# LLMEffiChecker: Understanding and Testing Efficiency Degradation of Large Language Models

XIAONING FENG, College of Computer Science and Technology (College of Data Science), Taiyuan University of Technology, Taiyuan, China
XIAOHONG HAN, College of Computer Science and Technology (College of Data Science), Taiyuan University of Technology, Taiyuan, China
SIMIN CHEN, Computer Science, The University of Texas at Dallas, Richardson, United States
WEI YANG, University of Texas at Dallas, Richardson, United States

Large Language Models (LLMs) have received much recent attention due to their human-level accuracy. While existing works mostly focus on either improving accuracy or testing accuracy robustness, the computation efficiency of LLMs, which is of paramount importance due to often vast generation demands and real-time requirements, has surprisingly received little attention. In this article, we make the first attempt to understand and test potential computation efficiency robustness in state-of-the-art LLMs. By analyzing the working mechanism and implementation of 20,543 public-accessible LLMs, we observe a fundamental property in LLMs that could be manipulated in an adversarial manner to reduce computation efficiency significantly. Our interesting observation is that the output length determines the computation efficiency of LLMs instead of the input, where the output length depends on two factors: an often sufficiently large yet pessimistic pre-configured threshold controlling the max number of iterations and a runtime-generated end of sentence (EOS) token. Our key motivation is to generate test inputs that could sufficiently delay the generation of EOS such that LLMs would have to go through enough iterations to satisfy the pre-configured threshold. We present `LLMEffiChecker`, which can work under both white-box setting and black-box setting. In the white-box scenario, `LLMEffiChecker` develops a gradient-guided technique that searches for a minimal and unnoticeable perturbation at character-level, token-level, and structure-level. In the black-box scenario, `LLMEffiChecker` employs a causal inference-based approach to find critical tokens and similarly applies three levels of imperceptible perturbation to them. Both the white-box and black-box settings effectively delay the appearance of EOS, compelling these inputs to reach the naturally unreachable threshold. To demonstrate the effectiveness of `LLMEffiChecker`, we conduct a systematic evaluation on nine publicly available LLMs: Google T5, AllenAI WMT14, Helsinki-NLP translator, Facebook FairSeq, UNICAMP-DL translator, MarianMT, Google FLAN-T5, MBZUAI LaMini-GPT, and Salesforce CodeGen. Experimental results show that `LLMEffiChecker` can increase on average LLMs' response latency and energy consumption by 325% to 3,244% and 344% to 3,616%, respectively, by perturbing just one character or token in the input sentence. Our case study shows that inputs generated by `LLMEffiChecker` significantly affect the battery power in real-world mobile devices (i.e., drain more than 30 times battery power than normal inputs).

Authors' Contact Information: Xiaoning Feng, College of Computer Science and Technology (College of Data Science), Taiyuan University of Technology, Taiyuan, Shanxi, China; e-mail: fengxiaoning1746@link.tyut.edu.cn; Xiaohong Han (Corresponding author), College of Computer Science and Technology (College of Data Science), Taiyuan University of Technology, Taiyuan, Shanxi, China; e-mail: hanxiaohong@tyut.edu.cn; Simin Chen (Corresponding author), Computer Science, The University of Texas at Dallas, Richardson, Texas, United States; e-mail: simin.chen@utdallas.edu; Wei Yang, University of Texas at Dallas, Richardson, Texas, United States; e-mail: wei.yang@utdallas.edu.

ACM 1049-331X/2024/08-ART186
https://doi.org/10.1145/3664812

CCS Concepts: • **Software and its engineering → Search-based software engineering**; **Software testing and debugging**; *Automatic programming*; *Software evolution;*

Additional Key Words and Phrases: Machine learning, software testing, large language model

**ACM Reference Format:**
Xiaoning Feng, Xiaohong Han, Simin Chen, and Wei Yang. 2024. `LLMEffiChecker`: Understanding and Testing Efficiency Degradation of Large Language Models. *ACM Trans. Softw. Eng. Methodol.* 33, 7, Article 186 (August 2024), 38 pages. https://doi.org/10.1145/3664812

## 1 INTRODUCTION

**Large Language Model (LLM)** is a promising approach that applies neural networks to resolve various text generation problems. LLMs have received significant recent attention from both academia [4, 10, 42, 53] and industry [2, 36, 46, 54, 67, 90, 93] due to its advantages over traditional text generation methods (e.g., N-gram language models [68]). For instance, due to being capable of capturing rather long dependencies in sentences, LLMs are seeing a wide adoption in commercial text generation including OpenAI's GPT products (e.g., ChatGPT) [6, 11, 57, 61] and Meta's LLaMA products [65, 72, 73].

Much research has been done on enhancing the accuracy of LLMs [47, 84]. Recently, research [30, 33, 34, 69] has been conducted to understand the accuracy robustness of existing LLMs by developing a series of adversarial test input generation frameworks that reduce the generation accuracy of existing LLMs. While accuracy robustness is clearly important, we observe that the computation efficiency of LLMs, particularly in terms of the latency and energy spent on generating an input with a specific length, is an equivalently critical property that has surprisingly received little attention. A common and unique characteristic of the LLMs domain is the need to process a huge amount of real-time requests (e.g., OpenAI's ChatGPT has an average monthly visit volume of 15 billion and an average daily consultation volume of approximately 270 million times [28, 50, 63]). The vast demand for generation requests combined with the real-time requirements naturally makes the computation efficiency of any LLM be one of the most critical optimization goals. In this article, we make the first attempt in understanding and testing potential vulnerabilities in terms of the computation efficiency of existing LLMs.

**Key observations revealing vulnerabilities on LLMs computation efficiency.** Our findings are motivated by several observations. Particularly, through analyzing the working mechanisms and detailed implementation of 20,543 public-accessible LLMs (e.g., Google FLAN-T5 [19], BigScience T0 [66]), we observe a fundamental property of LLMs that could be manipulated in an adversarial manner to significantly reduce computation efficiency. Specifically, we observe that the computation efficiency of LLMs is highly sensitive to different inputs, even those exhibiting just minor differences. For instance, slightly modifying an input could incur an order of magnitude more computation demand (e.g., as shown in Figure 2, inserting a character "b" in token "Genäckstück" will increase the latency of HuggingFace's LLM from 0.876 s to 20.382 s, representing an over 20× latency increase). Such dramatic impact on computation efficiency may occur fundamentally because LLMs often need to invoke the underlying decoder with non-deterministic numbers of iterations to generate outputs [49, 75]. Intuitively, the computation efficiency of LLMs is determined by the output length instead of the input, where the output length depends on two factors: an often sufficiently large yet pessimistic pre-configured threshold controlling the max number of iterations (e.g., as shown in Figure 3, a dominant number of our studied LLMs set this threshold to be over 300, which is significantly larger than the actual output length in most cases); and a runtime-generated **end of sentence (EOS)** token. By observing such properties, our key motivation is that it may be

possible to generate test inputs that could sufficiently delay the generation of EOS such that LLMs would have to go through max iterations to satisfy the pessimistic pre-configured threshold.

This implies an important yet unexplored vulnerability of LLMs: adversarially designed inputs that may cause enormous, abnormal computation demand in existing LLMs, thus significantly wasting the computational resources and energy and may adversely impair user experience and even service availability. Such adversarial inputs could result in devastating consequences for many real-world applications (also proved by our experiments). For example, abusing computational resources on commercial text generation service providers (e.g., *HuggingFace* [82]) could negatively impact the quality of service (e.g., enormously long response time or even denial of service). For application domains that are sensitive to latency or energy, such as mobile and IoT devices, abusing computational resources might consume battery in an unaffordable, fast manner.

Motivated by these observations, we aim to systematically develop a framework that generates inputs to test the robustness w.r.t. computation efficiency of LLMs. The generated test inputs may significantly increase the computational demand and thus hinder the computation efficiency regarding response latency, energy consumption, and availability. To make such testing practical, any generated LLMs test inputs shall not be attack-obvious. One objective is thus to make trivial or unnoticeable modifications on normal textual inputs to generate such test inputs. We present `LLMEffiChecker`, which effectively achieves our objectives. `LLMEffiChecker` is developed based on the aforementioned observation. Specifically, LLMs iteratively compute the output token until either the system generates an EOS token or a pre-configured threshold controlling the max number of iterations has been met. For our studied 20,543 LLMs[1] the appearance of EOS is computed from the underlying DNNs output probability. `LLMEffiChecker` develops techniques that could perturb input sentences to change the underlying DNNs output probability and sufficiently delay the generation of EOS, thus forcing these inputs to reach the naturally unreachable threshold. In the white-box setting, `LLMEffiChecker` further develops a gradient-guided technique that searches for a minimal perturbation (including character-level, token-level, and structure-level ones) that can effectively delay the generation of EOS. In the black-box setting, `LLMEffiChecker` utilizes a causal inference-based method to identify crucial tokens without relying on gradient information and correspondingly applies three levels of imperceptible perturbation to effectively degrade the efficiency of LLMs. Applying the above minimal perturbation on the seed input would result in significantly longer output, costing LLMs more computational resources and thus reducing computation efficiency.

**Implementation and evaluation.** We have conducted extensive experiments to evaluate the effectiveness of `LLMEffiChecker`. Particularly, we applied `LLMEffiChecker` on nine real-world publicly available and widely used (e.g., with more than 2,714,275 downloads in November 2023) LLMs (i.e., Google T5 [29, 62], AllenAI WMT14 [1], Helsinki-NLP [35], Facebook Fairseq [55], UNICAMP-DL Translator [51], MarianMT [52], Google FLAN-T5 [19], MBZUAI LaMini-GPT [83], and Salesforce CodeGen [56]). The selected LLMs are trained with different corpus and feature diverse DNN architectures as well as various configurations. We compare `LLMEffiChecker` against four state-of-the-art methods that focus on testing LLMs' accuracy and correctness. Evaluation results show that `LLMEffiChecker` is highly effective in generating test inputs to degrade the computation efficiency of the LLMs under test. Specifically, `LLMEffiChecker` generates test inputs that could increase the LLMs' CPU latency, CPU energy consumption, GPU latency, and GPU energy consumption by 322% to 3,154%, 366% to 3,053%, 327% to 1,969%, and 322% to 1,966%, respectively, through only perturbing one character or token in any seed input sentences. Our

[1] https://huggingface.co/models?pipeline_tag=text2text-generation&sort=downloads

case study shows that inputs generated by LLMEffiChecker significantly affect the battery power in real-world mobile devices (i.e., drain more than 30 times battery power than normal inputs).

**Contribution.** Our contributions are summarized as follows:

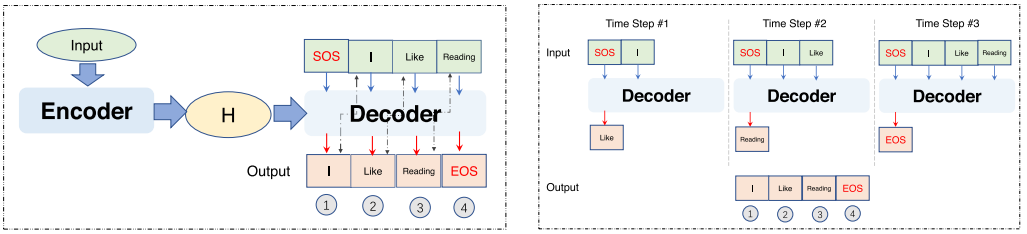
- Characterization: We are the first to study and characterize the computation efficiency vulnerability in state-of-the-art LLMs, which may critically impair latency and energy performance, as well as user experience and service availability. Such vulnerability is revealed by conducting extensive empirical studies on 20,543 publicly available LLMs, which have been downloaded more than 3,260,064 times in November 2023. The results show that the revealed vulnerability could widely exist due to a fundamental property of LLMs.
- Approach: We design and implement LLMEffiChecker, the first framework for testing LLMs' computation efficiency. Specifically, given a seed input, LLMEffiChecker applies gradient-guided and causal inference-based methods to mutate the seed input to generate test inputs in white-box and black-box settings, respectively. Test inputs generated by LLMEffiChecker only perturb one to three tokens in any seed inputs.
- Evaluation: We evaluate LLMEffiChecker on nine real-world publicly available LLMs (i.e., Google T5, AllenAI WMT14, Helsinki-NLP, Facebook FairSeq, U-DL Translator, MarianMT, FLAN-T5, LaMini-GPT, and CodeGen) against four correctness-based testing methods. In addition, we propose a series of metrics (Equation (5)) to quantify the effectiveness of the triggered computation efficiency degradation. Evaluation results suggest existing correctness-based testing methods cannot generate test inputs that impact computation efficiency. In contrast, LLMEffiChecker generates test inputs that increase LLMs' latency and energy consumption by 291% to 12,536% and 207% to 11,172%, respectively.
- Mitigation: We propose a lightweight method to mitigate possible computation efficiency degradation: running a detector at runtime for input validation. We evaluate the performance of our proposed mitigation method in terms of accuracy and additional overheads. Results confirm the efficacy and efficiency of our proposed mitigation method.

This article represents a substantial expansion of our prior research featured in ESEC/FSE'22 [15]. This extension encompasses several key advancements: (1) Diversification of Testing Scope: We have broadened our focus from efficiency testing specific to **neural machine translation (NMT)** models to encompass a broader range, specifically targeting General **Large Language Models (LLMs)**. The scope of our study is now more inclusive, as detailed in the Section 3. (2) Introduction of a Black-box Approach: In addition to the original white-box methodology, we have introduced a novel black-box approach, as explained in Section 5.3. This innovative methodology is designed to operate effectively under realistic scenarios, offering a more robust evaluation of the model's performance. (3) Expanded Subject Evaluation: Going beyond the confines of NMT models, our research evaluates our proposed framework across a wider array of subjects. This includes a comprehensive assessment of the framework's applicability to LLMs for diverse applications, such as sentence completion and code generation.

## 2 BACKGROUND

### 2.1 Working Mechanism of Large Language Models

Much recent research has been done towards developing more accurate and efficient LLMs [9, 49, 60, 70, 74, 75, 84]. The language model computes the conditional probability $P(Y|X)$, where $X = [x_1, x_2, \ldots, x_m]$ is the input token sequence and $Y = [y_1, y_2, \ldots, y_n]$ is the output token sequence. Modern LLMs apply the neural networks to approximate such conditional probability $P(Y|X)$. As shown in Figure 1, The structure of LLMs can be broadly categorized into two types: the Encoder-Decoder architecture (e.g., Google T5 series) and the Decoder-Only




(a) The Encoder-Decoder architecture (b) The Decoder-Only architecture

Fig. 1. Working mechanism of LLMs.

architecture (e.g., OpenAI GPT series). The encoder $f_{en}(\cdot)$ encodes the source input $X$ into hidden representation $H$, then $H$ is fed into the decoder for decoding. Notably, the attention layers in the encoder possess the capacity to analyze all words within the initial sentence, whereas the attention layers of the decoder $f_{de}(\cdot)$ can only access the words positioned before a given word in the input. Consequently, these two architectures are often chosen for different tasks. The Encoder-Decoder architecture is well-suited for tasks involving sequence-to-sequence mappings (e.g., translation and summarization). While the Decoder-Only architecture is more fitting for autoregressive generation tasks, characterized by the sequential generation of output sequences (e.g., text continuation and dialogue systems), it excels in predicting the next piece of text based on the sequence that has already been generated (or a given initial text). An implementation example of LLMs' decoding process is shown in Listing 1.[2] From the code snippet, we observe that the decoding process **starts with a special token (SOS)** and iteratively accesses $H$ for an auto-regressive generation of each token $y_i$ until the **end of sequence token (EOS)** or the maximum iteration (e.g., `max_length`) is reached (whichever condition is reached earlier). To improve LLMs' accuracy, a common practice is to apply the beam search algorithm to search multiple top tokens at each iteration and select the best one after the whole decoding process.

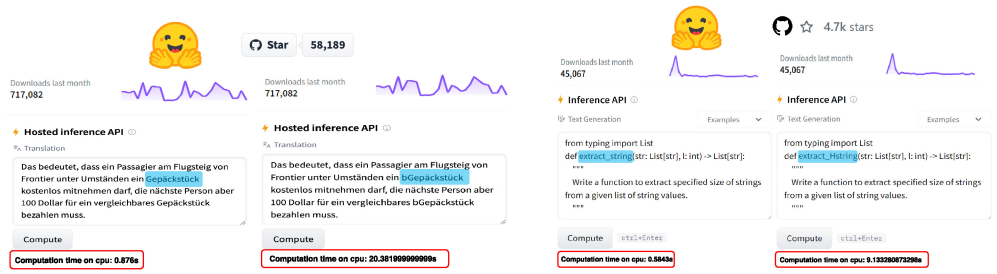
```
'''
Encoding process
'''
decoded_words = ['<SOS>']
for di in range(max_length):
    decoder_output, decoder_hidden = decoder( decoder_input, decoder_hidden,
    encoder_outputs)
    topv, topi = decoder_output.data.topk(1)
    if topi.item() == EOS_token:
        decoded_words.append('<EOS>')
        break
    else:
        decoded_words.append(index2word[topi.item()])
        decoder_input = topi.squeeze().detach()
return decoded_words
```

Listing 1. Source Code Example of LLMs Implementation.

## 2.2 Robustness Testing for NLP Systems

Although modern NLP systems demonstrate human-level performance in terms of accuracy, NLP systems are still far from robust due to the complexity and intractability of the underlying

[2]The code snippet is downloaded from PyTorch LLM tutorial.




(a) Model in Translation (b) Model in Code Generation

Fig. 2. Examples illustrating LLMs' efficiency degradation by inserting one character (using HuggingFace API).

neural networks. To improve the robustness of NLP systems, a series of testing methods have been proposed, which focus on accuracy testing. The core idea of existing work is to perturb seed input sentences with different perturbations and detect output inconsistency between perturbed and seed outputs. At high-level, the perturbations in existing work can be categorized into three types: *(i) character-level:* This type of perturbations [4, 20, 21, 44, 95] represents the natural typos and noises in textual inputs. For example, character swap (e.g., noise → nosie), order random (e.g., noise → nisoe), character insertions (e.g., noise → noisde), and keyboard typo (e.g., noise → noide); *(ii) token-level:* This type of perturbations [18, 44, 64, 69, 88, 91] replaces a few tokens in the seed sentences with other tokens. However, token replacement sometimes would completely change the semantic of the input text; thus, this type of perturbation usually appears in adversary scenarios; *(iii) structure-level:* Different from the above two perturbations, this type of perturbations [30, 33, 34, 45] seeks to generate legal sentences that do not contain lexical or syntactic errors. For example, Reference [33] proposes a structure invariant testing method to perturb seed inputs with `Bert` [40], and the perturbed sentences will exhibit similar sentence structure with the seed sentences.

## 3 MOTIVATION & PRELIMINARY STUDY

In this section, we first give a motivating example in detail to show efficiency degradation issues in real-world LLMs. We then present a comprehensive empirical study based on 20,543 state-of-the-art LLMs, which reveals an important vulnerability in existing LLMs that may suffer from significant efficiency degradation.

### 3.1 Motivating Example

Figure 2 illustrates the efficiency degradation issue that HuggingFace LLMs APIs may experience due to unnoticeable perturbations. Sub-figure (a) depicts Helsinki's model[3] specialized in translating from German to English, while sub-figure (b) showcases Salesforce's CodeGen model[4] tailored for code synthesis tasks. The selected LLMs APIs are rather popular among developers, with 717,082 and 45,067 downloads merely in February 2024. Figure 2 shows the computation time of LLMs in different scenarios using two input sentences, where a normal (abnormal) input is used in the left (right) part of the sub-figure. Note that the abnormal input differs from the normal input by only one character, "b" or "H" (highlighted in blue). Nonetheless, due to such a one-character

[3] https://huggingface.co/Helsinki-NLP/opus-mt-de-en
[4] https://huggingface.co/Salesforce/codegen-350M-mono

Table 1. Top 10 Popular LLMs on HuggingFace Website

| Rank | Model Name | max_length | # of Downloads |
|---|---|---|---|
| **1** | gpt2 | 50 | 23,723,037 |
| **2** | tiiuae/falcon-7b-instruct | 2,048 | 8,068,318 |
| **3** | distilgpt2 | 50 | 4,812,521 |
| **4** | Kyle1668/boss-toxicity-t5-large | 300 | 4,400,913 |
| **5** | facebook/mbart-large-50 | 200 | 4,080,895 |
| **6** | stabilityai/StableBeluga-7B | 4,096 | 3,480,702 |
| **7** | Kyle1668/boss-sentiment-t5-large | 200 | 3,402,617 |
| **8** | t5-small | 300 | 2,714,275 |
| **9** | t5-base | 300 | 2,132,545 |
| **10** | google/flan-t5-base | 300 | 1,307,572 |

The Order is based on the Number of Downloads.

difference in the input, the computation time increases from 0.876 s to 20.382 s (a 2,226.7% increase) and 0.5843 s to 9.133 s (a 1,474.1% increase). This real-world example reveals that state-of-the-art LLMs may have critical yet unrevealed vulnerabilities that negatively impact computation efficiency.

As we discussed in Section 2.1, the working mechanism of LLMs is to iteratively call the decoder $f_{de}(\cdot)$ to generate output tokens until either the particular token EOS is reached or the pre-configured threshold is met. Thus, LLMs with more decoder calls (i.e., denoted as $||f_{de}(\cdot)||$) will consume more computational resources and incur longer computational times. An intuitive approach to mitigate the efficiency degradation issue in Figure 2 is to set a small threshold to limit $||f_{de}(\cdot)||$. However, this solution is impractical due to inherently significant differences of $||f_{de}(\cdot)||$ in the text generation corpus. According to our empirical study of 20,543 LLMs (detailed in Section 3.2), the majority of them set `max_length` over 300. To further understand why this intuitive approach does not work, we conduct a comprehensive empirical study using 20,543 state-of-the-art LLMs. Specifically, we focus on answering the following two research questions:

— **RQ 1.1:** *What are the current engineering configurations in real-world LLMs that control* $||f_{de}(\cdot)||$? (Section 3.2)
— **RQ 1.2:** *Why is small threshold impractical to mitigate efficiency degradation?* (Section 3.3)

### 3.2 Current Engineering Configurations

*3.2.1 Study Methodology.* We investigate the configurations of existing mainstream LLMs. Specifically, we study 20,543 LLMs (e.g., Google Flan-T5, BigScience BLOOMZ) from HuggingFace online LLMs service.[5] HuggingFace is a commercial platform that provides third-party real-time NLP service, which covers almost all LLMs architectures. LLMs on the HuggingFace platform are open-source and widely used by public, as shown in Table 1 (e.g., the most popular LLMs in HuggingFace have been downloaded for more than 23,723,037 times in November 2023). HuggingFace provides high-level abstraction API for LLMs service. List 2 shows code snippets of using HuggingFace API to load Google T5 service. All language model classes are inherited from a common parent class, `GenerationMixin`, which contains all functions supporting text generation. We parse the source code of the `GenerationMixin.generate` function and observe that the generation flow could be divided into nine parts. Among all nine parts, we find that the eighth part determines

[5]https://huggingface.co/

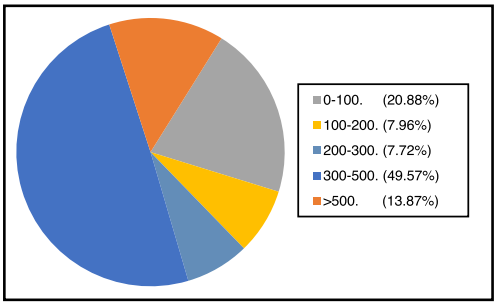



Fig. 3. The distribution of max_length values.

the critical stopping criteria. The source code of the eighth part is shown in List 3. From the source code, we observe that two variables, max_length and max_time, determine the stopping criteria. max_length is a variable from the LLMs' configuration file that determines the maximum length of the sequence to be generated, equivalent to the maximum number of decoder calls mentioned earlier. Similarly, max_time is a variable that determines the maximum computation time. According to HuggingFace programming specifications, only one of these two fields needs to be set. Finally, we select all LLMs in the Text2Text Generation column from HuggingFace's API services[6] and parse each LLM's configuration file to check how max_length and max_time have been set.

```
# HuggingFace high-level API for text generation
model = AutoModelWithLMHead.from_pretrained("t5-base")
s = "CS is the study of computational systems"
input_tk = tokenizer(s, return_tensors="pt").input_ids
res_tk = model.generate(input_tk)
```

Listing 2. HuggingFace libraries high-level text generation API.

```
# 8. prepare stopping criteria
stopping_criteria = self._get_stopping_criteria(
                max_length=max_length,
                max_time=max_time,
                stopping_criteria=stopping_criteria)
```

Listing 3. Stopping criteria in text generation.

*3.2.2 Study Results.* Among all 20, 826 LLMs, we successfully collect 20, 543 configuration files, where 14, 266 of them include the max_length field and none of them includes the max_time field. This is mainly because the max_time field is hardware-dependent. The statistical results of the max_length values are shown in Figure 3. We have the following two observations: First, there is a significant variance in the max_length value (ranging from 8 to 16,896); second, the majority of LLMs (63.44%) configure the max_length to values surpassing 300, i.e., the maximum decoder invocation exceeds this threshold. Furthermore, if there are no specifications for max_length in the model configuration, then it potentially indicates a bug, as this omission could lead to unpredictable behavior and may not align with the user's expectations for the generated text. We present the following two evidences: First, when utilizing HuggingFace's transformers library to load a model (e.g., List 2), if max_length is not specified in the model configuration file, then the default

[6]https://huggingface.co/models?pipeline_tag=text2text-generation&sort=downloads

Table 2. Statistical Results of Efficiency Differences in LLMs

| Language | | # of pairs | Quantile of Target Length | | | | Quantile of Length Ratio | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Src | Tgt | | 10% | 50% | 90% | 100% (max) | 1% (min) | 10% | 50% | 90% | 100% (max) |
| fr | en | 13,172,019 | 4.00 | 24.00 | 52.00 | 97.00 | 0.50 | 0.87 | 1.10 | 1.47 | 3.00 |
| zh | en | 9,564,315 | 11.00 | 41.00 | 87.00 | 179.00 | 0.90 | 1.38 | 1.83 | 3.00 | 8.26 |
| zh | es | 9,847,770 | 10.00 | 40.00 | 87.00 | 176.00 | 0.75 | 1.19 | 1.57 | 2.68 | 8.50 |
| zh | fr | 9,690,914 | 11.00 | 41.00 | 88.00 | 178.00 | 0.74 | 1.21 | 1.63 | 2.85 | 8.29 |
| zh | ru | 9,557,007 | 10.00 | 42.00 | 90.00 | 180.00 | 0.62 | 1.60 | 2.25 | 5.00 | 13.75 |

1%, 10%, 50%, 90%, 100% Represent Quantile.

value is set to 20. It is strongly advised in the official documentation to set an appropriate value manually.[7] The default small value is a conservative choice to facilitate a quick start for users, as longer outputs necessitate increased computational resources (time and memory) for generation, processing, and storage. However, this default value is insufficient to convey adequate information, necessitating users to define a reasonable `max_length` manually. Detailed arguments on this matter will be provided in Section 3.3. Second, decoder-only LLMs also return the input prompt as part of the output. Consequently, if the input length exceeds 20 tokens, then the model will not produce any output and trigger a UserWarning: "Input length exceeds the default max_length (=20)." This may result in unexpected behavior. Note that real-world LLMs prefer to set such a large threshold just to prevent unresponsiveness (e.g., dead-loop). However, in most cases with normal inputs, such a threshold will not yield any real impact, as the EOS token often appears much earlier (e.g., in code generation applications, setting the `max_length` of LLMs to 512 is a widely adopted practice [8, 48, 94]).

### 3.3 Feasibility Analysis of an Intuitive Solution

*3.3.1 Study Methodology.* An intuitive solution to mitigate the efficiency degradation is to limit $||f_{de}(\cdot)||$ (i.e., the `max_length` field). In this section, we conduct a statistical analysis to prove that such an intuitive solution is infeasible. We analyze the distribution of `max_length` of the target sentence (ground truth) in the training corpus. We select the MultiUN dataset [22] as the subject in our empirical study because of the following criteria: *(i)* the datasets are open-source and publicly available; *(ii)* the datasets are widely studied in recent works (with more than 1,000 citations until November 2023); *(iii)* the datasets are diverse in covering various areas (e.g., different languages, concepts, etc.). MultiUN dataset is a collection of translated documents from the United Nations. It includes seven languages with 489,334 files and a total number of 81.41M sentence fragments. We parse the source/target sentence pairs in the MultiUN dataset and measure the length of all target sentences.

*3.3.2 Study Results.* The statistic results of the output length are shown in Table 2 (full results could be found in an anonymous website.[8]) Column "Quantile of Target Length" shows the target sentence length under different quantiles, and Column "Quantile of Length Ratio" shows the ratio of sentence length between the source and target. From the results, we observe that the lengths of target sentences (ground truth) are in sparse distributions. Particularly, the ratio of sentence length between the source and target exhibits rather large variance. For instance, the length of target sentence varies from 4 to 97, and the ratio is from 0.62 to 13.75 for language `fr` and `en`. As a

[7] https://huggingface.co/docs/transformers/v4.38.1/en/llm_tutorial
[8] https://github.com/Cap-Ning/LLMEffiChecker

result, setting a small `max_length` field will lead to low-precision generation results. For instance, in the last line of Table 2, i.e., generating `zh` to `ru`, if setting `max_length` to 42, at least 50% of data will not be generated completely. Thus, we can conclude that the intuitive solution, i.e., setting a small `max_length` field, is impractical to avoid efficiency degradation issues. On the contrary, setting a sufficiently large `max_length` can address the limitation of incomplete text generation while not incurring efficiency issues for any ordinary inputs due to the EOS mechanism.

## 4 PROBLEM FORMULATION

Our goal is to generate test inputs that can degrade the computation efficiency of LLMs. Our proposed method seeks to perturb a seed sentence to craft test inputs. The perturbed test inputs will incur significantly longer computation time, thus impairing user experience and even causing service unavailability. Note that we allow general and unnoticeable perturbation patterns, including adding limited number of characters (e.g., 1–3 characters) at arbitrary positions and replacing arbitrary tokens using semantic-equivalent alternatives. As we discussed in Section 2, LLMs' computation efficiency depends on the output length, where a lengthier output implies less computation efficiency. Thus, our goal can be achieved through increasing LLMs' output length through generating effective test inputs. We thus formulate our problem of generating test inputs for computation efficiency testing as the following optimization:

$$\Delta = \text{argmax}_{\delta} \quad ||f_{de}(x+\delta)|| \qquad s.t.\ ||\delta|| \leq \epsilon, \tag{1}$$

where $x$ is the seed input, $f_{de}(\cdot)$ is the decoder of the LLMs under test, $\epsilon$ is the maximum allowed perturbation, and $||f_{de}(\cdot)||$ measures the number of times of LLMs's decoders being called. Our proposed `LLMEffiChecker` tries to search a perturbation $\Delta$ that maximizes the decoders' calling times (decreasing target LLMs efficiency) within a minimum allowable perturbation threshold (which ensures unnoticeable perturbations).

## 5 METHODOLOGY

We now present `LLMEffiChecker`, designed for both white-box and black-box scenarios. It provides three specific implementations: character-level perturbation, token-level perturbation, and structure-level perturbation.

### 5.1 Design Overview

`LLMEffiChecker` demonstrates practicality by functioning seamlessly in both white-box and black-box settings. In either scenario, `LLMEffiChecker` employs an iterative process where it systematically perturbs a single token within a seed sentence using various types of perturbations. A design overview of the procedural steps for each iteration is presented in Figure 4. This illustration encapsulates three pivotal steps applicable to both white-box and black-box settings:

(1) *Finding critical tokens.* For each seed sentence, we feed it to LLMs under test. In the white-box setting, `LLMEffiChecker` applies a gradient-based approach to identify critical tokens with the highest impact on the computation efficiency of LLMs. Conversely, in the black-box setting, `LLMEffiChecker` employs a casual inference-based instead of a gradient-based approach to pinpoint critical tokens that significantly influence LLMs' computational efficiency.
(2) *Mutating seed input sentences.* After identifying the critical tokens in the seed sentences, we mutate the seed sentences with three types of perturbations and generate three lists of similar sentences.
(3) *Detecting efficiency degradation.* We feed the mutated sentences and the seed sentence into LLMs and detect any efficiency degradation.

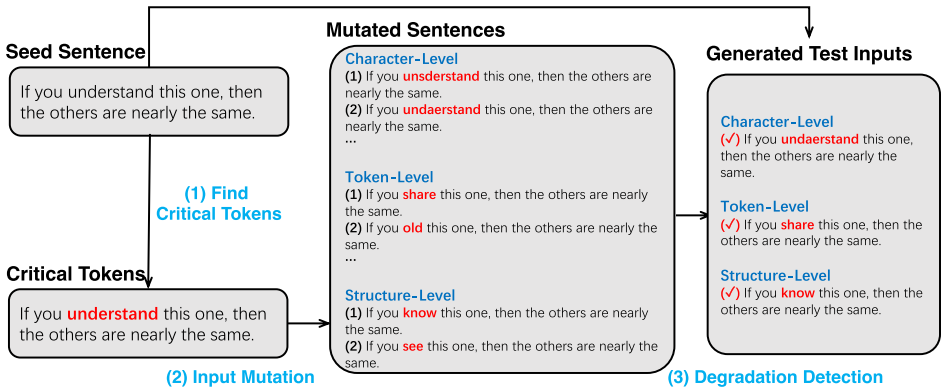



Fig. 4. Design overview of LLMEffiChecker.

### 5.2 White-box Detail Design

**Finding Critical Tokens:** Given a seed sentence $x = [tk_1, \ldots, tk_m]$, the first step is to identify tokens that are critical to LLMs' efficiency. As we discussed earlier, LLMs' computation efficiency depends on the corresponding output length given any input, which is determined by the pre-configured threshold and the EOS token. In Section 3, we showed that the pre-configured threshold is set as a fixed value in the configuration files of LLMs. Thus, to generate effective testing inputs, our objective is to decrease the probability that the EOS token would appear given a specific input to reduce LLMs' computation efficiency.

Formally, let LLM's output probability be a sequence of vectors, i.e., $[p_1, p_2, \ldots, p_n]$, and the probability of EOS token appearance be $[p_1^{eos}, p_2^{eos}, \ldots, p_n^{eos}]$. We seek to find the importance of each token $tk_i$ in $x$ to this probability sequence. We also observe that the output token sequence will affect EOS's probability [27]. Specifically, LLMs generate tokens in the generated sequences based on the generated probability distribution. When the generated sequence is semantically complete or matches a common grammatical structure that typically ends, the model may predict a higher probability for the EOS token. To encourage deviations from the original generated token sequence and focus more on other possible candidate tokens, we incorporate $p_i^{o_i}$ into $f(x)$ to enhance the output uncertainty on each generated token, promoting longer, more complex sequences. Thus, we define the importance score of token $tk_i$ as $g_i$, shown in Equation (2).

$$o_i = \text{argmax}(p_i) \qquad f(x) = \frac{1}{n}\sum_{i}^{n}\left(p_i^{eos} + p_i^{o_i}\right) \qquad g_i = \sum_{j}\frac{\partial f(x)}{\partial tk_i^j}, \tag{2}$$

where $[o_1, o_2, \ldots, o_n]$ is the current output token, $f(x)$ is the probability we seek to minimize; it can delay the generation of the EOS token and introduce more uncertainty for each generated token in the prediction process to break the existing output dependency, thereby maximizing the generation of longer sentences to the fullest extent. $tk_i^j$ is the $j$th dimension of $tk$'s embeddings, and $g_i$ is the derivative of $f(x)$ to $i$th token's embedding. The score $g_i$ assesses the importance of the token $tk_i^j$ for the output length. It is calculated by summing the gradients, which quantify the sensitivity of $f(x)$ to variations in each dimension of the token's embedding.

**Input Mutation:** After identifying important tokens, the next step is to mutate the important token with unnoticeable perturbations. In this step, we get a set of perturbation candidate $L$ after we perturb the most important tokens in the original input. We consider three kinds of perturbations, i.e., character-level perturbation, token-level perturbation, and structure-level perturbation.

Table 3. Examples of Character-level, Token-level, and Structure-level Perturbation under Different Size

| Original | $\epsilon$ | Do you know who Rie Miyazawa is? |
|---|---|---|
| Character-Level | 1 | Do you know who Rie Miya-zawa is? |
| | 2 | Do you know whoo Rie Miya-zawa is? |
| Token-Level | 1 | Do Hello know who Rie Miyazawa is? |
| | 2 | Do Hello know who Hill Miyazawa is? |
| Structure-Level | 1 | Do you remember who Rie Miyazawa is? |
| | 2 | Do you remember what Rie Miyazawa is? |

Table 3 shows some examples of character-level, token-level, and structure-level perturbations with different perturbation sizes $\epsilon$ (the perturbation is highlighted with the color red).

For character-level perturbation, we consider character insertion perturbation. Specifically, we insert one character $c$ into token $tk$ to get another token $\delta$. The character-insert perturbation is common in the real world when typing quickly and can be unnoticeable without careful examination. Because character insertion is likely to result in **out-of-vocabulary (OOV)**, it is thus challenging to compute the token replacement increment at token-level. Instead, we enumerate possible $\delta$ after character insertion to get a candidate set $L$. Specifically, we consider all letters and digits as the possible character $c$, because humans can type these characters through the keyboard, and we consider all positions as the potential insertion position. Clearly, for token $tk$, which contains $l$ characters, there are $(l+1) \times ||C||$ perturbation candidates, where $||C||$ denotes the size of all possible characters. For token-level perturbation, we consider replacing the original token $tk$ with another token $\delta$. To compute the target token $\delta$, we define token replace increment $\mathcal{I}_{src,tgt}$ to measure the efficiency degradation of replacing token $src$ to $tgt$. As shown in Equation (3), $E(\cdot)$ is the function to obtain the corresponding token's embedding, $E(tgt) - E(src)$ represents the vector increment in the embedding space, capturing the semantic and syntactic variation and measuring the impact of the replacement on the sentence's meaning and structure. It explores a wider range of potential outputs, further breaking the original output dependency, leading to more diverse and complex sequences, making it difficult for LLMs to converge to a coherent output. Recall that Equation (2), $\frac{\partial f(x)}{\partial tk_i^j}$ indicates the sensitivity of output length to each embedding dimension. Therefore, $\mathcal{I}_{src,tgt}$ denotes the total benefits of replacing token $src$ with $tgt$. We search the target token $\delta$ in the vocabulary to maximize the token replace increment with the source token $tk$.

$$\mathcal{I}_{src,tgt} = \sum_j (E(tgt) - E(src)) \times \frac{\partial f(x)}{\partial tk_i^j} \qquad \delta = \text{argmax}_{tgt}\ \mathcal{I}_{tk,tgt}; \tag{3}$$

For structure-level perturbation, we follow existing work [33, 69] to parse the seed input sentence as a constituency tree and replace the critical token with another token based on `Bert` [5]. Unlike token-level perturbation, the structure-level perturbation ensures the constituency structure of the perturbed sentence is the same as the seed one. Figure 5 shows an example of the structure-level perturbation. To enhance clarity, our explanation utilizes the left section of the tree as an illustrative example. At the apex, the "S" symbolizes the sentence in its entirety. Descending from the top, the sentence splits into a **noun phrase (NP)** and a **verb phrase (VP)**, representing the basic **Subject-Verb-Object (SVO)** pattern inherent to elementary English structure. The NP itself breaks down further into a possessive pronoun "PRP$" (our) and a common noun "NN" (group), indicating "our group" as the subject of the sentence. Within this seed sentence, "group" has been identified as the critical token. After feeding the parsed information from the sentence

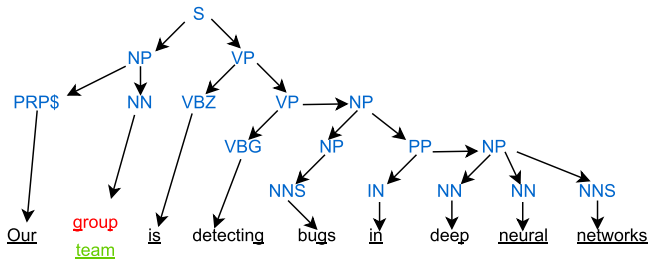



Fig. 5. Constituency tree of sentence.

constituent tree into the BERT model, the token "team" is produced as a structural perturbation. This method of critical token replacement retains the original sentence structure, affirming the integrity of the constituency tree post-perturbation.

**Efficiency Degradation Detection:** After collecting candidate perturbations $L$, we select an optimal perturbation from the collected candidate sets. Since our objective is searching this perturbation candidate set that will produce a longer output length, we straightforwardly test all perturbations in this set and select the optimal perturbation that produces the maximum output length.

### 5.3 Black-box Detail Design

**Finding Critical Tokens:** Note that selecting critical tokens is relatively straightforward in a white-box scenario, since it can be easily accomplished by inspecting the gradients of LLMs, while most other tokens are irrelevant. However, in the more common black-box setup, model gradients are unavailable. In black-box settings, employing random mutation to generate test inputs often proves ineffective due to the vastness of the search space. To overcome this challenge, we propose a novel approach grounded in the concepts of delta debugging [76] and causal inference [87] to identify the critical tokens with the utmost impact on the computational efficiency of LLMs. Additionally, our approach is based on the fundamental conclusion discussed in Section 2, which states that the computational efficiency of LLMs depends on the resulting output length for any given input. Longer outputs necessitate more frequent invocations of the decoder during input processing, thereby demanding a higher volume of **floating-point operations (FLOPs)**. Specifically, we first decompose the input by removing each token from the original input sentence, breaking it down into multiple subsets. By comparing the output length of each subset with the original output length, we pinpoint the sentence with the most substantial difference in output length from the seed sentence. Subsequently, we identify the missing token in this sentence, which constitutes the critical tokens we are seeking. Through this strategic division of the search process, our approach adeptly identifies the critical tokens in black-box scenarios.

Formally, given a seed sentence $S_{\text{orig}} = [tk_1, tk_2, \ldots, tk_m]$, we generate debugging subsets $S_i$ by removing the token $tk_i$ from $S_{\text{orig}}$. Subsequently, we feed each $S_i$ and $S_{\text{orig}}$ into the target LLM to obtain the corresponding output lengths $O_i$ and $O$. Our objective is to identify the index $j$ that maximizes $\gamma_j$. Once this index $j$ is determined, the critical token is $tk_j$ in $S_{\text{orig}}$ (refer to Equation (4)).

$$\gamma_i = |O_i - O| \qquad j = argmax_i \gamma_i \tag{4}$$

Specifically, we conceptualize LLMs as a sequence of mappings that transition from an input domain to an output domain, with each distinct input eliciting a unique output [40]. By employing causal inference methods, we modify the inputs and monitor the resultant variations in the outputs. This process enables us to infer the correlation between diverse inputs and their corresponding

Table 4. The LLMs under Test in Our Experiments

| Model | Task Category | Model_size | Vocab Size | Max_length | URL |
|---|---|---|---|---|---|
| H-NLP | En-Zh Translation | 298 MB | 65,001 | 512 | https://huggingface.co/Helsinki-NLP/opus-mt-en-de |
| AllenAi | En-De Translation | 235 MB | 42,024 | 200 | https://huggingface.co/allenai/wmt16-en-de-dist-12-1 |
| T5 | En-Zh Translation | 242 MB | 32,100 | 200 | https://huggingface.co/t5-small |
| U-DL | En-Pt Translation | 892 MB | 32,128 | 200 | https://huggingface.co/unicamp-dl/translation-en-pt-t5 |
| FairSeq | En-De Translation | 1.08 GB | 42,024 | 200 | https://huggingface.co/facebook/wmt19-en-de |
| MarianMT | En-Zh Translation | 310 MB | 65,001 | 512 | https://huggingface.co/DDDSSS/translation_en-zh |
| Flan-T5 | Sentence Completion | 308 MB | 32,128 | 300 | https://huggingface.co/google/flan-t5-small |
| LaMini-GPT | Sentence Completion | 510 MB | 50,258 | 200 | https://huggingface.co/MBZUAI/LaMini-GPT-124M |
| CodeGen | Code Generation | 797 MB | 51,200 | 200 | https://huggingface.co/Salesforce/codegen-350M-mono |

output lengths, which serve as indicators of the LLMs' computational efficiency. Through this analytical approach, we aim to pinpoint the critical tokens that are instrumental in this dynamic.

**Input Mutation:** The character-level perturbations and structure-level perturbations described in Section 5.2 are well-suited for black-box settings. Consequently, we focus specifically on modifying token-level perturbations in this section. Our intuition is that even in the black-box scenario, obtaining the model's vocabulary is relatively straightforward. This is because models performing the same task in the same language typically share similar vocabularies, and the tokens within it are visible in the model input. Consequently, upon identification of the critical tokens, we proceed to randomly select tokens from the vocabulary to effect replacements.

**Efficiency Degradation Detection:** Upon compiling a set of candidate perturbations, denoted as $L$, we proceed to select the optimal perturbation from this collection. Since our aim is to identify a perturbation candidate that leads to a longer output length, we systematically assess all perturbations within this set and choose the one that yields the maximum output length.

## 6 EVALUATION

We evaluate `LLMEffiChecker` and answer the following research questions:

- ***RQ 2.1 (Severity)***: How severe will `LLMEffiChecker` degrade LLMs efficiency?
- ***RQ 2.2 (Effectiveness)***: How effective is `LLMEffiChecker` in generating test samples that degrade LLMs efficiency?
- ***RQ 2.3 (Sensitivity)***: Can `LLMEffiChecker` generate useful test samples that decrease LLMs efficiency under different LLMs' configurations?
- ***RQ 2.4 (Overheads)***: What is the overhead of `LLMEffiChecker` in generating test samples?
- ***RQ 2.5 (Ablation Study)***: How much does each component in `LLMEffiChecker` contribute to the overall performance?

### 6.1 Experimental Setup

**Models and Datasets.** As shown in Table 4, we consider the following nine public LLMs as our evaluation models: Google's T5 [62], AllenAI's WMT14 Transformer [55], and Helsinki-NLP's H-NLP Translator [41], Facebook's Fairseq Transformer [55], UNICAMP-DL's U-DL Translator [51], Fine-tuned MarianMT [52], Google's FLAN-T5 [19], Mohamed Bin Zayed University's LaMini-GPT [83], and Salesforce's CodeGen [56]. The first six models are employed for translation tasks, and the subsequent two models are capable of handling various downstream Natural Language Processing tasks. In this article, our focus is on sentence completion as the subject of investigation. The last model is specialized in code generation. Individually, T5 is released by Google, which is first pre-trained with multiple language problems and then fine-tuned on the English-German

translation task. We apply English sentences from dataset ZH19 as seed inputs to generate test samples. AllenAI's WMT14 is one of LLMs from the company AllenAI, which is trained on the WMT19 shared news translation task based on the transformer architecture. We select the WMT14 en-de model as our evaluation model, which is designed for the English-German translation task. H-NLP is a seq2seq model, where the source language is English and the target language is Chinese. For each experimental subject, we randomly select 1,000 inputs from the test dataset as the seed inputs.

To further validate the efficiency loopholes in LLMs for translation, we have additionally chosen three publicly available and high-performing translation LLMs. Fairseq is one of the language models that Facebook FAIR submitted to the WMT19 shared news translation task, and it is based on the FFN transformer architecture. We select Fairseq's en-de model as our victim model, which is designed for the English-German translation task. U-DL, developed by Natural Language and Deep Learning Process Laboratory of Universidade Estadual de Campinas, is a model built on the T5 architecture and fine-tuned for tasks involving English and Portuguese translation. Marian is a Neural Machine Translation framework, which is mainly developed by the Microsoft Translator team, and it is released under MIT License. MarianMT Framework's flexibility and efficiency have made it exceptionally popular in the translation field. We choose English-Chinese translator as our evaluation model. To ensure experiment consistency, we randomly selected 1,000 English sentences from the ZH19 dataset as seed inputs.

In addition, we selected three open source LLMs for other application scenarios: Flan-T5 (Encoder-Decoder) instruction-finetune on a collection of data sources using a diverse set of instruction templates. Its performance and ability to generalize to unseen tasks are notably superior to those of the baseline T5 model. LaMini-GPT (Decoder-Only), released by Mohamed Bin Zayed University of Artificial Intelligence, is built on the GPT-2 framework, fine-tuned and distilled with a large-scale instruction dataset derived from ChatGPT, all while being more compact and efficient carried out within the structure. We employ the dataset HellaSwag [89] to assess the sentence completion tasks for the two aforementioned large language models. Likewise, we randomly select 1,000 data samples from this dataset as initial seed inputs. CodeGen, a creation of company Salesforce, is part of the CodeGen family, specializing in autoregressive language models for code generation. Our evaluation of this model involves the utilization of the mbpp dataset [3], which comprises 427 Python programming challenges and is a widely recognized benchmark for code generation tasks. It is important to note that this dataset falls into the category of "zero-shot" datasets, as it lacks any input/output demonstrations within its prompts. To improve the efficiency of our experiments, we have implemented a modification in the prompt format. In particular, we processed each problem by incorporating a function header and converting the language instructions into function docstrings. Note that this same modification is also used in existing works [8].

We select subjects (i.e., model, dataset) following policies below.

- **Availability and Accessibility:** The selected subjects are publicly available, ensuring our research can be widely replicated and expanded upon.
- **Adoption and Prevalence:** The chosen subjects are widely used across various fields. For example, the H-NLP model had 263,348 downloads on Huggingface in February 2024 [41], Flan-T5 has been cited over 1,300 times [19], and the MBPP dataset represents the mainstream benchmarks for evaluating code generation models and has gained widespread utilization in prior research [26, 58, 80].
- **Diversity and Representativeness:** Our selection of datasets and models emphasizes diversity and representativeness across various dimensions. Specifically, for LLMs used in translation tasks, our chosen models have different model architectures, training corpora, translation languages, and training processes. Such a strategic selection underpins the

universality and reliability of our results. For sentence completion applications, we have chosen flagship models representing the two principal architectures in contemporary text generation: Google's Flan-T5, embodying the encoder-decoder framework, and MBZUAI's LaMini_GPT, a decoder-only model. Notably, LaMini_GPT has undergone extensive fine-tuning with high-quality ChatGPT instructions, achieving performance metrics that eclipse those of OpenAI's GPT-2 [83]. In the realm of code generation, our selection includes models from the CodeGen family. Upon its release, CodeGen was recognized as a leading state-of-the-art model in code generation, showcasing remarkable capabilities in automating programming tasks and epitomizing the cutting edge of the field [58].

**Comparison Baselines.** A branch of existing works has been proposed for testing LLMs [4, 18, 30, 33, 34, 69]. However, all of them focus on testing LLMs' correctness. To the best of our knowledge, we are the first to study LLMs' efficiency degradation issue. To show that existing correctness testing methods can not generate test inputs that trigger efficiency degradation for LLMs. We compare `LLMEffiChecker` against four state-of-the-art correctness testing methods, which are designed to generate testing inputs that produce incorrect results. Specifically, we choose `SIT` [33], `TransRepair` [69], `Seq2Sick` [18], and `SynError` [4] as our comparison baselines. `SIT` proposes a structure-invariant testing method, which is a metamorphic testing approach for validating language models. Given a seed sentence, `SIT` first generates a list of similar sentences by modifying tokens in the seed sentence. After that, `SIT` compares the structure of the original outputs and the generated outputs to detect generation errors. `TransRepair` is similar to `SIT`, with the difference that the unperturbed parts of the sentences preserve their adequacy and fluency modulo the mutated tokens. Thus, any perturbed input sentence violating this assumption will be treated as incorrect. `Seq2Sick` replaces the tokens in seed inputs to produce adversarial generation outputs that are entirely different from the original outputs. `SynError` is a character-level testing method, which minimizes the LLMs's accuracy (BLEU score) by introducing synthetic noise. Specifically, `SynError` introduces four character-level perturbations: swap, fully random, and keyboard typos to perturb seed inputs to decrease the BLEU score.

**Experimental Procedure.** We run `LLMEffiChecker` in both white-box and black-box settings to test the above-mentioned nine LLMs. Given a seed input, `LLMEffiChecker` perturbs the seed input with different types of perturbations. `LLMEffiChecker` has one hyper-parameter ($\epsilon$) that is configurable. In our experiments, we follow existing works [44] and set perturbation size (i.e., $\epsilon$) from 1 to 3, representing different degrees of perturbation. For RQ1 (severity), we measure the percentage of the average and maximum increased computational resource in terms of iteration loops, latency, and energy consumption (Equation (5)), due to the generated test inputs compared to the seed inputs. For RQ2 (effectiveness), we measure the degradation success ratio (Equation (6)), which quantifies the percentage of the test inputs out of all generated by `LLMEffiChecker` that can degrade the efficiency to a degree that is larger than a pre-defined threshold. A higher ratio would imply better efficacy in generating useful test inputs. For RQ3 (sensitivity), we run `LLMEffiChecker` on LLMs with different configurations to study whether the efficacy of `LLMEffiChecker` is sensitive to configurations. For RQ4 (overheads), we measure the average overheads of running `LLMEffiChecker` to generate test inputs. For RQ5 (ablation study), we conduct an ablation study to validate the contribution of each component in `LLMEffiChecker`. It is worth noting that, due to the unique nature of code generation tasks, for the evaluation of the CodeGen model, we have made modifications to the stopping criteria. Specifically, we have expanded the list of default EOS tokens (i.e., "`<|endoftext|>`," "`\ndef`," "`\n#`," "`\nif`," and "`\nclass`"). This method finds widespread application in code generation works [8, 26, 48, 80] and proves to be effective in enhancing the efficiency of the model.

**Implementation.** We implement LLMEffiChecker with the PyTorch library, using a server with Intel Xeon E5-26 CPU and eight Nvidia A4500 GPUs. For the baseline methods, we implement SIT and TransRepair using the authors' open sourced code [32, 33]. We re-implement Seq2sick and SynError according to the corresponding papers, as the original implementations are not open sourced. For LLMs used in our evaluation, we download the pre-trained models using the HuggingFace APIs, and we configure LLMs using both default configurations and varied configurations to answer RQ3.

### 6.2 RQ 2.1: Severity

**Metrics.** Our evaluation considers both hardware-independent metrics (i.e., number of iteration loops) and hardware-dependent metrics (i.e., latency and energy consumption), which quantitatively represent LLMs' efficiency. The number of iteration loops is a widely used hardware-independent metric for measuring software computational efficiency [81]. In this experiment, the focus is on calculating the number of decoder calls presented in Section 2.1, which corresponds to the number of output tokens. Higher decoder calls indicate that LLMs cast more FLOPs to handle the input text, which leads to less efficiency [16]. Response latency (i.e., the output generation time) and energy consumption are two widely used hardware-dependent metrics for measuring systems efficiency. Larger latency and energy consumption clearly indicate less efficiency.

$$\begin{aligned} \text{I-Loops} &= \frac{\text{Loops}(x') - \text{Loops}(x)}{\text{Loops}(x)} \times 100\% \\ \text{I-Latency} &= \frac{\text{Latency}(x') - \text{Latency}(x)}{\text{Latency}(x)} \times 100\% \\ \text{I-Energy} &= \frac{\text{Energy}(x') - \text{Energy}(x)}{\text{Energy}(x)} \times 100\% \end{aligned} \tag{5}$$

We use I-Loops, I-Latency, and I-Energy to denote the number of iteration loops, response latency, and energy consumption, respectively. The formal definitions of I-Loops, I-Latency, and I-Energy are shown in Equation (5), where $x$ denotes the seed input and $x'$ represents the perturbed input under LLMEffiChecker. Loops$(\cdot)$, Latency$(\cdot)$, and Energy$(\cdot)$ denote the functions that calculate the average number of iteration loops, latency, and energy consumption, respectively. Larger values of I-Loops, I-Latency, and I-Energy indicate a more severe efficiency degradation caused by the test inputs generated under LLMEffiChecker. In our evaluation, we measure the hardware-dependent efficiency metrics (i.e., latency and energy consumption) on two popular hardware platforms: Intel Xeon E5-2660v3 CPU and Nvidia A4500 GPU. For precise measurement of energy consumption on both CPU and GPU, we employ advanced monitoring libraries. Intel's **Running Average Power Limit (RAPL)** interface is used for the CPU, offering an effective method to observe and manage the power usage of its various components. For the GPU, we utilize Nvidia's **Python Library for NVIDIA Management Library (PyNVML)**, which serves as a Python wrapper for NVML, enabling accurate tracking and analysis of energy consumption. This rigorous methodology allows us to capture comprehensive data on the energy efficiency of these platforms across different operational scenarios, providing critical insights into their performance dynamics and sustainability footprint. Furthermore, to mitigate potential biases introduced by hardware dependencies in the evaluated metrics, we enhanced the reliability and reproducibility of our measurements by averaging the experimental results over three runs.

**Results.** Table 5 and Table 6 showcase the average and maximum efficiency degradation results under varied perturbations for LLMs, respectively. Specifically, we recorded the required I-Loops, I-Latency, and I-energy for all test inputs, providing their mean and best-performing outcomes.

Table 5. The Average Effectiveness Results of LLMEffiChecker in Degrading LLMs Performance

| Subject | Methods | I-Loops | | | I-Latency(CPU) | | | I-Energy(CPU) | | | I-Latency(GPU) | | | I-Energy(GPU) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $\epsilon = 1$ | $\epsilon = 2$ | $\epsilon = 3$ | $\epsilon = 1$ | $\epsilon = 2$ | $\epsilon = 3$ | $\epsilon = 1$ | $\epsilon = 2$ | $\epsilon = 3$ | $\epsilon = 1$ | $\epsilon = 2$ | $\epsilon = 3$ | $\epsilon = 1$ | $\epsilon = 2$ | $\epsilon = 3$ |
| H-NLP | Seq2Sick | 4.31 | 5.84 | 12.28 | 4.83 | 8.85 | 19.55 | 4.84 | 8.85 | 21.47 | 3.73 | 5.90 | 13.24 | 3.77 | 5.96 | 13.33 |
| | SynError | 19.09 | 19.59 | 19.59 | 19.35 | 19.82 | 19.82 | 19.63 | 20.10 | 20.10 | 14.14 | 14.52 | 14.52 | 14.27 | 14.65 | 14.65 |
| | SIT | 11.83 | 5.99 | 5.35 | −1.68 | −8.53 | −11.21 | 8.17 | 6.32 | 7.41 | 9.84 | 5.50 | 5.75 | 9.90 | 5.58 | 5.83 |
| | TransRepair | 0.17 | 0.17 | 0.17 | 0.76 | 0.10 | 0.10 | 0.93 | 0.33 | 0.33 | −0.07 | 0.00 | 0.00 | −0.07 | 0.00 | 0.00 |
| | LLMEffiChecker(C) | 564.45 | 995.45 | 1,357.77 | 764.92 | 1,487.92 | 2,015.70 | 785.60 | 1,471.26 | 1,967.05 | 462.24 | 851.80 | 1,116.80 | 406.39 | 755.18 | 972.92 |
| | LLMEffiChecker(T) | 2,697.77 | 3,735.38 | 3,917.91 | 3,153.97 | 4,481.93 | 4,681.28 | 3,052.62 | 4,544.65 | 4,759.71 | 1,953.57 | 2,729.83 | 2,854.89 | 1,532.91 | 2,137.53 | 2,221.66 |
| | LLMEffiChecker(S) | 142.31 | 311.06 | 612.08 | 146.51 | 451.93 | 877.79 | 147.70 | 461.30 | 870.72 | 101.21 | 275.58 | 523.04 | 95.05 | 259.88 | 508.80 |
| | LLMEffiChecker-B (C) | 907.89 | 1,483.58 | 2,032.41 | 815.15 | 1,561.45 | 2,139.55 | 1,026.17 | 1,948.84 | 2,653.57 | 684.34 | 1,330.43 | 1,823.64 | 681.23 | 1,326.60 | 1,821.57 |
| | LLMEffiChecker-B (T) | 2,556.56 | 3,064.19 | 3,043.95 | 2,557.49 | 3,099.16 | 2,996.62 | 3,106.02 | 3,807.87 | 3,700.27 | 1,968.85 | 2,451.50 | 2,416.83 | 1,965.72 | 2,451.54 | 2,419.52 |
| | LLMEffiChecker-B (S) | 200.15 | 450.31 | 809.79 | 181.05 | 389.76 | 766.62 | 233.63 | 488.25 | 973.90 | 172.32 | 369.99 | 683.68 | 172.01 | 369.75 | 683.28 |
| AllenAI | Seq2Sick | 1.72 | 2.22 | 2.15 | 1.48 | 2.06 | 1.35 | 1.19 | 1.76 | 1.10 | 1.57 | 1.41 | 0.38 | 1.70 | 1.57 | 0.57 |
| | SynError | 0.38 | 0.38 | 0.38 | 1.89 | 1.89 | 1.89 | 1.75 | 1.75 | 1.75 | −0.85 | −0.85 | −0.85 | −0.71 | −0.71 | −0.71 |
| | SIT | 7.06 | 4.12 | 6.67 | 1.73 | −3.24 | −4.64 | 1.73 | −3.24 | −4.60 | 3.95 | 14.25 | −2.05 | 4.12 | 14.64 | −1.60 |
| | TransRepair | 0.08 | 0.08 | 0.08 | −0.37 | −0.37 | −0.37 | −0.55 | −0.55 | −0.55 | −0.15 | −0.15 | −0.15 | −0.14 | −0.14 | −0.14 |
| | LLMEffiChecker(C) | 35.16 | 74.90 | 103.36 | 26.69 | 45.77 | 85.09 | 27.48 | 48.09 | 86.00 | 21.82 | 35.43 | 91.48 | 22.12 | 43.21 | 98.46 |
| | LLMEffiChecker(T) | 24.83 | 42.04 | 56.75 | 49.12 | 62.84 | 67.98 | 49.99 | 62.65 | 69.06 | 30.65 | 41.32 | 46.09 | 31.00 | 41.81 | 49.66 |
| | LLMEffiChecker(S) | 66.21 | 108.67 | 128.60 | 86.05 | 139.03 | 164.57 | 84.17 | 135.71 | 160.95 | 69.57 | 112.88 | 132.68 | 68.79 | 115.23 | 137.06 |
| | LLMEffiChecker-B (C) | 31.78 | 84.30 | 116.75 | 225.80 | 935.84 | 143.91 | 43.25 | 132.30 | 194.46 | 27.64 | 97.85 | 148.22 | 27.41 | 97.47 | 147.72 |
| | LLMEffiChecker-B (T) | 71.37 | 131.54 | 121.88 | 56.40 | 119.95 | 146.99 | 85.25 | 163.81 | 196.13 | 60.21 | 123.44 | 152.83 | 59.93 | 123.10 | 152.54 |
| | LLMEffiChecker-B (S) | 65.82 | 76.05 | 90.41 | 78.24 | 90.00 | 80.83 | 110.20 | 123.05 | 113.93 | 80.90 | 92.03 | 86.14 | 80.57 | 91.73 | 85.87 |
| T5 | Seq2Sick | 7.09 | 6.28 | −6.03 | 7.21 | 6.04 | −5.97 | 8.55 | 6.88 | −5.16 | 9.01 | 8.00 | −3.97 | 8.85 | 16.94 | 4.50 |
| | SynError | 2.18 | 2.18 | 2.18 | 3.20 | 3.20 | 3.20 | 2.11 | 2.11 | 2.11 | 1.02 | 1.02 | 1.02 | 1.13 | 1.13 | 1.13 |
| | SIT | −8.06 | 1.05 | 6.27 | −4.51 | 7.79 | 7.38 | −3.79 | 9.84 | 10.59 | −10.99 | 3.57 | 7.74 | −10.90 | 3.78 | 8.07 |
| | TransRepair | 3.73 | 8.06 | 8.06 | 4.90 | 9.47 | 9.26 | 6.42 | 11.39 | 10.74 | 3.70 | 8.34 | 8.35 | 3.76 | 8.42 | 8.39 |
| | LLMEffiChecker(C) | 168.92 | 198.36 | 205.37 | 191.05 | 225.48 | 233.01 | 194.45 | 228.02 | 234.04 | 164.61 | 194.79 | 202.28 | 165.38 | 195.77 | 203.29 |
| | LLMEffiChecker(T) | 307.27 | 328.94 | 328.94 | 352.14 | 376.55 | 376.55 | 347.74 | 373.85 | 373.85 | 305.37 | 325.61 | 325.61 | 331.85 | 352.25 | 352.25 |
| | LLMEffiChecker(S) | 77.67 | 80.56 | 82.52 | 85.72 | 89.11 | 91.38 | 86.90 | 90.29 | 92.56 | 75.77 | 78.68 | 80.66 | 68.79 | 73.03 | 74.56 |
| | LLMEffiChecker-B (C) | 231.95 | 255.70 | 259.05 | 239.17 | 257.96 | 259.95 | 279.77 | 303.05 | 305.42 | 233.26 | 257.03 | 261.46 | 233.86 | 257.69 | 262.09 |
| | LLMEffiChecker-B (T) | 318.94 | 293.67 | 257.92 | 331.77 | 304.63 | 272.73 | 384.44 | 350.53 | 311.98 | 319.07 | 294.36 | 259.99 | 319.65 | 294.84 | 260.43 |
| | LLMEffiChecker-B (S) | 252.44 | 279.53 | 289.54 | 260.69 | 288.66 | 295.31 | 300.39 | 333.39 | 341.33 | 252.44 | 279.23 | 288.46 | 252.48 | 279.22 | 288.41 |
| U-DL | Seq2Sick | 0.59 | 1.22 | 2.71 | −0.30 | 0.97 | 2.47 | 2.70 | 4.07 | 5.80 | 0.96 | 1.72 | 3.17 | 0.98 | 1.74 | 3.20 |
| | SynError | 0.02 | 0.02 | 0.02 | −0.97 | −0.97 | −0.97 | 2.01 | 2.01 | 2.01 | −0.25 | −0.25 | −0.25 | −0.24 | −0.24 | −0.24 |
| | SIT | 16.73 | 6.40 | 4.26 | 15.50 | 4.15 | 2.45 | 18.90 | 9.37 | 7.56 | 17.16 | 6.30 | 4.81 | 17.18 | 6.33 | 4.82 |
| | TransRepair | 11.36 | 10.66 | 10.82 | 7.53 | 7.09 | 7.09 | 10.07 | 9.94 | 9.95 | 11.46 | 11.45 | 11.59 | 11.43 | 11.41 | 11.55 |
| | LLMEffiChecker(C) | 258.07 | 390.60 | 469.24 | 261.02 | 405.80 | 494.30 | 288.15 | 439.72 | 532.81 | 253.46 | 383.78 | 461.51 | 253.45 | 383.82 | 461.64 |
| | LLMEffiChecker(T) | 604.17 | 642.38 | 642.38 | 655.13 | 696.56 | 696.56 | 697.90 | 741.86 | 741.86 | 595.88 | 634.44 | 634.44 | 596.49 | 635.08 | 635.08 |
| | LLMEffiChecker(S) | 406.92 | 592.52 | 702.89 | 438.23 | 632.64 | 753.88 | 465.04 | 673.76 | 800.25 | 404.42 | 583.65 | 694.40 | 401.74 | 583.99 | 694.84 |
| | LLMEffiChecker-B (C) | 488.81 | 495.81 | 502.85 | 522.33 | 517.47 | 526.17 | 559.45 | 556.72 | 567.10 | 483.54 | 490.87 | 499.09 | 483.52 | 490.88 | 499.15 |
| | LLMEffiChecker-B (T) | 502.85 | 502.85 | 494.97 | 536.77 | 527.40 | 511.22 | 579.50 | 570.67 | 557.33 | 498.65 | 499.99 | 496.64 | 498.56 | 499.90 | 496.46 |
| | LLMEffiChecker-B (S) | 466.97 | 490.19 | 501.44 | 507.14 | 517.67 | 525.86 | 541.40 | 558.12 | 567.17 | 463.23 | 490.47 | 511.64 | 463.31 | 490.69 | 511.80 |
| FairSeq | Seq2Sick | 0.61 | 0.64 | −1.33 | 0.46 | −0.75 | −1.48 | 3.60 | 2.91 | 2.04 | 0.05 | 0.15 | −1.25 | 0.08 | 0.20 | −1.20 |
| | SynError | 0.25 | 0.25 | 0.25 | −2.67 | −2.57 | −2.57 | −0.06 | 0.03 | 0.03 | 0.07 | 0.07 | 0.07 | 0.08 | 0.08 | 0.08 |
| | SIT | −1.15 | −2.15 | −1.56 | −4.46 | −7.26 | −6.73 | 0.13 | −1.90 | −1.63 | −2.34 | −1.51 | −2.37 | −2.37 | −1.52 | −2.37 |
| | TransRepair | 0.26 | 0.25 | 0.25 | 0.02 | 0.01 | 0.05 | 0.67 | 0.66 | 0.69 | −0.03 | −0.01 | −0.01 | −0.03 | 0.00 | 0.00 |
| | LLMEffiChecker(C) | 22.53 | 37.68 | 59.26 | 15.87 | 29.07 | 49.18 | 20.47 | 34.62 | 55.44 | 18.07 | 31.53 | 51.79 | 18.08 | 31.55 | 51.85 |
| | LLMEffiChecker(T) | 33.73 | 62.13 | 76.41 | 23.97 | 55.26 | 70.28 | 29.79 | 63.19 | 79.41 | 28.62 | 59.42 | 75.47 | 28.60 | 59.42 | 75.47 |
| | LLMEffiChecker(S) | 19.42 | 30.87 | 37.82 | 14.01 | 23.67 | 31.31 | 18.23 | 28.59 | 36.73 | 14.84 | 24.87 | 31.68 | 14.86 | 24.91 | 31.72 |
| | LLMEffiChecker-B (C) | 33.09 | 43.60 | 72.18 | 26.76 | 34.61 | 69.78 | 31.94 | 40.82 | 79.01 | 29.29 | 39.99 | 76.06 | 29.36 | 40.19 | 76.36 |
| | LLMEffiChecker-B (T) | 41.71 | 66.82 | 87.19 | 32.97 | 58.12 | 75.42 | 39.15 | 66.41 | 85.81 | 36.04 | 64.10 | 85.87 | 36.10 | 64.18 | 85.95 |
| | LLMEffiChecker-B (S) | 19.04 | 31.29 | 38.51 | 12.24 | 20.73 | 26.06 | 16.94 | 26.55 | 32.40 | 15.71 | 26.25 | 32.80 | 15.72 | 26.29 | 32.85 |
| MarianMT | Seq2Sick | −0.06 | −1.78 | −6.61 | −2.69 | −4.85 | −9.87 | 2.07 | 0.05 | −4.26 | 1.12 | −1.24 | −4.58 | 1.09 | −1.26 | −4.60 |
| | SynError | 3.00 | 3.66 | 3.62 | 1.09 | 1.31 | 1.15 | 3.75 | 4.12 | 3.95 | 1.78 | 2.30 | 2.13 | 1.75 | 2.27 | 2.12 |
| | SIT | 1.94 | −0.27 | −0.82 | −1.91 | −3.95 | −5.03 | 3.95 | 2.00 | 0.71 | 1.23 | −0.59 | −1.64 | 1.20 | −0.59 | −1.62 |
| | TransRepair | 0.03 | 0.95 | 0.68 | −0.89 | −0.65 | −1.06 | 1.50 | 1.77 | 1.36 | 0.06 | 0.24 | −0.02 | 0.05 | 0.23 | −0.03 |
| | LLMEffiChecker(C) | 54.10 | 113.20 | 222.04 | 41.58 | 102.38 | 226.68 | 51.56 | 119.53 | 274.26 | 52.98 | 113.90 | 210.12 | 52.89 | 113.88 | 210.12 |
| | LLMEffiChecker(T) | 231.65 | 550.07 | 726.61 | 234.93 | 564.34 | 770.28 | 269.58 | 660.70 | 893.87 | 223.56 | 544.49 | 728.90 | 223.48 | 544.28 | 728.68 |
| | LLMEffiChecker(S) | 42.77 | 72.19 | 89.17 | 33.89 | 72.33 | 90.33 | 40.27 | 84.89 | 106.84 | 41.21 | 77.28 | 95.68 | 41.19 | 77.21 | 95.61 |
| | LLMEffiChecker-B (C) | 65.70 | 185.56 | 264.37 | 55.72 | 173.09 | 298.05 | 67.01 | 200.74 | 338.54 | 55.62 | 164.33 | 282.05 | 55.46 | 163.84 | 282.20 |
| | LLMEffiChecker-B (T) | 223.05 | 517.87 | 722.71 | 229.07 | 580.01 | 752.90 | 260.24 | 660.71 | 860.89 | 199.48 | 508.85 | 674.54 | 199.44 | 509.79 | 675.87 |
| | LLMEffiChecker-B (S) | 42.43 | 68.25 | 78.04 | 35.36 | 64.01 | 65.18 | 44.70 | 77.00 | 78.26 | 36.10 | 64.71 | 69.38 | 36.06 | 64.67 | 69.41 |
| Flan-T5 | Seq2Sick | 6.64 | 10.31 | 13.09 | 5.97 | 9.68 | 11.76 | 14.05 | 18.79 | 22.16 | 5.76 | 9.35 | 11.96 | 5.74 | 9.34 | 11.92 |
| | SynError | 0.05 | 0.05 | 0.05 | −1.73 | −1.73 | −1.73 | 6.02 | 6.02 | 6.02 | 5.62 | 5.62 | 5.62 | −7.51 | −7.51 | −7.51 |
| | SIT | 87.52 | 43.57 | 42.95 | 86.40 | 40.87 | 40.36 | 85.17 | 59.87 | 55.88 | 82.96 | 43.18 | 44.38 | 83.07 | 43.33 | 44.51 |
| | TransRepair | −1.48 | −1.77 | −1.48 | −1.32 | −1.73 | −1.32 | 1.99 | 1.44 | 1.84 | −1.47 | −1.81 | −1.51 | −1.54 | −1.87 | −1.56 |
| | LLMEffiChecker(C) | 327.55 | 566.82 | 625.69 | 329.99 | 564.27 | 621.78 | 381.37 | 647.48 | 715.92 | 333.91 | 574.26 | 634.28 | 334.03 | 574.53 | 634.51 |
| | LLMEffiChecker(T) | 1,209.50 | 1,306.26 | 1,349.04 | 1,229.54 | 1,327.42 | 1,372.96 | 1,409.23 | 1,524.04 | 1,578.49 | 1,227.99 | 1,325.01 | 1,368.67 | 1,229.06 | 1,326.23 | 1,369.93 |
| | LLMEffiChecker(S) | 554.58 | 937.63 | 1,063.39 | 552.96 | 952.73 | 1,087.15 | 637.50 | 1,094.72 | 1,253.29 | 564.39 | 948.81 | 1,076.25 | 564.90 | 949.32 | 1,076.86 |
| | LLMEffiChecker-B (C) | 421.97 | 628.14 | 850.14 | 426.32 | 629.20 | 845.69 | 495.38 | 726.82 | 980.89 | 426.82 | 637.66 | 897.14 | 426.87 | 638.74 | 868.34 |
| | LLMEffiChecker-B (T) | 1,242.18 | 1,338.54 | 1,341.69 | 1,240.60 | 1,333.61 | 1,322.22 | 1,422.94 | 1,543.97 | 1,533.38 | 1,256.64 | 1,378.93 | 1,389.51 | 1,258.92 | 1,381.33 | 1,391.86 |
| | LLMEffiChecker-B (S) | 572.45 | 892.17 | 1,082.76 | 564.03 | 879.97 | 1,064.21 | 652.28 | 1,015.91 | 1,235.05 | 567.71 | 884.32 | 1,076.55 | 568.44 | 885.57 | 1,077.70 |
| LaMini-GPT | Seq2Sick | 21.91 | 21.81 | 15.24 | 121.46 | 121.66 | 88.75 | 140.43 | 140.48 | 101.57 | 98.56 | 97.73 | 69.70 | 98.85 | 98.03 | 69.97 |
| | SynError | 87.81 | 88.68 | 96.68 | 377.07 | 403.58 | 429.18 | 438.06 | 468.04 | 498.53 | 318.71 | 333.48 | 354.60 | 319.06 | 333.86 | 355.09 |
| | SIT | 153.80 | 152.34 | 137.53 | 867.02 | 916.92 | 747.65 | 1,014.25 | 1,069.92 | 870.79 | 659.99 | 651.73 | 552.49 | 660.31 | 652.99 | 553.72 |
| | TransRepair | 29.39 | 26.51 | 26.74 | 144.03 | 140.82 | 140.73 | 168.75 | 163.40 | 164.34 | 130.07 | 119.77 | 130.57 | 130.42 | 120.23 | 131.11 |
| | LLMEffiChecker(C) | 323.34 | 367.46 | 376.55 | 2,091.80 | 2,643.98 | 2,707.52 | 2,403.58 | 3,098.30 | 3,169.53 | 1,598.12 | 1,997.75 | 2,043.54 | 1,597.50 | 1,995.33 | 2,041.12 |
| | LLMEffiChecker(T) | 368.67 | 379.73 | 379.73 | 2,539.10 | 2,588.35 | 2,588.35 | 3,066.39 | 3,130.40 | 3,130.40 | 2,106.89 | 2,148.49 | 2,148.49 | 2,104.61 | 2,146.16 | 2,146.16 |
| | LLMEffiChecker(S) | 347.41 | 366.07 | 366.42 | 2,157.36 | 2,371.21 | 2,372.00 | 2,510.74 | 2,792.73 | 2,793.60 | 1,746.59 | 1,919.43 | 1,920.28 | 1,747.96 | 1,919.32 | 1,920.17 |
| | LLMEffiChecker-B (C) | 140.67 | 208.14 | 240.05 | 764.30 | 1,109.99 | 1,211.88 | 943.59 | 1,385.40 | 1,525.22 | 733.99 | 1,111.11 | 1,260.97 | 734.45 | 1,112.28 | 1,263.29 |
| | LLMEffiChecker-B (T) | 242.05 | 232.44 | 246.41 | 1,225.62 | 1,129.45 | 1,223.03 | 1,510.56 | 1,401.18 | 1,513.43 | 1,224.47 | 1,143.59 | 1,249.03 | 1,225.86 | 1,145.94 | 1,252.04 |
| | LLMEffiChecker-B (S) | 191.80 | 208.14 | 225.11 | 998.56 | 1,069.81 | 1,127.94 | 1,222.32 | 1,328.29 | 1,407.41 | 962.31 | 1,051.01 | 1,138.46 | 962.76 | 1,052.40 | 1,140.30 |
| CodeGen | Seq2Sick | 5.10 | 5.10 | 5.10 | 13.34 | 13.34 | 13.34 | 16.68 | 16.68 | 16.68 | 17.06 | 17.06 | 17.06 | 17.10 | 17.10 | 17.10 |
| | SynError | 26.48 | 26.49 | 27.54 | 83.33 | 83.26 | 85.38 | 89.39 | 89.34 | 91.54 | 87.61 | 87.54 | 89.78 | 87.65 | 87.58 | 89.81 |
| | SIT | 54.06 | 41.45 | 44.71 | 202.97 | 145.02 | 158.70 | 213.45 | 153.11 | 168.58 | 223.83 | 161.93 | 181.19 | 223.90 | 162.00 | 181.28 |
| | TransRepair | 77.64 | 80.47 | 80.84 | 286.84 | 294.79 | 294.88 | 296.86 | 308.83 | 309.09 | 309.52 | 330.97 | 333.95 | 309.68 | 331.24 | 334.22 |
| | LLMEffiChecker(C) | 109.93 | 139.88 | 168.23 | 321.08 | 434.06 | 533.72 | 336.01 | 453.64 | 558.45 | 351.20 | 478.08 | 592.39 | 351.46 | 478.36 | 592.75 |
| | LLMEffiChecker(T) | 182.42 | 182.42 | 182.42 | 578.30 | 578.30 | 578.30 | 602.68 | 602.68 | 602.68 | 639.91 | 639.91 | 639.91 | 640.26 | 640.26 | 640.26 |
| | LLMEffiChecker(S) | 176.30 | 187.59 | 187.59 | 575.42 | 615.61 | 615.61 | 593.11 | 635.62 | 635.62 | 607.64 | 653.51 | 653.51 | 607.97 | 653.91 | 653.91 |
| | LLMEffiChecker-B (C) | 147.75 | 170.76 | 175.70 | 463.20 | 518.89 | 523.96 | 481.57 | 540.51 | 549.87 | 501.99 | 582.24 | 603.35 | 501.99 | 582.47 | 603.57 |
| | LLMEffiChecker-B (T) | 118.40 | 154.85 | 146.01 | 378.35 | 496.82 | 451.94 | 394.97 | 517.96 | 475.82 | 405.33 | 551.65 | 528.71 | 405.63 | 552.12 | 529.16 |
| | LLMEffiChecker-B (S) | 152.02 | 166.56 | 158.82 | 488.73 | 523.34 | 494.20 | 507.41 | 547.59 | 519.25 | 521.24 | 578.62 | 563.35 | 521.44 | 578.88 | 563.62 |

Table 6. The Maximum Effectiveness Results of LLMEffiChecker in Degrading LLMs Performance

| Subject | Methods | I-Loops | | | I-Latency(CPU) | | | I-Energy(CPU) | | | I-Latency(GPU) | | | I-Energy(GPU) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $\epsilon = 1$ | $\epsilon = 2$ | $\epsilon = 3$ | $\epsilon = 1$ | $\epsilon = 2$ | $\epsilon = 3$ | $\epsilon = 1$ | $\epsilon = 2$ | $\epsilon = 3$ | $\epsilon = 1$ | $\epsilon = 2$ | $\epsilon = 3$ | $\epsilon = 1$ | $\epsilon = 2$ | $\epsilon = 3$ |
| H-NLP | **Seq2Sick** | 1,922.22 | 1,922.22 | 2,000.00 | 2,557.01 | 2,557.01 | 2,557.01 | 3,081.88 | 3,081.88 | 3,081.88 | 2,031.63 | 2,031.63 | 2,031.63 | 2,035.46 | 2,035.46 | 2,035.46 |
| | **SynError** | 975.00 | 975.00 | 975.00 | 398.24 | 398.24 | 398.24 | 492.18 | 492.18 | 492.18 | 397.80 | 397.80 | 397.80 | 396.97 | 396.97 | 396.97 |
| | **SIT** | 1,416.67 | 1,666.67 | 540.00 | 2,100.68 | 1,383.50 | 361.29 | 2,541.75 | 1,675.15 | 474.76 | 1,876.29 | 1,251.05 | 388.86 | 1,872.66 | 1,249.04 | 387.36 |
| | **TransRepair** | 66.67 | 81.56 | 81.56 | 36.92 | 32.87 | 86.91 | 53.98 | 57.88 | 120.45 | 45.82 | 52.88 | 69.05 | 45.78 | 52.57 | 68.90 |
| | **LLMEffiChecker(C)** | 8,433.33 | 8,433.33 | 8,433.33 | 7,969.98 | 7,969.98 | 7,969.98 | 10,554.40 | 10,554.40 | 10,554.40 | 7,691.45 | 7,691.45 | 7,691.45 | 7,673.37 | 7,673.37 | 7,673.37 |
| | **LLMEffiChecker(T)** | 12,700.00 | 12,700.00 | 12,700.00 | 8,258.55 | 8,258.55 | 8,258.55 | 10,486.75 | 10,486.75 | 10,486.75 | 5,990.07 | 5,990.07 | 5,990.07 | 5,984.31 | 5,984.31 | 5,984.31 |
| | **LLMEffiChecker(S)** | 3,100.00 | 6,300.00 | 8,433.33 | 4,162.90 | 6,704.67 | 7,357.19 | 5,193.61 | 8,286.11 | 8,810.90 | 2,945.15 | 5,305.42 | 5,305.42 | 2,944.76 | 5,292.81 | 5,292.81 |
| | **LLMEffiChecker-B (C)** | 10,140.00 | 12,700.00 | 10,140.00 | 8,456.70 | 8,220.03 | 8,240.21 | 11,418.85 | 11,059.20 | 11,091.90 | 6,244.01 | 6,231.62 | 7,711.93 | 6,180.36 | 6,214.44 | 7,697.90 |
| | **LLMEffiChecker-B (T)** | 10,140.00 | 10,140.00 | 10,140.00 | 9,579.03 | 9,388.97 | 9,286.91 | 12,430.16 | 12,355.92 | 12,241.07 | 6,483.82 | 6,480.49 | 6,717.57 | 6,483.59 | 6,487.76 | 6,700.88 |
| | **LLMEffiChecker-B (S)** | 3,100.00 | 6,300.00 | 12,700.00 | 3,599.50 | 5,417.91 | 9,132.01 | 4,452.36 | 6,740.22 | 12,109.99 | 3,513.36 | 4,508.14 | 6,781.85 | 3,507.85 | 4,512.49 | 6,767.54 |
| AllenAI | **Seq2Sick** | 33.33 | 33.33 | 45.16 | 17.98 | 20.79 | 28.50 | 46.67 | 50.97 | 60.87 | 13.57 | 15.48 | 41.13 | 13.28 | 15.44 | 41.08 |
| | **SynError** | 12.81 | 12.81 | 12.81 | 9.89 | 9.89 | 9.89 | 31.33 | 31.33 | 31.33 | 12.27 | 12.27 | 12.27 | 12.21 | 12.21 | 12.21 |
| | **SIT** | 23.81 | 28.57 | 28.57 | 34.08 | 26.92 | 16.56 | 63.45 | 46.50 | 43.62 | 25.41 | 15.28 | 13.28 | 25.3 | 15.22 | 13.24 |
| | **TransRepair** | 14.29 | 14.29 | 14.29 | 4.34 | 4.34 | 4.34 | 27.52 | 27.52 | 27.52 | 8.95 | 8.95 | 8.95 | 8.91 | 8.91 | 8.91 |
| | **LLMEffiChecker(C)** | 93.33 | 247.37 | 440.54 | 128.38 | 140.21 | 332.09 | 165.37 | 194.69 | 442.04 | 172.72 | 202.17 | 368.06 | 172.1 | 201.49 | 368.99 |
| | **LLMEffiChecker(T)** | 173.33 | 285.71 | 683.33 | 83.94 | 266.41 | 611.60 | 119.43 | 336.99 | 789.79 | 90.91 | 283.61 | 602.78 | 90.85 | 282.94 | 603.10 |
| | **LLMEffiChecker(S)** | 175.00 | 414.29 | 589.66 | 142.19 | 292.91 | 573.46 | 182.65 | 386.68 | 703.88 | 160.93 | 377.36 | 538.89 | 161.42 | 377.22 | 539.05 |
| | **LLMEffiChecker-B (C)** | 157.58 | 852.38 | 852.38 | 184.80 | 1,021.41 | 1,021.18 | 269.93 | 1,357.36 | 1,349.91 | 212.59 | 1,052.79 | 1,023.75 | 211.45 | 1,047.56 | 1,018.38 |
| | **LLMEffiChecker-B (T)** | 1,233.33 | 1,233.33 | 900.00 | 1,032.59 | 996.48 | 846.78 | 1,394.01 | 1,356.31 | 1,130.11 | 1,109.14 | 1,074.40 | 798.5 | 1,100.56 | 1,066.07 | 796.15 |
| | **LLMEffiChecker-B (S)** | 761.54 | 1,233.33 | 1,186.67 | 920.27 | 756.81 | 729.93 | 1,193.81 | 958.69 | 954.42 | 891.05 | 851.72 | 852.57 | 889.57 | 848.38 | 849.46 |
| T5 | **Seq2Sick** | 90.91 | 121.15 | 247.62 | 87.43 | 130.42 | 252.47 | 127.68 | 156.29 | 294.40 | 95.72 | 118.58 | 248.67 | 95.87 | 118.50 | 249.29 |
| | **SynError** | 61.54 | 61.54 | 61.54 | 63.81 | 63.81 | 63.81 | 70.27 | 70.27 | 70.27 | 61.74 | 61.74 | 61.74 | 61.75 | 61.75 | 61.75 |
| | **SIT** | 945.45 | 576.47 | 576.47 | 1,063.35 | 565.74 | 608.96 | 1,323.72 | 666.54 | 714.72 | 961.39 | 570.17 | 567.80 | 963.22 | 571.87 | 568.55 |
| | **TransRepair** | 422.73 | 422.73 | 422.73 | 439.91 | 439.91 | 439.91 | 493.85 | 493.85 | 493.85 | 425.92 | 425.92 | 425.92 | 425.95 | 425.95 | 425.95 |
| | **LLMEffiChecker(C)** | 945.45 | 945.45 | 945.45 | 1,470.96 | 1,470.96 | 1,470.96 | 1,751.91 | 1,751.91 | 1,751.91 | 885.92 | 885.92 | 885.92 | 879.21 | 879.21 | 879.21 |
| | **LLMEffiChecker(T)** | 1,816.67 | 1,816.67 | 1,816.67 | 1,829.31 | 1,829.31 | 1,829.31 | 2,443.06 | 2,443.06 | 2,443.06 | 1,796.62 | 1,796.62 | 1,796.62 | 1,798.34 | 1,798.34 | 1,798.34 |
| | **LLMEffiChecker(S)** | 945.45 | 945.45 | 945.45 | 1,010.14 | 1,010.14 | 1,010.14 | 1,271.41 | 1,271.41 | 1,271.41 | 963.27 | 963.27 | 963.27 | 964.11 | 964.11 | 964.11 |
| | **LLMEffiChecker-B (C)** | 1,816.67 | 1,816.67 | 1,816.67 | 1,898.90 | 1,837.56 | 1,920.93 | 2,500.77 | 2,524.75 | 2,598.89 | 1,835.70 | 1,831.20 | 1,824.22 | 1,840.98 | 1,836.47 | 1,828.48 |
| | **LLMEffiChecker-B (T)** | 1,816.67 | 945.45 | 784.62 | 1,892.24 | 1,000.72 | 866.48 | 2,569.88 | 1,272.44 | 1,060.74 | 1,783.49 | 962.94 | 785.66 | 1,786.73 | 964.82 | 787.57 |
| | **LLMEffiChecker-B (S)** | 945.45 | 945.45 | 945.45 | 992.67 | 1,001.37 | 992.03 | 1,259.61 | 1,276.53 | 1,264.36 | 958.68 | 960.53 | 959.96 | 960.48 | 961.78 | 961.22 |
| U-DL | **Seq2Sick** | 27.03 | 36.84 | 75.76 | 27.75 | 52.99 | 90.40 | 31.39 | 56.60 | 91.85 | 28.21 | 39.38 | 74.23 | 28.30 | 39.58 | 74.31 |
| | **SynError** | 15.79 | 15.79 | 15.79 | 12.12 | 12.12 | 12.12 | 17.72 | 17.72 | 17.72 | 16.95 | 16.95 | 16.95 | 16.85 | 16.85 | 16.85 |
| | **SIT** | 545.16 | 327.27 | 327.27 | 579.53 | 361.57 | 379.32 | 416.79 | 377.14 | 393.91 | 553.80 | 323.51 | 323.51 | 553.80 | 322.16 | 322.16 |
| | **TransRepair** | 850.00 | 850.00 | 850.00 | 523.63 | 566.02 | 566.02 | 612.79 | 686.76 | 686.76 | 855.37 | 937.09 | 937.09 | 852.58 | 934.06 | 934.06 |
| | **LLMEffiChecker(C)** | 4,900.00 | 4,900.00 | 4,900.00 | 4,065.42 | 4,065.42 | 4,065.42 | 5041.52 | 5041.52 | 5,041.52 | 4,692.21 | 4,692.21 | 4,692.21 | 4,663.56 | 4,663.56 | 4,663.56 |
| | **LLMEffiChecker(T)** | 4,900.00 | 4,900.00 | 4,900.00 | 3,978.94 | 3,978.94 | 3,978.94 | 4,831.58 | 4,831.58 | 4,831.58 | 4,448.49 | 4,448.49 | 4,448.49 | 4,444.52 | 4,444.52 | 4,444.52 |
| | **LLMEffiChecker(S)** | 1,718.18 | 4,900.00 | 4,900.00 | 1,984.05 | 3,844.74 | 3,844.74 | 2,176.62 | 4,694.89 | 4,694.89 | 1,720.49 | 4,566.83 | 4,566.83 | 1,722.57 | 4557.84 | 4,557.84 |
| | **LLMEffiChecker-B (C)** | 4,900.00 | 4,900.00 | 4,900.00 | 4,976.52 | 4,979.55 | 4,902.19 | 6,293.19 | 6,183.34 | 6,153.30 | 4,889.23 | 4,884.94 | 4,823.49 | 4,826.34 | 4,893.77 | 4,916.49 |
| | **LLMEffiChecker-B (T)** | 4,900.00 | 4,900.00 | 4,900.00 | 4,961.40 | 4,942.95 | 4,899.74 | 6,161.10 | 5,931.57 | 6,154.24 | 4,858.21 | 4,884.94 | 4,887.12 | 4,820.22 | 4,847.27 | 4,848.92 |
| | **LLMEffiChecker-B (S)** | 4,900.00 | 4,900.00 | 4,900.00 | 5,120.57 | 4,991.13 | 4,945.80 | 6,359.07 | 6,255.39 | 6,063.02 | 4,825.21 | 4,876.52 | 4,952.21 | 4,819.50 | 4,878.73 | 4,952.97 |
| FairSeq | **Seq2Sick** | 39.13 | 39.13 | 39.13 | 65.76 | 65.76 | 65.76 | 68.53 | 68.53 | 68.53 | 42.23 | 42.23 | 42.23 | 42.19 | 42.19 | 42.19 |
| | **SynError** | 10.00 | 10.00 | 10.00 | 19.33 | 19.33 | 19.33 | 15.98 | 15.98 | 15.98 | 12.53 | 12.53 | 12.53 | 12.67 | 12.67 | 12.67 |
| | **SIT** | 37.50 | 37.50 | 37.50 | 64.17 | 36.24 | 49.49 | 73.91 | 40.20 | 50.67 | 20.97 | 27.18 | 21.13 | 20.90 | 26.82 | 21.10 |
| | **TransRepair** | 13.16 | 13.16 | 13.16 | 22.00 | 22.00 | 22.00 | 18.89 | 18.89 | 18.89 | 7.36 | 7.27 | 7.27 | 7.36 | 7.25 | 7.25 |
| | **LLMEffiChecker(C)** | 80.00 | 150.00 | 589.66 | 75.28 | 89.34 | 371.87 | 91.66 | 101.22 | 402.55 | 81.85 | 90.86 | 403.03 | 81.70 | 90.73 | 404.14 |
| | **LLMEffiChecker(T)** | 769.57 | 1,011.11 | 1,011.11 | 576.82 | 1,071.13 | 1,071.13 | 661.12 | 1,157.19 | 1,157.19 | 713.87 | 1,030.73 | 1,030.73 | 713.45 | 1,030.07 | 1,030.07 |
| | **LLMEffiChecker(S)** | 62.50 | 100.00 | 125.00 | 51.89 | 78.08 | 86.72 | 59.25 | 87.21 | 107.37 | 58.85 | 76.68 | 100.38 | 58.56 | 76.63 | 100.20 |
| | **LLMEffiChecker-B (C)** | 769.57 | 386.96 | 818.75 | 739.45 | 469.90 | 1094.49 | 796.85 | 513.22 | 1179.19 | 777.23 | 542.11 | 1007.78 | 777.18 | 542.04 | 1012.06 |
| | **LLMEffiChecker-B (T)** | 769.57 | 1,011.11 | 1,328.57 | 751.56 | 1,074.00 | 1,322.30 | 836.38 | 1,140.75 | 1,434.50 | 731.03 | 1,005.78 | 1,230.04 | 733.89 | 1,004.80 | 1,229.26 |
| | **LLMEffiChecker-B (S)** | 62.50 | 100.00 | 125.00 | 55.40 | 74.50 | 92.28 | 62.82 | 80.25 | 103.05 | 56.03 | 75.48 | 84.32 | 56.00 | 75.43 | 84.30 |
| MarianMT | **Seq2Sick** | 57.69 | 56.52 | 36.36 | 56.00 | 51.82 | 39.27 | 76.41 | 65.69 | 48.38 | 74.30 | 74.30 | 87.19 | 74.06 | 74.06 | 87.02 |
| | **SynError** | 44.44 | 66.67 | 71.05 | 40.48 | 67.97 | 70.53 | 51.09 | 79.28 | 79.28 | 43.19 | 63.05 | 63.96 | 43.18 | 63.04 | 64.56 |
| | **SIT** | 77.78 | 65.00 | 60.87 | 54.35 | 36.79 | 53.29 | 61.71 | 50.24 | 62.93 | 50.84 | 32.85 | 59.25 | 50.72 | 32.84 | 59.20 |
| | **TransRepair** | 22.22 | 35.71 | 34.78 | 34.95 | 32.78 | 28.83 | 40.93 | 40.95 | 40.95 | 43.33 | 33.71 | 32.67 | 43.29 | 33.71 | 32.48 |
| | **LLMEffiChecker(C)** | 200.00 | 1,869.23 | 6,300.00 | 228.97 | 2,323.96 | 7,306.69 | 276.90 | 2,699.20 | 9,529.47 | 256.83 | 1,664.78 | 4,687.01 | 256.63 | 1,669.21 | 4,685.35 |
| | **LLMEffiChecker(T)** | 3,557.14 | 6,300.00 | 6,300.00 | 3,877.44 | 5,235.32 | 5,235.32 | 4,489.69 | 6,768.05 | 6,768.05 | 3,459.49 | 5,290.32 | 5,290.32 | 3,452.27 | 5,280.51 | 5,280.51 |
| | **LLMEffiChecker(S)** | 757.89 | 926.67 | 926.67 | 490.25 | 1,673.06 | 1,673.06 | 429.09 | 1,945.97 | 1,945.97 | 561.70 | 1,336.00 | 1,336.00 | 562.66 | 1,336.00 | 1,336.00 |
| | **LLMEffiChecker-B (C)** | 225.00 | 4,166.67 | 2,594.74 | 247.93 | 3,995.95 | 2,619.07 | 290.05 | 4,728.15 | 3,003.46 | 232.88 | 3,211.10 | 2,563.35 | 230.91 | 3,196.36 | 2,559.47 |
| | **LLMEffiChecker-B (T)** | 3,313.33 | 4,554.55 | 4,554.55 | 3,292.26 | 4,630.19 | 4,953.01 | 3,767.91 | 5,451.64 | 6,056.81 | 2,905.84 | 4,062.96 | 4,062.26 | 2,905.01 | 4,072.67 | 4,073.25 |
| | **LLMEffiChecker-B (S)** | 757.89 | 926.67 | 752.63 | 728.78 | 1,593.17 | 710.01 | 812.40 | 1,798.47 | 786.78 | 599.04 | 1,379.27 | 618.82 | 597.73 | 1,376.98 | 621.54 |
| Flan-T5 | **Seq2Sick** | 283.33 | 283.33 | 283.33 | 250.63 | 250.63 | 250.63 | 299.03 | 299.03 | 299.03 | 274.07 | 274.07 | 274.07 | 275.03 | 275.03 | 275.03 |
| | **SynError** | 18.18 | 18.18 | 18.18 | 19.95 | 19.95 | 19.95 | 30.01 | 30.01 | 30.01 | 31.38 | 31.38 | 31.38 | 31.65 | 31.65 | 31.65 |
| | **SIT** | 1328.57 | 1,233.33 | 1,233.33 | 1,288.39 | 1,197.20 | 1,271.50 | 1,440.45 | 1,344.44 | 1,443.66 | 1,433.72 | 1,255.22 | 1,254.68 | 1,437.51 | 1,257.34 | 1,256.80 |
| | **TransRepair** | 108.33 | 108.33 | 108.33 | 70.30 | 89.02 | 89.02 | 81.55 | 94.70 | 94.70 | 106.82 | 106.82 | 106.82 | 105.81 | 105.81 | 105.81 |
| | **LLMEffiChecker(C)** | 2,400.00 | 2,400.00 | 2,400.00 | 2,454.15 | 2,454.15 | 2,454.15 | 2,989.28 | 2,989.28 | 2,989.28 | 2,508.12 | 2,508.12 | 2,508.12 | 2,507.51 | 2,507.51 | 2,507.51 |
| | **LLMEffiChecker(T)** | 2,400.00 | 2,400.00 | 2,757.14 | 2,481.39 | 2,481.39 | 2,904.22 | 3,086.71 | 3,086.71 | 3,691.62 | 2,477.39 | 2,477.39 | 2,826.32 | 2,482.17 | 2,482.17 | 2,830.94 |
| | **LLMEffiChecker(S)** | 2,400.00 | 2,400.00 | 2,400.00 | 2,396.28 | 2,917.05 | 2,917.05 | 2,998.45 | 3,444.31 | 3,444.31 | 2,496.46 | 2,496.46 | 2,496.46 | 2,498.29 | 2,498.29 | 2,498.29 |
| | **LLMEffiChecker-B (C)** | 2,400.00 | 2,400.00 | 2,400.00 | 2,138.76 | 2,158.41 | 2,369.58 | 2,561.77 | 2,596.17 | 2,972.43 | 2,404.92 | 2,393.24 | 2,778.14 | 2,405.80 | 2,403.45 | 2,776.39 |
| | **LLMEffiChecker-B (T)** | 2,400.00 | 2,400.00 | 2,757.14 | 2,369.71 | 2,661.01 | 2,710.52 | 2,950.62 | 3,290.37 | 3,424.51 | 2,569.68 | 3,145.83 | 3,212.41 | 2,572.49 | 3,151.03 | 3,210.04 |
| | **LLMEffiChecker-B (S)** | 2,400.00 | 2,400.00 | 2,400.00 | 2,416.88 | 2,480.45 | 2,538.83 | 3,006.46 | 3,072.52 | 3,167.72 | 2,440.13 | 2,450.42 | 2,442.96 | 2,444.45 | 2,454.10 | 2,447.29 |
| LaMini-GPT | **Seq2Sick** | 669.23 | 669.23 | 426.32 | 3,856.26 | 3,856.26 | 3,856.26 | 4,367.19 | 4,367.19 | 4,367.19 | 3,019.68 | 3,019.68 | 3,019.68 | 3,028.41 | 3,028.41 | 3,028.41 |
| | **SynError** | 769.57 | 769.57 | 769.57 | 3,478.30 | 3,718.72 | 3,718.72 | 4,012.68 | 4,211.54 | 4,211.54 | 2,824.70 | 2,824.70 | 2,824.70 | 2,822.29 | 2,822.29 | 2,822.29 |
| | **SIT** | 952.63 | 952.63 | 952.63 | 6,166.94 | 7,827.18 | 7,934.35 | 8,319.36 | 9,972.64 | 10,092.55 | 5,200.85 | 5,214.04 | 5,231.60 | 5,181.36 | ,5206.77 | 5,224.30 |
| | **TransRepair** | 525.00 | 525.00 | 525.00 | 3,933.96 | 3,933.96 | 3,933.96 | 4,394.69 | 4,394.69 | 4,394.69 | 3,053.31 | 3,053.31 | 3,053.31 | 3,072.67 | 3,072.67 | 3,072.67 |
| | **LLMEffiChecker(C)** | 952.63 | 1,076.47 | 1,076.47 | 8,352.47 | 18,318.66 | 18,318.66 | 10,466.74 | 25,835.15 | 25,835.15 | 6,769.64 | 15,538.18 | 15,538.18 | 6,752.79 | 15,377.14 | 15,377.14 |
| | **LLMEffiChecker(T)** | 952.63 | 1,076.47 | 1,076.47 | 18,081.36 | 18,081.36 | 18,081.36 | 25,469.76 | 25,469.76 | 25,469.76 | 15,373.20 | 15,373.20 | 15,373.20 | 15,204.52 | 15,204.52 | 15,204.52 |
| | **LLMEffiChecker(S)** | 852.38 | 852.38 | 852.38 | 12,388.72 | 14,396.41 | 14,396.41 | 14,680.90 | 20,952.22 | 20,952.22 | 6,434.13 | 14,818.49 | 14,818.49 | 6,432.25 | 14,664.10 | 14,664.10 |
| | **LLMEffiChecker-B (C)** | 471.43 | 525.00 | 525.00 | 3,720.21 | 4,762.53 | 4,658.27 | 4,554.85 | 6,175.32 | 6,084.00 | 3,267.67 | 5,363.85 | 5,350.09 | 3,267.87 | 5,345.96 | 5,338.68 |
| | **LLMEffiChecker-B (T)** | 525.00 | 525.00 | 525.00 | 4,582.59 | 4,235.27 | 5,073.44 | 6,016.12 | 5,630.82 | 6,404.08 | 5,302.61 | 4,055.18 | 5,344.19 | 5,280.60 | 4,053.63 | 5,337.63 |
| | **LLMEffiChecker-B (S)** | 669.23 | 669.23 | 669.23 | 4,243.52 | 4,533.69 | 4,061.00 | 5,612.34 | 6,003.08 | 5,566.46 | 4,338.83 | 4,335.70 | 4,354.52 | 4,330.33 | 4,327.21 | 4345.99 |
| CodeGen | **Seq2Sick** | 189.86 | 189.86 | 189.86 | 583.95 | 583.95 | 583.95 | 610.69 | 610.69 | 610.69 | 540.78 | 540.78 | 540.78 | 541.35 | 541.35 | 541.35 |
| | **SynError** | 334.78 | 334.78 | 334.78 | 1,492.46 | 1,492.46 | 1,492.46 | 1,582.08 | 1,582.08 | 1,582.08 | 1,571.08 | 1,571.08 | 1,571.08 | 1,568.21 | 1,568.21 | 1,568.21 |
| | **SIT** | 334.78 | 334.78 | 334.78 | 1,732.38 | 1,558.46 | 1,717.67 | 1,840.58 | 1,649.79 | 1,828.54 | 1,854.63 | 1,788.77 | 2,056.57 | 1,852.92 | 1,789.52 | 2,059.23 |
| | **TransRepair** | 334.78 | 334.78 | 334.78 | 1,575.88 | 1,577.57 | 1,577.57 | 1,659.61 | 1,666.67 | 1,666.67 | 1,860.50 | 1,860.50 | 1,860.50 | 1,860.02 | 1,860.02 | 1,860.02 |
| | **LLMEffiChecker(C)** | 354.55 | 354.55 | 354.55 | 1,742.25 | 1,739.49 | 1,685.36 | 1,843.62 | 1,841.95 | 1,843.32 | 1815.81 | 1,815.81 | 1,815.81 | 1,827.49 | 1,827.49 | 1,827.49 |
| | **LLMEffiChecker(T)** | 354.55 | 354.55 | 354.55 | 1,736.78 | 1,736.78 | 1,736.78 | 1,843.88 | 1,843.88 | 1,843.88 | 1,820.21 | 1,820.21 | 1,820.21 | 1,829.98 | 1,829.98 | 1,829.98 |
| | **LLMEffiChecker(S)** | 354.55 | 354.55 | 354.55 | 1,745.30 | 1,745.30 | 1,745.30 | 1,843.20 | 1,843.20 | 1,843.20 | 1,843.31 | 1,843.31 | 1,843.31 | 1,845.11 | 1,845.11 | 1,845.11 |
| | **LLMEffiChecker-B (C)** | 354.55 | 354.55 | 354.55 | 1,737.79 | 1,727.37 | 1,716.29 | 1,833.57 | 1,829.78 | 1,823.24 | 1,761.06 | 1,764.48 | 1,759.02 | 1,762.84 | 1,772.37 | 1,760.79 |
| | **LLMEffiChecker-B (T)** | 354.55 | 325.53 | 354.55 | 1,738.85 | 1,724.79 | 1,743.39 | 1,839.45 | 1,829.15 | 1,855.67 | 1,891.48 | 1,904.63 | 1,799.27 | 1,894.36 | 1,906.88 | 1,807.42 |
| | **LLMEffiChecker-B (S)** | 354.55 | 354.55 | 354.55 | 1,749.17 | 1,740.43 | 1,652.96 | 1,844.60 | 18,44.65 | 1,765.17 | 1,765.54 | 1,761.92 | 1,765.10 | 1,772.19 | 1,768.46 | 1,771.65 |

Table 7. The Examples of Test Samples Generated by LLMEffiChecker

| Subject | Methods | Test samples |
|---|---|---|
| FairSeq | **Original** | women over 55 are pickier about their partners than at any other time in their lives. |
| | **LLMEffiChecker (C)** | women over 55 are 5pickier about their partners than at any other time in their lives. |
| | **LLMEffiChecker (T)** | women dinger 55 are pickier about their partners than at any other time in their lives. |
| | **LLMEffiChecker (S)** | women over 55 are pickier because their partners than at any other time in their lives. |
| | **LLMEffiChecker-B (C)** | women Gover 55 are pickier about their partners than at any other time in their lives. |
| | **LLMEffiChecker-B (T)** | structures over 55 are pickier about their partners than at any other time in their lives. |
| | **LLMEffiChecker-B (S)** | research over 55 are pickier about their partners than at any other time in their lives. |
| Flan-T5 | **Original** | A woman is sitting at a table in a fast food restaurant while eating. She continually speaks to nobody as she eats. she |
| | **LLMEffiChecker (C)** | A woman is sitting at a table in a fast food restaurant while _eating. She continually speaks to nobody as she eats. she |
| | **LLMEffiChecker (T)** | authorities woman is sitting at a table in a fast food restaurant while eating. She continually speaks to nobody as she eats. she |
| | **LLMEffiChecker (S)** | A woman is sitting at a table in a fast food restaurant while eating. She continually speaks to nobody as she eats. It |
| | **LLMEffiChecker-B (C)** | A woman is sitting at a table in a fast food restaurant while eating. She continually speaks to nobody as szhe eats. she |
| | **LLMEffiChecker-B (T)** | exhibitors woman is sitting at a table in a fast food restaurant while eating. She continually speaks to nobody as she eats. she |
| | **LLMEffiChecker-B (S)** | A woman is sitting at a table in a fast food restaurant while eating. She continually speaks to nobody as she eats. It |
| CodeGen | **Original** | def sum_div(number: int) -> int:\n """\n\tWrite a function to return the sum of all divisors of a number.\n\t"""\n |
| | **LLMEffiChecker (C)** | degf sum_div(number: int) -> int:\n """\n\tWrite a function to return the sum of all divisors of a number.\n\t"""\n |
| | **LLMEffiChecker (T)** | def umption_div(number: int) -> int:\n """\n\tWrite a function to return the sum of all divisors of a number.\n\t"""\n |
| | **LLMEffiChecker (S)** | def sum_div(number: int) -> int:\n """\n\tWrite a function to return the sum until all divisors of a number.\n\t"""\n |
| | **LLMEffiChecker-B (C)** | def sum_div(number: int) -> int:\n 1"""\n\tWrite a function to return the sum of all divisors of a number.\n\t"""\n |
| | **LLMEffiChecker-B (T)** | Huge sum_div(number: int) -> int:\n """\n\tWrite a function to return the sum of all divisors of a number.\n\t"""\n |
| | **LLMEffiChecker-B (S)** | def sum_div(number: int) -> int:\n """\n\tWrite a function to return the death of all divisors of a number.\n\t"""\n |

Furthermore, Table 7 showcases examples of test samples generated by LLMEffiChecker, with Original denoting seed sentences and red font highlighting perturbed segments. To elaborate, LLMEffiChecker(C), LLMEffiChecker(T), LLMEffiChecker(S) denote character-level, token-level, and structure-level perturbations in white-box settings, respectively. Similarly, LLMEffiChecker-B (C), LLMEffiChecker-B (T), LLMEffiChecker-B (S) represent character-level, token-level, and structure-level perturbations in black-box settings. From the results, we have the following observations: *(i)* For all LLMs under test, LLMEffiChecker generates test samples that trigger more severe efficiency degradation by a large margin compared to the baseline methods. Specifically, LLMEffiChecker generates test inputs that on average increase LLMs for translation (i.e., the first six models') CPU latency, CPU energy consumption, GPU latency, and GPU energy consumption by 100% to 776%, 101% to 768%, 96% to 537%, and 82% to 539%, respectively, through only perturbing one character or token in any seed input sentences. Correspondingly, the LLMs for the sentence completion task (i.e., Flan-T5 and LaMini-GPT) can increase 547% to 1,890%, 662% to 2,245%, 534% to 1,682%, 534% to 1,682%, respectively. For code-generated LLMs (i.e., CodeGen), the increases are 321% to 578%, 336% to 603%, 351% to 640%, and 351% to 640%. Notably, LLMEffiChecker-B demonstrates performance comparable to LLMEffiChecker, signifying LLMEffiChecker-B equally effectively influences the efficiency of LLMs. In addition, LLMEffiChecker-B proves more effective than LLMEffiChecker in character type perturbations (i.e., +32.49%). This indicates our success in finding critical tokens within the black-box scenario presented in Section 5.3. However, baseline methods could not effectively impact efficiency, since they are designed to reduce LLMs' accuracy, not efficiency. *(ii)* With an increased perturbation size, the corresponding test samples generated by LLMEffiChecker effectively degrade LLMs' efficiency to a larger degree. *(iii)* The maximum effectiveness of our methods is far greater than the average case for most scenarios. Additionally, the computational efficiency of LLMs can be dramatically compromised through specific perturbations (e.g., employing the LLMEffiChecker-B (C) on the H-NLP model, where a single character perturbation can lead to a maximum increase of 11,418% in CPU energy consumption).

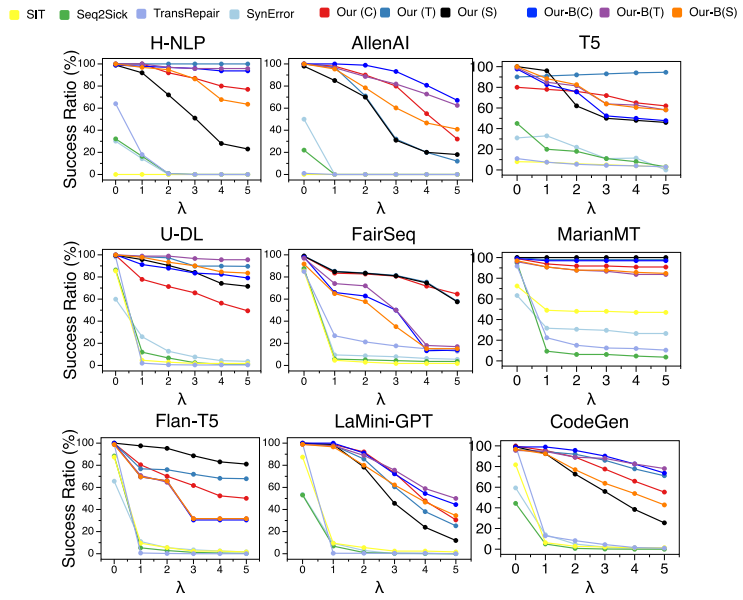



Fig. 6. Degradation success ratio under different settings.

Answers to **RQ2.1**: Test samples generated by LLMEffiChecker in both white-box and black-box settings significantly degrade LLMs efficiency in number of iteration loops, latency, and energy consumption.

### 6.3 RQ2.2: Effectiveness

This section evaluates the effectiveness of LLMEffiChecker in generating useful test samples that successfully degrade the efficiency of LLMs.

**Metrics.** We define a metric of degradation success ratio ($\eta$) to evaluate the effectiveness of LLMEffiChecker.

$$\eta = \frac{\sum_{x \in \mathcal{X}} \mathbb{I}([\text{Loop}(x') - \text{Loop}(x)] \geq \lambda \times \text{MSE}_{orig})}{||\mathcal{X}||} \times 100\% \tag{6}$$

As shown in Equation (6), $\mathcal{X}$ is a randomly selected seed input set, Loop($x$) is the function that measures the iteration number of LLMs in handling input $x$, $\text{MSE}_{orig}$ is the Mean Squared Error of the iteration number in the seed datasets that have the same input length as $x$, and $\mathbb{I}(\cdot)$ is the indicator function, which returns one if the statement is true, zero otherwise. The above equation assumes that the computational costs required by an LLM given perturbed inputs shall be within $\lambda$ times the MSE produced by the seed inputs with the same input length. Otherwise, the perturbed inputs trigger efficiency degradation. Note that this same assumption is also used in existing works [71].

**Results.** The results on the degradation successful ratio ($\eta$) under different $\lambda$ values are shown in Figure 6. We observe that for all experimental settings, LLMEffiChecker outperforms the baseline methods by a significant margin in both white-box and black-box settings. For example, for U-DL and $\lambda = 5$, LLMEffiChecker achieves a degradation success ratio over 50% with all type

perturbations in both white-box and black-box scenarios; while all the comparison baseline methods' degradation success ratios are below 5%. The results indicate that LLMEffiChecker effectively generates useful test samples to trigger LLMs' efficiency degradation. Another observation is that when $\lambda = 0$, baselines may generate some test samples that require more computations than seed inputs ($\eta \geq 50$ for H-NLP). However, such extra computations are not significant enough to indicate efficiency degradation. As we studied in Section 3, the natural efficiency variance in the LLM task could be significant, and the degree of extra computations incurred under baseline methods is within the range of natural efficiency variance. As $\lambda$ grows, $\eta$ under baseline methods drop quickly. However, this observation does not hold for LLMEffiChecker, where the average degradation success ratio of LLMEffiChecker is still 72.32% when $\lambda = 3$. Recall that from the statistical prospective [39], 99.73% of the inputs will locate in the range of $3\text{MSE}_{orig}$. Thus, these results clearly show that LLMEffiChecker successfully triggers LLMs' efficiency degradation.

> Answers to **RQ2.2**: LLMEffiChecker effectively generates test samples that trigger LLMs' efficiency degradation in both white-box and black-box settings.

### 6.4 RQ2.3: Sensitivity

In this section, we implement two prevalent decoding methods from LLMs with comprehensive hyperparameter settings to thoroughly evaluate the performance of LLMEffiChecker: Beam Search and Temperature Sampling.

**Experimental Setup.** In the first configuration, we investigate the impact of varying the beam search size on the efficiency of LLMs. As we introduced in Section 2, modern LLMs apply the beam search algorithm to generate the output token. The beam search algorithm requires one hyper-parameter, the beam search size (num_beams), to define the search space. In Section 6.3, we evaluate the effectiveness of LLMEffiChecker under each LLMs' default num_beams. In this section, we evaluate whether LLMEffiChecker is sensitive to the configuration of num_beams. We configure each LLM under test with different num_beams (ranging from 1 to 5) and measure the I-Loops, GPU latency, and GPU energy consumption of the generated test samples. In the second configuration, we focus on the effects of enabling sampling (do_sample = true) and varying the temperature parameter (i.e., 0.1, 0.3, 0.5, 0.7, and 0.9) to understand its impact on LLMEffiChecker. The temperature parameter controls the level of randomness in the sampling process, with lower temperatures leading to less variability and higher temperatures allowing for more diverse outputs.

**Experimental Results.** The results of I-Loops, GPU-Latency, and GPU-Energy for different beam sizes under Beam Search are, respectively, presented in Figure 7, Figure 8, and Figure 9. Similarly, the results of I-Loops, GPU-Latency, and GPU-Energy for different temperatures under Temperature Sampling are, respectively, presented in Figure 10, Figure 11, and Figure 12. From the results, we have the following observations: *(i)* When the beam search size num_beams is set to 1, the test samples generated by LLMEffiChecker can degrade LLMs efficiency slightly more than other beam search size settings in both white-box and black-box scenarios. This is because when num_beams=1, the token generation process is a gradient-smooth process, and the token search space is limited. Thus, our gradient-guided and causal inference-based approach can trigger more severe efficiency degradation under this configuration. *(ii)* In temperature-controlled sampling, setting the temperature to 0.1 allows for the generation of test inputs that slightly improve the reduction of LLMs' computational efficiency. This is because at a lower temperature (i.e., 0.1), the sampling process becomes more deterministic, making the model more likely to choose tokens

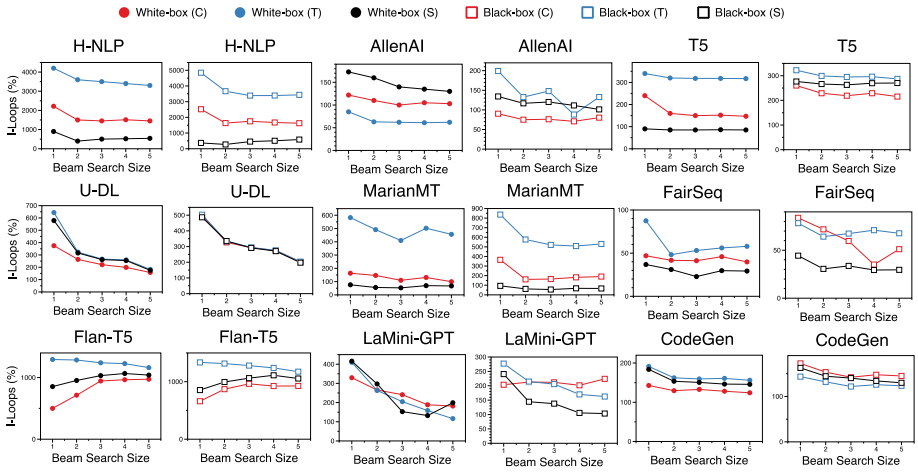



Fig. 7. I-Loops under different beam search sizes.

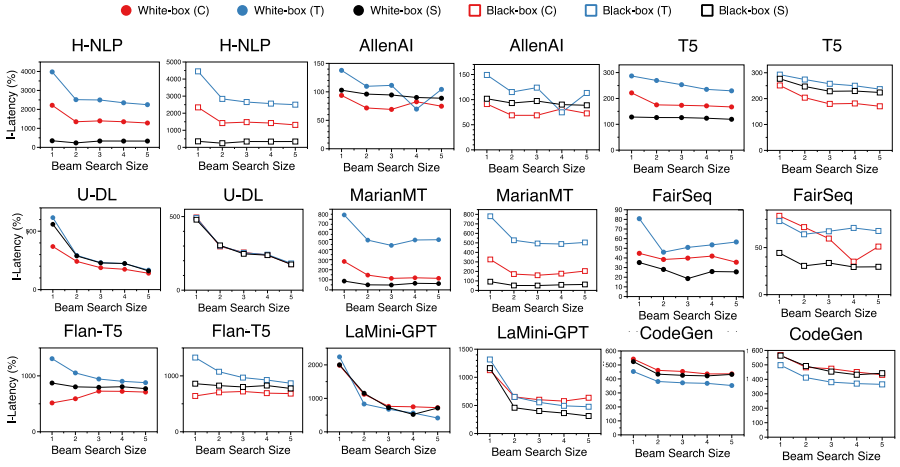



Fig. 8. GPU-Latency under different beam search sizes.

with the highest probability. This can lead to generating sequences that are highly structured. The generated test samples are more focused and consistent to triggering inefficient computation paths within the LLMs. *(iii)* Across both sets of results, it is evident that LLMEffiChecker consistently and significantly degrades the computational efficiency of the LLMs across a diverse range of beam search size settings and temperature configurations. (e.g., T5 requires more than 100% and 300% computations).

> Answers to **RQ2.3**: LLMEffiChecker can generate test samples that degrade LLMs efficiency under various decoding methods with comprehensive hyperparameter settings in both white-box and black-box settings. Moreover, the efficiency degradation is more severe when the beam search size is configured as one or temperature is set to 0.1.

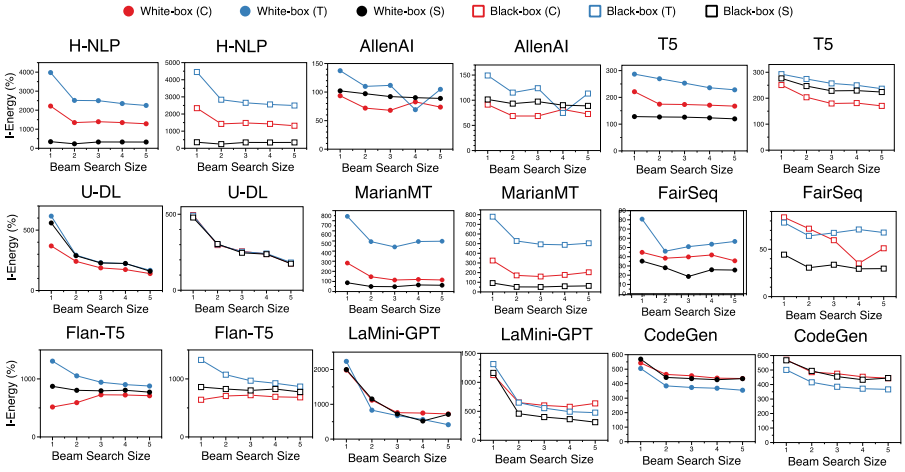



Fig. 9. GPU-Energy under different beam search sizes.

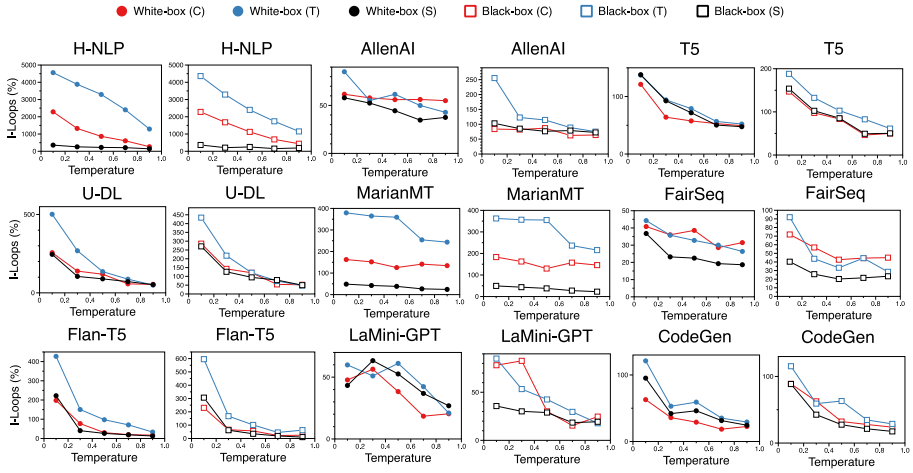



Fig. 10. I-Loops under different temperatures.

### 6.5 RQ2.4: Overheads

Table 8 and Table 9 show the average overhead of LLMEffiChecker when generating test inputs in white-box and black-box scenarios, respectively. We report only the overhead of `LLMEffiChecker`, because the comparison baselines cannot degrade LLMs' efficiency. The measured overhead of `LLMEffiChecker` is rather reasonable (ranging from 2.25 s to 191.32 s) and may increase linearly as the perturbation size increases. The linearly increasing overheads are due to the fact that `LLMEffiChecker` is an iterative approach (iteration number equals to $\epsilon$), and the overhead within each iteration is stable. Additionally, the overhead of `LLMEffiChecker-B` is reduced by 16.74% compared to `LLMEffiChecker`, as it eliminates the need for gradient calculations. Note that such reasonable overhead is not a concern, since perturbed test inputs are generated by `LLMEffiChecker` offline.

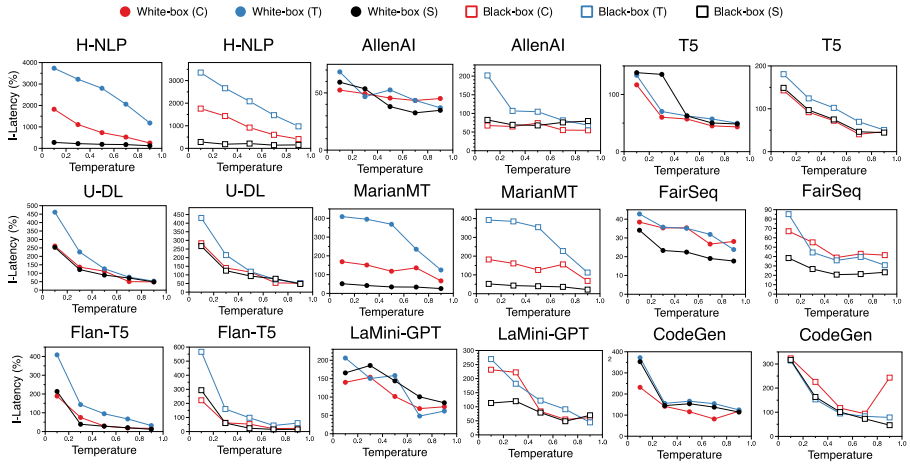



Fig. 11. GPU-Latency under different temperatures.

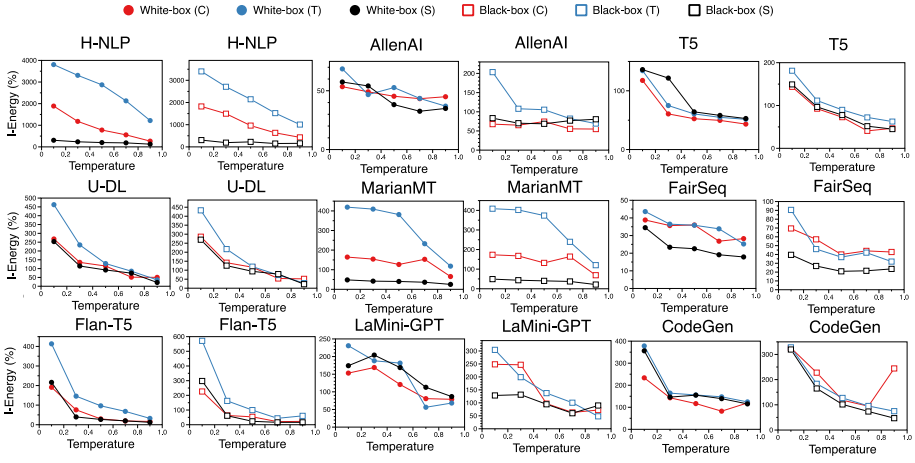



Fig. 12. GPU-Energy under different temperatures.

Table 8. Average Overheads of LLMEffiChecker (s)

| Methods | $\epsilon$ | H-NLP | AllenAi | T5 | U-DL | FairSeq | MarianMT | Flan-T5 | LaMini-GPT | CodeGen | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LLMEffiChecker(C) | 1 | 11.40 | 21.14 | 18.50 | 9.00 | 12.40 | 10.05 | 5.21 | 2.57 | 20.07 | 11.13 |
| | 2 | 31.80 | 44.66 | 45.59 | 22.09 | 28.05 | 22.83 | 12.77 | 8.42 | 46.98 | 26.52 |
| | 3 | 59.76 | 69.56 | 74.48 | 42.26 | 47.70 | 39.91 | 20.84 | 14.63 | 75.27 | 44.74 |
| LLMEffiChecker(T) | 1 | 7.50 | 18.45 | 22.62 | 31.56 | 52.80 | 38.92 | 17.85 | 5.94 | 26.70 | 22.33 |
| | 2 | 31.41 | 39.48 | 61.86 | 66.19 | 108.75 | 84.74 | 39.16 | 13.99 | 67.80 | 51.54 |
| | 3 | 62.50 | 62.54 | 100.01 | 101.28 | 165.80 | 131.74 | 62.09 | 22.35 | 110.76 | 82.21 |
| LLMEffiChecker(S) | 1 | 10.52 | 39.19 | 6.73 | 24.74 | 25.91 | 18.83 | 12.62 | 8.01 | 30.65 | 17.82 |
| | 2 | 23.33 | 75.21 | 17.45 | 59.05 | 53.85 | 39.83 | 29.49 | 19.17 | 69.94 | 38.93 |
| | 3 | 38.93 | 106.35 | 27.71 | 93.07 | 82.87 | 61.92 | 49.97 | 30.19 | 111.21 | 60.52 |

Table 9. Average Overheads of LLMEffiChecker-B (s)

| Methods | $\epsilon$ | H-NLP | AllenAi | T5 | U-DL | FairSeq | MarianMT | Flan-T5 | LaMini-GPT | CodeGen | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|
| LLMEffiChecker-B (C) | 1 | 9.73 | 17.49 | 6.05 | 10.85 | 24.51 | 15.27 | 2.59 | 2.25 | 10.73 | 10.05 |
| | 2 | 20.57 | 42.29 | 10.44 | 18.02 | 55.37 | 34.38 | 5.66 | 4.57 | 16.78 | 21.01 |
| | 3 | 31.12 | 70.62 | 14.76 | 24.93 | 92.63 | 56.17 | 9.03 | 6.88 | 22.88 | 33.20 |
| LLMEffiChecker-B (T) | 1 | 6.86 | 58.78 | 6.62 | 20.97 | 63.68 | 45.49 | 10.82 | 1.89 | 8.34 | 22.45 |
| | 2 | 11.52 | 113.03 | 10.61 | 23.80 | 130.30 | 86.04 | 12.85 | 3.34 | 12.97 | 40.65 |
| | 3 | 15.69 | 157.59 | 11.97 | 25.93 | 191.32 | 121.11 | 14.68 | 4.82 | 16.66 | 56.28 |
| LLMEffiChecker-B (S) | 1 | 3.19 | 30.84 | 16.35 | 32.84 | 29.78 | 25.58 | 15.73 | 9.66 | 24.26 | 18.92 |
| | 2 | 7.60 | 63.07 | 31.01 | 61.34 | 62.74 | 52.28 | 31.12 | 19.04 | 45.42 | 37.56 |
| | 3 | 13.51 | 94.78 | 46.55 | 85.30 | 98.08 | 80.89 | 45.43 | 28.42 | 65.11 | 56.11 |

Answers to **RQ2.4**: The overheads of LLMEffiChecker are reasonable and may increase linearly as the perturbation size increases. Specifically, when $\epsilon = 1$, LLMEffiChecker costs 17.01, 16.19, and 18.81 seconds to generate character-level, token-level, and structure-level test samples. Correspondingly, LLMEffiChecker-B costs 10.05, 22.45, and 18.92 seconds to generate samples of the same type.

### 6.6 RQ2.5: Ablation Study

In this experiment, we carried out ablation studies to assess the efficacy of $p_i^{o_i}$ in LLMEffiChecker for identifying critical tokens, as illustrated in Equation (2). The inspiration for this component came from recent research, which showed that the sequence of tokens output by a model also affects the generation of the EOS token [27]. To validate this idea's effectiveness for LLMEffiChecker and ensure it aligns with our overarching goals, we remove $p_i^{o_i}$ from the function $f(x)$ in Equation (2) and then apply it to generate test inputs.

**Experimental Setup.** In our evaluation of various LLMs, we randomly choose 1,000 seed inputs and apply LLMEffiChecker (with $p_i^{o_i}$ removed from $f(x)$) to generate 1,000 abnormal inputs for each type of perturbation. We denote the approach with the removed $p_i^{o_i}$ as Removed and our original approach as Original. The evaluation metrics employed adhere to those detailed in Section 6.2. Correspondingly, we average the experimental outcomes over three runs.

**Experimental Results.** The results are shown in Table 10. From the results, we make two observations: *(i)* The test samples generated in the ablation study exhibit a weaker degradation in computational efficiency for LLMs. Specifically, out of 27 control experiments conducted, 20 confirm this finding. On average, the required loops, CPU latency, CPU energy consumption, GPU latency, and GPU energy consumption decreased by 18.21%, 20.75%, 20.44%, 20.04%, and 20.11%, respectively. *(ii)* The decoder-only models are more sensitive to such components. Notably, during the ablation study, the GPU latency of LaMini-GPT saw a significant decrease of 74.07% compared to control experiments. In contrast, models based on an encoder-decoder architecture exhibited a maximum reduction of only 45.83%. This heightened sensitivity in decoder-only models can be attributed to their autoregressive nature, which makes them more susceptible to the influence of output context. Therefore, the results demonstrate the effectiveness of the $p_i^{o_i}$ component in LLMEffiChecker for identifying critical tokens.

Answers to **RQ2.5**: Each component within LLMEffiChecker aligns with the overall design goal and effectively contributes to its performance enhancement.

Table 10. The Ablation Results of Ours-Ablation in Degrading LLMs Performance

| Subject | Methods | I-Loops | | | I-Latency(CPU) | | | I-Energy(CPU) | | | I-Latency(GPU) | | | I-Energy(GPU) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $\epsilon = 1$ | $\epsilon = 2$ | $\epsilon = 3$ | $\epsilon = 1$ | $\epsilon = 2$ | $\epsilon = 3$ | $\epsilon = 1$ | $\epsilon = 2$ | $\epsilon = 3$ | $\epsilon = 1$ | $\epsilon = 2$ | $\epsilon = 3$ | $\epsilon = 1$ | $\epsilon = 2$ | $\epsilon = 3$ |
| H-NLP | Original (C) | 564.45 | 995.45 | 1,357.77 | 764.92 | 1,487.92 | 2,015.70 | 785.60 | 1,471.26 | 1,967.05 | 462.24 | 851.80 | 1,116.80 | 406.39 | 755.18 | 972.92 |
| | Removed (C) | **493.90** | **934.79** | **1,244.70** | **592.94** | **1,133.14** | **1,547.26** | **613.07** | **1,117.62** | **1,506.78** | **352.26** | **668.56** | **888.26** | **309.12** | **592.12** | **773.13** |
| | Original (T) | 2,697.77 | 3,735.38 | 3,917.91 | 3,153.97 | 4,481.93 | 4,681.28 | 3,052.62 | 4,544.65 | 4,759.71 | 1,953.57 | 2,729.83 | 2,854.89 | 1,532.91 | 2,137.53 | 2,221.66 |
| | Removed (T) | **1,594.55** | **2,271.86** | **2,382.88** | **1,621.64** | **2,355.45** | **2,460.22** | **1,555.09** | **2,359.78** | **2,471.45** | **1,058.38** | **1,518.38** | **1,587.94** | **830.35** | **1,188.80** | **1,235.59** |
| | Original (S) | 142.31 | 311.06 | 612.08 | 146.51 | 451.93 | 877.79 | 147.70 | 461.30 | 870.72 | 101.21 | 275.58 | 523.04 | 95.05 | 259.88 | 508.80 |
| | Removed (S) | **127.09** | **280.58** | **593.16** | **136.92** | **340.42** | **632.91** | **136.97** | **346.46** | **624.23** | **98.81** | **219.31** | **408.83** | **92.76** | **206.57** | **397.26** |
| AllenAI | Original (C) | 35.16 | 74.90 | 103.36 | 26.69 | 45.77 | 85.09 | 27.48 | 48.09 | 86.00 | 21.82 | 35.43 | 91.48 | 22.12 | 43.21 | 98.46 |
| | Removed (C) | **44.62** | **88.40** | **127.77** | **34.73** | **73.64** | **111.38** | **59.37** | **108.81** | **154.37** | **38.93** | **77.78** | **115.71** | **38.80** | **77.59** | **115.48** |
| | Original (T) | 24.83 | 42.04 | 56.75 | 49.12 | 62.84 | 67.98 | 49.99 | 62.65 | 69.06 | 30.65 | 41.32 | 46.09 | 31.00 | 41.81 | 49.66 |
| | Removed (T) | **34.73** | **58.45** | **89.32** | **25.04** | **50.13** | **86.52** | **45.17** | **75.62** | **118.70** | **29.57** | **55.62** | **92.69** | **29.45** | **55.46** | **92.50** |
| | Original (S) | 66.21 | 108.67 | 128.60 | 86.05 | 139.03 | 164.57 | 84.17 | 135.71 | 160.95 | 69.57 | 112.88 | 132.68 | 68.79 | 115.23 | 137.06 |
| | Removed (S) | **67.48** | **99.92** | **131.33** | **79.34** | **115.42** | **144.89** | **111.56** | **153.76** | **189.10** | **84.90** | **120.76** | **152.29** | **84.67** | **120.48** | **151.97** |
| T5 | Original (C) | 168.92 | 198.36 | 205.37 | 191.05 | 225.48 | 233.01 | 194.45 | 228.02 | 234.04 | 164.61 | 194.79 | 202.28 | 165.38 | 195.77 | 203.29 |
| | Removed (C) | **155.44** | **188.32** | **190.63** | **168.33** | **215.73** | **218.59** | **173.43** | **218.39** | **220.48** | **150.10** | **183.60** | **185.25** | **151.30** | **185.13** | **186.71** |
| | Original (T) | 307.27 | 328.94 | 328.94 | 352.14 | 376.55 | 376.55 | 347.74 | 373.85 | 373.85 | 305.37 | 325.61 | 325.61 | 331.85 | 352.25 | 352.25 |
| | Removed (T) | **294.68** | **315.47** | **315.47** | **334.63** | **357.83** | **357.83** | **328.58** | **353.25** | **353.25** | **294.72** | **314.25** | **314.25** | **320.11** | **339.79** | **339.79** |
| | Original (S) | 77.67 | 80.56 | 82.52 | 85.72 | 89.11 | 91.38 | 86.90 | 90.29 | 92.56 | 75.77 | 78.68 | 80.66 | 68.79 | 73.03 | 74.56 |
| | Removed (S) | **78.47** | **81.24** | **83.37** | **85.44** | **88.73** | **91.24** | **87.53** | **90.81** | **93.25** | **76.75** | **79.71** | **82.14** | **69.65** | **73.96** | **75.90** |
| U-DL | Original (C) | 258.07 | 390.60 | 469.24 | 261.02 | 405.80 | 494.30 | 288.15 | 439.72 | 532.81 | 253.46 | 383.78 | 461.51 | 253.45 | 383.82 | 461.64 |
| | Removed (C) | **154.67** | **274.00** | **350.74** | **157.17** | **282.11** | **366.04** | **170.59** | **308.67** | **399.07** | **156.19** | **274.11** | **352.54** | **156.32** | **274.10** | **352.59** |
| | Original (T) | 604.17 | 642.38 | 642.38 | 655.13 | 696.56 | 696.56 | 697.90 | 741.86 | 741.86 | 595.88 | 634.44 | 634.44 | 596.49 | 635.08 | 635.08 |
| | Removed (T) | **595.70** | **595.70** | **595.70** | **635.50** | **635.50** | **635.50** | **678.60** | **678.60** | **678.60** | **590.44** | **590.44** | **590.44** | **590.44** | **590.44** | **590.44** |
| | Original (S) | 406.92 | 592.52 | 702.89 | 438.23 | 632.64 | 753.88 | 465.04 | 673.76 | 800.25 | 404.42 | 583.65 | 694.40 | 401.74 | 583.99 | 694.84 |
| | Removed (S) | **329.64** | **467.00** | **501.56** | **350.83** | **494.42** | **532.38** | **374.35** | **533.90** | **573.56** | **332.55** | **469.61** | **504.01** | **332.57** | **469.57** | **504.01** |
| FairSeq | Original (C) | 22.53 | 37.68 | 59.26 | 15.87 | 29.07 | 49.18 | 20.47 | 34.62 | 55.44 | 18.07 | 31.53 | 51.79 | 18.08 | 31.55 | 51.85 |
| | Removed (C) | **19.79** | **34.08** | **51.59** | **14.24** | **25.78** | **41.86** | **18.16** | **32.69** | **48.79** | **16.51** | **30.98** | **46.45** | **16.51** | **28.00** | **46.49** |
| | Original (T) | 33.73 | 62.13 | 76.41 | 23.97 | 55.26 | 70.28 | 29.79 | 63.19 | 79.41 | 28.62 | 59.42 | 75.47 | 28.60 | 59.42 | 75.47 |
| | Removed (T) | **23.65** | **37.82** | **58.70** | **16.25** | **28.96** | **47.25** | **21.01** | **34.80** | **54.37** | **18.99** | **32.82** | **52.96** | **18.96** | **32.81** | **52.94** |
| | Original (S) | 19.42 | 30.87 | 37.82 | 14.01 | 23.67 | 31.31 | 18.23 | 28.59 | 36.73 | 14.84 | 24.87 | 31.68 | 14.86 | 24.91 | 31.72 |
| | Removed (S) | **19.17** | **30.64** | **36.33** | **13.11** | **22.57** | **28.64** | **16.96** | **27.46** | **34.15** | **14.26** | **23.90** | **30.21** | **14.75** | **23.91** | **30.22** |
| MarianMT | Original (C) | 54.10 | 113.20 | 222.04 | 41.58 | 102.38 | 226.68 | 51.56 | 119.53 | 274.26 | 52.98 | 113.90 | 210.12 | 52.89 | 113.88 | 210.12 |
| | Removed (C) | **60.13** | **78.00** | **105.23** | **59.41** | **78.72** | **103.52** | **70.11** | **91.22** | **118.31** | **56.65** | **74.42** | **96.50** | **56.78** | **74.55** | **96.66** |
| | Original (T) | 231.65 | 550.07 | 726.61 | 234.93 | 564.34 | 770.28 | 269.58 | 660.70 | 893.87 | 223.56 | 544.49 | 728.90 | 223.48 | 544.28 | 728.68 |
| | Removed (T) | **207.33** | **284.10** | **327.79** | **235.99** | **315.13** | **398.78** | **264.59** | **353.72** | **449.18** | **201.15** | **274.95** | **346.98** | **201.14** | **274.88** | **346.94** |
| | Original (S) | 42.77 | 72.19 | 89.17 | 33.89 | 72.33 | 90.33 | 40.27 | 84.89 | 106.84 | 41.21 | 77.28 | 95.68 | 41.19 | 77.21 | 95.61 |
| | Removed (S) | **34.13** | **46.25** | **55.67** | **29.43** | **38.89** | **48.23** | **36.48** | **46.97** | **57.39** | **30.14** | **39.65** | **48.94** | **30.12** | **39.62** | **48.91** |
| Flan-T5 | Original (C) | 327.55 | 566.82 | 625.69 | 329.99 | 564.27 | 621.78 | 381.37 | 647.48 | 715.92 | 333.91 | 574.26 | 634.28 | 334.03 | 574.53 | 634.51 |
| | Removed (C) | **440.92** | **597.31** | **696.80** | **440.07** | **591.31** | **694.97** | **507.54** | **680.23** | **800.62** | **445.48** | **599.07** | **703.96** | **446.51** | **600.50** | **705.66** |
| | Original (T) | 1,209.50 | 1,306.26 | 1,349.04 | 1,229.54 | 1,327.42 | 1,372.96 | 1,409.23 | 1,524.04 | 1,578.49 | 1,227.99 | 1,325.01 | 1,368.67 | 1,229.06 | 1,326.23 | 1,369.93 |
| | Removed (T) | **1,195.08** | **1,336.93** | **1,392.43** | **1,217.34** | **1,363.25** | **1,422.13** | **1,401.04** | **1,571.49** | **1,642.44** | **1,203.52** | **1,348.72** | **1,410.90** | **1,206.12** | **1,351.65** | **1,413.88** |
| | Original (S) | 554.58 | 937.63 | 1063.39 | 552.96 | 952.73 | 1,087.15 | 637.50 | 1,094.72 | 1,253.29 | 564.39 | 948.81 | 1,076.25 | 564.90 | 949.32 | 1,076.86 |
| | Removed (S) | **572.68** | **965.17** | **1,101.25** | **579.51** | **967.04** | **1,109.02** | **672.54** | **1,115.38** | **1,278.07** | **575.43** | **972.25** | **1,108.37** | **576.77** | **974.47** | **1110.90** |
| LaMini-GPT | Original (C) | 323.34 | 367.46 | 376.55 | 2,091.80 | 2,643.98 | 2,707.52 | 2,403.58 | 3,098.30 | 3,169.53 | 1,598.12 | 1,997.75 | 2,043.54 | 1,597.50 | 1,995.33 | 2,041.12 |
| | Removed (C) | **132.15** | **151.14** | **151.33** | **582.31** | **727.35** | **728.12** | **734.07** | **924.54** | **925.36** | **624.50** | **760.50** | **761.69** | **623.63** | **758.88** | **760.06** |
| | Original (T) | 368.67 | 379.73 | 379.73 | 2,539.10 | 2,588.35 | 2,588.35 | 3,066.39 | 3,130.40 | 3,130.40 | 2,106.89 | 2,148.49 | 2,148.49 | 2,104.61 | 2,146.16 | 2,146.16 |
| | Removed (T) | **109.46** | **124.69** | **128.23** | **538.88** | **620.64** | **638.58** | **681.90** | **783.73** | **805.72** | **546.36** | **629.52** | **650.19** | **545.67** | **628.73** | **649.42** |
| | Original (S) | 347.41 | 366.07 | 366.42 | 2,157.36 | 2,371.21 | 2,372.00 | 2,510.74 | 2,792.73 | 2,793.60 | 1,746.59 | 1,919.43 | 1,920.28 | 1,747.96 | 1,919.32 | 1,920.17 |
| | Removed (S) | **104.26** | **137.73** | **144.21** | **527.32** | **697.10** | **722.20** | **668.76** | **884.99** | **916.18** | **545.94** | **727.06** | **750.39** | **545.60** | **726.90** | **750.66** |
| CodeGen | Original (C) | 109.93 | 139.88 | 168.23 | 321.08 | 434.06 | 533.72 | 336.01 | 453.64 | 558.45 | 351.20 | 478.08 | 592.39 | 351.46 | 478.36 | 592.75 |
| | Removed (C) | **97.58** | **117.32** | **132.52** | **306.48** | **367.30** | **404.72** | **319.78** | **383.35** | **422.02** | **327.14** | **396.39** | **435.68** | **327.33** | **396.65** | **435.95** |
| | Original (T) | 182.42 | 182.42 | 182.42 | 578.30 | 578.30 | 578.30 | 602.68 | 602.68 | 602.68 | 639.91 | 639.91 | 639.91 | 640.26 | 640.26 | 640.26 |
| | Removed (T) | **131.58** | **160.20** | **167.04** | **407.21** | **512.63** | **541.95** | **424.16** | **535.21** | **566.01** | **443.26** | **562.78** | **595.73** | **443.52** | **563.03** | **596.04** |
| | Original (S) | 176.30 | 187.59 | 187.59 | 575.42 | 615.61 | 615.61 | 593.11 | 635.62 | 635.62 | 607.64 | 653.51 | 653.51 | 607.97 | 653.91 | 653.91 |
| | Removed (S) | **150.11** | **169.18** | **178.25** | **501.52** | **560.08** | **580.77** | **521.86** | **582.85** | **604.39** | **536.43** | **604.63** | **627.90** | **536.68** | **604.95** | **628.28** |

Table 11. Input Sentences for Experiments on Mobile Devices

| | |
|---|---|
| **Seed Input** | Death comes often to the soldiers and marines who are fighting in anbar province, which is roughly the size of louisiana and is the most intractable region in iraq. |
| **Test Input** | Death comes often to the soldiers and marines who are fighting in anbar province, which is roughly the (size of of louisiana and is the most intractable region in iraq. |

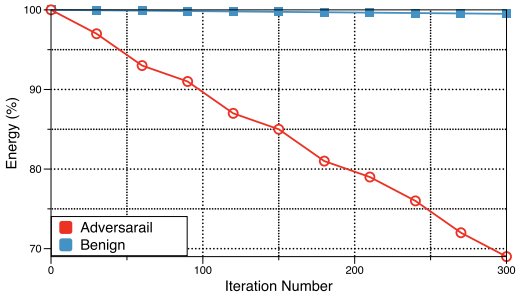



Fig. 13. Remaining battery power of the mobile device after T5 original seed and perturbed sentences.

## 7 REAL-WORLD STUDY AND POSSIBLE MITIGATION STRATEGY

In this section, we present a real-world case study to discuss how LLMs' efficiency degradation will impact real-world devices' battery power and the computational latency of commercial models. After that, we show how developers could apply `LLMEffiChecker` to improve LLMs' efficiency robustness and mitigate computational resource waste. Finally, we discuss potential threats that might threaten the applicability of `LLMEffiChecker` and how we alleviate them.

### 7.1 Real-world Case Study on Mobile Devices

**Experimental Setup.** We select Google T5 as our evaluation LLM in this case study. We first deploy the model on the Samsung Galaxy S9+, which has 6 GB RAM and a battery capacity of 3,500 mAh. After that, we select one sentence from the dataset `ZH19` as our seed input; we then apply `LLMEffiChecker` to perturb the seed input with character-level perturbation and obtain the corresponding test sample. The seed sentence and the corresponding test sample are shown in Table 11, where the perturbation is colored in red. Notice the test sample inserts only one character in the seed sentence. This one-character perturbation is very common in the real world due to a user's typo. Finally, we feed the seed input and test sample to the deployed LLMs and measure the mobile device's battery consumption rate.

**Experimental Results.** The mobile phone's battery consumption status is shown in Figure 13. The red line is for the perturbed input, and the blue one is for the original seed input. The results show that the perturbed input consumes the mobile's battery power significantly more quickly than the seed input. Specifically, after 300 iterations, the perturbed input consumes 30% of the battery power, while the seed input consumes less than 1%. The results demonstrate the vulnerability of the efficiency degradation for mobile devices. Recall that the perturbed example used in our experiment only inserts one character in the seed sentence, which would mimic many practical scenarios (e.g., typo). Thus, the results suggest the criticality and the necessity of improving LLMs' efficiency robustness.

### 7.2 Real-world Case Study on Commercial Model

**Experimental Setup.** In this case study, we select OpenAI's GPT-3.5 as the evaluation model. We randomly choose 500 entries from the test set of the HellaSwag [89] dataset as seed inputs. Given its status as a commercial model not available in open source, we opt for three types of black-box test methods from LLMEffiChecker (i.e., LLMEffiChecker-B (C), LLMEffiChecker-B (T), and LLMEffiChecker-B (S)), with the perturbation level set to 1. For the baseline, we employ all the black-box methods (i.e., SynError, SIT, and TransRepair) in our research. The evaluation metrics include I-Loops and I-Latency, as discussed in Section 6.2. Specifically, I-Loops is calculated using the *completion_tokens* from GPT-3.5's returned JSON, which reflects the number of decoder invocations, correlating with the computational demands (i.e., required FLOPs). Concurrently, I-Latency is calculated from another field in the returned JSON, namely, *response_ms*, which represents the time required for the model to generate data upon receiving input.

**Experimental Results.** Table 14 shows the average efficiency reduction results of GPT-3.5 under various perturbations. The results indicate that the perturbations generated by LLMEffiChecker-B lead to a notably steeper decline in computational efficiency compared to the baseline methods. Specifically, perturbations produced by LLMEffiChecker-B can increase GPT-3.5's I-Loops and I-Latency by an average of 25.64% to 176.92% and 19.96% to 156.53%, respectively. It is noteworthy that the perturbations set in this experiment are minimal, at the level of a single character or token. Furthermore, the test inputs conceived by LLMEffiChecker-B (S) not only replicate the structural essence of the original sentences without introducing any grammatical or lexical inaccuracies but also succeed in catalyzing a 66.18% surge in computational latency. Therefore, the results substantiate LLMEffiChecker's efficacy and underscore the prevalent issue of computational efficiency vulnerabilities within real-world LLMs.

### 7.3 Mitigating Efficiency Degradation with LLMEffiChecker

This section shows how developers leverage LLMEffiChecker to develop runtime abnormal input detector, which mitigates possible efficiency degradation and computational waste under the adversary scenario (e.g., DOS attack). In detail, we propose an approach to filter out test inputs that require abnormal computational resources at runtime. Because the abnormal inputs are forced to quit at early stage, the computational resources waste is avoided. The idea of applying input validation to improve DNNs' correctness robustness has been studied in recent works [77, 78]. However, existing input validation techniques may not be suitable for improving LLMs' efficiency robustness due to the high overheads. Our intuition is that although normal inputs and the computational resource heavy inputs look similar in human eyes, the latent representations of these two categories of inputs are quite different [77]. Thus, we can leverage the latent representations of these two category inputs to train a light-weighted SVM classifier and apply the classifier to distinguish abnormal inputs at runtime. Because the classifier should be light-weighted, getting each input's latent representations is preferable without additional computations. Specifically, in LLMs, the hidden layer converts input data into a higher-level abstract representation, effectively capturing the essential features and patterns of the input sentences. We propose to use the information in the hidden layer as the latent representation to train a lighted-weighted SVM classifier.

**Experimental Setup.** For each LLMs in our evaluation, we randomly choose 1,000 seed inputs and apply LLMEffiChecker to generate 1,000 abnormal inputs for each perturbation type. We randomly select 80% of the seed inputs and the abnormal inputs as the training data to train the SVM classifier and use the rest 20% for testing. We run the trained SVM classifier on the testing dataset and measure the detectors' AUC score, extra computation overheads.

Table 12. The Accuracy and Extra Overheads of the LLMEffiChecker Detector

| Methods | H-NLP | | | | AllenAI | | | | T5 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | AUC | Overheads | Energy | Acc | AUC | Overheads | Energy | Acc | AUC | Overheads | Energy |
| LLMEffiChecker(C) | 99.98 | 100.00 | 0.17 | 0.09 | 100.00 | 100.00 | 0.17 | 0.11 | 99.97 | 100.00 | 0.08 | 0.05 |
| LLMEffiChecker(T) | 99.99 | 100.00 | 0.32 | 0.17 | 100.00 | 100.00 | 0.08 | 0.05 | 100.00 | 100.00 | 0.06 | 0.04 |
| LLMEffiChecker(S) | 99.98 | 100.00 | 0.18 | 0.12 | 87.00 | 98.32 | 0.49 | 0.30 | 99.99 | 100.00 | 0.03 | 0.02 |
| Mixed | 99.98 | 100.00 | 0.74 | 0.48 | 98.00 | 100.00 | 0.86 | 0.79 | 100.00 | 100.00 | 0.18 | 0.11 |
| **Methods** | **U-DL** | | | | **FairSeq** | | | | **MarianMT** | | | |
| | Acc | AUC | Overheads | Energy | Acc | AUC | Overheads | Energy | Acc | AUC | Overheads | Energy |
| LLMEffiChecker(C) | 100.00 | 100.00 | 0.00 | 0.19 | 100.00 | 100.00 | 0.02 | 0.11 | 100.00 | 100.00 | 0.01 | 0.03 |
| LLMEffiChecker(T) | 100.00 | 100.00 | 0.00 | 0.54 | 100.00 | 100.00 | 0.01 | 0.27 | 100.00 | 100.00 | 0.00 | 0.06 |
| LLMEffiChecker(S) | 100.00 | 100.00 | 0.01 | 0.31 | 100.00 | 100.00 | 0.01 | 0.16 | 100.00 | 100.00 | 0.03 | 0.02 |
| Mixed | 100.00 | 100.00 | 0.03 | 0.83 | 100.00 | 100.00 | 0.10 | 0.52 | 98.50 | 100.00 | 0.01 | 0.15 |
| **Methods** | **Flan-T5** | | | | **LaMini-GPT** | | | | **CodeGen** | | | |
| | Acc | AUC | Overheads | Energy | Acc | AUC | Overheads | Energy | Acc | AUC | Overheads | Energy |
| LLMEffiChecker(C) | 83.65 | 89.42 | 0.01 | 0.02 | 100.00 | 100.00 | 0.00 | 0.06 | 100.00 | 100.00 | 0.13 | 0.09 |
| LLMEffiChecker(T) | 91.00 | 91.38 | 0.04 | 0.09 | 100.00 | 100.00 | 0.01 | 0.25 | 100.00 | 100.00 | 0.37 | 0.43 |
| LLMEffiChecker(S) | 90.50 | 93.98 | 0.04 | 0.06 | 92.50 | 100.00 | 0.01 | 0.14 | 98.47 | 100.00 | 0.27 | 0.14 |
| Mixed | 92.66 | 97.38 | 0.24 | 0.26 | 99.00 | 100.00 | 0.05 | 0.47 | 100.00 | 100.00 | 0.71 | 0.79 |

Table 13. The Accuracy and Extra Overheads of the LLMEffiChecker-B Detector

| Methods | H-NLP | | | | AllenAI | | | | T5 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | AUC | Overheads | Energy | Acc | AUC | Overheads | Energy | Acc | AUC | Overheads | Energy |
| LLMEffiChecker-B (C) | 100.00 | 100.00 | 0.14 | 0.07 | 100.00 | 100.00 | 0.14 | 0.13 | 100.00 | 100.00 | 0.06 | 0.04 |
| LLMEffiChecker-B (T) | 100.00 | 100.00 | 0.30 | 0.16 | 100.00 | 100.00 | 0.12 | 0.08 | 100.00 | 100.00 | 0.08 | 0.04 |
| LLMEffiChecker-B (s) | 100.00 | 100.00 | 0.19 | 0.09 | 95.00 | 100.00 | 0.52 | 0.37 | 97.50 | 100.00 | 0.03 | 0.03 |
| Mixed | 100.00 | 100.00 | 0.69 | 0.39 | 97.50 | 100.00 | 0.81 | 0.67 | 97.50 | 100.00 | 0.17 | 0.12 |
| **Methods** | **U-DL** | | | | **FairSeq** | | | | **MarianMT** | | | |
| | Acc | AUC | Overheads | Energy | Acc | AUC | Overheads | Energy | Acc | AUC | Overheads | Energy |
| LLMEffiChecker-B (C) | 100.00 | 100.00 | 0.00 | 0.21 | 100.00 | 100.00 | 0.03 | 0.14 | 100.00 | 100.00 | 0.02 | 0.09 |
| LLMEffiChecker-B (T) | 100.00 | 100.00 | 0.02 | 0.58 | 100.00 | 100.00 | 0.02 | 0.23 | 100.00 | 100.00 | 0.04 | 0.05 |
| LLMEffiChecker-B (S) | 100.00 | 100.00 | 0.00 | 0.42 | 100.00 | 100.00 | 0.00 | 0.15 | 100.00 | 100.00 | 0.02 | 0.04 |
| Mixed | 97.50 | 100.00 | 0.03 | 0.96 | 100.00 | 100.00 | 0.09 | 0.56 | 100.00 | 100.00 | 0.07 | 0.18 |
| **Methods** | **Flan-T5** | | | | **LaMini-GPT** | | | | **CodeGen** | | | |
| | Acc | AUC | Overheads | Energy | Acc | AUC | Overheads | Energy | Acc | AUC | Overheads | Energy |
| LLMEffiChecker-B (C) | 100.00 | 100.00 | 0.07 | 0.07 | 100.00 | 100.00 | 0.03 | 0.05 | 100.00 | 100.00 | 0.15 | 0.12 |
| LLMEffiChecker-B (T) | 100.00 | 100.00 | 0.12 | 0.09 | 100.00 | 100.00 | 0.06 | 0.32 | 100.00 | 100.00 | 0.42 | 0.45 |
| LLMEffiChecker-B (S) | 100.00 | 100.00 | 0.05 | 0.04 | 94.47 | 98.46 | 0.05 | 0.18 | 99.87 | 100.00 | 0.29 | 0.20 |
| Mixed | 100.00 | 100.00 | 0.17 | 0.29 | 98.34 | 99.73 | 0.08 | 0.53 | 99.98 | 100.00 | 0.84 | 0.86 |

**Experimental Results.** The experimental results in white-box and black-box scenarios are shown in Table 12 and Table 13, respectively. Each column in Table 12 and Table 13 represents the performance in detecting one specific perturbation type, and "Mixed" represents the performance in detecting a mixed set of three perturbation types. We observe that the proposed detector achieves almost perfect detection accuracy with a lowest accuracy of 83.65%. Moreover, the proposed detector's overheads and energy consumption are negligible compared to those incurred under the LLM. All experimental subjects' extra overheads and the energy consumption are merely at most 1% of the original LLMs' overheads in generation normal sentences. The results show that our

Table 14. The Average Effectiveness Results of LLMEffiChecker on GPT-3.5

| Subject | Methods | I-Loops | I-Latency |
|---|---|---|---|
| GPT3.5 | **SynError** | −1.19 | −1.46 |
| | **SIT** | 6.98 | 6.64 |
| | **TransRepair** | −1.3 | −11.21 |
| | **LLMEffiChecker-B (C)** | 25.64 | 19.96 |
| | **LLMEffiChecker-B (T)** | 176.92 | 156.53 |
| | **LLMEffiChecker-B (S)** | 90.93 | 66.18 |

validation-based approach can effectively filter out the abnormal input sentences with negligible overheads.

### 7.4 Threat Analysis

Our selection of the nine LLMs, namely, Google T5, AllenAI WMT14, H-NLP, U-DL, Facebook FairSeq, MarianMT, Flan-T5, LaMini-GPT, and CodeGen, might threaten the *external validity* of our experimental conclusions. We alleviate this threat by the following efforts: (1) the nine LLMs are very popular and have been widely used among developers (with more than 2,714,275 downloads in November 2023); (2) their underlying DNN models are state-of-the-art models; (3) these systems differ from each other by diverse topics (e.g., model architecture, language, training corpus, training process). Therefore, our experimental conclusions should generally hold, although specific data could be inevitably different for other subjects. Our *internal threat* mainly comes from our definition of different perturbation types. Our introduced perturbation may not always be grammatically correct (e.g., inserting one character may result in an unknown token). However, as discussed in Section 2, such perturbations may not be typical but exist in the real-world (e.g., user typos, adversarial manner). Thus, it is meaningful to understand LLMs' efficiency degradation with such realistic perturbations. Moreover, all three perturbation types are well studied in related works [20, 21, 30, 33, 34, 64, 69, 88, 91, 95].

## 8 RELATED WORK

**Adversarial Attacks & DNN Robustness.** Recent works [7, 17, 59, 71, 86, 88, 91] show that DNN-based applications are not robust under adversarial attacks, which generate adversarial examples to fool the state-of-the-art DNN-based applications. Existing adversarial attacks can be grouped as *white-box*, and *black-box* attacks based on their access to the DNN parameters. To improve DNNs' robustness and mitigate the threats of adversarial attacks, a series of defense approaches [12, 23, 43, 78, 85] have been proposed. For example, FeatureSqueeze [85] introduces a series of feature squeeze approaches to mitigate the adversarial perturbations during DNN runtime. NNMutate[78] identifies that adversarial examples are the data points close to the DNN decision boundary and thus proposes applying model mutation techniques to detect adversarial samples.

**DNN's Efficiency.** Recently, the efficiency of DNNs has raised much concern due to their substantial *inference-time* costs. To improve DNNs' *inference-time* efficiency, many existing works have been proposed, categorized into two major techniques. The first category [38, 92] of techniques prune the DNNs offline to identify important neurons and remove unimportant ones. After pruning, the smaller size DNNs could achieve competitive accuracy compared to the original DNNs while incurring significantly less computational costs. Another category of techniques [24, 25, 79], called input-adaptive techniques, dynamically skip a certain part of the DNNs to reduce the number of computations during inference time. By skipping certain parts of the DNNs, the

input-adaptive DNNs can trade off between accuracy and computational costs. However, recent studies [13, 14, 16, 31, 37] show input-adaptive DNNs are not robust against the adversary attack, which implies the input-adaptive will not save computational costs under attacks.

## 9 CONCLUSIONS

In this work, we study the efficiency robustness of LLMs. Specifically, we present `LLMEffiChecker`, a comprehensive framework designed to function effectively in both white-box and black-box scenarios. This innovative framework introduces imperceptible perturbations to seed inputs, strategically reducing the computational efficiency of LLMs. Evaluation on nine publicly available LLMs shows that `LLMEffiChecker` can generate effective test inputs that may significantly decrease LLMs' efficiency.

## REFERENCES


[1] AllenAI. 2022. Retrieved from https://huggingface.co/allenai/wmt16-en-de-dist-12-1

[2] Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Benjamin Mann, Nova DasSarma, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Jackson Kernion, Kamal Ndousse, Catherine Olsson, Dario Amodei, Tom B. Brown, Jack Clark, Sam McCandlish, Chris Olah, and Jared Kaplan. 2021. A general language assistant as a laboratory for alignment. *CoRR* abs/2112.00861 (2021).

[3] Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR* abs/2108.07732 (2021).

[4] Yonatan Belinkov and Yonatan Bisk. 2018. Synthetic and natural noise both break neural machine translation. In *6th International Conference on Learning Representations (ICLR'18)*. OpenReview.net. Retrieved from https://openreview.net/forum?id=BJ8vJebC-

[5] Wieland Brendel, Jonas Rauber, Matthias Kümmerer, Ivan Ustyuzhaninov, and Matthias Bethge. 2019. Accurate, reliable and fast robustness evaluation. In *Annual Conference on Neural Information Processing Systems (NeurIPS'19)*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 12841–12851. Retrieved from https://proceedings.neurips.cc/paper/2019/hash/885fe656777008c335ac96072a45be15-Abstract.html

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems Annual Conference on Neural Information Processing Systems (NeurIPS'20)*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). December 6-12, 2020, virtual, 2020.

[7] Nicholas Carlini and David A. Wagner. 2017. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (SP'17)*. IEEE Computer Society, 39–57. DOI: https://doi.org/10.1109/SP.2017.49

[8] Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q. Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. 2023. MultiPL-E: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Trans. Softw. Eng.* 49, 7 (2023), 3675–3691. DOI: https://doi.org/10.1109/TSE.2023.3267446

[9] Isaac Caswell and Bowen Liang. 2020. Recent Advances in Google Translate. Retrieved from https://ai.googleblog.com/2020/06/recent-advances-in-google-translate.html

[10] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Kaijie Zhu, Hao Chen, Linyi Yang, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. 2023. A survey on evaluation of large language models. *CoRR* abs/2307.03109 (2023).

[11] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati,

Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR* abs/2107.03374 (2021).

[12] Simin Chen, Soroush Bateni, Sampath Grandhi, Xiaodi Li, Cong Liu, and Wei Yang. 2020. DENAS: Automated rule generation by knowledge extraction from neural networks. In *28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'20)*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 813–825. DOI: https://doi.org/10.1145/3368089.3409733

[13] Simin Chen, Hanlin Chen, Mirazul Haque, Cong Liu, and Wei Yang. 2023. The dark side of dynamic routing neural networks: Towards efficiency backdoor injection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'23)*. IEEE, 24585–24594. DOI: https://doi.org/10.1109/CVPR52729.2023.02355

[14] Simin Chen, Mirazul Haque, Cong Liu, and Wei Yang. 2022. DeepPerform: An efficient approach for performance testing of resource-constrained neural networks. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE'22)*. ACM, 31:1–31:13. DOI: https://doi.org/10.1145/3551349.3561158

[15] Simin Chen, Cong Liu, Mirazul Haque, Zihe Song, and Wei Yang. 2022. NMTSloth: Understanding and testing efficiency degradation of neural machine translation systems. In *30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022)*, Abhik Roychoudhury, Cristian Cadar, and Miryung Kim (Eds.). ACM, 1148–1160. DOI: https://doi.org/10.1145/3540250.3549102

[16] Simin Chen, Zihe Song, Mirazul Haque, Cong Liu, and Wei Yang. 2022. NICGSlowDown: Evaluating the efficiency robustness of neural image caption generation models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'22)*. IEEE, 15344–15353. DOI: https://doi.org/10.1109/CVPR52688.2022.01493

[17] Yiming Chen, Simin Chen, Zexin Li, Wei Yang, Cong Liu, Robby T. Tan, and Haizhou Li. 2023. Dynamic transformers provide a false sense of efficiency. In *61st Annual Meeting of the Association for Computational Linguistics (ACL'23)*, Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, 7164–7180. DOI: https://doi.org/10.18653/V1/2023.ACL-LONG.395

[18] Minhao Cheng, Jinfeng Yi, Pin-Yu Chen, Huan Zhang, and Cho-Jui Hsieh. 2020. Seq2Sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. In *34th AAAI Conference on Artificial Intelligence (AAAI'20), 32nd Innovative Applications of Artificial Intelligence Conference (IAAI'20), 10th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI'20)*. AAAI Press, 3601–3608. DOI: https://doi.org/10.1609/AAAI.V34I04.5767

[19] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Y. Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models. *CoRR* abs/2210.11416 (2022).

[20] Javid Ebrahimi, Daniel Lowd, and Dejing Dou. 2018. On adversarial examples for character-level neural machine translation. In *27th International Conference on Computational Linguistics (COLING'18)*, Emily M. Bender, Leon Derczynski, and Pierre Isabelle (Eds.). Association for Computational Linguistics, 653–663. Retrieved from https://aclanthology.org/C18-1055/

[21] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. HotFlip: White-box adversarial examples for text classification. In *56th Annual Meeting of the Association for Computational Linguistics (ACL'18)*, Iryna Gurevych and Yusuke Miyao (Eds.). Association for Computational Linguistics, 31–36. DOI: https://doi.org/10.18653/V1/P18-2006

[22] Andreas Eisele and Yu Chen. 2010. MultiUN: A multilingual corpus from United Nation documents. In *International Conference on Language Resources and Evaluation (LREC'10)*, Nicoletta Calzolari, Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias (Eds.). European Language Resources Association. Retrieved from http://www.lrec-conf.org/proceedings/lrec2010/summaries/686.html

[23] Reuben Feinman, Ryan R. Curtin, Saurabh Shintre, and Andrew B. Gardner. 2017. Detecting adversarial samples from artifacts. *CoRR* abs/1703.00410 (2017).

[24] Michael Figurnov, Maxwell D. Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry P. Vetrov, and Ruslan Salakhutdinov. 2017. Spatially adaptive computation time for residual networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'17)*. IEEE Computer Society, 1790–1799. DOI: https://doi.org/10.1109/CVPR.2017.194

[25] Daniel Fojo, Víctor Campos, and Xavier Giró-i-Nieto. 2018. Comparing fixed and adaptive computation time for recurrent neural networks. In *6th International Conference on Learning Representations (ICLR'18)*. OpenReview.net. Retrieved from https://openreview.net/forum?id=SkZq3vyDf

[26] Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. 2023. InCoder: A generative model for code infilling and synthesis. In *11th International Conference on Learning Representations (ICLR'23)*. OpenReview.net. Retrieved from https://openreview.net/pdf?id=hQwb-lbM6EL

[27] Kuofeng Gao, Yang Bai, Jindong Gu, Shu-Tao Xia, Philip Torr, Zhifeng Li, and Wei Liu. 2024. Inducing high energy-latency of large vision-language models with verbose images. In *12th International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=BteuUysuXX

[28] A. Shaji George and A. S. Hovan George. 2023. A review of ChatGPT AI's impact on several business sectors. *Partn. Univ. Int. Innov. J.* 1, 1 (2023), 9–23.
[29] Google. 2022. Retrieved from https://huggingface.co/t5-small
[30] Shashij Gupta. 2020. Machine translation testing via pathological invariance. In *42nd International Conference on Software Engineering (ICSE'20)*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 107–109. DOI: https://doi.org/10.1145/3377812.3382162
[31] Mirazul Haque, Anki Chauhan, Cong Liu, and Wei Yang. 2020. ILFO: Adversarial attack on adaptive neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'20)*. Computer Vision Foundation/IEEE, 14252–14261. DOI: https://doi.org/10.1109/CVPR42600.2020.01427
[32] Pinjia He. 2022. RobustNLP Library. Retrieved from https://github.com/RobustNLP/TestTranslation
[33] Pinjia He, Clara Meister, and Zhendong Su. 2020. Structure-invariant testing for machine translation. In *42nd International Conference on Software Engineering (ICSE'20)*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 961–973. DOI: https://doi.org/10.1145/3377811.3380339
[34] Pinjia He, Clara Meister, and Zhendong Su. 2021. Testing machine translation via referential transparency. In *43rd IEEE/ACM International Conference on Software Engineering (ICSE'21)*. IEEE, 410–422. DOI: https://doi.org/10.1109/ICSE43902.2021.00047
[35] Helsinki-NLP. 2022. Retrieved from https://huggingface.co/Helsinki-NLP/opus-mt-en-de
[36] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training compute-optimal large language models. *CoRR* abs/2203.15556 (2022).
[37] Sanghyun Hong, Yigitcan Kaya, Ionut-Vlad Modoranu, and Tudor Dumitras. 2021. A panda? No, it's a sloth: Slowdown attacks on adaptive multi-exit neural network inference. In *9th International Conference on Learning Representations (ICLR'21)*. OpenReview.net. Retrieved from https://openreview.net/forum?id=9xC2tWEwBD
[38] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. *CoRR* abs/1704.04861 (2017).
[39] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Vol. 112. Springer.
[40] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is BERT really robust? A strong baseline for natural language attack on text classification and entailment. In *34th AAAI Conference on Artificial Intelligence (AAAI'20), 32nd Innovative Applications of Artificial Intelligence Conference (IAAI'20), 10th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI'20)*. AAAI Press, 8018–8025. DOI: https://doi.org/10.1609/AAAI.V34I05.6311
[41] Mintong Kang, Nezihe Merve Gürel, Ning Yu, Dawn Song, and Bo Li. 2024. C-rag: Certified generation risks for retrievalaugmented language models. *arXiv preprint arXiv:2402.03181*.
[42] Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. Large language models are few-shot testers: Exploring LLM-based general bug reproduction. In *45th IEEE/ACM International Conference on Software Engineering (ICSE'23)*. IEEE, 2312–2323. DOI: https://doi.org/10.1109/ICSE48619.2023.00194
[43] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *41st International Conference on Software Engineering (ICSE'19)*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE/ACM, 1039–1049. DOI: https://doi.org/10.1109/ICSE.2019.00108
[44] Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2019. TextBugger: Generating adversarial text against real-world applications. In *26th Annual Network and Distributed System Security Symposium (NDSS'19)*. The Internet Society. Retrieved from https://www.ndss-symposium.org/ndss-paper/textbugger-generating-adversarial-text-against-real-world-applications/
[45] Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. 2020. BERT-ATTACK: Adversarial attack against BERT using BERT. In *Conference on Empirical Methods in Natural Language Processing (EMNLP'20)*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 6193–6202. DOI: https://doi.org/10.18653/V1/2020.EMNLP-MAIN.500
[46] Yujia Li, David H. Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, R'emi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with alphacode. *CoRR*, abs/2203.07814.
[47] Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Sal Candido, and Alexander Rives. 2022. Language models of protein sequences at the scale of evolution enable accurate structure prediction. *BioRxiv*, 2022:500902, 2022.

[48] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation. In *Annual Conference on Neural Information Processing Systems (NeurIPS'23)*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). Retrieved from http://papers.nips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html

[49] Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pre-training for neural machine translation. *Trans. Assoc. Comput. Ling.* 8 (2020), 726–742. DOI: https://doi.org/10.1162/TACL_A_00343

[50] Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, Zihao Wu, Lin Zhao, Dajiang Zhu, Xiang Li, Ning Qiang, Dinggang Shen, Tianming Liu, and Bao Ge. 2023. Summary of chatgpt-related research and perspective towards the future of large language models. *Meta-Radiology*, 1, 2 (2023), 100017.

[51] Alexandre Lopes, Rodrigo Frassetto Nogueira, Roberto de Alencar Lotufo, and Hélio Pedrini. 2020. Lite training strategies for Portuguese-English and English-Portuguese translation. In *5th Conference on Machine Translation (WMT@EMNLP'20)*, Loïc Barrault, Ondrej Bojar, Fethi Bougares, Rajen Chatterjee, Marta R. Costa-Jussà, Christian Federmann, Mark Fishel, Alexander Fraser, Yvette Graham, Paco Guzman, Barry Haddow, Matthias Huck, Antonio Jimeno-Yepes, Philipp Koehn, André Martins, Makoto Morishita, Christof Monz, Masaaki Nagata, Toshiaki Nakazawa, and Matteo Negri (Eds.). Association for Computational Linguistics, 833–840. Retrieved from https://aclanthology.org/2020.wmt-1.90/

[52] MarianMT. 2023. Marianmt: translation_en-zh. https://huggingface.co/DDDSSS/translation_en-zh

[53] Jesse G. Meyer, Ryan J. Urbanowicz, Patrick C. N. Martin, Karen O'Connor, Ruowang Li, Pei-Chen Peng, Tiffani J. Bright, Nicholas P. Tatonetti, Kyoung-Jae Won, Graciela Gonzalez-Hernandez, and Jason H. Moore. 2023. ChatGPT and large language models in academia: Opportunities and challenges. *BioData Min.* 16, 1 (2023). DOI: https://doi.org/10.1186/S13040-023-00339-9

[54] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2021. WebGPT: Browser-assisted question-answering with human feedback. *CoRR* abs/2112.09332 (2021).

[55] Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook FAIR's WMT19 news translation task submission. In *4th Conference on Machine Translation (WMT'19)*, Ondrej Bojar, Rajen Chatterjee, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno-Yepes, Philipp Koehn, André Martins, Christof Monz, Matteo Negri, Aurélie Névéol, Mariana L. Neves, Matt Post, Marco Turchi, and Karin Verspoor (Eds.). Association for Computational Linguistics, 314–319. DOI: https://doi.org/10.18653/V1/W19-5333

[56] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. CodeGen: An open large language model for code with multi-turn program synthesis. In *11th International Conference on Learning Representations (ICLR'23)*. OpenReview.net. Retrieved from https://openreview.net/pdf?id=iaYcJKpY2B_

[57] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Annual Conference on Neural Information Processing Systems 2022 (NeurIPS'22)*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.). Retrieved from http://papers.nips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html

[58] Papers with Code. 2024. Code generation on mbpp. https://paperswithcode.com/sota/code-generation-on-mbpp

[59] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2019. DeepXplore: Automated whitebox testing of deep learning systems. *Commun. ACM* 62, 11 (2019), 137–145. DOI: https://doi.org/10.1145/3361566

[60] Jeff Pitman. 2021. Google Translate: One billion installs, one billion stories. Retrieved from https://blog.google/products/translate/one-billion-installs/

[61] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1, 8 (2019), 9.

[62] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67. Retrieved from http://jmlr.org/papers/v21/20-074.html

[63] Partha Pratim Ray. 2023. Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems* 3 (2023), 121–154.

[64] Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating natural language adversarial examples through probability weighted word saliency. In *57th Conference of the Association for Computational Linguistics (ACL'19)*, Anna Korhonen, David R. Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, 1085–1097. DOI: https://doi.org/10.18653/V1/P19-1103

[65] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code Llama: Open foundation models for code. *CoRR* abs/2308.12950 (2023).

[66] Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M. Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal V. Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Févry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M. Rush. 2022. Multitask prompted training enables zero-shot task generalization. In *10th International Conference on Learning Representations (ICLR'22)*. OpenReview.net. Retrieved from https://openreview.net/forum?id=9Vrb9D0WI4

[67] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilic, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, et al. 2022. BLOOM: A 176B-parameter open-access multilingual language model. *CoRR* abs/2211.05100 (2022).

[68] Vesa Siivola and Bryan L. Pellom. 2005. Growing an n-gram language model. In *9th European Conference on Speech Communication and Technology (INTERSPEECH'05—Eurospeech)*. ISCA, 1309–1312. DOI: https://doi.org/10.21437/INTERSPEECH.2005-24

[69] Zeyu Sun, Jie M. Zhang, Mark Harman, Mike Papadakis, and Lu Zhang. 2020. Automatic testing and improvement of machine translation. In *42nd International Conference on Software Engineering (ICSE'20)*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 974–985. DOI: https://doi.org/10.1145/3377811.3380420

[70] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Annual Conference on Neural Information Processing Systems (NeurIPS'14)*, Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger (Eds.). 3104–3112. Retrieved from https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html

[71] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In *40th International Conference on Software Engineering (ICSE'18)*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 303–314. DOI: https://doi.org/10.1145/3180155.3180220

[72] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and efficient foundation language models. *CoRR* abs/2302.13971 (2023).

[73] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR* abs/2307.09288 (2023).

[74] Barak Turovsky. 2016. Ten years of Google Translate. Retrieved from https://www.blog.google/products/translate/ten-years-of-google-translate/

[75] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Annual Conference on Neural Information Processing Systems*

*(NeurIPS'17)*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. Retrieved from https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[76] Guancheng Wang, Ruobing Shen, Junjie Chen, Yingfei Xiong, and Lu Zhang. 2021. Probabilistic delta debugging. In *29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'21)*, Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta (Eds.). ACM, 881–892. DOI: https://doi.org/10.1145/3468264.3468625

[77] Huiyan Wang, Jingwei Xu, Chang Xu, Xiaoxing Ma, and Jian Lu. 2020. Dissector: Input validation for deep learning applications by crossing-layer dissection. In *42nd International Conference on Software Engineering (ICSE'20)*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 727–738. DOI: https://doi.org/10.1145/3377811.3380379

[78] Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. 2019. Adversarial sample detection for deep neural network through model mutation testing. In *41st International Conference on Software Engineering (ICSE'19)*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE/ACM, 1245–1256. DOI: https://doi.org/10.1109/ICSE.2019.00126

[79] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. 2018. SkipNet: Learning dynamic routing in convolutional networks. In *15th European Conference on Computer Vision (ECCV'18) (Lecture Notes in Computer Science,* Vol. 11217*)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer, 420–436. DOI: https://doi.org/10.1007/978-3-030-01261-8_25

[80] Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP'21)*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 8696–8708. DOI: https://doi.org/10.18653/V1/2021.EMNLP-MAIN.685

[81] Elaine J. Weyuker and Filippos I. Vokolos. 2000. Experience with performance testing of software systems: Issues, an approach, and case study. *IEEE Trans. Softw. Eng.* 26, 12 (2000), 1147–1156. DOI: https://doi.org/10.1109/32.888628

[82] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 38–45.

[83] Minghao Wu, Abdul Waheed, Chiyu Zhang, Muhammad Abdul-Mageed, and Alham Fikri Aji. 2024. LaMini-LM: A diverse herd of distilled models from large-scale instructions. In *18th Conference of the European Chapter of the Association for Computational Linguistics (EACL'24)*, Yvette Graham and Matthew Purver (Eds.). Association for Computational Linguistics, 944–964. Retrieved from https://aclanthology.org/2024.eacl-long.57

[84] Guangxuan Xiao, Ji Lin, Mickaël Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning (ICML'23) (Proceedings of Machine Learning Research,* Vol. 202*)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 38087–38099. Retrieved from https://proceedings.mlr.press/v202/xiao23c.html

[85] Weilin Xu, David Evans, and Yanjun Qi. 2018. Feature squeezing: Detecting adversarial examples in deep neural networks. In *25th Annual Network and Distributed System Security Symposium (NDSS'18)*. The Internet Society. Retrieved from https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_03A-4_Xu_paper.pdf

[86] Zhen Yang, Wei Chen, Feng Wang, and Bo Xu. 2018. Generative adversarial training for neural machine translation. *Neurocomputing* 321 (2018), 146–155. DOI: https://doi.org/10.1016/J.NEUCOM.2018.09.006

[87] Liuyi Yao, Zhixuan Chu, Sheng Li, Yaliang Li, Jing Gao, and Aidong Zhang. 2021. A survey on causal inference. *ACM Trans. Knowl. Discov. Data* 15, 5 (2021), 74:1–74:46. DOI: https://doi.org/10.1145/3444944

[88] Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. 2020. Word-level textual adversarial attacking as combinatorial optimization. In *58th Annual Meeting of the Association for Computational Linguistics (ACL'20)*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 6066–6080. DOI: https://doi.org/10.18653/V1/2020.ACL-MAIN.540

[89] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *57th Conference of the Association for Computational Linguistics (ACL'19)*, Anna Korhonen, David R. Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, 4791–4800. DOI: https://doi.org/10.18653/V1/P19-1472

[90] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open pre-trained transformer language models. *CoRR* abs/2205.01068 (2022).

[91] Xinze Zhang, Junzhe Zhang, Zhenhua Chen, and Kun He. 2021. Crafting adversarial examples for neural machine translation. In *59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL/IJCNLP'21)*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, 1967–1977. DOI: https://doi.org/10.18653/V1/2021.ACL-LONG.153

[92] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'18)*. Computer Vision Foundation/IEEE Computer Society, 6848–6856. DOI: https://doi.org/10.1109/CVPR.2018.00716

[93] Zhengyan Zhang, Yuxian Gu, Xu Han, Shengqi Chen, Chaojun Xiao, Zhenbo Sun, Yuan Yao, Fanchao Qi, Jian Guan, Pei Ke, Yanzheng Cai, Guoyang Zeng, Zhixing Tan, Zhiyuan Liu, Minlie Huang, Wentao Han, Yang Liu, Xiaoyan Zhu, and Maosong Sun. 2021. CPM-2: Large-scale cost-effective pre-trained language models. *AI Open* 2 (2021), 216–224. DOI: https://doi.org/10.1016/J.AIOPEN.2021.12.003

[94] Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Lei Shen, Zihan Wang, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. 2023. CodeGeeX: A pre-trained model for code generation with multilingual benchmarking on HumanEval-X. In *29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'23)*, Ambuj K. Singh, Yizhou Sun, Leman Akoglu, Dimitrios Gunopulos, Xifeng Yan, Ravi Kumar, Fatma Ozcan, and Jieping Ye (Eds.). ACM, 5673–5684. DOI: https://doi.org/10.1145/3580305.3599790

[95] Wei Zou, Shujian Huang, Jun Xie, Xinyu Dai, and Jiajun Chen. 2020. A reinforced generation of adversarial examples for neural machine translation. In *58th Annual Meeting of the Association for Computational Linguistics (ACL'20)*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 3486–3497. DOI: https://doi.org/10.18653/V1/2020.ACL-MAIN.319

Received 9 December 2023; revised 28 April 2024; accepted 2 May 2024