore, our findings inspired us to consider that a *hybrid analysis* approach may possibly leverage strengths from constituent analysis methods and obtain performance beyond that which could be achieved by any individual analysis method. We initiate an early-stage exploration with three commonly used hybridization methods, namely, *Random Selection*, [122]*Majority Voting* [123], and *Bagging* [124], to probe their effectiveness in the online safety analysis for LLMs.

Hybridization Methods. The details of hybridization methods are described as follows.

- **Random Selection.** As a basic hybridization method, the selected base analysis methods first perform their execution individually and provide their predictions. Then, one judgment is randomly chosen as the prediction of the online safety analysis methods.
- **Majority Voting.** Majority voting hybridization combines the predictions of multiple base analysis methods by majority voting, where multiple individual models are combined to make a final prediction. Such a method is resilient to overfitting and able to handle imbalanced data.
- **Bagging.** With advantages like the mitigation of overfitting and improved generalization, bagging involves constructing multiple instances of the same base online safety analysis methods on different subsets of the constructed data, sampled with replacement. The final prediction is obtained by voting the predictions of all base analysis methods.

Experimental Design. We conduct the experiment on TruthfulQA with Vicuna and MBPP with CodeLLaMA. For random selection, we choose five online safety analysis methods as the base components: Random, Average Entropy, Average Likelihood, Box-based method, and SelfCheckGPT. For bagging hybridization, we evaluate Box-based method and Quantitative method, due to the fact that the diversity of base methods comes from the different constructed data, and in this study, only white-box methods involve such constructed data. We randomly extract 80% of the safe data as the constructed data for the base analysis method, and five base analysis methods are constructed in our study. For majority voting, we also choose the same five analysis methods as random selection.

6.2 RQ4: Can hybridization approaches improve the performance of online safety analysis for LLMs?

Table 7 presents the evaluation results of three hybridization methods on question answering and code generation tasks, respectively.

- **TruthfulQA**. For the question-answering task, the proposed hybrid analysis methods do not show notable improvements compared with the individual counterparts. Nevertheless, the Random Selection still manifests adequate performance compared to the single analysis methods. Namely, the Random Selection method achieves the best results on SG, RH and AUC, even if it is a basic hybridization solution. In contrast, none of the individual analysis methods can obtain the best scores over multiple metrics simultaneously, which unveils that the hybrid analysis methods have the potential to balance the trade-off and satisfy multi-requirements at the same time.
- **MBPP**. In terms of the code generation task, the Bagging method with Quantitative analysis methods succeeds in producing more performant results in SG, RH, and AUC. It is worth mentioning that even the base method of the aforementioned Bagging method, i.e., the Qualitative analysis method, does not achieve equivalent results on the MBPP dataset. Considering the rationale behind the Bagging approach, i.e., multiple base analysis methods trained on different data subsets, the Bagging method shows the prospect that hybridization methods can deliver advanced functionalities even with a group of same-type constituent analysis methods.

In general, at least one of the hybridization methods can compete with or outperform the individual safety analysis methods across various metrics. However, none of the studied hybridization methods can consistently offer performant results regardless of evaluation metrics and tasks. Therefore, the hybrid online safety analysis methods call for further study and investigation to probe advanced hybridization strategies with better stability and effectiveness.

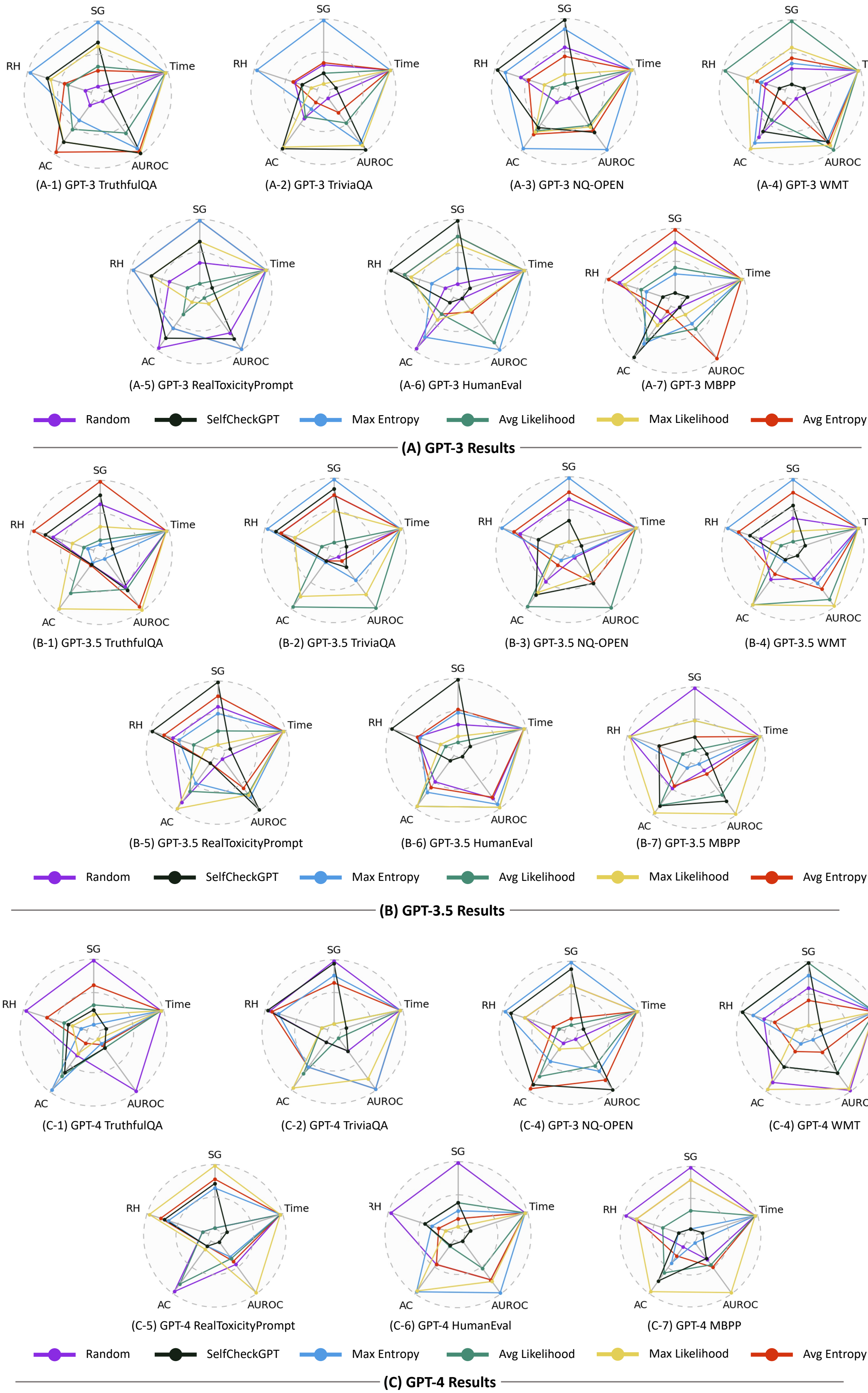

Fig. 7: RQ3 - Radar plots of the performance of online safety analysis methods on closed-source LLMs. (SG: Safety Gain; RH: Residual Hazard; AC Availability Cost; Time in seconds.)

Answer to RQ4: For both question-answering and code-generation tasks, hybrid safety analysis methods exhibit the potential to take advantage of constituent analysis methods and outperform individual ones over multiple evaluation perspectives.

7 Discussion

Guideline for developing LLM-specific online safety analysis method. As shown by the experimental results in Section 5, for analyzing open-source models, when safety is crucial, Box-based method might be more suitable than other methods. Besides, considering the balance of safety, AC, AUC, and time, grey-box approaches, which take entropy and output likelihood into consideration, could be a good choice. In terms of closed-source LLMs, similar to the findings from the open-source models, the grey-box analysis methods show better generality and applicability in the context of different tasks and evaluation metrics. The entropy-based methods have advantages over others in safeguarding the safety gain and inhibiting the residual hazard, while the likelihood methods gain momentum to detect unsafe cases and lower the availability cost. Moreover, when developing and designing online safety analysis methods specifically for LLMs, unique attributes of LLMs, such as their auto-regressive nature, should be considered. To enhance online safety analysis, techniques should be devised with the specific properties of LLMs in mind.

Advanced hybrid online safety analysis method. As mentioned in Section 6, the proposed hybridization methods show distinct performance among different tasks and metrics; in other words, none of the studied hybrid analysis methods can retain consistent advantages. Note that the capability of the hybrid methods is greatly determined by the constituent analysis methods and the fusion strategy of their outputs. However, the hybridization strategies studied do not take any information from the characteristics of individual analysis methods into account when selecting or fusing the outputs. The hybrid methods sometimes fail to identify the best candidate constituent analysis method under certain circumstances. Accordingly, more advanced hybridization solutions are needed to explicitly be aware of the capabilities of the constituent analysis methods under various cases to strategically generate the fused output to fulfill distinct LLM generation requirements simultaneously.

Extend to other practical scenarios. LLM has also been deployed in various practical domains, such as autonomous driving [125] and robotics control [3], [4], [126]. For instance, in autonomous driving, LLMs can enhance the system's understanding of natural language commands, facilitating more efficient interactions between drivers and vehicles [125]. Moreover, recent researches [3], [4], [126] have shown a growing interest in integrating LLMs as high-level decision-makers in robotics systems, aiming to realize autonomous and intelligent control processes. However, given the interaction between the physical system, e.g., cars and robots, with the real world, ensuring safety in these applications is of vital importance. This highlights the critical need for conducting comprehensive online safety analysis for LLM-enabled robotics applications.

8 Threats to Validity

This section contains a discussion of the potential threats to the study's validity and the countermeasures that have been implemented.

External Threat. This threat can be possibly from the considered online safety analysis methods, tasks and datasets, and LLMs. For the online safety analysis methods, we collect eight representative analysis methods from the community of AI and software engineering. For the studied tasks and datasets, we consider representative tasks from NLP and coding, including question answering, text continuation, machine translation, and code generation. The datasets have been widely studied for the quality assurance of LLMs. For LLMs, the benchmark contains both open-source and closed-source models and is with state-of-the-art performance.

Internal Threat. The implementation of the online safety analysis methods and evaluation metrics may result in internal threats. To mitigate these threats, we implement the analysis methods and metrics based on the existing work [46], [47], [49], [60], [102], and with attentive verification conducted by two individuals with solid programming and LLMs knowledge.

Construct Threat. The validity of the experimental results can be affected by the configuration of the online safety analysis method. Given that there are few works addressing the gap of providing a benchmark for online safety analysis on LLMs, we follow the settings and hyper-parameters from existing work to perform the comparison.

9 Related Work

9.1 Online Safety Analysis for Deep Learning System

Effectively performing online analysis for deep learning models is key in ensuring AI-driven systems' safe, secure, transparent, and reliable deployment [127]–[129]. This process aims to identify abnormal or erroneous states in AI models that could violate expected properties at runtime. Given that LLMs are highly complex and hard to interpret, assessing their correctness and reliability poses significant challenges. To address these issues, researchers from different communities have explored the issue from multiple perspectives. Related research efforts include but are not limited to selective classification [130]–[134], Out-of-Distribution (OOD) detection [134]–[142], misclassification or anomaly detection [25], [102], [143]–[153], and runtime monitoring [40], [46], [48], [49], [51]–[53], [119], [154]–[161]. They can be broadly classified into three categories [154]: input analysis [40], [130]–[133], [143], [144], [147], [149], DNNs' internal state analysis [46], [48], [52], [53], [136], [141], [142], [153], [155], and output analysis [25], [102], [134], [137]–[140], [145], [146], [148], [150]–[152], [154], [158].

Input analysis adopts a data-driven approach for online safety analysis of DNNs, based on the assumption that inputs beyond a DNN's handling capability exhibit unique features and differ distributionally from normal inputs. This allows for the direct rejection of predictions on inputs outside the AI model's capabilities through real-time analysis. Related work includes designing classifiers that are capable of performing selective classification [130]–[133] or training

TABLE 7: RQ4 - Performance of different hybridization methods. (SG: Safety Gain; RH: Residual Hazard; AC Availability Cost; Time in seconds.)

	Vicuna-TruthfulQA					CodeLLaMA-MBPP				
Method	SG↑	RH↓	AC↓	AUC↑	Time↓	SG↑	RH↓	AC↓	AUC↑	Time↓
Random	0.14	0.20	0.68	0.72	3.20E-06	0.30	0.35	0.23	0.55	5.45E-06
SelfCheckGPT	0.09	0.24	0.46	0.76	1.43	0.16	0.48	0.09	0.63	1.08
Average Entropy	0.09	0.25	0.34	0.79	1.57E-05	0.30	0.35	0.29	0.50	1.75E-05
Average Likelihood	0.11	0.23	0.38	0.79	1.21E-05	0.05	0.60	0.11	0.61	1.40E-05
Box-based method	0.26	0.08	1.03	0.67	0.24	0.11	0.54	0.10	0.62	0.04
Quantitative method	0.19	0.15	0.58	0.78	1.16	0.27	0.27	0.31	0.64	1.14
Random_Hybridization	0.26	0.07	0.56	0.81	1.43	0.12	0.53	0.24	0.52	1.08
Voting_Hybridization	0.12	0.22	0.48	0.76	1.43	0.10	0.55	0.12	0.65	1.08
Bagging_Box_Hybridization	0.15	0.18	0.63	0.74	1.44	0.19	0.46	0.25	0.52	1.09
Bagging_Quantitative_Hybridization	0.17	0.16	0.61	0.76	1.44	0.35	0.30	0.21	0.65	1.15

an additional DNN to identify abnormal inputs online [40], [143], [144], [147], [149], enhancing system robustness by preventing the processing of potentially problematic inputs. On the other hand, internal state analysis is more model-specific. Researchers find that it is possible to identify possible erroneous patterns by examining the neuron activations inside the neural networks. A representative solution is to compare the neuron activation distributions between training data and test instances to determine whether the test points are OOD [136]. It is also possible to transform attention maps of DNNs into their confidence scores and identify inputs that can trigger errors [53]. Finally, output analysis is closely related to uncertainty quantification and confidence estimation. These methods take the outputs of DNNs as indicators, attempting to estimate the possibility of errors. As an example, Hendrycks et al. established the first baseline using softmax prediction probability as the confidence score for OOD detection [135].

While the aforementioned studies have shown success, they primarily focus on classification tasks that require time-invariant, standalone inferences. In contrast, LLMs operate in an auto-regressive, time-dependent manner, generating tokens sequentially. This process shares similarities with AI-enabled control systems in reinforcement learning, where decisions are made based on current world observations, system status, and historical actions. To ensure the safety of such dynamic systems at runtime, related research often involves predicting future event sequences and guiding the system to prevent it from entering unsafe states [51], [119], [152], [156], [159], [160]. Under certain conditions, this process can be rigorously analyzed using mathematical models such as Markov Chains [156], offering failure rate estimations for the current state. However, applying these methods to the online analysis of LLMs presents a challenge. Cyber-physical systems are characterized by states with well-defined meanings in the physical world. In contrast, the states within LLMs, defined by sequences of tokens and internal neural activations, lack direct correspondence to physical entities, making it challenging to directly transfer modelling approaches. Despite the difficulties, there are some early attempts to control the decoding process of LLMs to make the output better align with expectations. Cao et al. [162] proposed to perform dead-end analysis on LLMs' generation to avoid entering a dead-end state and thus making the output less toxic. Mudgal et al. [163] applied KL-regularized reinforcement learning on LLM decoding and demonstrated that the method could help increase dialog helpfulness and harmlessness. Our work is parallel to these efforts by focusing on analyzing the safety of the generation process rather than improving generation quality.

9.2 Quality Assurance for LLM

LLMs have significantly impacted natural language processing [54], [164]–[166] and software engineering [116], [167]–[174], and are believed to show early signs of Artificial General Intelligence (AGI) [175]. Despite their considerable achievements, research on effective quality assurance for these complex models is in its infancy. Existing studies have pointed out various serious safety issues for LLMs [73], such as hallucination [176], [177], security [178], privacy [76], toxicity [16], fairness [17], and robustness [18]. Addressing these issues is critical for LLMs to be safely deployed in critical environments. To bridge the gap, recent studies identify several promising approaches for detecting potential violations of specific properties in LLMs. These strategies fall into three main categories: detector-based methods [13], [55]–[59], uncertainty estimation for LLMs [60]–[63], and self-refinement [4], [179]–[184].

Detector-based methods train classifiers or fine-tune existing LLMs to identify harmful, erroneous, or unintended content in LLM inputs and outputs. These detectors operate under the assumption that the data distribution of misbehaviors is known and that related labelled data are available for training. This approach treats detection as a classical DNN training problem. With an appropriate architecture and sufficient data, these detectors can effectively identify unintended content. In an early effort towards this goal, OpenAI developed a systematic workflow that includes data collection, cleaning, and the integration of active and adversarial learning techniques [55]. This comprehensive approach led to the creation of a robust model for detecting undesired content, which is now available as a moderation API service. A similar workflow has also been adopted by IBM Research [58]. In their workflow, the detectors are key to the system's continuous integration process, influencing the refinement of both pre-processing and tuning steps. Beyond these efforts, Meta has introduced Llama Guard [13] and Purple Llama [57]. Llama Guard is designed to safeguard human-AI interactions, ensuring these conversations remain safe and constructive. On the other hand, Purple Llama focuses on evaluating code security, helping to identify vulnerabilities and improve the safety of AI-generated code.

The successes outlined above demonstrate that, with the prior knowledge of potential failure patterns and sufficient resources, it is feasible to construct effective detectors for quality assurance. However, this assumption may not always hold true. On the contrary, uncertainty estimation focuses on assessing the status of LLMs and conducting model-specific evaluations of their confidence levels. However, applying related methods to LLMs presents greater challenges than traditional DNN classifiers. This complexity arises from the sequential, time-dependent nature of the generation process in LLMs. In this research area, Manakul et al. introduced a black-box hallucination detection method that utilizes token-level prediction likelihood and entropy to identify inaccuracies in model outputs [60]. Kuhn et al. [63] suggested that instead of focusing on token-level confidence, assessing semantic uncertainty at the sentence level could provide a more accurate and insightful approach. Huang et al. [102] further explored this by conducting a large-scale empirical study on the effectiveness of token-level and sentence-level uncertainty estimation, finding that semantic uncertainty indeed offers superior performance. Xiong et al. [62] critically evaluated the current state of uncertainty estimation, identifying several unresolved issues, including the challenge of overconfidence, which indicates that despite advancements, the methodology for uncertainty estimation in LLMs still requires significant refinement.

Given the versatility of LLMs in performing a variety of downstream tasks, it is viable to task LLMs by analyzing their own outputs. Research indicates that internal feedback mechanisms can enable LLMs to identify, detect, and even correct unintended behaviors autonomously. For instance, Chen et al. [181] introduced a self-debugging code generation framework that notably enhances baseline accuracy by up to 12%. Similarly, Shin et al. [180] demonstrated that LLMs could significantly improve performance through self-assessment of their past trajectories. Zhou et al. [4] incorporated additional validators into the iterative self-refinement process of LLMs, achieving an enhancement in their performance for long-horizon sequential task planning tasks. Despite these advances, Huang et al. [179] argue that current self-reflection capabilities in LLMs are not yet perfect, suggesting it might be overly optimistic to depend heavily on these techniques at this stage.

Nevertheless, all the aforementioned methods focus on post-generation analysis of LLM outputs, whereas our work is dedicated to conducting online safety analysis for LLMs. This distinction underscores a fundamentally different approach: instead of evaluating outputs after generation, we aim to monitor and ensure safety in real-time as LLMs operate.

9.3 Benchmark for LLM

Developing benchmarks for LLMs is a crucial step in understanding the current limitations and providing insights for further improvements. Given the multifaceted capabilities of LLMs to tackle a wide range of tasks, their benchmarks are inherently more complex compared to traditional DL models. These benchmarks encompass a variety of tasks, including natural language understanding [185]–[187], reasoning [188]–[190], and code generation [116], [191], [192], evaluated from multiple perspectives, such as correctness [186], factuality [75], [193], robustness [74], [194], fairness [73], and privacy [195]. These datasets may be human-labeled [116], [196], [197], illustrating the meticulous effort for better quality and relevance. Others are extracted from external resources, leveraging existing repositories of knowledge [19], [198]. Some datasets are transformed from other datasets, undergoing modifications to better suit specific tasks or objectives [199], [200]. Additionally, there are datasets that are labeled or generated by AI models themselves, reflecting a growing trend of leveraging AI to prepare data for evaluation [16], [75].

Among the notable studies in this field, HELM [186] is an important work that provides an extensive evaluation of LLMs. HELM performed evaluation across seven metrics in 42 different scenarios for 30 language models, offering comprehensive insights into the capabilities and limitations of current LLMs. Another benchmark, DecodingTrust [74], focuses on assessing LLMs from eight perspectives of trustworthiness, highlighting the previously unknown vulnerabilities to trustworthiness threats. TrustLLM [73] is a more recent benchmark that evaluates 16 LLMs using over 30 datasets across six critical dimensions: truthfulness, safety, fairness, robustness, privacy, and machine ethics. These benchmarks collectively contribute to a deeper understanding of LLM performance and ethical implications, guiding the development of more reliable and equitable AI systems. To the best of our knowledge, our study is the first to assess the effectiveness of various online safety analysis methods applied to both open-source and closed-source LLMs. Furthermore, our evaluation encompasses tasks within both the natural language processing and software engineering domains, offering a broad and diverse testing ground for future research.

10 Conclusion

In this paper, we introduce the first benchmark of online safety analysis methods for LLMs in different domains, which can be deemed as a base assessment framework for developing and enhancing new online safety analysis methods. We start with a pilot study to show the presence of unsafe output indeed emerges at the early stage of generation and motivate the necessity of applying online safety analysis. Based on the constructed benchmark, which contains a series of analysis methods, LLMs, datasets, and evaluation metrics, we systematically and extensively measure the effectiveness and performance of state-of-the-art online safety analysis methods. We also propose simple yet effective hybridization approaches, which can leverage the advantages of diverse individual analysis methods and achieve better performance. We believe that the novel constructed benchmark and empirical evaluation can offer developers and academic researchers valuable insights on developing and understanding more LLM-specific online safety analysis methods and boost the development of quality assurance for LLM.

References

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat

et al., "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *Chi conference on human factors in computing systems extended abstracts*, 2022, pp. 1–7.

[3] A. Z. Ren, A. Dixit, A. Bodrova, S. Singh, S. Tu, N. Brown, P. Xu, L. Takayama, F. Xia, J. Varley *et al.*, "Robots that ask for help: Uncertainty alignment for large language model planners," *arXiv preprint arXiv:2307.01928*, 2023.

[4] Z. Zhou, J. Song, K. Yao, Z. Shu, and L. Ma, "Isr-llm: Iterative self-refined large language model for long-horizon sequential task planning," *arXiv preprint arXiv:2308.13724*, 2023.

[5] M. Cascella, J. Montomoli, V. Bellini, and E. Bignami, "Evaluating the feasibility of chatgpt in healthcare: an analysis of multiple clinical and research scenarios," *Journal of medical systems*, vol. 47, no. 1, p. 33, 2023.

[6] S. Wu, O. Irsoy, S. Lu, V. Dabravolski, M. Dredze, S. Gehrmann, P. Kambadur, D. Rosenberg, and G. Mann, "Bloomberggpt: A large language model for finance," *arXiv preprint arXiv:2303.17564*, 2023.

[7] G. Wenzek, M.-A. Lachaux, A. Conneau, V. Chaudhary, F. Guzmán, A. Joulin, and E. Grave, "Ccnet: Extracting high quality monolingual datasets from web crawl data," *arXiv preprint arXiv:1911.00359*, 2019.

[8] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[9] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima *et al.*, "The pile: An 800gb dataset of diverse text for language modeling," *arXiv preprint arXiv:2101.00027*, 2020.

[10] B. Goertzel, "Artificial general intelligence: concept, state of the art, and future prospects," *Journal of Artificial General Intelligence*, vol. 5, no. 1, pp. 1–48, 2014.

[11] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.

[12] X. Huang, W. Ruan, W. Huang, G. Jin, Y. Dong, C. Wu, S. Bensalem, R. Mu, Y. Qi, X. Zhao *et al.*, "A survey of safety and trustworthiness of large language models through the lens of verification and validation," *arXiv preprint arXiv:2305.11391*, 2023.

[13] H. Inan, K. Upasani, J. Chi, R. Rungta, K. Iyer, Y. Mao, M. Tontchev, Q. Hu, B. Fuller, D. Testuggine *et al.*, "Llama guard: Llm-based input-output safeguard for human-ai conversations," *arXiv preprint arXiv:2312.06674*, 2023.

[14] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[15] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.

[16] S. Gehman, S. Gururangan, M. Sap, Y. Choi, and N. A. Smith, "Realtoxicityprompts: Evaluating neural toxic degeneration in language models," *arXiv preprint arXiv:2009.11462*, 2020.

[17] Y. Li, M. Du, R. Song, X. Wang, and Y. Wang, "A survey on fairness in large language models," *arXiv preprint arXiv:2308.10149*, 2023.

[18] H. Wang, G. Ma, C. Yu, N. Gui, L. Zhang, Z. Huang, S. Ma, Y. Chang, S. Zhang, L. Shen *et al.*, "Are large language models really robust to word-level perturbations?" *arXiv preprint arXiv:2309.11166*, 2023.

[19] T. Liu, Y. Zhang, C. Brockett, Y. Mao, Z. Sui, W. Chen, and B. Dolan, "A token-level reference-free hallucination detection benchmark for free-form text generation," *arXiv preprint arXiv:2104.08704*, 2021.

[20] M. Rateike, C. Cintas, J. Wamburu, T. Akumu, and S. Speakman, "Weakly supervised detection of hallucinations in llm activations," *arXiv preprint arXiv:2312.02798*, 2023.

[21] G. News, "B.c. lawyer who used fake, ai-generated cases faces law society probe, possible costs," 2024. [Online]. Available: https://globalnews.ca/news/10263897/fake-ai-cases-b-c-supreme-court/

[22] N. Ousidhoum, X. Zhao, T. Fang, Y. Song, and D.-Y. Yeung, "Probing toxic content in large pre-trained language models," 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)*, 2021, pp. 4262–4274.

[23] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18.

[24] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*, 2018, pp. 120–131.

[25] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1039–1049.

[26] B. Li, P. Qi, B. Liu, S. Di, J. Liu, J. Pei, J. Yi, and B. Zhou, "Trustworthy ai: From principles to practices," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–46, 2023.

[27] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella, "Testing machine learning based systems: a systematic mapping," *Empirical Software Engineering*, vol. 25, pp. 5193–5254, 2020.

[28] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer, 2017, pp. 97–117.

[29] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE symposium on security and privacy (SP)*. IEEE, 2018, pp. 3–18.

[30] L. E. Lwakatare, A. Raj, I. Crnkovic, J. Bosch, and H. H. Olsson, "Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions," *Information and software technology*, vol. 127, p. 106368, 2020.

[31] X. Gao, Y. Feng, Y. Yin, Z. Liu, Z. Chen, and B. Xu, "Adaptive test selection for deep neural networks," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 73–85.

[32] Z. Yang, J. Shi, M. H. Asyrofi, and D. Lo, "Revisiting neuron coverage metrics and quality of deep neural networks," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 408–419.

[33] V. Riccio and P. Tonella, "When and why test generators for deep learning produce invalid inputs: an empirical study," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1161–1173.

[34] J. Wang, H. Qiu, Y. Rong, H. Ye, Q. Li, Z. Li, and C. Zhang, "Bet: black-box efficient testing for convolutional neural networks," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022, pp. 164–175.

[35] J. Wang, J. Chen, Y. Sun, X. Ma, D. Wang, J. Sun, and P. Cheng, "Robot: Robustness-oriented testing for deep learning systems," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 300–311.

[36] Z. Wang, H. You, J. Chen, Y. Zhang, X. Dong, and W. Zhang, "Prioritizing test inputs for deep neural networks via mutation analysis," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 397–409.

[37] Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, and Z. Chen, "Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020, pp. 177–188.

[38] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, 2018, pp. 303–314.

[39] M. Sotoudeh and A. V. Thakur, "Provable repair of deep neural networks," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021, pp. 588–603.

[40] A. Stocco, M. Weiss, M. Calzana, and P. Tonella, "Misbehaviour prediction for autonomous driving systems," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, 2020, pp. 359–371.

[41] B. Sun, J. Sun, L. H. Pham, and J. Shi, "Causality-based neural network repair," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 338–349.

[42] J. Kim, N. Humbatova, G. Jahangirova, P. Tonella, and S. Yoo, "Repairing dnn architecture: Are we there yet?" in *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2023, pp. 234–245.

[43] X. Yang, T. Yamaguchi, H.-D. Tran, B. Hoxha, T. T. Johnson, and D. Prokhorov, "Neural network repair with reachability analysis," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2022, pp. 221–236.

[44] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Computer Science Review*, vol. 37, p. 100270, 2020.

[45] B. Sun, J. Sun, T. Dai, and L. Zhang, "Probabilistic verification of neural networks against group fairness," in *Formal Methods: 24th International Symposium, FM 2021, Virtual Event, November 20–26, 2021, Proceedings 24*. Springer, 2021, pp. 83–102.

[46] T. A. Henzinger, A. Lukina, and C. Schilling, "Outside the box: Abstraction-based monitoring of neural networks," *arXiv preprint arXiv:1911.09032*, 2019.

[47] A. Lukina, C. Schilling, and T. A. Henzinger, "Into the unknown: Active monitoring of neural networks," in *International Conference on Runtime Verification*. Springer, 2021, pp. 42–61.

[48] C.-H. Cheng, G. Nührenberg, and H. Yasuoka, "Runtime monitoring neuron activation patterns," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 300–303.

[49] J. Guerin, R. S. Ferreira, K. Delmas, and J. Guiochet, "Unifying evaluation of machine learning safety monitors," in *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2022, pp. 414–422.

[50] J. Guerin, K. Delmas, and J. Guiochet, "Evaluation of runtime monitoring for uav emergency landing," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 9703–9709.

[51] A. Zolfagharian, M. Abdellatif, L. C. Briand *et al.*, "Smarla: A safety monitoring approach for deep reinforcement learning agents," *arXiv preprint arXiv:2308.02594*, 2023.

[52] K. Aslansefat, I. Sorokos, D. Whiting, R. Tavakoli Kolagari, and Y. Papadopoulos, "Safeml: safety monitoring of machine learning classifiers through statistical difference measures," in *International Symposium on Model-Based Safety and Assessment*. Springer, 2020, pp. 197–211.

[53] A. Stocco, P. J. Nunes, M. d'Amorim, and P. Tonella, "Thirdeye: Attention maps for safe autonomous driving systems," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–12.

[54] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[55] T. Markov, C. Zhang, S. Agarwal, F. E. Nekoul, T. Lee, S. Adler, A. Jiang, and L. Weng, "A holistic approach to undesired content detection in the real world," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 12, 2023, pp. 15 009–15 018.

[56] C. Zhou, G. Neubig, J. Gu, M. Diab, F. Guzmán, L. Zettlemoyer, and M. Ghazvininejad, "Detecting hallucinated content in conditional neural sequence generation," in *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Online: Association for Computational Linguistics, Aug. 2021, pp. 1393–1404. [Online]. Available: https://aclanthology.org/2021.findings-acl.120

[57] M. Bhatt, S. Chennabasappa, C. Nikolaidis, S. Wan, I. Evtimov, D. Gabi, D. Song, F. Ahmad, C. Aschermann, L. Fontana *et al.*, "Purple llama cyberseceval: A secure coding benchmark for language models," *arXiv preprint arXiv:2312.04724*, 2023.

[58] S. Achintalwar, A. A. Garcia, A. Anaby-Tavor, I. Baldini, S. E. Berger, B. Bhattacharjee, D. Bouneffouf, S. Chaudhury, P.-Y. Chen, L. Chiazor *et al.*, "Detectors for safe and reliable llms: Implementations, uses, and limitations," *arXiv preprint arXiv:2403.06009*, 2024.

[59] Z. Zhang, Y. Lu, J. Ma, D. Zhang, R. Li, P. Ke, H. Sun, L. Sha, Z. Sui, H. Wang *et al.*, "Shieldlm: Empowering llms as aligned, customizable and explainable safety detectors," *arXiv preprint arXiv:2402.16444*, 2024.

[60] P. Manakul, A. Liusie, and M. J. Gales, "Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models," *arXiv preprint arXiv:2303.08896*, 2023.

[61] Z. Lin, S. Trivedi, and J. Sun, "Generating with confidence: Uncertainty quantification for black-box large language models," *arXiv preprint arXiv:2305.19187*, 2023.

[62] M. Xiong, Z. Hu, X. Lu, Y. LI, J. Fu, J. He, and B. Hooi, "Can LLMs express their uncertainty? an empirical evaluation of confidence elicitation in LLMs," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=gjeQKFxFpZ

[63] L. Kuhn, Y. Gal, and S. Farquhar, "Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=VD-AYtP0dve

[64] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[65] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[66] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[67] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[68] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[69] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[70] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[71] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[72] U. Germann, "Greedy decoding for statistical machine translation in almost linear time," in *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, 2003, pp. 72–79.

[73] L. Sun, Y. Huang, H. Wang, S. Wu, Q. Zhang, C. Gao, Y. Huang, W. Lyu, Y. Zhang, X. Li *et al.*, "Trustllm: Trustworthiness in large language models," *arXiv preprint arXiv:2401.05561*, 2024.

[74] B. Wang, W. Chen, H. Pei, C. Xie, M. Kang, C. Zhang, C. Xu, Z. Xiong, R. Dutta, R. Schaeffer, S. T. Truong, S. Arora, M. Mazeika, D. Hendrycks, Z. Lin, Y. Cheng, S. Koyejo, D. Song, and B. Li, "Decodingtrust: A comprehensive assessment of trustworthiness in GPT models," in *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. [Online]. Available: https://openreview.net/forum?id=kaHpo8OZw2

[75] S. Lin, J. Hilton, and O. Evans, "Truthfulqa: Measuring how models mimic human falsehoods," *arXiv preprint arXiv:2109.07958*, 2021.

[76] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A survey on large language model (llm) security and privacy: The good, the bad, and the ugly," *High-Confidence Computing*, p. 100211, 2024.

[77] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, P. Isabelle, E. Charniak, and D. Lin, Eds. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 311–318. [Online]. Available: https://aclanthology.org/P02-1040

[78] D. Hendrycks, C. Burns, S. Basart, A. Critch, J. Li, D. Song, and J. Steinhardt, "Aligning ai with shared human values," *arXiv preprint arXiv:2008.02275*, 2020.

[79] T. Bolukbasi, K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai, "Man is to computer programmer as woman is to homemaker? debiasing word embeddings," *Advances in neural information processing systems*, vol. 29, 2016.

[80] A. Azaria and T. Mitchell, "The internal state of an llm knows when it's lying," 2023.

[81] K. Liu, S. Casper, D. Hadfield-Menell, and J. Andreas, "Cognitive dissonance: Why do language model outputs disagree with internal representations of truthfulness?" *arXiv preprint arXiv:2312.03729*, 2023.

[82] J. Welbl, A. Glaese, J. Uesato, S. Dathathri, J. Mellor, L. A. Hendricks, K. Anderson, P. Kohli, B. Coppin, and P.-S. Huang, "Challenges in detoxifying language models," *arXiv preprint arXiv:2109.07445*, 2021.

[83] T. Madiega, "Artificial intelligence act," *European Parliament: European Parliamentary Research Service*, 2021.

[84] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 1–36, 2020.

[85] H. B. Braiek and F. Khomh, "On testing machine learning programs," *Journal of Systems and Software*, vol. 164, p. 110542, 2020.

[86] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "Testing deep neural networks," *arXiv preprint arXiv:1803.04792*, 2018.

[87] J. Wang, G. Dong, J. Sun, X. Wang, and P. Zhang, "Adversarial sample detection for deep neural network through model mutation testing," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1245–1256.

[88] S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter, "Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification," *Advances in Neural Information Processing Systems*, vol. 34, pp. 29 909–29 921, 2021.

[89] H.-D. Tran, X. Yang, D. Manzanas Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, "Nnv: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems," in *International Conference on Computer Aided Verification*. Springer, 2020, pp. 3–17.

[90] D. Shriver, S. Elbaum, and M. B. Dwyer, "Dnnv: A framework for deep neural network verification," in *International Conference on Computer Aided Verification*. Springer, 2021, pp. 137–150.

[91] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer, 2017, pp. 3–29.

[92] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[93] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[94] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 187–210, 2018.

[95] S. Wang, Z. Zhao, X. Ouyang, Q. Wang, and D. Shen, "Chatcad: Interactive computer-aided diagnosis on medical image using large language models," *arXiv preprint arXiv:2302.07257*, 2023.

[96] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, "Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications," *Lectures on Runtime Verification: Introductory and Advanced Topics*, pp. 135–175, 2018.

[97] C. Sánchez, G. Schneider, W. Ahrendt, E. Bartocci, D. Bianculli, C. Colombo, Y. Falcone, A. Francalanza, S. Krstić, J. M. Lourenço *et al.*, "A survey of challenges for runtime verification from advanced application domains (beyond software)," *Formal Methods in System Design*, vol. 54, pp. 279–335, 2019.

[98] D. Ratasich, F. Khalid, F. Geissler, R. Grosu, M. Shafique, and E. Bartocci, "A roadmap toward the resilient internet of things for cyber-physical systems," *IEEE Access*, vol. 7, pp. 13 260–13 283, 2019.

[99] Y. Falcone, S. Krstić, G. Reger, and D. Traytel, "A taxonomy for classifying runtime verification tools," *International Journal on Software Tools for Technology Transfer*, vol. 23, no. 2, pp. 255–284, 2021.

[100] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.

[101] R. Salay, M. Angus, and K. Czarnecki, "A safety analysis method for perceptual components in automated driving," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2019, pp. 24–34.

[102] Y. Huang, J. Song, Z. Wang, H. Chen, and L. Ma, "Look before you leap: An exploratory study of uncertainty measurement for large language models," *arXiv preprint arXiv:2307.10236*, 2023.

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

[104] Google, "Perspective api," 2024. [Online]. Available: https://support.perspectiveapi.com/s/about-the-api?language=en_US

[105] K. Krishna and M. N. Murty, "Genetic k-means algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 29, no. 3, pp. 433–439, 1999.

[106] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Stanford alpaca: An instruction-following llama model," https://github.com/tatsu-lab/stanford_alpaca, 2023.

[107] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing, "Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality," March 2023. [Online]. Available: https://lmsys.org/blog/2023-03-30-vicuna/

[108] ShareGPT, "Sharegpt," 2024. [Online]. Available: https://sharegpt.com/

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

[110] OpenAI, "Gpt 3.5," 2023. [Online]. Available: https://platform.openai.com/docs/models/gpt-3-5

[111] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Alpaca: A strong, replicable instruction-following model," *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, vol. 3, no. 6, p. 7, 2023.

[112] M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer, "Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension," *arXiv preprint arXiv:1705.03551*, 2017.

[113] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee *et al.*, "Natural questions: a benchmark for question answering research," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 453–466, 2019.

[114] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand *et al.*, "Findings of the 2014 workshop on statistical machine translation," in *Proceedings of the ninth workshop on statistical machine translation*, 2014, pp. 12–58.

[115] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[116] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[117] J. Guérin, K. Delmas, R. Ferreira, and J. Guiochet, "Out-of-distribution detection is not all you need," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 37, no. 12, 2023, pp. 14 829–14 837.

[118] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[119] X. Xie, J. Song, Z. Zhou, F. Zhang, and L. Ma, "Mosaic: Model-based safety analysis framework for ai-enabled cyber-physical systems," *arXiv preprint arXiv:2305.03882*, 2023.

[120] Z. Zhang, P. Arcaini, and X. Xie, "Online reset for signal temporal logic monitoring," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4421–4432, 2022.

[121] J. V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia, "Robust online monitoring of signal temporal logic," *Formal Methods in System Design*, vol. 51, pp. 5–30, 2017.

[122] M. A. Ganaie, M. Hu, A. K. Malik, M. Tanveer, and P. N. Suganthan, "Ensemble deep learning: A review," *Engineering Applications of Artificial Intelligence*, vol. 115, p. 105151, 2022.

[123] T. G. Dietterich *et al.*, "Ensemble learning," *The handbook of brain theory and neural networks*, vol. 2, no. 1, pp. 110–125, 2002.

[124] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley interdisciplinary reviews: data mining and knowledge discovery*, vol. 8, no. 4, p. e1249, 2018.

[125] J. Wu, C. Lu, A. Arrieta, T. Yue, and S. Ali, "Reality bites: Assessing the realism of driving scenarios with large language models," *arXiv preprint arXiv:2403.09906*, 2024.

[126] J. Song, Z. Zhou, J. Liu, C. Fang, Z. Shu, and L. Ma, "Self-refined large language model as automated reward function designer for deep reinforcement learning in robotics," *arXiv preprint arXiv:2309.06687*, 2023.

[127] A. Serban, K. van der Blom, H. Hoos, and J. Visser, "Adoption and effects of software engineering best practices in machine learning," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020, pp. 1–12.

[128] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 291–300.

[129] Y. Liu, L. Ma, and J. Zhao, "Secure deep learning engineering: A road towards quality assurance of intelligent systems," in *Formal Methods and Software Engineering: 21st International Conference on Formal Engineering Methods, ICFEM 2019, Shenzhen, China, November 5–9, 2019, Proceedings 21*. Springer, 2019, pp. 3–15.

[130] C. Cortes, G. DeSalvo, and M. Mohri, "Learning with rejection," in *Algorithmic Learning Theory: 27th International Conference, ALT 2016, Bari, Italy, October 19-21, 2016, Proceedings 27*. Springer, 2016, pp. 67–82.

[131] Y. Geifman and R. El-Yaniv, "Selectivenet: A deep neural network with an integrated reject option," in *International conference on machine learning*. PMLR, 2019, pp. 2151–2159.

[132] ——, "Selective classification for deep neural networks," *Advances in neural information processing systems*, vol. 30, 2017.

[133] A. Fisch, T. S. Jaakkola, and R. Barzilay, "Calibrated selective classification," *Transactions on Machine Learning Research*, 2022. [Online]. Available: https://openreview.net/forum?id=zFhNBs8GaV

[134] G. Xia and C.-S. Bouganis, "Augmenting softmax information for selective classification with out-of-distribution data," in *Proceedings of the Asian Conference on Computer Vision*, 2022, pp. 1995–2012.

[135] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=Hkg4TI9xl

[136] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," *Advances in neural information processing systems*, vol. 31, 2018.

[137] S. Liang, Y. Li, and R. Srikant, "Enhancing the reliability of out-of-distribution image detection in neural networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=H1VGkIxRZ

[138] Y.-C. Hsu, Y. Shen, H. Jin, and Z. Kira, "Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 951–10 960.

[139] W. Liu, X. Wang, J. Owens, and Y. Li, "Energy-based out-of-distribution detection," *Advances in neural information processing systems*, vol. 33, pp. 21 464–21 475, 2020.

[140] D. Berend, X. Xie, L. Ma, L. Zhou, Y. Liu, C. Xu, and J. Zhao, "Cats are not fish: Deep learning testing calls for out-of-distribution awareness," in *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*, 2020, pp. 1041–1052.

[141] Y. Sun, C. Guo, and Y. Li, "React: Out-of-distribution detection with rectified activations," *Advances in Neural Information Processing Systems*, vol. 34, pp. 144–157, 2021.

[142] Y. H. Ahn, G.-M. Park, and S. T. Kim, "Line: Out-of-distribution detection by leveraging important neurons," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2023, pp. 19 852–19 862.

[143] M. Sabokrou, M. Khalooei, M. Fathy, and E. Adeli, "Adversarially learned one-class classifier for novelty detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3379–3388.

[144] D. Hendrycks, M. Mazeika, and T. Dietterich, "Deep anomaly detection with outlier exposure," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HyxCxhRcY7

[145] X. Zhang, X. Xie, L. Ma, X. Du, Q. Hu, Y. Liu, J. Zhao, and M. Sun, "Towards characterizing adversarial defects of deep learning software from the lens of uncertainty," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 739–751.

[146] F. Granese, M. Romanelli, D. Gorla, C. Palamidessi, and P. Piantanida, "Doctor: A simple method for detecting misclassification errors," *Advances in Neural Information Processing Systems*, vol. 34, pp. 5669–5681, 2021.

[147] J. Chen, F. Liu, B. Avci, X. Wu, Y. Liang, and S. Jha, "Detecting errors and estimating accuracy on unlabeled data with self-training ensembles," *Advances in Neural Information Processing Systems*, vol. 34, pp. 14 980–14 992, 2021.

[148] Z. Nado, N. Band, M. Collier, J. Djolonga, M. W. Dusenberry, S. Farquhar, Q. Feng, A. Filos, M. Havasi, R. Jenatton *et al.*, "Uncertainty baselines: Benchmarks for uncertainty & robustness in deep learning," *arXiv preprint arXiv:2106.04015*, 2021.

[149] F. Zhu, Z. Cheng, X.-Y. Zhang, and C.-L. Liu, "Openmix: Exploring outlier samples for misclassification detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 12 074–12 083.

[150] Y. Huang, L. Ma, and Y. Li, "Patchcensor: Patch robustness certification for transformers via exhaustive testing," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 6, pp. 1–34, 2023.

[151] J. Kim, R. Feldt, and S. Yoo, "Evaluating surprise adequacy for deep learning system testing," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 2, pp. 1–29, 2023.

[152] E. D. C. Gomes, M. Romanelli, G. Pichler, and P. Piantanida, "A data-driven measure of relative uncertainty for misclassification detection," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=ruGY8v10mK

[153] Y. Liu, C. X. TIAN, H. Li, L. Ma, and S. Wang, "Neuron activation coverage: Rethinking out-of-distribution detection and generalization," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=SNGXbZtK6Q

[154] R. S. Ferreira, J. Arlat, J. Guiochet, and H. Waeselynck, "Benchmarking safety monitors for image classifiers with machine learning," in *2021 IEEE 26th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2021, pp. 7–16.

[155] Y. Xiao, I. Beschastnikh, D. S. Rosenblum, C. Sun, S. Elbaum, Y. Lin, and J. S. Dong, "Self-checking deep neural networks in deployment. in 2021 ieee/acm 43rd international conference on software engineering (icse)," *IEEE, 372ś384*, 2021.

[156] S. Junges, H. Torfah, and S. A. Seshia, "Runtime monitors for markov decision processes," in *International Conference on Computer Aided Verification*. Springer, 2021, pp. 553–576.

[157] A. Balakrishnan, J. Deshmukh, B. Hoxha, T. Yamaguchi, and G. Fainekos, "Percemon: online monitoring for perception systems," in *Runtime Verification: 21st International Conference, RV 2021, Virtual Event, October 11–14, 2021, Proceedings 21*. Springer, 2021, pp. 297–308.

[158] J. Ayerdi, A. Iriarte, P. Valle, I. Roman, M. Illarramendi, and A. Arrieta, "Metamorphic runtime monitoring of autonomous driving systems," *arXiv preprint arXiv:2310.07414*, 2023.

[159] B. Könighofer, J. Rudolf, A. Palmisano, M. Tappler, and R. Bloem, "Online shielding for reinforcement learning," *Innovations in Systems and Software Engineering*, vol. 19, no. 4, pp. 379–394, 2023.

[160] S. Gronauer, T. Haider, F. S. da Roza, and K. Diepold, "Reinforcement learning with ensemble model predictive safety certification," *arXiv preprint arXiv:2402.04182*, 2024.

[161] Y. Sun, C. M. Poskitt, X. Zhang, and J. Sun, "Redriver: Runtime enforcement for autonomous vehicles," *arXiv preprint arXiv:2401.02253*, 2024.

[162] M. Cao, M. Fatemi, J. C. Cheung, and S. Shabanian, "Systematic rectification of language models via dead-end analysis," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=k8_yVW3Wqln

[163] S. Mudgal, J. Lee, H. Ganapathy, Y. Li, T. Wang, Y. Huang, Z. Chen, H.-T. Cheng, M. Collins, T. Strohman *et al.*, "Controlled decoding from language models," *arXiv preprint arXiv:2310.17022*, 2023.

[164] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, 2023.

[165] W. Zhang, Y. Deng, B. Liu, S. J. Pan, and L. Bing, "Sentiment analysis in the era of large language models: A reality check," *arXiv preprint arXiv:2305.15005*, 2023.

[166] X. Pu, M. Gao, and X. Wan, "Summarization is (almost) dead," *arXiv preprint arXiv:2309.09558*, 2023.

[167] C. Lemieux, J. P. Inala, S. K. Lahiri, and S. Sen, "Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models," in *International conference on software engineering (ICSE)*, 2023.

[168] Z. Liu, C. Chen, J. Wang, X. Che, Y. Huang, J. Hu, and Q. Wang, "Fill in the blank: Context-aware automated text input generation for mobile gui testing," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1355–1367.

[169] J. A. Prenner, H. Babii, and R. Robbes, "Can openai's codex fix bugs? an evaluation on quixbugs," in *Proceedings of the Third International Workshop on Automated Program Repair*, 2022, pp. 69–75.

[170] D. Sobania, M. Briesch, C. Hanna, and J. Petke, "An analysis of the automatic bug fixing performance of chatgpt," *arXiv preprint arXiv:2301.08653*, 2023.

[171] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," 2023.

[172] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems," *arXiv preprint arXiv:2310.03533*, 2023.

[173] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," 2024.

[174] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software testing with large language models: Survey, landscape, and vision," *IEEE Transactions on Software Engineering*, 2024.

[175] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg *et al.*, "Sparks of artificial general intelligence: Early experiments with gpt-4," *arXiv preprint arXiv:2303.12712*, 2023.

[176] Y. Zhang, Y. Li, L. Cui, D. Cai, L. Liu, T. Fu, X. Huang, E. Zhao, Y. Zhang, Y. Chen *et al.*, "Siren's song in the ai ocean: a survey on hallucination in large language models," *arXiv preprint arXiv:2309.01219*, 2023.

[177] Z. Xu, S. Jain, and M. Kankanhalli, "Hallucination is inevitable: An innate limitation of large language models," *arXiv preprint arXiv:2401.11817*, 2024.

[178] Y. Liu, G. Deng, Z. Xu, Y. Li, Y. Zheng, Y. Zhang, L. Zhao, T. Zhang, and Y. Liu, "Jailbreaking chatgpt via prompt engineering: An empirical study," *arXiv preprint arXiv:2305.13860*, 2023.

[179] J. Huang, X. Chen, S. Mishra, H. S. Zheng, A. W. Yu, X. Song, and D. Zhou, "Large language models cannot self-correct reasoning yet," *arXiv preprint arXiv:2310.01798*, 2023.

[180] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[181] X. Chen, M. Lin, N. Schärli, and D. Zhou, "Teaching large language models to self-debug," *arXiv preprint arXiv:2304.05128*, 2023.

[182] L. Pan, M. Saxon, W. Xu, D. Nathani, X. Wang, and W. Y. Wang, "Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies," *arXiv preprint arXiv:2308.03188*, 2023.

[183] Z. Gou, Z. Shao, Y. Gong, yelong shen, Y. Yang, N. Duan, and W. Chen, "CRITIC: Large language models can self-correct with tool-interactive critiquing," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=Sx038qxjek

[184] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang *et al.*, "Self-refine: Iterative refinement with self-feedback," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[185] Y. Bang, S. Cahyawijaya, N. Lee, W. Dai, D. Su, B. Wilie, H. Lovenia, Z. Ji, T. Yu, W. Chung, Q. V. Do, Y. Xu, and P. Fung, "A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity," 2023.

[186] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, B. Newman, B. Yuan, B. Yan, C. Zhang, C. A. Cosgrove, C. D. Manning, C. Re, D. Acosta-Navas, D. A. Hudson, E. Zelikman, E. Durmus, F. Ladhak, F. Rong, H. Ren, H. Yao, J. WANG, K. Santhanam, L. Orr, L. Zheng, M. Yuksekgonul, M. Suzgun, N. Kim, N. Guha, N. S. Chatterji, O. Khattab, P. Henderson, Q. Huang, R. A. Chi, S. M. Xie, S. Santurkar, S. Ganguli, T. Hashimoto, T. Icard, T. Zhang, V. Chaudhary, W. Wang, X. Li, Y. Mai, Y. Zhang, and Y. Koreeda, "Holistic evaluation of language models," *Transactions on Machine Learning Research*, 2023, featured Certification, Expert Certification. [Online]. Available: https://openreview.net/forum?id=iO4LZibEqW

[187] C. Qin, A. Zhang, Z. Zhang, J. Chen, M. Yasunaga, and D. Yang, "Is chatgpt a general-purpose natural language processing task solver?" *arXiv preprint arXiv:2302.06476*, 2023.

[188] S. Frieder, L. Pinchetti, R.-R. Griffiths, T. Salvatori, T. Lukasiewicz, P. Petersen, and J. Berner, "Mathematical capabilities of chatgpt," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[189] H. Liu, R. Ning, Z. Teng, J. Liu, Q. Zhou, and Y. Zhang, "Evaluating the logical reasoning ability of chatgpt and gpt-4," *arXiv preprint arXiv:2304.03439*, 2023.

[190] Y. Fu, L. Ou, M. Chen, Y. Wan, H. Peng, and T. Khot, "Chain-of-thought hub: A continuous effort to measure large language models' reasoning performance," *arXiv preprint arXiv:2305.17306*, 2023.

[191] X. Du, M. Liu, K. Wang, H. Wang, J. Liu, Y. Chen, J. Feng, C. Sha, X. Peng, and Y. Lou, "Classeval: A manually-crafted benchmark for evaluating llms on class-level code generation," *arXiv preprint arXiv:2308.01861*, 2023.

[192] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[193] O. Honovich, R. Aharoni, J. Herzig, H. Taitelbaum, D. Kukliansy, V. Cohen, T. Scialom, I. Szpektor, A. Hassidim, and Y. Matias, "True: Re-evaluating factual consistency evaluation," *arXiv preprint arXiv:2204.04991*, 2022.

[194] K. Zhu, J. Wang, J. Zhou, Z. Wang, H. Chen, Y. Wang, L. Yang, W. Ye, N. Z. Gong, Y. Zhang *et al.*, "Promptbench: Towards evaluating the robustness of large language models on adversarial prompts," *arXiv preprint arXiv:2306.04528*, 2023.

[195] A. Hamid, H. R. Samidi, T. Finin, P. Pappachan, and R. Yus, "Genaipabench: A benchmark for generative ai-based privacy assistants," *arXiv preprint arXiv:2309.05138*, 2023.

[196] S. Santhanam, B. Hedayatnia, S. Gella, A. Padmakumar, S. Kim, Y. Liu, and D. Hakkani-Tür, "Rome was built in 1776: A case study on factual correctness in knowledge-grounded response generation," 2021.

[197] O. Honovich, L. Choshen, R. Aharoni, E. Neeman, I. Szpektor, and O. Abend, "Q2: Evaluating factual consistency in knowledge-grounded dialogues via question generation and question answering," *arXiv preprint arXiv:2104.08202*, 2021.

[198] J. Thorne, A. Vlachos, C. Christodoulopoulos, and A. Mittal, "Fever: a large-scale dataset for fact extraction and verification," *arXiv preprint arXiv:1803.05355*, 2018.

[199] B. Wang, C. Xu, S. Wang, Z. Gan, Y. Cheng, J. Gao, A. H. Awadallah, and B. Li, "Adversarial glue: A multi-task benchmark for robustness evaluation of language models," *arXiv preprint arXiv:2111.02840*, 2021.

[200] X. Wang, Q. Liu, T. Gui, Q. Zhang, Y. Zou, X. Zhou, J. Ye, Y. Zhang, R. Zheng, Z. Pang, Q. Wu, Z. Li, C. Zhang, R. Ma, Z. Fei, R. Cai, J. Zhao, X. Hu, Z. Yan, Y. Tan, Y. Hu, Q. Bian, Z. Liu, S. Qin, B. Zhu, X. Xing, J. Fu, Y. Zhang, M. Peng, X. Zheng, Y. Zhou, Z. Wei, X. Qiu, and X. Huang, "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*, H. Ji, J. C. Park, and R. Xia, Eds. Online: Association for Computational Linguistics, Aug. 2021, pp. 347–355. [Online]. Available: https://aclanthology.org/2021.acl-demo.41

[201] L. Zhuang, L. Wayne, S. Ya, and Z. Jun, "A robustly optimized bert pre-training approach with post-training," in *Proceedings of the 20th chinese national conference on computational linguistics*, 2021, pp. 1218–1227.

[202] M. Ahmed, R. Seraj, and S. M. S. Islam, "The k-means algorithm: A comprehensive survey and performance evaluation," *Electronics*, vol. 9, no. 8, p. 1295, 2020.

[203] U. Arora, W. Huang, and H. He, "Types of out-of-distribution texts and how to detect them," *arXiv preprint arXiv:2109.06827*, 2021.

[204] V. Raunak, A. Menezes, and M. Junczys-Dowmunt, "The curious case of hallucinations in neural machine translation," 2021.

[205] M. Post, "A call for clarity in reporting bleu scores," *arXiv preprint arXiv:1804.08771*, 2018.

APPENDIX

COLLECTED SAFETY ANALYSIS METHODS

In this section, we present more details about the online safety analysis methods studied. The safety analysis methods can be divided into three categories: black-box (Random and SelfCheckGPT), white-box (Box-based method and Quantitative method), and grey-box (Average Entropy, Maximum Entropy, Average Likelihood, and Maximum Likelihood).

- **Random**: Random method is a naive black prediction method.
- **SelfCheckGPT** [60]: It is a representative example of black-box methods, which is designed as a sampling-based hallucination detection approach that operates independently of external resources. The key idea is to compare sampled responses: if the model has knowledge of a concept, responses are expected to be consistent; otherwise, they may diverge. Several comparison methods are used to decide whether there exists a divergence, e.g., BERTScore [201]. Here, we focus on the best-performing method SelfCheckGPT with prompt. In particular, another LLM is queried to assess whether the given sentence is supported by the context using the prompt: *Context: ... Sentence: ... Is the sentence supported by the context above?* In this work, to adapt to diverse safety requirements, we modify the prompt to the queried LLM according to different safety requirements. An example of its application in toxicity detection is that, given a partial output, we ask the queried LLM using this prompt: *Are the following sentences toxic? ...* If the queried LLM returns *Yes*, we consider the safety requirement is violated, otherwise, it is safetisfied. We choose Alpaca as the query LLM due to the budget limit.
- **Box-based method** [46]: Box-based method [46] is a typical example of this type of technique. It is proposed to analyze neural networks to detect novel inputs. The method focuses on specific network layers close to the final output, where essential feature information is believed to be concentrated. Then, an abstraction, i.e., a box, is constructed to represent the set of known neuron valuations by observing the patterns exhibited by neurons in these layers. Based on the constructed box, the analyzer is trained to recognize typical input patterns for each class during runtime. If the observed pattern deviates significantly from the expected behavior, the method raises a warning about a possible novelty in the input. To perform an online safety analysis for LLM, we build the box on the collected safe input. At runtime, if a new coming prompt triggers generation that is out-of-box, we consider it as an unsafe one and raise a warning. we count the number of tokens that are out of the box and compare them to a predefined threshold to judge whether it is abnormal.
- **Quantitative method** [47]: This approach is proposed for novelty detection. Compared to traditional qualitative methods (like box-based), it assigns a numerical score to each observed input-output pair. Similar to the Box-based method, it first collects a set of internal states of the selected layer by running a small set of prompts. Then, it performs clustering, e.g., KMeans [202] on the states set and gets the centers of different clustering. At runtime, given a new input, for the selected token, it measures the distances between the current states and different centers. The minimum distance is compared with a pre-defined threshold: if the distance is larger, it is considered to be a potential violation of trustworthiness.
- **Average Entropy** [60], [102], [203]: Huang et al. [102] investigate the correlation between uncertainty and LLMs' performance on NLP tasks and code tasks. Manakul et al. [60] demonstrate that entropy and token likelihood can be used to detect hallucinations. Hence, we adapt entropy and token likelihood into the online safety analysis for LLM. We compute the sentence-level average entropy and compare it with a pre-defined threshold, to decide whether it could be a violation.

$$Avg(\mathcal{H})_i = \frac{1}{J}\sum_j \mathcal{H}_{ij}, \tag{9}$$

  where $\mathcal{H}_{ij}$ is the entropy of this token's probability distribution. The thresholds are different for each pair of tasks and models. To ease the presentation, we put the threshold on the project website.
- **Maximum Entropy** [60], [102], [203]: The maximum entropy is computed as

$$Max(\mathcal{H})_i = \max_j[\mathcal{H}_{ij}]. \tag{10}$$

- **Average Likelihood** [60], [102]: The average likelihood is computed as

$$Avg(-\log p)_i = -\frac{1}{J}\sum_j \log p_{ij}, \tag{11}$$

  where p_{ij} is the probability of a token at position j of a sentence i.
- **Maximum Likelihood** [60], [102]: The maximum likelihood is computed as

$$Max(-\log p)_i = \max_j(-\log p_{ij}). \tag{12}$$

COLLECTED METRICS

The full definition of the collected metrics is as follows. The goal of the evaluation metrics is to evaluate the online safety analysis methods from different perspectives.

For Safety Gain (SG), Residual Hazard (RH), and Availability Cost (AC), we follow the definitions of Guerin et al. [49]. The original definition is for single inference of classic DNNs and it can be adapted to the context of LLM.

SG is used to measure the safety addition from the method, which is defined as

$$SG = \int_{\mathcal{D}} p(x)\left(B^{\mathcal{S}}_{(N,m_N)}(x) - B^{\mathcal{S}}_{N}(x)\right)dx, \tag{13}$$

where $\mathcal{D}$ is an entire operational domain of the ML model, $B^{\mathcal{S}}_{\_}$ is the safety return of the model running with/without the online safety analysis method, N is the ML model, and m_N is the online safety analysis method, (N, m_N) is the model running under the supervision of the online safety analysis method. The safety return $B^{\mathcal{S}}_{\_}$ is a measurement of the safety of the model. In particular, in our study, it is defined as:

$$B^{\mathcal{S}}_{(N,m_N)}(x) = \begin{cases} 0 & \text{else,} \\ 1 & \text{if } \textit{unsafe} \text{ and } m_N(x) = 1. \end{cases} \tag{14}$$

In other words, the online safety analysis method gets safety scores only if it successfully reports an unsafe case. $m_N(x) = 1$ means the online safety analysis method reports the model is in an unsafe scenario. $B_N^{\mathcal{S}}(x) = 0$ in the study. A greater SG means the online safety analysis method provides more safety to the execution of the model. This metric represents the safety benefits of using the online safety analysis method. It focuses on how the online safety analysis method helps in preventing hazardous situations by detecting prediction errors and raising alerts when necessary. A higher Safety Gain indicates that the online safety analysis method is effectively improving the safety of the system.

RH is a measurement of the remaining unsafety, which is defined as:

$$RH = \int_{\mathcal{D}} p(x) \left(B_{N^*}^{\mathcal{S}}(x) - B_{(N,m_N)}^{\mathcal{S}}(x) \right) dx, \tag{15}$$

where $\mathcal{D}$ is an entire operational domain of the ML model, $B^{\mathcal{S}}$ is the safety return of the model running with/without the online safety analysis method, N^* is the ideal ML model that can avoid all unsafe cases, and m_N is the online safety analysis method. $B_{N^*}^{\mathcal{S}}(x)$ is defined as:

$$B_{f^*}^{\mathcal{S}}(x) = 1 \text{ if } \textit{unsafe}. \tag{16}$$

A greater RH indicates that the online safety analysis method misses more dangerous events during the execution of the online safety analysis method. The Residual Hazard metric measures the remaining safety gaps despite using the online safety analysis method. It compares the safety of the monitored model against the safety of an optimal model. A lower Residual Hazard value indicates that the online safety analysis method is successful in reducing the amount of hazard still present in the system.

AC is the decrease of the system performance, which is defined as:

$$AC = \int_{\mathcal{D}} p(x) \left(B_{N}^{\mathcal{M}}(x) - B_{(N,m_N)}^{\mathcal{M}}(x) \right) dx, \tag{17}$$

where $\mathcal{D}$ is an entire operational domain of the ML model, $B^{\mathcal{M}}$ is the mission return of the model running with/without the online safety analysis method, N is the ML model, and m_N is the online safety analysis method, (N, m_N) is the model running under the supervision of the online safety analysis method. A greater AC implies that the operation of the online safety analysis method is more costly. $B^{\mathcal{M}}$ is the mission return by applying the online safety analysis method on the system, which is defined as:

$$B_{(N,m_N)}^{\mathcal{M}}(x) = \begin{cases} 0.2 & \text{else}, \\ -2 & \text{if } \textit{safe} \text{ and } m_N(x) = 1. \end{cases} \tag{18}$$

That is to say, if the online safety analysis method performs a wrong reporting, the mission return will decrease by 2, while when the reporting is correct, the mission return will increase by 0.2. $B_N^{\mathcal{M}}(x) = 0$ in our case. Availability Cost quantifies the negative impact of the online safety analysis method on the system's performance. It evaluates how the online safety analysis method affects the system's ability to perform its mission by comparing the availability of the monitored model with the availability of the initial system. A lower Availability Cost suggests that the online safety analysis method is minimizing the performance impact on the system.

We also include classic metrics, i.e., Area Under the receiver operating characteristic Curve (AUC) and time cost. AUC [118] is a traditional classification task metric that summarizes the binary classifier's performance. The ROC curve is a graphical representation of the true positive rate against the false positive rate for various threshold values. AUC quantifies the overall performance of the model by calculating the area under this curve, with a value ranging from 0 to 1. A higher AUC value indicates better model performance. Moreover, time cost [119]–[121] is another important metric to measure the overhead of the online safety analysis method. The overhead of the online safety analysis method should be as small as possible, so that it does not bring additional debt to the model.

COLLECTED MODELS

For open-source model, max_new_token is set to 200 (for MBPP and HumanEval) and 100, otherwise. $temperature$ is set to be 1.

For the closed-source models, we use the following specific model type: davinci-002 (GPT-3), gpt-3.5-turbo-0125 (GPT-3.5), and gpt-4-0125-preview (GPT-4). We set the following parameters when calling the model, max_new_token is set to 500 (for MBPP and HumanEval) and 100, otherwise. $temperature$ is set to be 1.

COLLECTED DATASETS

- **Open-source model evaluation**: For RealToxicityPrompt, we use a version provided by Wang et al. [74], which contain 1,200 toxic task prompts and 1,200 non-toxic task prompts.
- **Closed-source model evaluation**: For closed-source model evaluation, due to the budget limit, for each dataset, we randomly choose 100 instances as the evaluation object.

SAFETY

The assessment of the safety is conducted as follows.

- **Truthfulness**: Truthfulness of LLM typically refers to the accuracy and fidelity of generated information [204]. The ground-truth of the instance is judged by a fine-tuned GPT-3 model (GPT-judge) [75], which is the gold standard in the field, to categorize each instance as truthful or untruthful.
- **Toxicity**: Toxicity [16] refers to the presence of harmful, offensive, or inappropriate language in textual data or generated outputs. We use Perspective API [104], an automated online model, accessed by Google APIs, for detecting hate speech and toxic words. It returns the probability that the instance is toxic and we use 0.5 as the classification threshold.
- **BLEU**: The BLEU (Bilingual Evaluation Understudy) [205] is a metric used to evaluate the quality of machine-translated text by comparing it to one or more human reference translations.
- **Pass@1**: Pass@1 [116] is a common metric to evaluate the quality of the code generation. It evaluates the quality of

the top suggestion or the first completion proposed by the model.

More Examples of Pilot Study

Example 2: In MBPP with code generation task, an instance is with the prompt:

```
-----------------------------------------------
Write a python function to check whether the
given array is monotonic or not.
assert is_Monotonic([6, 5, 4, 4]) == True.
-----------------------------------------------
```

The prompt is given as the input to CodeLLaMA and the first 25% of the output is shown in Listing 1. When the length of the input array is 2 , the function simply returns `True`, which is wrong.

```python
def is_monotonic(arr):
  if len(arr) == 1:
    return True
  if len(arr) == 2:
    return True
  if len(arr) == 3:
    ...
```

Listing 1: Example of code generation.