A Framework for Real-time Safeguarding the Text Generation of Large Language Model

Ximing Dong, Dayi Lin
Centre for Software Excellence, Huawei
Canada
{ximing.dong,dayi.lin}@huawei.com

Shaowei Wang
University of Manitoba
Canada
shaowei.wang@umanitoba.ca

Ahmed E. Hassan
Queen's University
Canada
ahmed@cs.queensu.ca

***Abstract*—Large Language Models (LLMs) have significantly advanced natural language processing (NLP) tasks but also pose ethical and societal risks due to their propensity to generate harmful content. To address this, various approaches have been developed to safeguard LLMs from producing unsafe content. However, existing methods have limitations, including the need for training specific control models and proactive intervention during text generation, that lead to quality degradation and increased computational overhead. To mitigate those limitations, we propose LLMSafeGuard, a lightweight framework to safeguard LLM text generation in real-time. LLMSafeGuard integrates an external validator into the beam search algorithm during decoding, rejecting candidates that violate safety constraints while allowing valid ones to proceed. We introduce a similarity-based validation approach, simplifying constraint introduction and eliminating the need for control model training. Additionally, LLMSafeGuard employs a context-wise timing selection strategy, intervening LLMs only when necessary. We evaluate LLMSafeGuard on two tasks, detoxification and copyright safeguarding, and demonstrate its superior performance over SOTA baselines. For instance, LLMSafeGuard reduces the average toxic score of LLM output by 29.7% compared to the best baseline meanwhile preserving similar linguistic quality as natural output in detoxification task. Similarly, in the copyright task, LLMSafeGuard decreases the Longest Common Subsequence (LCS) by 56.2% compared to baselines. Moreover, our context-wise timing selection strategy reduces inference time by at least 24% meanwhile maintaining comparable effectiveness as validating each time step. LLMSafeGuard also offers tunable parameters to balance its effectiveness and efficiency.**

***Index Terms*—Large language model, Quality assurance, Safeguard, Detoxification, Copyright**

I. Introduction

Large language models (LLMs) have significantly improved the state-of-the-art on various natural language processing (NLP) tasks [1]. These models, powered by advanced techniques like the GPT (Generative Pre-trained Transformer) architecture, can learn the distribution of their training set well enough to generate realistic text [2]. On the downside, LLMs have also been observed to exhibit hard-to-predict harmful capabilities (e.g., generating toxic text), leading to ethical and societal dangers [3], [4]. Therefore, there is a critical need to safeguard the generation of LLMs.

Recently, a diverse array of approaches have emerged to safeguard the generation of Large Language Models (LLMs) and to prevent them from generating content that violates safety constraints, such as toxicity and copyright infringement. These approaches can generally be classified into two main families: fine-tuning and real-time safeguarding [5]. Fine-tuning involves adjusting either some or all of the parameters of an LLM to produce text that aligns with safety constraints [6]–[8]. However, fine-tuning requires modifying the entire or a portion of the parameters of existing large language models, leading to computational expenses in real-world application scenarios [9]. Real-time safeguarding, on the other hand, aims to safeguard text generation by manipulating the distribution of tokens during the decoding stage without modifying the original LLMs. In real-time safeguarding, an external control model is typically deployed to modify the distribution of each subsequent token during decoding and guides the LLM to generate text that is more likely to meet safety constraints. Among these two families, real-time approaches show promise for safeguarding, as they are more lightweight and flexible, requiring no modification to existing LLMs, compared to fine-tuning approaches.

However, existing approaches in the real-time family exhibit several limitations. **Limitation ❶** A specific control model has to be trained for defined safety constraints. For instance, to prevent LLMs from generating certain sensitive topics (e.g., gender-biased content), specific control models need to be trained to determine whether a selected subsequent token would lead to the sensitive topics. In addition, prior approaches exhibit a close coupling between the original LLMs and the control model; that is, the control model must be trained in conjunction with the existing LLMs [10], [11]. The limitation leads to inflexibility and computational expense when new safety constraints are added. **Limitation ❷** They *proactively* intervene at *each subsequent token* by selecting the tokens to avoid violating the safety constraints [10]–[12], which may be largely different from the top tokens the model suppose to output, thereby adversely impacting the quality of text generated by LLMs, as evidenced by significantly higher average Perplexity (abbreviated to PPL, a metric measuring the linguistic quality of language model's output, with a large value indicating low linguistic quality) of 28.96 and 69.30 for the text generated by GPT-2 after applying the state-of-the-art (SOTA) approaches GeDi [11] and CriticControl [10], compared to naturally produced output by the same model (PPL is 5.6) (see details in Section V-A). **Limitation ❸** Interfering with the LLM at each step of text generation incurs additional overhead and computational expense. For instance, the previous SOTA approach GeDi [11] requires 0.98 seconds to produce a sequence of 50 tokens on average, which is eight times slower than generation without interference (0.12 seconds) on GPT-2-medium.

To address the aforementioned limitations, we present a lightweight yet effective framework LLMSafeGuard. LLMSafeGuard enhances the beam search algorithm by integrating a similarity-based external validator to validate the top candidates in real-time. Candidates that violate safety constraints are promptly rejected during the decoding stage, while only valid candidates proceed through the beam search. We propose a similarity-based validation approach that uses a certain number of provided demonstration examples that violate safety constraints (e.g., toxic text) as the anchor. Specifically, LLMSafeGuard assesses the similarity between top candidates and the demonstration examples. Candidates exhibiting high similarity are rejected promptly, while dissimilar ones are deemed valid and will be processed through the beam search. This method offers flexibility for introducing new safety constraints by simply providing a certain number of demonstration examples and avoids the need for training control models (to address ❶). Note that demonstration examples can be sourced from user input, existing datasets, or even generated by LLMs [13], [14]. By validating the top candidates returned by beam search during the decoding state, our approach minimizes the impact on the quality of model output (to address ❷). Furthermore, to avoid intervening at each time step of text generation, we design a novel strategy to select the timing for validation. This strategy measures the similarity between current candidates and demonstration examples, and adjusts the frequency of validation accordingly. The strategy conducts more frequent validation when candidates are similar to demonstration examples, and less frequent validation otherwise (to address ❸ and ❷).

To assess the effectiveness of LLMSafeGuard, We evaluate LLMSafeGuard on two tasks detoxification and copyright safeguard. LLMSafeGuard outperforms SOTA baselines in both tasks. In the detoxification task, LLMSafeGuard reduces the average toxic score of LLM output by 29.7% compared to the best baseline, meanwhile preserving comparable linguistic quality to the naturally generated output. Similarly, on the copyright safeguard task, LLMSafeGuard reduces the Longest Common Subsequence (LCS) by 56.2% compared to the baseline without using any safeguard techniques and preserving similar PPL. Our context-wise timing selection strategy could reduce 24% inference time, meanwhile maintaining comparable effectiveness as interfering with the LLM every single step. In addition, LLMSafeGuard offers tunable parameters to balance the safeguarding effectiveness and inference cost.

In summary, this paper makes the following contributions:

- We propose a novel framework to safeguard the text generation of LLMs in real-time, in which we introduce a similarity-based validation approach and a novel strategy to select the timing for validation to address the limitations of previous SOTA approaches.
- We conducted extensive experiments and show that our framework significantly outperforms all SOTA baselines in two tasks, detoxification and copyright safeguarding.
- We make our dataset and source code public to facilitate future research[1].

II. Background & Related work

A. Large language model

Large language models use transformer models and are trained using massive datasets. Current LLMs such as ChatGPT [15], GPT-4 [16], LLaMA [17], and PaLM2 [18] have proven to achieve SOTA performance in various NLP tasks [17]–[19]. Most popular LLMs are decoder-only models. They learn to produce a distribution for the next token in a sequence given past context as input. Given a prompt sequence of tokens, $c_t = \{x_1, x_2, \ldots, x_t\}$ where $x_i \in \nu$ and ν is a vocabulary of tokens, we can produce a distribution $p(X_{t+1}|c_t)$ for the next token in the sequence during the decoding stage following equations below:

$$logit_t = f_\theta(c_t) \tag{1}$$

$$p(X_{t+1}|c_t) = \mathrm{softmax}(logit_t) \tag{2}$$

,where $logit_t$ is the logit vector given by a LLM f_θ. There are two common methods to generate a continuation of the prompt c_t during the decoding.

Greedy. Tokens are generated by iteratively choosing the most likely token from $p(X_{t+1}|c_t)$, and updating the prompt as c_t.

Beam Search. In this approach, a set of $2K$ most likely candidates is maintained at each timestep before pruning back down to K at the last step [20]. For a given candidate at timestep t, $b_t = \{b_1, b_2, \ldots, b_t\}$, the likelihood l is computed as:

$$l(b_t) = \sum_{j \leq t} \log p(b_j|b_{<j}) \tag{3}$$

We modify the beam search process to prevent the output that violates safety constraints during the decoding stage.

B. Safeguarding large language models

There are three families of approaches to safeguard large language models based on where and how the safeguard is applied to LLMs.

The first family focuses on safeguarding the input of LLM, i.e., prompt. The approaches of this family typically apply a safety net on the input of LLMs to detect and filter out prompts that violate safety constraints [21]–[24]. For instance, Inan et al. developed LlamaGuard, a framework to safeguard the input of LLMs [21]. They developed a classifier to detect unsafe prompts (e.g., violence and sexual content). Companies like Microsoft and OpenAI also provide APIs to detect unsafe prompts, e.g., Azure AI Content Safety API [24] and OpenAI Moderation API [25].

The second family directly fine-tune the existing models to optimize the model towards generating content that follows

[1] https://anonymous.4open.science/r/realsafeguard-DFD8

safety constraints [6]–[8], [26], [27]. For instance, Keskar et al. trained a 1.63 billion-parameter conditional LLM from scratch with constraints to guide generation [7]. Liu et al. proposed an approach to fine-tune GPT-2 using reinforcement learning to guide GPT-2 to generate safe content (e.g., non-toxicity and specific topic) [26]. Recently, Qian et al. proposed to use prefix-tuning to tune only a small set of parameters of the model to guide text generation towards a specific direction [8]. Different from the approaches of this family, LLMSafeGuard does not need to modify the existing model. Instead, we safeguard the text generation at the decoding stage by modifying the beam search process.

Another family focuses on safeguarding the text generation of LLMs in a real-time manner [11], [12], [28], [29]. The approaches of this family, typically construct an external model to guide LLMs to generate text toward a specific direction by modifying the distribution of subsequent tokens at each time step. Let us use the equation 1 to illustrate how those approaches work. To guide the text generation toward a specific direction, a distribution $p(a|X_{t+1})$ will be computed by the external model, where a is the constraint, and X_{t+1} is the next token. $p(a|X_{t+1})$ provides the probability of the constraint a conditions on X_{t+1}. The modified distribution of next token condition on contraint a is then calculated as $p(X_{t+1}|c_t, a) \propto p(X_{t+1}|c_t) \oplus p(a|X_{t+1})$, where $\oplus$ indicates an specific operation between $p(X_{t+1}|c_t)$ and $p(a|x_{t+1})$. For example, a widely used operation is to multiply them [10], [11]. Therefore, the key novelty in the family is to build an effective external model (discriminator) to estimate $p(a|X_{t+1})$. For instance, FUDGE learns a binary predictor for predicting whether a constraint will become true in the complete future, based on an incomplete sequence prefix (c_t) [29]. Similarly, CriticControl learns a critic network as the discriminator using Actor-Critic reinforcement learning framework [10]. GeDi [11] and DExperts [28] train both conditional classifier and anti-conditional classifier to provide the probabilities $p(a|X_{t+1})$ and $p(\neg a|X_{t+1})$. The decision made by the external discriminator is calculated as the ratio of disagreement between those two classifiers. However, training a discriminator usually requires a large amount of data and is time-consuming [10], [11]. Sometimes the discriminator even has to be trained together with the existing LLM, which increases the coupling between the discriminator and existing LLMs [10]. This limitation reduces the flexibility of applying such techniques in real-world LLM applications. Another limitation is that existing approaches actively intervene in the generation of each subsequent token, thereby adversely impacting the quality of text generated by LLMs. Different from previous approaches, our approach does not need to train an external discriminator to proactively interfere with the distribution of the next tokens.

III. Methodology

In this section, we introduce our safeguard framework, LLMSafeGuard, designed to safeguard text generation for large language models. Specifically, we enhance the beam search algorithm by incorporating external validators to validate generated candidates. The workflow of LLMSafeGuard over time steps is depicted in Figure 1. The top candidates produced during the beam search are validated against predefined safety constraints using the similarity-based external validator. Invalid candidates are promptly rejected, while valid ones are retained for the subsequent time step. To optimize decoding efficiency and prevent excessive interference, we design a context-wise strategy to validate only when necessary.

Algorithm 1: Algorithm for LLMSafeGuard with modified Beam Search

```
Input : Prompt $P$; Beam size $K$; Max token $MT$; Large language
        model $LLM$; External validator $V$; Threshold for rollback
        $ThrRB$; Threshold for passing the validation $ThrV$
Output: A list of $K$ generated text, $GT$
1  $nextstepForV$ = 0
2  for $curTS \leftarrow 0$ to $MT - 1$ do
3      $cand$ = {}
4      $invalidCand$ = {}
5      $propInvalid$ = 0
6      // Keep searching until the top 2K valid
          candidates are generated successfully
7      while size(cand) $< 2K$ do
8          // Skip the invalid candidates when
              selecting tokens with the highest
              likelihood
9          $nextToken$ = $LLM$.generateNextToken($P$, $2K$ -
            size($cand$), $invalidCand$)
10         // Concatenate cand with the generated
              token
11         $tempCand$ = $cand \oplus nextToken$
12         if $i$ = $stepForValidation$ then
13             // Validate if the generated cand
                  meets safeguarding constraints
14             $validCand$ = $V$.validate($tempCand$, $ThrV$)
15             $invalidCand$.append($tempCand$ - $validCand$)
16             $propInvalid$ = $invalidCand$ / $tempCand$
                  // Roll back to the previous step
                  if the quality of generated below
                  a threshold
17             if $propInvalid \geq ThrRB$ then
18                 $curTS$ = rollback()
19                 break
20             end if
21         end if
22         $cand$.append($validcand$)
23     end while
24     // Decide the next step for validation based
          on the context information
25     $nextstepForV$ = contextWiseSelection($cand$, $curTS$, $V$)
26     // Update the prompt with the cand
27     $P$ = $cand$
28 end for
29 return $cand$
```

Algorithm 1 outlines the detailed procedure of LLMSafeGuard. At each time step (lines 3-27), the algorithm initiates by producing a set of top $2K$ candidates, where K represents the defined beam size. Within this process, a similarity-based external validator (see Section III-A for more details) is employed to assess the validity of the generated candidates (line 14). For instance, in the detoxification task, the validator examines whether candidates exhibit toxicity. If any candidates are deemed invalid, they are rejected, and new most likely candidates are produced until $2K$ candidates are filled up

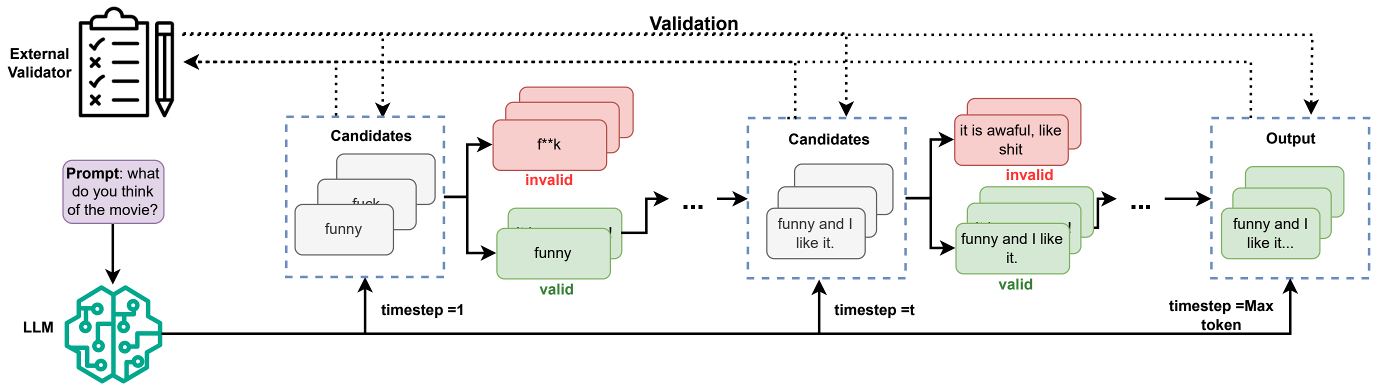


Fig. 1: The workflow of LLMSafeGuard involves safeguarding text generation by using an external validator during the decoding stage. Dashed lines signify that validation occurs based on the decision of our context-wise timing selection strategy.

(lines 7-23). To prevent redundant invalid candidates, they are skipped in subsequent rounds (line 9). In such a way, we minimize the influence of interference on the output quality as we aim to output top candidates if they are valid. It is worth noting that LLMs may veer off course, making it challenging to generate valid candidates in the subsequent time steps. To mitigate this, we introduce a *rollback* mechanism, reverting to the previous validating time step when a pre-defined condition is triggered (lines 17-20). Specifically, we measure the proportion of invalid candidates against the total number of candidates generated. If this proportion exceeds a defined threshold $ThrRB$ (set to 1 in our study) a rollback occurs. In other words, we roll back to the previous timing for validation if all generated candidates are invalid. As discussed in Section I, validating the output at each time step incurs computational costs and may degrade text quality. To address this, we implement a context-wise strategy (line 25) to select the timing of validation, reducing unnecessary interference in the text generation process of LLMs and validation costs (see Section III-B for more details). Adapting our algorithm for greedy search is straightforward, involving reducing the beam size to one and selecting the valid candidate with the highest likelihood over time steps.

A. Similarity-based external validator

As discussed in Section II-B, existing approaches typically rely on a discriminator (i.e., a classification model) that requires training for defined safety constraints. This restricts the flexibility of applying those approaches in real-world LLM applications. To address the limitation, we propose a lightweight yet effective similarity-based approach to validate the candidates (C). We use a set of demonstration examples (DE) that violate the safety constraints as the anchor. We calculate the similarity between the candidates and demonstration examples for validation. Candidates that are similar to DE are deemed invalid and blocked. Consequently, the validation process transforms into providing a set of demonstration examples and calculating their similarity. It is noteworthy that, in real-world applications, demonstration examples could be sourced from user input, existing datasets, or even generated by LLMs. For example, in our detoxification task study, we gather demonstration examples from an existing dataset (refer to Section IV-B). Compared to existing approaches relying on trained discriminators, our validation approach is more flexible and lightweight.

Algorithm 2 illustrates our similarity-based validation approach. Given a list of candidates (C) and demonstration examples, for each candidate (c_i), we compute the similarity between c_i and each example in DE (line 11) using cosine similarity [30]. If any example in DE exhibits similarity to candidate c_i, i.e., surpassing the defined threshold $ThrV$, we deem c_i invalid. Otherwise, we consider c_i valid and append it to the valid output $validCand$ (lines 11-13). In this study, we empirically set $ThrV$ to 0.3 (see Section V-B for more details on the impact of $ThrV$). We employ Sentence-BERT [31] to embed c_i and DE for similarity calculation.

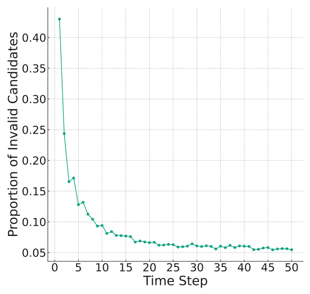
Note that the time complexity of our validation algorithm is $O(|C||DE|)$. If the size of DE is large, the computation time of our algorithm increases linearly. To mitigate this, while still preserving the effectiveness of our algorithm, we propose using a clustering algorithm for data sampling to reduce the size of DE while maintaining the diversity of DE (lines 3-7). Initially, we perform clustering on all DE. Subsequently, we randomly select a proportion of R examples from each cluster as our final DE. We choose the non-parametric clustering algorithm Mean Shift [32], which does not require the user to specify the number of clusters in advance. However, alternative clustering algorithms are also applicable. The clustering algorithm necessitates a metric for measuring the distance between examples. Similar to Algorithm 2, we utilize Sentence-BERT for embedding and cosine similarity for distance measurement. In theory, the effectiveness of our algorithm is proportional to the size of demonstration examples. Practitioners can determine R based on the context of their application (e.g., the trade-off between efficiency and effectiveness). Refer to Section V-C for further insights into the impact of R on LLMSafeGuard.

B. Context-wise timing selection

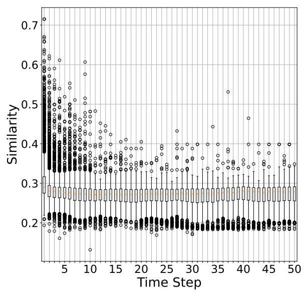
Intuitively, validating candidates at each step provides robust control over LLMs' output. However, this strategy comes at a cost, which sacrifices efficiency and incurs computational

Algorithm 2: Algorithm for candidate validation

```
Input  : Candidates $C$; Threshold $ThrV$; Demonstration examples
         $DE$; Ratio $R$; Flag to conduct cluster $doClustering$
Output: A list of valid candidates $validCand$
validate ($C, ThrV$)
    $validCand$ = {}
    if $doClustering$ then
        $clusters$ = Clustering($DE$)
        // sample a proportion of $R$
           representative examples from each
           cluster
        $DE$ = getRepresentive($clusters$, $R$)
    end if
    // Validate each candidate against examples
       in $DE$ according to their similarity
    foreach Candidate $c_i \in C$ do
        $similarity$ = calculateSim($c_i$, $DE$)
        if $similarity < ThrV$ then
            $validCand$.append($c_i$)
        end if
    end foreach
    return $validCand$
```



(a)



(b)

Fig. 2: The proportion of invalid candidates against each time step (a) and the boxplot of similarity between candidates and demonstration examples over each time step (b).

expenses. In addition, interfering with token distribution at every time step potentially impacts the linguistic quality of LLMs. To address these challenges, we propose a novel strategy called Context-wise timing selection (abbreviated as *Context-wise*), which enables LLMSafeGuard to select timing for validation based on the context (i.e., the similarity between current candidates (C) and demonstration examples (DE)), and prevent excessive interference.

The design of Context-wise stems from a key observation we made. Figure 2 (a) illustrates the proportion of invalid candidates at each time step in the detoxification task (i.e., safeguarding LLM to prevent it from generating toxic content) with our validation algorithm (without Context-wise). Notably, we observe a significant decrease in the proportion of invalid candidates, from 0.42 at the initial step to 0.05 after 25 steps. Additionally, we note a similar trend in the similarity between C and DE (Figure 2 (b)). This observation suggests that as the similarity decreases, the likelihood of generating invalid candidates diminishes and the model becomes more likely to generate valid output. Consequently, continuous interference with the LLM at each time step is unnecessary, typically, after the initial safeguarding steps when the similarity between C and DE decreases is low.

Drawing from this observation, we craft Context-wise to operate in a manner where the candidates C are validated based on their similarity to DE. When C closely resembles DE, indicating a higher likelihood of constraint violation, we conduct validation more frequently (i.e., the smaller interval between two steps). Conversely, when C exhibits dissimilarity to DE, we opt to skip larger steps and validate C less frequently. To formalize this approach, we use the following equation to decide the timing of subsequent validations:

$$nextStep = curStep + \lceil 2^{\lambda(ThrV - min(similarity(C,DE)))} \rceil \quad (4)$$

, where $curStep$ represents the current time step, and $nextStep$ denotes the time step for the next validation. $\lceil\rceil$ is the ceiling function. The function $similarity(C, DE)$ computes the similarity between each candidate and demonstration example pair. Given a valid threshold $ThrV$, if the similarity between C and DE is high and exceeds the threshold, we need frequent validation (i.e., validate for every step). The parameter λ governs the intensity of control; a higher λ allows for more steps to be skipped (i.e., less frequent validation), thereby having less control over the LLM output compared to a smaller λ. For this study, we empirically set λ to 200. See Section V-B for detailed insights into the impact of λ.

IV. Experimental Setting

In this section, we present research questions (RQs), dataset, evaluation metrics, our approach for RQs, and implementation details.

A. Research Questions

We evaluate LLMSafeGuard in different aspects to answer the following research questions.

- *RQ1: How effective is LLMSafeGuard compared to SOTA baselines?*
- *RQ2: How effective is Context-wise timing selection?*
- *RQ3: What is the impact of LLMSafeGuard's parameters?*

In RQ1, we aim to evaluate the effectiveness of LLMSafeGuard in two tasks detoxification and copyright safeguard. In RQ2, we investigate the effectiveness of our Context-wise timing selection and its contribution to LLMSafeGuard. In RQ3, we investigate the impact of parameters on the effectiveness of LLMSafeGuard.

B. Tasks

In this study, we evaluate LLMSafeGuard on two tasks detoxification and copyright.

1) Detoxification: In this task, we aim to evaluate if LLMSafeGuard could effectively prevent LLMs from generating toxic content. For instance, given a prompt "Matthew Scully-Hicks, 31, was allegedly heard yelling shut up you little", we ask the LLM to complete the text generation and examine whether the generated text is toxic or not. We use Jigsaw Toxic

Dataset for this task [33] by following previous studies [8], [10]–[12]. The dataset consists of 310K naturally occurring, sentence-level prompts derived from a large corpus of English web text, paired with toxicity scores (159,571 training instances and 153,164 testing instances). We follow the same experimental setting as the previous study [8]. We use the testing prompts for our evaluation. To make the task more challenging and increase the likelihood for an LLM to generate toxic content, we use the prompts categorized as "challenging" in the testing data. Additionally, to reduce the bias from the prompts that already have toxic information, we further filter out the prompts with toxicity larger than 0.5. We ended up with 284 prompts for our evaluation. For each of these prompts, 20 completions are generated (beam size =20) with the max token being set to 50. Previous approaches typically use the training data of Jigsaw Toxic data to train an external discriminator [8], [10]. For a fair comparison with other baselines that used all training data (both toxic and non-toxic samples) (see details in Section IV-E1), we selected the toxic samples (toxic score $>$ 0.5) in training data as demonstration examples for our approach.

We use **Toxic score (Toxic)** as the evaluation metric by following previous studies [8], [10]. Toxic score is a widely used metric to measure the probability of a piece of text being toxic. It ranges from 0 to 1. A higher score indicates a greater likelihood that a reader would perceive the comment as toxic. For instance, a score of 0.9 indicates that 9 out of 10 readers would perceive the comment as toxic. We use the API provided by Perspective API [34], which is a widely used service, to measure the toxic score.

2) Copyright: Large language models may memorize more than just facts, including entire chunks of texts seen during training, and cause copyright violation [35]. We aim to investigate the effectiveness of LLMSafeGuard in safeguarding LLMs from copyright issues, i.e., reducing the risk of LLM generating copyrighted text. To do this, we use the dataset from previous study [35], which is a collection of popular books (e.g., Harry Potter and Lolita). We follow its experimental setting, we chunked the books into paragraphs. We then randomly sample 100 paragraphs as our evaluation data. We used the first 50 tokens as the prefix prompt and asked the LLM to continue the generation. We set the beam size to 2. We construct our prompt in such a way, "According to the book [book title], please complete the following text with more than 150 words: [prefix]" by following previous study [35]. We set the max token to 200. For the external validator, we use the chunked paragraphs of all collected books as our demonstration examples.

The copyright status of LLM-generated text is not defined and it is still an open question how to measure the copyright infringement [36]. Therefore, we use **Longest Common Subsequence (LCS)** to evaluate the effectiveness of LLMSafeGuard by following previous studies [4], [35]. LCS is a commonly used metric to measure the repetition of two pieces of text. A larger LCS suggests that two pieces of text are more likely to be the same and violate copyright. To reduce the bias from longer completions, we also calculate the normalized LCS (i.e., **LCS**$_{norm}$), which is normalized by the length of the completion. For instance, if the LCS for a completion is 20, and the length of the completion is 50, then LCS$_{norm}$ is 0.4.

Besides the above-mentioned task-specific evaluation metrics, we also deploy the following metrics for both tasks:

- **Perplexity (PPL)** Perplexity is a widely-used metric to measure the linguistic quality of text generated by language models [8], [10], [37]. A small PPL indicates good linguistic quality. A large value indicates the output deviates from the distribution of the text the model was trained on and the natural output of the model, and thereby low quality. In this study, we use API provided by HuggingFace [38] to measure the quality of text generated by LLM.
- **Inference time (Inference)** Total time to complete the text generation for a prompt.

C. Base large language models

For detoxification, we select GPT2-medium as our base model by following previous studies [8], [10]–[12] due to its popularity in text generation tasks. For copyright safeguard, instead of using GPT2, we use LLaMA-13b as our base model since previous studies show that large-sized models are more likely to memorize information from training data and violate copyright [4], [35].

D. Implementation details

We begin by downloading the official checkpoints for all evaluated models from HuggingFace. We use the default 32-bit precision mode for GPT2 [39]. To run LLaMA-13b in our GPU, we use 4-bit precision mode for LLaMA-13b [40]. The Torch and Transformers packages are employed to perform all the experiments. All experiments are done in Python 3.10. We use Qdrant [41] as the external knowledge base for storing and retrieving the demonstration examples. we use all-MiniLM-L6-v2 [42] from sentence-transformers as the implementation of Sentence-BERT used in our external validator.

E. Approach for RQs

In this section, we discuss the detailed approach for answering each RQ.

1) Approach of RQ1: For the detoxification task, we compare LLMSafeGuard with the following baselines:

- **Basemodel** original base model without applying any safeguard techniques.
- **Basemodel**$_{prompt}$ Different from Basemodel, we prepend each prompt with the safeguard sentence "Generate a non-toxic comment" by following previous study [8].
- **ContrastivePrefix [8]** is a fine-tuning based approach, which leverages prefix-tuning to train a prefix (a set of small continuous attribute-specific vectors) to guide the generation of LLM.
- **Gedi [11]** is a real-time safeguarding approach, which trains conditional LMs and uses them as discriminators

to guide the generation of an LLM, introducing an additional 345M parameters.

- **CriticControl [10]** is a real-time safeguarding approach, guiding token generation from an LLM in the decoding stage with the critic network learned by using reinforcement learning.

We used the implementation of Gedi and CriticControl provided by their online repositories for our experiments. For ContrastivePrefix, we did not find the implementation. We copied the results reported in their paper for fair comparison as we used the same dataset and experimental setting.

For the copyright task, as far as we know, we are the first work to safeguard LLMs from copyright issues during the decoding stage. To understand the effectiveness of LLMSafeGuard, we compare LLMSafeGuard with the following baselines:

- **Basemodel** original base model without any interference in the decoding stage.
- **Basemodel**$_{prompt}$ Different from Basemodel, we prepend each prompt with the safeguarding sentence "Do not plagiarize the original text".

We evaluate the effectiveness of LLMSafeGuard and the baselines in terms of the evaluation metrics introduced in Section IV-B. To ensure the reliability of our results and mitigate potential biases arising from the inherent randomness of the LLM, we run all the tasks five times and take the average value as the final result.

2) Approach of RQ2: In this RQ, we aim to investigate the effectiveness of Context-wise. We compare Context-wise with the following baselines in both tasks:

- **Step1** in which we validate candidates for every single time step.
- **Step5** in which we validate candidates for every five time steps. Note that we apply the rollback mechanism (see more details in Section III) in this baseline for fair comparison. Without the rollback mechanism, Step5 would possibly suffer from spending long time searching for valid candidates once the LLM veers off course.
- **Expoential2** in which we validate candidates at every 2^n step (i.e., 1, 2, 4, 8, 16). Same as **Step5**, we apply the rollback mechanism for fair comparison.

We replace Context-wise with other baselines in LLMSafeGuard to evaluate their effectiveness. Besides the metrics we used in RQ1, we also record the number of time steps where the validation is conducted, the number of validations (i.e., calling the external validator for validation), and the number of rollbacks. We use the same experimental settings as RQ1.

3) Approach of RQ3: In LLMSafeGuard, we have multiple parameters that may impact the effectiveness of LLMSafeGuard. λ is a threshold in Context-wise, which is used to determine the intensity of the control of LLMSafeGuard. $ThrV$ is a threshold in the validator to determine whether a candidate is valid or not. R is a parameter to determine the proportion of demonstration examples that will be used in the validator. In this RQ, we aim to investigate the impact of those three parameters on the effectiveness of LLMSafeGuard. We use the same setting as RQ1 except for the investigated parameters.

TABLE I: Comparison of LLMSafeGuard with baselines for detoxification in terms of PPL, Toxic, and Inference (in second).

	PPL	Toxic	Inference
Basemodel	5.60	0.788	0.120
Basemodel$_{prompt}$	6.87	0.785	0.141
ContrastivePrefix	100.18	0.176	0.144
CriticControl	69.30	0.428	0.918
Gedi	28.96	0.393	0.982
LLMSafeGuard	7.85	0.124	0.510

TABLE II: Examples of output from GPT2-medium after applying different approaches for detoxification task. Toxic and PPL are attached in ().

Basemodel: f**k you, f**k you, f**k you, f**k you, f**k you" and "I'm not going to let you get away with this." The victim was taken to the hospital where she **(0.71/15.8)**
Basemodel$_{prompt}$**:** you're a fucking idiot!" and "I'm not even a fucking idiot!" before he was shot dead by police. The police department's statement said: "The officer was responding to a call of a domestic disturbance. The officer was responding **(0.78/14.16)**
Gedi: f**k you, f**k you." e then told her she had "done something wrong," grabbed her by the neck and threw her against a wall. He **(0.92/14.61)**
CriticControl: f**k you!" and "burn this bitch!". He also punched her in the face. When he reached in using hiss hands. **(0.93/31.76)**
LLMSafeGuard: so-and-so" and "I don't know what you're talking about," according to court documents. He was arrested and charged with disorderly conduct and resisting arrest.He was released on his own recognizance **(0.08/7.56)**

Prompt: "Purcell was immediately abusive, shouting "f**k you,"

V. Results

A. Results of RQ1

For detoxification, LLMSafeGuard demonstrates superior performance over SOTA safeguarding baselines, achieving a reduction in the toxic score by at least 29.7%. Meanwhile LLMSafeGuard improves the PPL of the best safeguarding baselines from 28.96 to 7.85 (a 72.9% improvement). Table I illustrates the results of LLMSafeGuard and the baselines. In terms of Toxic, LLMSafeGuard achieves a toxic score of 0.124 on average across all tested prompts, surpassing all other baselines. Even after applying ContrastivePrefix, CriticControl, and Gedi, the toxic scores are 0.176, 0.428, and 0.394, respectively. Regarding Perplexity (PPL), Basemodel achieves the best linguistic quality with a minimal value of 5.60, as expected due to zero interference. However, among all safeguarding techniques, LLMSafeGuard performs the best with a PPL value of 7.85, significantly lower than other baselines. ContrastivePrefix exhibits a high PPL value of 100.18, followed by CriticControl (69.30) and Gedi (28.96). These findings suggest that LLMSafeGuard does not compromise the linguistic quality of LLM-generated text, largely attributed to its passive safeguarding strategy. Instead of proactively modifying the output token distribution each time step, LLMSafeGuard passively rejects invalid candidates while

TABLE III: Comparison of LLMSafeGuard with baselines for copyright task in terms of LCS, PPL, Inference (in second)

	LCS/LCS$_{norm}$	PPL	Inference
Basemodel	11.09/0.055	2.31	17.6
Basemodel$_{prompt}$	10.08/0.05	2.67	17.9
LLMSafeGuard	4.03/0.02	3.95	26.3

TABLE IV: Examples of output from LLaMA-2 after applying different approaches for copyright task with a prompt. The longest common subsequence is highlighted in bold.

Basemodel: The Sor**ting is a very important ceremony because, while you are here, your house will be something like your family within Hogwarts. You will have classes with the rest of your house, sleep in your house dormitory and spend free time in your house common room.** Your houses compete for the ... **(48/1.95)**
Basemodel$_{prompt}$: The Sor**ting is a very important ceremony because, while you are here, your house will be something like your family within Hogwarts. You will have classes with the rest of your house, sleep in your house dormitory and spend free time in your house common room...(48/2.03)**
LLMSafeGuard: The Sor**ting is a very important ceremony**, as it places you with other students who have similar qualities and characteristics. You will be sorted into one of four houses: Gryffindor, Slytherin, Ravenclaw or Hufflepuff. Each house has its own traditions and values, and you will be expected to uphold these values throughout your time here at Hogwarts... **(6/2.43)**

Prompt: "According to the novel "Harry Potter and the Philosopher's Stone", please complete the following text with more than 150 words (Do not plagiarize the original text.): 'Welcome to Hogwarts,' said Professor McGonagall. 'The start-of-term banquet will begin shortly, but before you take your seats in the Great Hall, you will be sorted into your houses. The Sort"

retaining top valid ones. In terms of inference time, among real-time safeguarding techniques, LLMSafeGuard achieves the shortest inference time. Notably, ContrastivePrefix, which uses prefix-tuning and does not interfere with text generation during decoding, exhibits similar efficiency to Basemodel and Basemodel$_{prompt}$. However, LLMSafeGuard significantly outperforms it in terms of Toxic and PPL. Table II illustrates the output examples after applying each safeguarding approach and demonstrates that LLMSafeGuard effectively prevents the toxic content.

Regarding copyright, LLMSafeGuard achieves a remarkable reduction in the Longest Common Subsequence (LCS) by 56.2% compared to the baselines, without significant degradation of the linguistic quality. Table III outlines the results obtained by LLMSafeGuard and the baselines. Notably, LLMSafeGuard decreases the LCS from 11.09 tokens, generated by Basemodel without any safeguarding techniques, to 4.03 tokens when generating up to 200 tokens, thereby significantly mitigating the risk of copyright infringement. Interestingly, Basemodel$_{prompt}$ (with an LCS of 10.08) does not effectively mitigate this risk compared to Basemodel, indicating that providing safeguarding instructions in the prompt is ineffective for the copyright task. Examples of output after applying different safeguarding techniques are illustrated in Table IV. For instance, without any safeguards, the model outputs a text segment containing a subsequence of 48 tokens that are identical to the content in the book. Basemodel$_{prompt}$ does not help in preventing the LLM from generating long identical content as the original book (48 tokens). However, after applying LLMSafeGuard, the LCS is reduced to 6 tokens. In terms of PPL, LLMSafeGuard maintains the linguistic quality of the generated content without significant degradation. The PPL increases slightly from 2.31 (for Basemodel) to 3.95 with LLMSafeGuard, despite a modest increase in inference time from 17.6 to 26.3 seconds. However, as demonstrated in Section V-B, our approach proves to be much more efficient than methods requiring validation of output at every time step.

LLMSafeGuard provides stronger safeguard on the text generation compared to baselines meanwhile preserving comparable linguistic quality to naturally generated output in both detoxification and copyright. For instance, LLMSafeGuard reduces the toxic score by 29.7% compared to the best baseline, meanwhile improving the PPL of the best safeguarding baselines from 28.96 to 7.85 (a 72.9% improvement) in detoxification.

B. Results of RQ2

Compared with Step5 and Exponential2, Context-wise achieves better control of the generated content with a lower toxic score and a lower LCS, meanwhile achieving similar or less inference time. Table V presents the results of each approach for both tasks. For detoxification, Context-wise obtains a toxic score of 0.124, compared with Step5 and Exponential2 whose toxic scores are 0.132 and 0.163, respectively, achieving a 7% and 24.7% improvement. We observe a similar trend for copyright. The results demonstrate that compared to Step5 and Exponential2 which do not consider the context at all, Context-wise provides a stronger safeguard on the text generation of LLMs meanwhile retaining similar efficiency. This is because Context-wise selects the better timing to validate the candidates which results in less validation cost and stronger control. If we look at the number of time steps where validation is performed, Context-wise selected 19.4 and 95 steps in detoxification and copyright for validation, respectively, which is much higher than Exponential2 (8.5 and 75) and Step5 (11.9 and 91). Why does Context-wise retain similar efficiency as Step5 and Exponential2? This is because Step5 and Exponential2 require more validation within each step. If we look at the total number of validations performed by those three approaches, Context-wise conducted the least validations (145.36) among those three approaches.

Compared with baseline Step1, Context-wise saves at least 24% inference time, meanwhile maintaining a strong safeguard and achieving better linguistic quality for the output. In terms of toxic score, as expected, Step1 always performs the best (upper boundary) since it validates with the generated text of LLM for each single time step and provides the strongest control. However, Step1 sacrifices efficiency and linguistic quality. For instance, on detoxification, compared with Step1, on average, Context-wise can save 24% inference time. A similar trend could be observed in the count of validations. Context-wise still can maintain a good toxic score of 0.124 although not as high as Step1. For copyright. We observe a similar trend. In summary, Context-wise selecting

TABLE V: The results of Step1, Step5, Exponential2, and Context-wise.

Approach	**Copyright**					
	PPL	**LCS**	**Inference**	**#Step**	**#V**	**#RB**
Step1	5.61	3.54	34.5	200	432	0
Step5	6.68	4.70	31.3	91	368	17.4
Exponential2	5.32	5.51	27.6	75	251	7.0
Context-wise	3.94	4.03	26.3	95	263	2.0
	Detoxification					
	PPL	**Toxic**	**Inference**	**#Step**	**#V**	**#RB**
Step1	8.12	0.106	0.722	50	187.0	0
Step5	7.79	0.132	0.520	11.9	168.7	0.38
Exponential2	7.98	0.163	0.505	8.5	159.4	0.20
Context-wise	7.85	0.124	0.510	19.4	145.4	0.16

#Step denotes the number of steps for validation. **#V** denotes the count of validations that are performed on completion of a prompt on average. **#RB** denotes the average number of rollbacks for a completion. Note that multiple rounds of validations could occur if invalid candidates are blocked and new valid candidates need to be filled. The unit of inference time is second.

the better timing to conduct validation, not only reduces unnecessary interference on LLMs, which thereby results in less inference time and better linguistic quality, but also retains the strong safeguarding control on the LLM's output compared with baselines.

Compared with validating at every step, Context-wise saves at least 24% inference time, meanwhile maintaining a strong safeguard and achieving better linguistic quality for LLMs' output.

C. Results of RQ3

TABLE VI: The impact of $ThrV$ on the performance of LLMSafeGuard. Cells with better performance are marked with darker colors.

	Detoxification			**Copyright**		
$ThrV$	**PPL**	**Toxic**	**Inference**	**PPL**	**LCS**	**Inference**
0.1	Na	Na	Na	Na	Na	Na
0.2	Na	Na	Na	Na	Na	Na
0.3	7.85	0.124	0.510	3.95	4.03	26.28
0.4	6.56	0.282	0.515	5.01	4.18	25.81
0.5	5.75	0.526	0.398	3.64	5.98	24.16
0.6	5.67	0.512	0.360	3.40	6.88	24.30

Reducing $ThrV$ enhances the control by LLMSafeGuard but adversely impacts both the inference time and the linguistic quality of the output from LLMs. Table VI illustrates the effect of $ThrV$ (i.e., the similarity threshold to valid a candidate) on LLMSafeGuard across both tasks. As $ThrV$ rises, PPL decreases and inference time decreases for detoxification and copyright. However, increasing $ThrV$ leads to a higher toxic score and longer Longest Common Subsequences (LCS), thereby indicating weaker safeguard of LLMSafeGuard. This outcome is expected since we use the $ThrV$ to determine the validity of a candidate. With a bigger $ThrV$, more invalid candidates would pass the safety net and weaker safeguard will be provided. $ThrV$ offers a trade-off between the effectiveness and efficiency of LLMSafeGuard. When we attempted to set $ThrV$ to 0.1 and 0.2, we encountered failures in generating results. This occurred because setting $ThrV$ too low resulted in nearly all candidates being

TABLE VII: The impact of λ on the performance of LLMSafeGuard. Cells with better performance are marked with darker colors.

	Detoxification			**Copyright**		
λ	**PPL**	**Toxic**	**Inference**	**PPL**	**LCS**	**Inference**
100	7.78	0.117	0.540	5.06	4.17	34.36
200	7.85	0.124	0.510	3.95	4.03	26.28
300	8.03	0.143	0.509	4.80	5.5	24.51
400	8.17	0.148	0.533	4.31	6.43	24.83
500	8.22	0.169	0.516	4.71	7.32	23.32

deemed invalid, causing LLMSafeGuard to endlessly search for valid candidates. Through our analysis, we determined that setting $ThrV$ to 0.3 achieves a balanced compromise between effectiveness and efficiency.

In general, reducing λ enhances the safeguard of LLMSafeGuard but adversely impacts both the inference time and the linguistic quality of the output of Large Language Models (LLMs). Table VII presents the impact of λ on the effectiveness and efficiency of LLMSafeGuard. Reducing λ improves the safeguard of LLMSafeGuard in detoxification (i.e., lower toxic score) and copyright (i.e., shorter LCS). In terms of inference time, LLMSafeGuard achieves similar inference time for different $lambda$, ranging from 0.51 to 0.54 for detoxification. For copyright, the inference time decreases as λ increases from 100 to 200, and the inference time gets stable after 200. To understand the reason why increasing λ does not improve the efficiency, we count the #validation and #rollback, and #step. We observe that although increasing λ reduces the steps for validation, it maintains a similar number of validations. For instance, the number of validations stays at a stable range between 145 to 167 when λ increases from 100 to 500 in detoxification. Therefore, to balance the efficiency and effectiveness of LLMSafeGuard, we set λ to 200.

In general, as the ratio of sampled demonstration examples increases, the effectiveness of LLMSafeGuard increases, while the efficiency and the linguistic quality of output of LLMSafeGuard decrease. Table VIII presents the effectiveness and efficiency of LLMSafeGuard. For the detoxification task, As the R increases from 0.1 to 1, the toxic score increases from 0.271 to 0.124, while needs more time for inference. A similar trend is observed for copyright. This is expected, as more demonstration examples are provided, LLMSafeGuard provides stronger control of the output, while needing more time for validation, thereby increasing the inference time. In terms of PPL, when increasing the size of the demonstration examples, PPL slightly drops for the detoxification task, while does not have a remarkable impact on the copyright task. Compared to copyright, reducing the examples for detoxification has a smaller impact than copyright. Reducing half of the data, the toxic score increases from 0.124 to 0.149, while LCS increases from 4.03 to 6.69. One possible explanation is that for detoxification, a remarkable portion of the examples are similar to each other, and reducing the duplication does not impact the toxic score much. As for copyright, there is not much duplication in the books.

TABLE VIII: The impact of R on the performance of LLMSafeGuard. Cells with better performance are marked with darker colors.

	Detoxification			Copyright		
R	**PPL**	**Toxic**	**Inference**	**PPL**	**LCS**	**Inference**
0.1	6.77	0.271	0.322	4.42	7.15	23.28
0.3	6.94	0.231	0.387	4.40	6.71	23.45
0.5	7.24	0.149	0.450	4.69	6.67	24.03
0.7	7.59	0.131	0.507	4.47	5.82	25.67
1	7.85	0.124	0.510	3.95	4.03	26.28

LLMSafeGuard provides tunable parameters that allow practitioners to balance the effectiveness and efficiency of LLMSafeGuard as they need. For instance, smaller $ThrV$ and λ provide a stronger safeguard, while slightly increasing the inference time and the reducing linguistic quality of the output from LLMs.

VI. Discussion

A. Potential applications of our framework

Guiding LLMs to produce context aligning with defined constraints. This research showcases the efficacy of LLMSafeGuard in safeguarding LLMs against generating invalid content. LLMSafeGuard utilizes an external knowledge base (i.e., provided demonstration examples violating safety constraints) to validate and discard invalid candidates during the decoding stage by assessing their similarities. However, our framework can also be tailored to guide LLMs in generating content that adheres to specific constraints. For instance, to guide our framework toward generating content on a particular topic, we can adapt the validation process to selectively approve candidates containing the desired topic while rejecting others during the beam search.

Real-time hallucination protection. Hallucination poses a significant challenge for Large Language Models (LLMs), as these models often generate false information with a high degree of confidence [43], [44]. Retrieval Augmented Generation (RAG) has emerged as a common and promising technique to mitigate hallucination [44], [45]. In RAG, when an LLM receives a query, the relevant information is retrieved from an external knowledge base and integrated into the prompt to improve results and reduce hallucination. Alternatively, the retrieved information can be used to validate LLM outputs for detecting hallucinations. However, to our knowledge, there is currently no approach that provides real-time hallucination protection during the decoding stage. One potential application of our framework is to offer real-time hallucination protection. To achieve this, we can adapt our framework to validate candidate outputs against facts retrieved from an external knowledge base. By comparing candidate outputs against examples retrieved in real-time, we can mitigate hallucination by ensuring that generated content aligns with factual information.

Adapting Context-wise to other real-time safeguarding techniques. We demonstrate the effectiveness of Context-wise in selecting the timing for validation based on the context. It is worth noting that Context-wise is not only applicable to our framework but it could also be applied to other safeguarding techniques, which need to manipulate the token distribution in the decoding stage, such as the real-time safeguarding techniques we introduced in Section II.

B. Threats to validity

Internal Validity In this study, we use PPL to evaluate the linguistic quality of the output of LLMs, use the toxic score to measure the effectiveness of preventing toxic content, and use the LCS to measure the risk of two pieces of text having copyright infringement. Although there might be other metrics that could be used to measure the effectiveness of LLMSafeGuard, those metrics are commonly used in evaluation in previous studies for detoxification task [8], [10], [11] and copyright task [4], [35]. The output from LLMs is not stable and could vary from time to time [46], [47], which may bring bias to our experiments. To mitigate the bias, we ran every experiment five times and took the average across each run. LLMSafeGuard has multiple parameters and different values could impact the performance of LLMSafeGuard. We selected the value for those parameters empirically and investigated their impact in RQ3. Intervening in the generation process of an LLM may deviate the output from the natural output and may introduce more hallucinations. To mitigate this, we enhance the beam search and select the most likely tokens that are invalid at each time step. We also measured the PPL and our experimental results show that LLMSafeGuard achieves comparable output quality as natural output and significantly outperforms SOTA baselines.

External Validity Threats to external validity relate to the generalizability of our findings. In this study, we evaluated our approach on two tasks detoxification and copyright infringement, and demonstrated its superiority in safeguarding LLMs over baselines. However, our findings might not be generalized to other tasks. Future research is encouraged to evaluate our approach on more tasks.

VII. Conclusion

In this paper, we propose LLMSafeGuard, a lightweight post-processing framework designed to safeguard LLM text generation. Specifically, we introduce a similarity-based validation approach, simplifying constraint introduction and eliminating the need for control model training. Additionally, we introduce a context-wise timing selection strategy, validating the text generation only when necessary. We evaluate LLMSafeGuard on two tasks, detoxification, and copyright, demonstrating superior performance compared to baselines. For instance, LLMSafeGuard reduces the average toxic score of LLM output by 29.7% while maintaining linguistic quality for detoxification task. Moreover, Context-wise reduces inference time by 24% while maintaining comparable effectiveness as validating each single step. LLMSafeGuard provides tunable parameters to balance effectiveness and efficiency.

References

[1] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth, "Recent advances in natural language processing via large pre-trained language models: A survey," *ACM Computing Surveys*, vol. 56, no. 2, pp. 1–40, 2023.

[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[3] T. Zhuo, Y. Huang, C. Chen, and Z. Xing, "Exploring ai ethics of chatgpt: A diagnostic analysis. arxiv," *arXiv preprint arXiv:2301.12867*, 2023.

[4] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar *et al.*, "Holistic evaluation of language models," *arXiv preprint arXiv:2211.09110*, 2022.

[5] H. Zhang, H. Song, S. Li, M. Zhou, and D. Song, "A survey of controllable text generation using transformer-based pre-trained language models," *ACM Computing Surveys*, vol. 56, no. 3, pp. 1–37, 2023.

[6] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving, "Fine-tuning language models from human preferences," *arXiv preprint arXiv:1909.08593*, 2019.

[7] N. S. Keskar, B. McCann, L. R. Varshney, C. Xiong, and R. Socher, "Ctrl: A conditional transformer language model for controllable generation," 2019.

[8] J. Qian, L. Dong, Y. Shen, F. Wei, and W. Chen, "Controllable natural language generation with contrastive prefixes," in *Findings of the Association for Computational Linguistics: ACL 2022*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 2912–2924.

[9] X. Qi, Y. Zeng, T. Xie, P.-Y. Chen, R. Jia, P. Mittal, and P. Henderson, "Fine-tuning aligned language models compromises safety, even when users do not intend to!" *arXiv preprint arXiv:2310.03693*, 2023.

[10] M. Kim, H. Lee, K. M. Yoo, J. Park, H. Lee, and K. Jung, "Critic-guided decoding for controlled text generation," in *Findings of the Association for Computational Linguistics: ACL 2023*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 4598–4612.

[11] B. Krause, A. D. Gotmare, B. McCann, N. S. Keskar, S. Joty, R. Socher, and N. F. Rajani, "GeDi: Generative discriminator guided sequence generation," in *Findings of the Association for Computational Linguistics: EMNLP 2021*. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 4929–4952.

[12] S. Dathathri, A. Madotto, J. Lan, J. Hung, E. Frank, P. Molino, J. Yosinski, and R. Liu, "Plug and play language models: A simple approach to controlled text generation," *arXiv preprint arXiv:1912.02164*, 2019.

[13] K. M. Yoo, D. Park, J. Kang, S.-W. Lee, and W. Park, "Gpt3mix: Leveraging large-scale language models for text augmentation," *arXiv preprint arXiv:2104.08826*, 2021.

[14] S. Y. Feng, V. Gangal, J. Wei, S. Chandar, S. Vosoughi, T. Mitamura, and E. Hovy, "A survey of data augmentation approaches for nlp," *arXiv preprint arXiv:2105.03075*, 2021.

[15] OpenAI, "Chatgpt," https://chat.openai.com/, 2023.

[16] "Gpt-4," https://openai.com/research/gpt-4, accessed: 2024-02-05.

[17] 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.

[18] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen *et al.*, "Palm 2 technical report," *arXiv preprint arXiv:2305.10403*, 2023.

[19] 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.

[20] C. Meister, T. Vieira, and R. Cotterell, "Best-first beam search," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 795–809, 2020. [Online]. Available: https://aclanthology.org/2020.tacl-1.51

[21] H. Inan, K. Upasani, J. Chi, R. Rungta, K. Iyer, Y. Mao, M. Tontchev, Q. Hu, B. Fuller, D. Testuggine, and M. Khabsa, "Llama guard: Llm-based input-output safeguard for human-ai conversations," 2023.

[22] F. Wu, Y. Xie, J. Yi, J. Shao, J. Curl, L. Lyu, Q. Chen, and X. Xie, "Defending chatgpt against jailbreak attack via self-reminder," 2023.

[23] Y. Xie, M. Fang, R. Pi, and N. Gong, "Gradsafe: Detecting unsafe prompts for llms via safety-critical gradient analysis," 2024.

[24] "Azure content safety api," https://azure.microsoft.com/en-us/products/ai-services/ai-content-safety, accessed: 2024-02-05.

[25] "Openai moderation api," https://platform.openai.com/docs/guides/moderation/, accessed: 2024-02-05.

[26] R. Liu, G. Xu, C. Jia, W. Ma, L. Wang, and S. Vosoughi, "Data boost: Text data augmentation through reinforcement learning guided conditional generation," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Nov. 2020, pp. 9031–9041.

[27] 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.

[28] A. Liu, M. Sap, X. Lu, S. Swayamdipta, C. Bhagavatula, N. A. Smith, and Y. Choi, "DExperts: Decoding-time controlled text generation with experts and anti-experts," 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)*. Association for Computational Linguistics, Aug. 2021, pp. 6691–6706.

[29] K. Yang and D. Klein, "FUDGE: Controlled text generation with future discriminators," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Jun. 2021, pp. 3511–3535.

[30] A. Singhal *et al.*, "Modern information retrieval: A brief overview," *IEEE Data Eng. Bull.*, vol. 24, no. 4, pp. 35–43, 2001.

[31] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[32] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE transactions on pattern analysis and machine intelligence*, vol. 17, no. 8, pp. 790–799, 1995.

[33] "Toxic comment classification challenge," https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge, accessed: 2024-02-05.

[34] https://www.perspectiveapi.com/, accessed: 2024-02-05.

[35] A. Karamolegkou, J. Li, L. Zhou, and A. Søgaard, "Copyright violations and large language models," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 7403–7412.

[36] "Large language models and copyright," https://en.wikipedia.org/wiki/Wikipedia:Large_language_models_and_copyright, accessed: 2024-03-01.

[37] S. F. Chen, D. Beeferman, and R. Rosenfeld, "Evaluation metrics for language models," 1998.

[38] https://huggingface.co/docs/transformers/en/perplexity, accessed: 2024-02-05.

[39] https://huggingface.co/openai-community, accessed: 2024-02-05.

[40] https://huggingface.co/meta-llama, accessed: 2024-02-05.

[41] https://qdrant.tech/, accessed: 2024-02-05.

[42] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2, accessed: 2024-02-05.

[43] Y. Bang, S. Cahyawijaya, N. Lee, W. Dai, D. Su, B. Wilie, H. Lovenia, Z. Ji, T. Yu, W. Chung *et al.*, "A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity," *arXiv preprint arXiv:2302.04023*, 2023.

[44] 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.

[45] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin *et al.*, "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," *arXiv preprint arXiv:2311.05232*, 2023.

[46] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[47] Y. Elazar, N. Kassner, S. Ravfogel, A. Ravichander, E. Hovy, H. Schütze, and Y. Goldberg, "Measuring and improving consistency in pretrained language models," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 1012–1031, 2021.