NMTSloth: Understanding and Testing Efficiency Degradation of Neural Machine Translation Systems

Simin Chen
simin.chen@UTDallas.edu
UT Dallas
Dallas, USA

Cong Liu
congl@ucr.edu
UT Dallas
Dallas, USA

Mirazul Haque
mirazul.haque@utdallas.edu
UT Dallas
Dallas, USA

Zihe Song
zihe.song@utdallas.edu
UT Dallas
Dallas, USA

Wei Yang
wei.yang@utdallas.edu
UT Dallas
Dallas, USA

ABSTRACT

Neural Machine Translation (NMT) systems 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 NMT systems, which is of paramount importance due to often vast translation demands and real-time requirements, has surprisingly received little attention. In this paper, we make the first attempt to understand and test potential computation efficiency robustness in state-of-the-art NMT systems. By analyzing the working mechanism and implementation of 1455 public-accessible NMT systems, we observe a fundamental property in NMT systems 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 NMT systems 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 NMT systems would have to go through enough iterations to satisfy the pre-configured threshold. We present `NMTSloth`, which develops a gradient-guided technique that searches for a minimal and unnoticeable perturbation at character-level, token-level, and structure-level, which sufficiently delays the appearance of EOS and forces these inputs to reach the naturally-unreachable threshold. To demonstrate the effectiveness of `NMTSloth`, we conduct a systematic evaluation on three public-available NMT systems: Google T5, AllenAI WMT14, and Helsinki-NLP translators. Experimental results show that `NMTSloth` can increase NMT systems' response latency and energy consumption by 85% to 3153% and 86% to 3052%, respectively, by perturbing just one character or token in the input sentence. Our case study shows that inputs generated by `NMTSloth` significantly affect the battery power in real-world mobile devices (*i.e.,* drain more than 30 times battery power than normal inputs).

CCS CONCEPTS

• **Software and its engineering → Software notations and tools**; • **Computing methodologies → Machine learning**.

KEYWORDS

Machine learning, software testing, neural machine translation

ACM Reference Format:
Simin Chen, Cong Liu, Mirazul Haque, Zihe Song, and Wei Yang. 2022. `NMTSloth`: Understanding and Testing Efficiency Degradation of Neural Machine Translation Systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3540250.3549102

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9413-0/22/11…$15.00
https://doi.org/10.1145/3540250.3549102

1 INTRODUCTION

Neural Machine Translation (NMT) is a promising approach that applies neural networks to resolve machine translation problems. NMT systems have received significant recent attention from both academia [2, 3, 29, 43] and industry [5, 14, 17–19, 35, 42], due to its advantages over traditional translation methods (*e.g.,* phrase-based translation models [30]). For instance, due to being capable of capturing rather long dependencies in sentences, NMT systems are seeing a wide adoption in commercial translation systems including Microsoft's translation products [14, 17–19] and Google Translate [5, 35, 42].

Much research has been done on enhancing the accuracy of NMT systems [36, 36, 43]. Recently, research [15, 21, 22, 39] has been conducted to understand the accuracy robustness of existing NMT systems by developing a series of adversarial test input generation frameworks that reduce the translation accuracy of existing NMT systems. While accuracy robustness is clearly important, we observe that the computation efficiency of NMT systems, particularly in terms of the latency and energy spent on translating an input with a specific length, is an equivalently critical property that has surprisingly received little attention. A common and unique characteristic of the machine translation domain is the need to process a huge amount of real-time requests (*e.g.,* Google Translate claims to have been translating over 100 billion words daily in 109 languages [5, 35, 42]). The vast demand for translation requests combined with the real-time requirements naturally makes the computation efficiency of any NMT system be one of the most critical optimization goals. In this paper, we make the first attempt in understanding and testing potential vulnerabilities in terms of computation efficiency of existing NMT systems.

Key observations revealing vulnerabilities on NMT computation efficiency. Our findings are motivated by several observations. Particularly, through analyzing the working mechanisms and detailed implementation of 1,455 public-accessible NMT systems (*e.g.,* Google T5 [12, 37], Meta [34]), we observe a fundamental property of NMT systems that could be manipulated in an adversarial manner to significantly reduce computation efficiency. Specifically, we observe that the computation efficiency of NMT systems 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 Fig. 2, inserting a character "b" in token "Genäckstück" will increase the latency of HuggingFace's NMT systems from 0.876s to 20.382s, representing an over 20× latency increase). Such dramatic impact on computation efficiency may occur fundamentally because NMT systems often need to invoke the underlying decoder with non-deterministic numbers of iterations to generate outputs [33, 43]. Intuitively, the computation efficiency of NMT systems 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 3, a dominant number of our studied NMT systems set this threshold to be 500-600, 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 NMT systems would have to go through max iterations to satisfy the pessimistic pre-configured threshold.

This implies an important yet unexplored vulnerability of NMT systems: adversarially-designed inputs that may cause enormous, abnormal computation demand in existing NMT systems, 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 machine translation service providers (*e.g., HuggingFace* [48]) 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 NMT systems. 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 NMT 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 `NMTSloth` that effectively achieves our objectives. `NMTSloth` is developed based on the aforementioned observation. Specifically, NMT systems iteratively compute the output token until either the system generates an end-of-sentence (EOS) token or a pre-configured threshold controlling the max number of iterations has been met. For our studied 1455 NMT systems [1], the appearance of EOS is computed from the underlying DNNs output probability. `NMTSloth` 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. `NMTSloth` further develops a gradient-guided technique that searches for a minimal perturbation (including both character-level, token-level, and structure-level ones) that can effectively delay the generation of EOS. Applying this minimal perturbation on the seed input would result in significantly longer output, costing NMT systems more computational resources and thus reducing computation efficiency.

Implementation and evaluation. We have conducted extensive experiments to evaluate the effectiveness of `NMTSloth`. Particularly, we applied `NMTSloth` on three real-world public-available and widely used (*e.g.,* with more than 592,793 downloads in Jan 2022) NMT systems (*i.e.,* Google T5 [12, 37], AllenAI WMT14 [1], and Helsinki-NLP [23]). The selected NMT systems are trained with different corpus and feature diverse DNN architectures as well as various configurations. We compare `NMTSloth` against four state-of-the-art methods that focus on testing NMT systems' accuracy and correctness. Evaluation results show that `NMTSloth` is highly effective in generating test inputs to degrade computation efficiency of the NMT systems under test. Specifically, `NMTSloth` generates test inputs that could increase the NMT systems' CPU latency, CPU energy consumption, GPU latency, and GPU energy consumption by 85% to 3153%, 86% to 3052%, 76% to 1953%, and 68% to 1532%, respectively, through only perturbing one character or token in any seed input sentences. Our case study shows that inputs generated by `NMTSloth` significantly affect the battery power in real-world mobile devices (*i.e.,* drain more than 30 times battery power than normal inputs).

Contribution. Our contribution are summarized as follows:

- Characterization: We are the first to study and characterize the computation efficiency vulnerability in state-of-the-art NMT systems, 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 1,455 public-available NMT systems, which have been downloaded more than 8,286,413 times in Jan/2022. The results show that the revealed vulnerability could widely exist due to a fundamental property of NMT systems.
- Approach: We design and implement `NMTSloth`, the first framework for testing NMT systems' computation efficiency. Specifically, given a seed input, `NMTSloth` applies a gradient-guided approach to mutate the seed input to generate test inputs. Test inputs generated by `NMTSloth` only perturb one to three tokens in any seed inputs.

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

- Evaluation: We evaluate NMTSloth on three real-world public-available NMT systems (*i.e.,* Google T5, AllenAI WMT14, and Helsinki-NLP) against four correctness-based testing methods. In addition, we propose a series of metrics (Eq.(4)) 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, NMTSloth generates test inputs that increase NMT systems' latency and energy consumption by 85% to 3153% and 86% to 3052%, respectively.
- Mitigation: We propose one 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.

2 BACKGROUND

2.1 Neural Machine Translation Systems

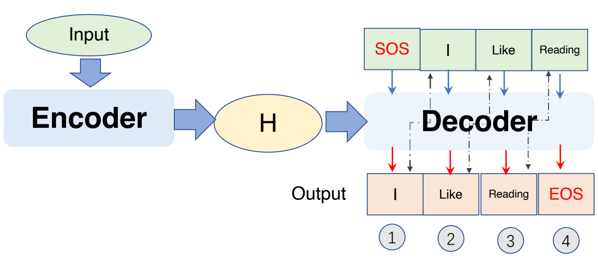

Figure 1: Working mechanism of NMT systems

Much recent research has been done towards developing more accurate and efficient machine translation systems [5, 33, 35, 40, 42, 43, 43]. The fundamental of machine translation systems is the language model, which computes the conditional probability $P(Y|X)$, where $X = [x_1, x_2, \cdots, x_m]$ is the input token sequence and $Y = [y_1, y_2, \cdots, y_n]$ is the output token sequence. Modern NMT systems apply the neural networks to approximate such conditional probability $P(Y|X)$. As shown in Fig. 1, a typical NMT system consists of an encoder $f_{en}(\cdot)$ and a decoder $f_{de}(\cdot)$. The encoder encodes the source input X into hidden representation H, then H is feed into the decoder for decoding. An implementation example of NMT systems' 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 NMT systems' 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.

[2]The code snippet is downloaded from PyTorch NMT tutorial

```
'''
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 of NMT Systems Implementation

2.2 Testing NMT Systems

Although modern NMT systems demonstrate human-level performance in terms of accuracy, NMT systems are still far from robust due to the complexity and intractability of the underlying neural networks. To improve the robustness of NMT 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 [3, 8, 9, 31, 52] 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 [7, 31, 38, 39, 49, 50] 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 [15, 21, 22, 32] seeks to generate legal sentences that do not contain lexical or syntactic errors. For example, [21] proposes a structure invariant testing method to perturb seed inputs with Bert [27], 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 NMT systems. We then present a comprehensive empirical study based on 1455 state-of-the-art NMT systems, which reveals an important vulnerability in existing NMT systems that may suffer from significant efficiency degradation.

3.1 Motivating Example

Fig. 2 illustrates the efficiency degradation issue that HuggingFace NMT API [3] may experience due to unnoticeable perturbations. This selected NMT API is rather popular among developers, with 136,902 downloads merely in Jan 2022. Fig. 2 shows the computation time using two input sentences, where a normal (abnormal) input is used

[3]https://huggingface.co/Helsinki-NLP/opus-mt-de-en

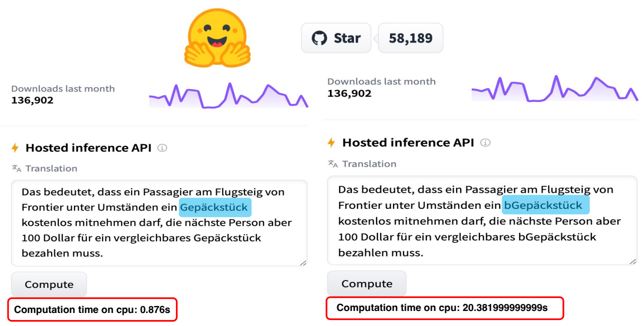

Figure 2: Example illustrating NMT systems' efficiency degradation by inserting one character (using HuggingFace API)

in the left (right) sub-figure. Note that the abnormal input differs from the normal input by only one character "b" (highlighted in blue). Nonetheless, due to such a one-character difference in the input, the computation time increases from 0.876s to 20.382s (a 2226.7% increase). This real-world example reveals that state-of-the-art NMT systems may have critical yet unrevealed vulnerabilities that negatively impact computation efficiency.

As we discussed in Sec. 2.1, the working mechanism of NMT systems 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, NMT systems 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 Fig. 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 translation corpus. According to our empirical study of 1,455 NMT systems (detailed in 3.2), 1,370 of them set `max_length` within a range of 500 to 600. To further understand why this intuitive approach does not work, we conduct a comprehensive empirical study using 1455 state-of-the-art NMT systems. Specifically, we focus on answering the following two research questions.

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

3.2 Current Engineering Configurations

3.2.1 Study Methodology. We investigate the configurations of existing mainstream NMT systems. Specifically, we studies 1,455 NMT systems (*e.g.,* Google T5, Meta FairSeq) from HuggingFace online NMT service [4]. HuggingFace is a commercial platform that provides third-party real-time translation service, which covers almost all NMT model architectures. NMT systems on the HuggingFace platform are open-source and widely used by public, as shown in Table 1 (*e.g.,* the most popular NMT systems in HuggingFace have been downloaded for more than 3,141,480 times in Jan 2022). HuggingFace provides high-level abstraction API for NMT service. List 2 shows code snippets of using HuggingFace API to load Google

[4]https://huggingface.co/

Table 1: Top 10 popular NMT systems on HuggingFace website (the order is based on the number of downloads)

Rank	Model Name	max_length	# of Downloads
1	Helsinki-NLP/opus-mt-zh-en	512	3141840
2	Google/t5-base	300	1736544
3	Helsinki-NLP/opus-mt-en-de	512	749228
4	Helsinki-NLP/opus-mt-en-ROMANCE	512	599267
5	Google/t5-small	300	592793
6	Helsinki-NLP/opus-mt-ar-en	512	196033
7	Helsinki-NLP/opus-mt-de-en	512	129923
8	Helsinki-NLP/opus-mt-es-en	512	111028
9	Helsinki-NLP/opus-mt-ROMANCE-en	512	92987
10	Helsinki-NLP/opus-mt-fr-en	512	91552

T5 translation service. All NMT model classes are inherited from a common parent class, `GenerationMixin`, which contains all functions supporting sentence translation. We parse the source code of the `GenerationMixin.generate` function and observe that the translation flow could be divided into nine parts. Among all nine parts, we find that the eighth part determines 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 NMT systems' 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 NMT models from HuggingFace's API services [5] and parse each NMT model's configuration file to check how `max_length` and `max_time` have been set.

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

Listing 2: HuggingFace libraries high-level translation 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 translation

3.2.2 Study Results. Among all 1, 455 NMT systems, we successfully collect 1, 438 configuration files, where 1, 400 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 Fig. 3. We have the following two observations. First, there is a significant variance in the `max_length` value (ranging from 20 to 1024); Second, almost all NMT systems (97.86%) set the `max_length` to be from 500 to 600, *i.e.,* maximum 500-600 decoder calls. Note that real-world NMT systems prefer to set such a large threshold just to prevent unresponsiveness (*e.g.,* dead-loop). However, in most cases

[5]https://huggingface.co/models?pipeline_tag=translation&sort=downloads

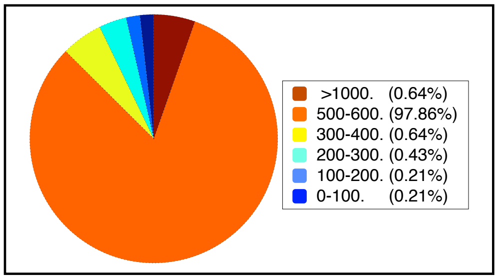

Figure 3: The distribution of `max_length` values

Table 2: Statistical results of efficiency differences in machine translation (1%, 10%, 50%, 90%, 100% represent quantile)

Language Src	Language Tgt	# of pairs	Quantile of Target Length 10%	50%	90%	100% (max)	Quantile of Length Ratio 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

with normal inputs, such a threshold will not yield any real impact as the EOS token often appears much earlier.

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 [10] as the subject in our empirical study because of the following criteria: *(i)* the datasets are open-source and public-available; *(ii)* the datasets are widely studied in recent works (with more than 1,000 citations until Jan 2022); *(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 [6]). Column "Target Length" shows the target sentence length under different quantiles, and Column "Target and Source 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 result, setting a small `max_length` field will lead to low-precision translation results. For instance, in the last line of Table 2, *i.e.,* translating `zh` to `ru`, if setting `max_length` to 42, at least 50% of data will not be translated 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 translation while not incurring efficiency issues for any ordinary inputs due to the EOS mechanism.

[6] https://github.com/SeekingDream/NMTSloth

4 PROBLEM FORMULATION

Our goal is to generate test inputs that can degrade computation efficiency of NMT systems. Our proposed method seeks to perturb a seed sentence to craft test inputs. The perturbed test inputs will incur significantly long computation time, thus impairing user experience and even cause 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 Sec. 2, NMT systems' computation efficiency depends on the output length, where a lengthier output implies less computation efficiency. Thus, our goal can be achieved through increasing NMT systems' 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 NMT system under test, ϵ is the maximum allowed perturbation, and $||f_{de}(\cdot)||$ measures the number of times of NMT's decoders being called. Our proposed `NMTSloth` tries to search a perturbation Δ that maximizes the decoders' calling times (decreasing target NMT systems efficiency) within a minimum allowable perturbation threshold (which ensures unnoticeable perturbations).

5 METHODOLOGY

We now present `NMTSloth` and provides three specific implementations including character-level perturbation, token-level perturbation, and structure-level perturbation.

5.1 Design Overview

`NMTSloth` is an iterative approach. During each iteration, `NMTSloth` perturbs one token in a seed sentence with different types of perturbations. An overview of the detailed procedure of each iteration is illustrated in Fig. 4, which contains three major steps:

(1) *Finding critical tokens.* For each seed sentence, we feed it to the NMT system under test and apply a gradient-based approach to search for the critical tokens that have the highest impact on NMT systems' computation 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 NMT systems and detect any efficiency degradation.

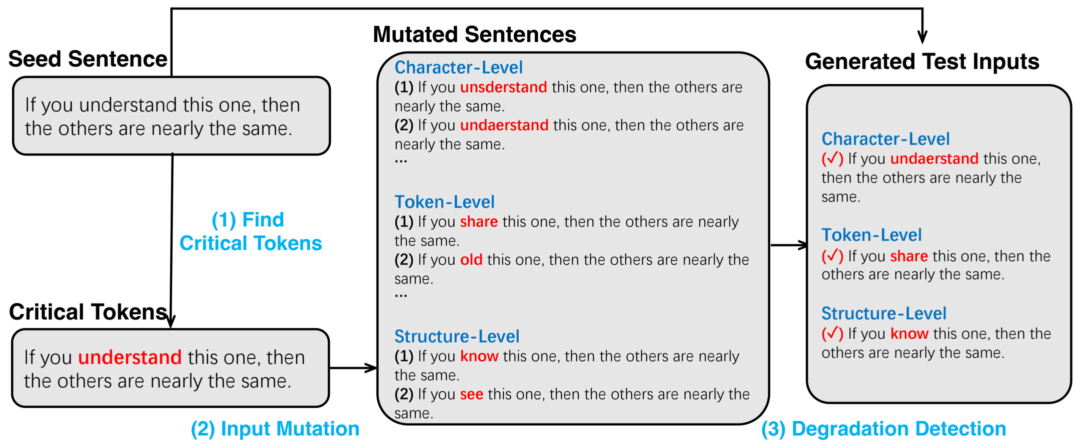

Figure 4: Design overview of `NMTSloth`

5.2 Detail Design

Finding Critical Tokens: Given a seed sentence $x = [tk_1, \cdots, tk_m]$, the first step is to identify tokens that are critical to NMT systems' efficiency. As we discussed earlier, NMT systems' computation efficiency depends on the corresponding output length given any input, which is determined by the pre-configured threshold and the EOS token. In Sec. 3, we showed that the pre-configured threshold is set as a fixed value in the configuration files of NMT systems. 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 NMT systems' computation efficiency.

Formally, let NMT system's output probability be a sequence of vectors, *i.e.,* $[p_1, p_2, \cdots, p_n]$, and the probability of EOS token appearance be $[p_1^{eos}, p_2^{eos}, \cdots, p_n^{eos}]$. 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. Thus, we define the importance score of token tk_i as g_i, shown in (2).

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

where $[o_1, o_2, \cdots, o_n]$ is the current output token, $f(x)$ is the probability we seek to minimize, 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.

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 two kinds of perturbations, *i.e.,*, token-level perturbation and character-level perturbation. Table 3 shows some examples of token-level and character-level perturbations with different perturbation sizes ϵ (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 δ. The character-inset 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 replace increment at token-level. Instead, we enumerate possible δ 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 δ. To compute the target token δ, we define token replace increment $\mathcal{I}_{src,tgt}$ to measure the efficiency degradation of replacing token src to tgt. As shown in (3), $E(\cdot)$ is the function to obtain the corresponding token's embedding, $E(tgt) - E(src)$ is the vector increment in the embedding space. Because $\frac{\partial f(x)}{\partial tk_i^j}$ indicates the sensitivity of output length to each embedding dimension, $\mathcal{I}_{src,tgt}$ denotes the total benefits of replacing token src with tgt. We search the target token δ in the vocabulary to maximize the token replace increment with the source token tk.

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

Table 3: Examples of token-level, character-level, and structure-level perturbation under different size

Original	ϵ	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?

For structure-level perturbation, we follow existing work [21, 39] to parse the seed input sentence as a constituency tree and replace the critical token with another token based on `Bert` [4]. Unlike token-level perturbation, the structure-level perturbation ensures the constituency structure of the perturbed sentence is the same as the seed one. Fig. 5 shows an example of the structure-level perturbation. After replacing the critical token, the constituency tree is the same as the seed one.

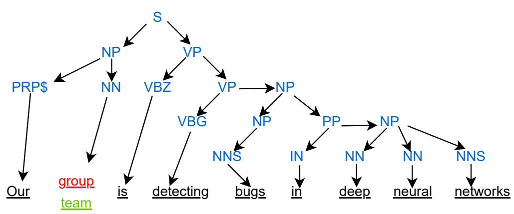

Figure 5: Constituency tree of sentence

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.

6 EVALUATION

We evaluate `NMTSloth` and answer the following research questions.

- ***RQ 2.1 (Severity)***: How severe will `NMTSloth` degrade NMT systems efficiency?
- ***RQ 2.2 (Effectiveness)***: How effective is `NMTSloth` in generating test samples that degrade NMT systems efficiency?
- ***RQ 2.3 (Sensitivity)***: Can `NMTSloth` generate useful test samples that decrease NMT systems efficiency under different NMT systems' configurations?
- ***RQ 2.4 (Overheads)***: What is the overhead of `NMTSloth` in generating test samples?

6.1 Experimental Setup

Models and Datasets. As shown in Table 4, we consider the following three public NMT systems as our evaluation models: Google's T5 [37], AllenAI's WMT14 Transformer [34], and Helsinki-NLP's H-NLP Translator [28]. 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 the NMT models 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.

Comparison Baselines. A branch of existing works have been proposed for testing NMT systems [3, 7, 15, 21, 22, 39]. However,

Table 4: The NMT systems under test in our experiments

Model	Source	Target	Vocab Size	`max_length`
H-NLP	En	De	65,001	512
AllenAi	En	De	42,024	200
T5	En	Zh	32,100	200

all of them focus on testing NMT systems' correctness. To the best of our knowledge, we are the first to study NMT systems' efficiency degradation issue. To show that existing correctness testing methods can not generate test inputs that trigger efficiency degradation for NMT systems. We compare `NMTSloth` against four state-of-the-art correctness testing methods, which are designed to generate testing inputs that produce incorrect translation results. Specifically, we choose `SIT` [21], `TransRepair` [39], `Seq2Sick` [7], and `SynError` [3] as our comparison baselines. `SIT` proposes a structure-invariant testing method, which is a metamorphic testing approach for validating machine translation software. 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 translation errors. `TransRepair` is similar to `SIT`, with a 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 translation outputs that are entirely different from the original outputs. `SynError` is a character-level testing method, which minimizes the NMT system's accuracy (BLUE 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 BLUE score.

Experimental Procedure. We run `NMTSloth` to test the above-mentioned three NMT systems. Given a seed input, `NMTSloth` perturbs the seed input with different types of perturbations. `NMTSloth` has one hyper-parameter (ϵ) that is configurable. In our experiments, we follow existing works [31] and set perturbation size (*i.e.,* ϵ) from 1 to 3, representing different degrees of perturbation. For RQ1 (severity), we measure the percentage of the increased computational resource, in terms of iteration loops, latency, and energy consumption (Eq.(4)), due to the generated test inputs compared to the seed inputs. For RQ2 (effectiveness), we measure the degradation success ratio (Eq.(5)), which quantifies the percentage of the test inputs out of all generated by `NMTSloth` 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 `NMTSloth` on NMT systems with different configurations to study whether the efficacy of `NMTSloth` is sensitive to configurations. For RQ4 (overheads), we measure the average overheads of running `NMTSloth` to generate test inputs.

Implementation. We implement `NMTSloth` with the `PyTorch` library, using a server with Intel Xeon E5-26 CPU and eight Nvidia 1080Ti GPUs. For the baseline methods, we implement `SIT` and `TransRepair` using the authors' open sourced code [20, 21]. We re-implement `Seq2sick` and `SynError` according to the corresponding papers as the original implementations are not open sourced.

For the NMT models used in our evaluation, we download the pre-trained models using the HuggingFace APIs, and we configure the NMT systems 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 NMT systems' efficiency. The number of iteration loops is a widely used hardware-independent metric for measuring software computational efficiency [47]. More iteration loops imply that more computations are required to be performed to handle an input, representing less efficiency. Response latency and energy consumption are two widely-used hardware-dependent metrics for measuring systems efficiency. Larger latency and energy consumption clearly indicate less efficiency.

$$\begin{aligned}\text{I-Loops} &= \frac{\text{Loops}(x') - \text{Loops}(x)}{\text{Loops}(x)} \times 100\% \\ \text{I-Latency} &= \frac{\text{Latency}(x') - \text{Latency}(x)}{\text{Latency}(x)} \times 100\% \\ \text{I-Energy} &= \frac{\text{Energy}(x') - \text{Energy}(x)}{\text{Energy}(x)} \times 100\%\end{aligned} \tag{4}$$

We use I-Loops, I-Latency, and I-energy to denote number of iteration loops, response latency, and energy consumption respectively. The formal definitions of I-Loops, I-Latency, and I-energy are shown in Eq.(4), where x denotes the seed input and x' represents the perturbed input under NMTSloth, Loops$(\cdot)$, Latency$(\cdot)$ and Energy$(\cdot)$ denote the functions which calculate the average number of iteration loops, latency, and energy consumption, respectively. Larger values of I-Loops, I-Latency, I-energy indicate a more severe efficiency degradation caused by the test inputs generated under NMTSloth. 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 1080Ti GPU, and we measure the energy consumption on CPU and GPU using Intel's RAPL interface and Nvidia's PyNVML library.

Results. The results of degrading NMT systems' efficiency are shown in Table 5, where NMTSloth (C), NMTSloth (T), NMTSloth (S) represent the character-level, token-level, structure-level perturbations, respectively. From the results, we have the following observations: *(i)* For all NMT systems under test, NMTSloth generates test samples that trigger more severe efficiency degradation by a large margin compared to the baseline methods. For instance, NMTSloth generates test inputs that on average increase NMT systems' CPU latency, CPU energy consumption, GPU latency, and GPU energy consumption by 85% to 3153%, 86% to 3052%, 76% to 1953% , and 68% to 1532%, respectively, through only perturbing one character or token in any seed input sentences. However, baseline methods could not effectively impact efficiency, since they are designed to reduce NMT systems' accuracy, not efficiency. *(ii)* With an increased perturbation size, the corresponding test samples generated by NMTSloth effectively degrade the NMT systems' efficiency to a larger degree.

Answers to **RQ2.1**: Test samples generated by NMTSloth significantly degrade NMT systems efficiency in number of iteration loops, latency, and energy consumption.

6.3 RQ2.2: Effectiveness

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

Metrics. We define a metric of degradation success ratio (η) to evaluate the effectiveness of NMTSloth.

$$\eta = \frac{\sum_{x \in \mathcal{X}} \mathbb{I}([\text{Loop}(x') - \text{Loop}(x)] \geq \lambda \times \text{MSE}_{orig})}{||\mathcal{X}||} \times 100\% \tag{5}$$

As shown in Eq.(5), $\mathcal{X}$ is a randomly selected seed input set, Loop(x) is the function that measures the iteration number of NMT systems in handling input x, 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 NMT system given perturbed inputs shall be within λ 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 [41].

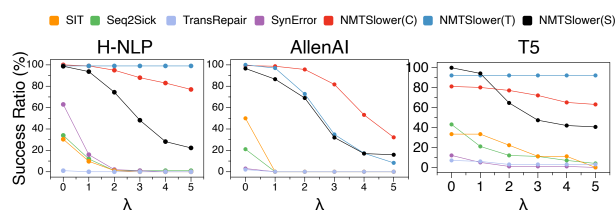

Figure 6: Degradation success ratio under different settings

Results. The results on the degradation successful ratio (η) under different λ values are shown in Fig. 6. We observe that for all experimental settings, NMTSloth outperforms the baseline methods by a significant margin. For example, for H-NLP and $\lambda = 5$, NMTSloth achieves a degradation success ratio of 76% and 98% with token and character level perturbations, respectively; while all the compassion baseline methods' degradation success ratios are below 5%. The results indicate that NMTSloth effectively generates useful test samples to trigger NMT systems' 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 Sec. 3, the natural efficiency variance in the NMT task could be significant, and the degree of extra computations incurred under baseline methods are within the range of natural efficiency variance. As λ grows, η under baseline methods drop quickly. However, this observation does not hold for NMTSloth, where the average degradation success ratio of NMTSloth is still 68.9% when $\lambda = 3$. Recall that from the statistical prospective [26], 99.73% of the inputs will locate in the

Table 5: The Effectiveness Results of Test Samples in Degrading NMT Performance

Subject	Method	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
	NMTSloth (C)	564.45	995.45	1357.77	764.92	1487.92	2015.70	785.60	1471.26	1967.05	462.24	851.80	1116.80	406.39	755.18	972.92
	NMTSloth (T)	2697.77	3735.38	3917.91	3153.97	4481.93	4681.28	3052.62	4544.65	4759.71	1953.57	2729.83	2854.89	1532.91	2137.53	2221.66
	NMTSloth (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
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
	NMTSloth (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
	NMTSloth (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
	NMTSloth (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
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
	NMTSloth (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
	NMTSloth (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
	NMTSloth (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

range of 3MSE_{orig}. Thus, these results clearly show that NMTSloth successfully triggers NMT systems' efficiency degradation.

> Answers to **RQ2.2**: NMTSloth effectively generates test samples that trigger NMT systems' efficiency degradation.

6.4 RQ2.3: Sensitivity

As we introduced in Sec. 2, modern NMT systems 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 Sec. 6.3, we evaluate the effectiveness of NMTSloth under each NMT systems' default num_beams. In this section, we evaluate whether NMTSloth is sensitive to the configuration of num_beams. We configure each NMT system under test with different num_beams (ranging from 1 to 5) and measure the I-Loops of the generated test samples. The experimental results are shown in Fig. 7. From the results, we observe that when the beam search size num_beams is set to 1, the test samples generated by NMTSloth can degrade the NMT systems efficiency slightly more than other beam search size settings. 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 approach can trigger more severe efficiency degradation under this configuration. Importantly, under other configurations where num_beams ranges from 2 to 5, NMTSloth can still trigger NMT systems' efficiency degradation in a stable and severe manner (*e.g.,* T5 requires more than 100% and 300% computations).

> Answers to **RQ2.3**: NMTSloth can generate test samples that degrade NMT systems efficiency under different beam search size configurations. Moreover, the efficiency degradation is more severe when the beam search size is configured as one.

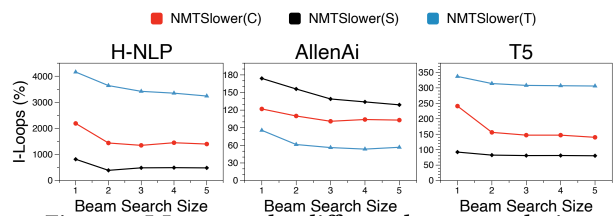

Figure 7: I-Loops under different beam search size

6.5 RQ2.4: Overheads

Table 6 shows the average overhead of NMTSloth in generating a test input, we report only the overhead of NMTSloth because the comparison baselines cannot degrade NMT systems' efficiency. The measured overhead of NMTSloth is rather reasonable (ranging from 7.5s to 106.35s) and may increase linearly as the perturbation size increases. The linearly increasing overheads are due to the fact that NMTSloth is an iterative approach (iteration number equals to ϵ), and the overhead within each iteration is stable. Note that such reasonable overhead is not a concern since perturbed test inputs are generated by NMTSloth offline.

Table 6: Average overheads of NMTSloth (s)

Method	ϵ	H-NLP	AllenAI	T5	Average
NMTSloth (C)	1	11.40	21.14	18.50	17.01
	2	31.80	44.66	45.59	40.68
	3	59.76	69.56	74.48	67.93
NMTSloth (T)	1	7.50	18.45	22.62	16.19
	2	31.41	39.48	61.86	44.25
	3	62.50	62.54	100.01	75.02
NMTSloth (S)	1	10.52	39.19	6.73	18.81
	2	23.33	75.21	17.45	38.66
	3	38.93	106.35	27.71	57.66

Answers to **RQ2.4**: The overheads of `NMTSloth` are reasonable and may increase linearly as the perturbation size increase. Specifically, when $\epsilon = 1$, `NMTSloth` costs 17.01, 16.19, and 18.81 seconds to generate character-level, token-level, and structure-level test samples.

7 DISCUSSION

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

7.1 Real-World Case Study

Table 7: 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.

Experimental Setup. We select Google T5 as our evaluation NMT model in this case study. We first deploy the model on the Samsung Galaxy S9+, which has 6GB RAM and a battery capacity of 3500 mAh. After that, we select one sentence from the dataset `ZH19` as our seed input; we then apply `NMTSloth` 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 7, 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 NMT system and measure the mobile device's battery consumption rate.

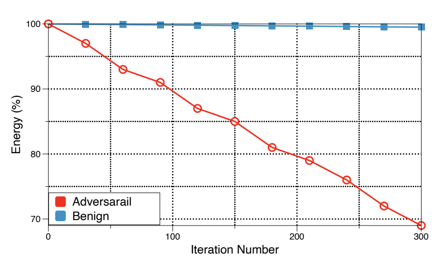

Figure 8: Remaining battery power of the mobile device after T5 translating seed and perturbed sentences

Experimental Results. The mobile phone's battery consumption status is shown in Fig. 8. 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 NMT systems' efficiency robustness.

7.2 Mitigation.

This section shows how developers leverage `NMTSloth` 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 a approach to filter out test inputs that require abnormal computational resources at runtime. Because the abnormal inputs are forced to quit at early stage, thus the computational resources waste are avoided. The idea of applying input validation to improve DNNs correctness robustness has been studied in recent works [44, 45]. However, existing input validation techniques may not be suitable for improving NMT systems 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 [44]. 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. As we introduced in Sec. 2, NMT systems run the encoder once and only once for each input sentence to get the hidden state (*i.e., h* in Fig. 1), we propose to use the output of the encoder as the latent representation to train a lighted-weighted SVM classifier.

Experimental Setup. For each NMT system in our evaluation, we randomly choose 1,000 seed inputs and apply `NMTSloth` to generate 1,000 abnormal inputs for each perturbation types. 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.

Experimental Results. The experimental results are shown in Table 8. Each column in Table 8 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 87.00%. Moreover, the proposed detector's overheads and energy consumption are negligible compared to those incurred under the NMT system. All experimental subjects' extra overheads and the energy consumption are merely at most 1% of the original NMT systems' overheads in translation normal sentences. The results show that our validation-based approach can effectively filter out the abnormal input sentences with negligible overheads.

Table 8: The accuracy and extra overheads of the detector

Subject	Metric (%)	Perturbation Type NMTSloth (C)	NMTSloth (T)	NMTSloth (S)	Mixed
H-NLP	Acc	99.98	99.99	99.98	99.98
	AUC	100.00	100.00	100.00	100.00
	Overheads	0.17	0.32	0.18	0.74
	Energy	0.09	0.17	0.12	0.48
AllenAI	Acc	100.00	100.00	87.00	98.00
	AUC	100.00	100.00	98.32	100.00
	Overheads	0.17	0.08	0.49	0.86
	Energy	0.11	0.05	0.30	0.79
T5	Acc	99.97	100.00	99.99	100.00
	AUC	100.00	100.00	100.00	100.00
	Overheads	0.08	0.06	0.03	0.18
	Energy	0.05	0.04	0.02	0.11

7.3 Threat Analyses.

Our selection of the three NMT systems, namely, Google T5, AllenAI WMT14, and H-NLP, might threaten the *external validity* of our experimental conclusions. We alleviate this threat by the following efforts: (1) the three NMT systems are very popular and have been widely used among developers (with more than 592,793 downloads in Jan 2022); (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 Sec. 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 NMT systems' efficiency degradation with such realistic perturbations. Moreover, all three perturbation types are well studied in related works [8, 9, 15, 21, 22, 38, 39, 49, 50, 52].

8 RELATED WORK

NMT Systems. A detailed overview of recent works on NMT systems and testing NMT systems have been given in Sec. 2.

DNN's Efficiency. Recently, the efficiency of DNNs has raised much concern due to their substantial *inference-time* costs. To improve DNN' *inference-time* efficiency, many existing works have been proposed, categorized into two major techniques. The first category [25, 51] 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 [11, 13, 46], 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 [6, 16, 24] show input-adaptive DNNs are not robustness 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 NMT systems. Specifically, we propose NMTSloth, a framework that introduces imperceptible perturbations to perturb seed inputs to reduce NMT systems' computation efficiency. Evaluation on three public-available NMT systems shows that NMTSloth can generate effective test inputs that may significantly decrease NMT systems' efficiency.

ACKNOWLEDGMENTS

This work was partially supported by Siemens Fellowship and NSF grant CCF-2146443.

REFERENCES

[1] AllenAI. 2022. https://huggingface.co/allenai/wmt16-en-de-dist-12-1. https://huggingface.co/allenai/wmt16-en-de-dist-12-1
[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations, ICLR 2015*. Openreview.net.
[3] Yonatan Belinkov and Yonatan Bisk. 2018. Synthetic and Natural Noise Both Break Neural Machine Translation. In *6th International Conference on Learning Representations, ICLR 2018*. OpenReview.net.
[4] Wieland Brendel, Jonas Rauber, Matthias Kümmerer, Ivan Ustyuzhaninov, and Matthias Bethge. 2019. Accurate, reliable and fast robustness evaluation. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*. 12841–12851.
[5] Isaac Caswell and Bowen Liang. 2020. Recent Advances in Google Translate. https://ai.googleblog.com/2020/06/recent-advances-in-google-translate.html
[6] Simin Chen, Zihe Song, Mirazul Haque, Cong Liu, and Wei Yang. 2022. NICGSlowDown: Evaluating the Efficiency Robustness of Neural Image Caption Generation Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 15365–15374.
[7] 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 *Proceedings of the AAAI Conference on Artificial Intelligence, AAAI 2020*, Vol. 34. 3601–3608.
[8] Javid Ebrahimi, Daniel Lowd, and Dejing Dou. 2018. On Adversarial Examples for Character-Level Neural Machine Translation. In *Proceedings of the 27th International Conference on Computational Linguistics, ACL 2018*. Association for Computational Linguistics, 653–663.
[9] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2018. HotFlip: White-Box Adversarial Examples for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018*. Association for Computational Linguistics, 31–36.
[10] Andreas Eisele and Yu Chen. 2010. MultiUN: A Multilingual Corpus from United Nation Documents. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*. European Language Resources Association (ELRA).
[11] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. 2017. Spatially Adaptive Computation Time for Residual Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. IEEE, 1039–1048.
[12] Google. 2022. https://huggingface.co/t5-small. https://huggingface.co/t5-small
[13] Alex Graves. 2016. Adaptive Computation Time for Recurrent Neural Networks. *arXiv preprint arXiv:1603.08983* (2016).
[14] Jiatao Gu, Hany Hassan Awadalla, and Jacob Devlin. 2018. Universal Neural Machine Translation for Extremely Low Resource Languages. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018*. Association for Computational Linguistics, 344–354.
[15] Shashij Gupta, Pinjia He, Clara Meister, and Zhendong Su. 2020. Machine translation testing via pathological invariance. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020*. IEEE, 863–875.
[16] Mirazul Haque, Anki Chauhan, Cong Liu, and Wei Yang. 2020. ILFO: Adversarial Attack on Adaptive Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020*. IEEE, 14252–14261.
[17] Hany Hassan Awadalla, Anthony Aue, Chang Chen, Vishal Chowdhary, Jonathan Clark, Christian Federmann, Xuedong Huang, Marcin Junczys-Dowmunt, Will Lewis, Mu Li, Shujie Liu, Tie-Yan Liu, Renqian Luo, Arul Menezes, Tao Qin, Frank Seide, Xu Tan, Fei Tian, Lijun Wu, Shuangzhi Wu, Yingce Xia, Dongdong Zhang, Zhirui Zhang, and Ming Zhou. 2018. Achieving Human Parity on Automatic Chinese to English News Translation. (2018). arXiv:1803.05567 arXiv.

[18] Hany Hassan Awadalla, Mostafa Elaraby, Ahmed Tawfik, Mahmoud Khaled, and Aly Osama. 2018. Gender Aware Spoken Language Translation Applied to English-Arabic. In *2nd International Conference on Natural Language and Speech Processing, ICNLSP 2018*. IEEE, 1–6.

[19] Hany Hassan Awadalla, Mostafa Elaraby, and Ahmed Y. Tawfik. 2017. Synthetic Data for Neural Machine Translation of Spoken-Dialects. In *Proceedings of the 14th International Conference on Spoken Language Translation, IWSLT 2017, Tokyo, Japan, December 14-15, 2017*. International Workshop on Spoken Language Translation, 82–89.

[20] Pinjia He. 2022. RobustNLP Library. https://github.com/RobustNLP/TestTranslation

[21] Pinjia He, Clara Meister, and Zhendong Su. 2020. Structure-invariant testing for machine translation. In *2020 IEEE/ACM 42nd International Conference on Software Engineering, ICSE 2020*. IEEE, 961–973.

[22] Pinjia He, Clara Meister, and Zhendong Su. 2021. Testing machine translation via referential transparency. In *2021 IEEE/ACM 43rd International Conference on Software Engineering, ICSE 2021*. IEEE, 410–422.

[23] Helsinki-NLP. 2022. https://huggingface.co/Helsinki-NLP/opus-mt-en-de. https://huggingface.co/Helsinki-NLP/opus-mt-en-de

[24] Sanghyun Hong, Yiğitcan Kaya, Ionuţ-Vlad Modoranu, and Tudor Dumitraş. 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 2021*. OpenReview.net.

[25] 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. *arXiv preprint arXiv:1704.04861* (2017).

[26] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An introduction to statistical learning*. Vol. 112. Springer.

[27] 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 *Proceedings of the AAAI conference on artificial intelligence, AAAI 2020*. 8018–8025.

[28] Tiedemann Jörg, Hardwick Sam, and Shleifer Sam. 2020. https://blogs.helsinki.fi/language-technology/. https://blogs.helsinki.fi/language-technology/

[29] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aäron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural Machine Translation in Linear Time. *CoRR* abs/1610.10099 (2016).

[30] Philipp Koehn, Franz J Och, and Daniel Marcu. 2003. *Statistical phrase-based translation*. Technical Report. University of Southern California Marina Del Rey Information Sciences Inst.

[31] 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 2019*. The Internet Society.

[32] Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. 2020. BERT-ATTACK: Adversarial Attack Against BERT Using BERT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020*. Association for Computational Linguistics, 6193–6202.

[33] 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. *Transactions of the Association for Computational Linguistics* (2020), 726–742.

[34] Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook FAIR's WMT19 News Translation Task Submission. In *Proceedings of the Fourth Conference on Machine Translation, WMT 2019, Florence, Italy, August 1-2, 2019 - Volume 2: Shared Task Papers, Day 1*. Association for Computational Linguistics, 314–319.

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

[36] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. In *arXiv*.

[37] 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.* (2020), 140:1–140:67.

[38] Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating Natural Language Adversarial Examples through Probability Weighted Word Saliency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, ACL 2019*. Association for Computational Linguistics, 1085–1097.

[39] Zeyu Sun, Jie M Zhang, Mark Harman, Mike Papadakis, and Lu Zhang. 2020. Automatic testing and improvement of machine translation. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE 2020*. IEEE, 974–985.

[40] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. 3104–3112.

[41] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering, ICSE 2018*. IEEE, 303–314.

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

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 5998–6008.

[44] Huiyan Wang, Jingwei Xu, Chang Xu, Xiaoxing Ma, and Jian Lu. 2020. Dissector: Input validation for deep learning applications by crossing-layer dissection. In *2020 IEEE/ACM 42nd International Conference on Software Engineering, ICSE 2020*. IEEE, 727–738.

[45] Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. 2019. Adversarial sample detection for deep neural network through model mutation testing. In *2019 IEEE/ACM 41st International Conference on Software Engineering, ICSE 2019*. IEEE, 1245–1256.

[46] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. 2018. Skipnet: Learning Dynamic Routing in Convolutional Networks. In *Proceedings of the European Conference on Computer Vision, ECCV 2018*. IEEE, 409–424.

[47] Elaine J Weyuker and Filippos I Vokolos. 2000. Experience with performance testing of software systems: issues, an approach, and case study. *IEEE transactions on software engineering* (2000), 1147–1156.

[48] 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 *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 38–45.

[49] Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. 2020. Word-level Textual Adversarial Attacking as Combinatorial Optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 6066–6080.

[50] Xinze Zhang, Junzhe Zhang, Zhenhua Chen, and Kun He. 2021. Crafting Adversarial Examples for Neural Machine Translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, 1967–1977.

[51] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition, CVPR 2018*. IEEE, 6848–6856.

[52] Wei Zou, Shujian Huang, Jun Xie, Xinyu Dai, and Jiajun Chen. 2020. A Reinforced Generation of Adversarial Examples for Neural Machine Translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020*. 3486–3497.