No-Skim: Towards Efficiency Robustness Evaluation on Skimming-based Language Models

Anonymous ACL submission

Abstract

To reduce the computation cost and the energy consumption in large language models (LLM), skimming-based acceleration dynamically drops unimportant tokens of the input sequence progressively along layers of the LLM while preserving the tokens of semantic importance. However, our work for the first time reveals the acceleration may be vulnerable to *Denial-of-Service* (DoS) attacks. In this paper, we propose *No-Skim*, a general framework to help the owners of skimming-based LLM to understand and measure the efficiency robustness of their acceleration scheme. Specifically, our framework searches minimal and unnoticeable perturbations to generate adversarial inputs that sufficiently increase the remaining token ratio, thus increasing the computation cost and energy consumption. With no direct access to the model internals, we further devise a time-based approximation algorithm to infer the remaining token ratio as the loss oracle. We systematically evaluate the vulnerability of the skimming acceleration in various LLM architectures including BERT and RoBERTa on the GLUE benchmark. In the worst case, the perturbation found by *No-Skim* substantially increases the running cost of LLM by over 103% on average.

1 Introduction

In Natural Language Processing, Transformer (Vaswani et al., 2017) has facilitated the birth of pre-trained language models, such as BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019) and GPT (Radford et al., 2018), which have brought significant improvements to various downstream applications. Despite the success on the effective performances, the computational complexity and model parameter size are massive, thus deploying these models to real-time service platforms and resource limited (i.e., energy, computation and memory resources) edge devices are very challenging.

To reduce the computation cost on language models, recent works (Goyal et al., 2020; Ye et al., 2021; Kim and Cho, 2020; Kim et al., 2022; Guan et al., 2022) propose the design of skimming-based language models. Skimming acceleration implements the intuition that human can comprehend the whole sentence by paying extra attention to only a few important words. As shown in Fig. 1, skimming acceleration dynamically and progressively drops unimportant tokens along different layers to reduce the computation budget and preserves the important tokens within the layers to maintain the semantic information. For example, the average FLOPs speed up is 281% on GLUE benchmark (Wang et al., 2018) when deploying skimming acceleration proposed in Guan et al. (2022).

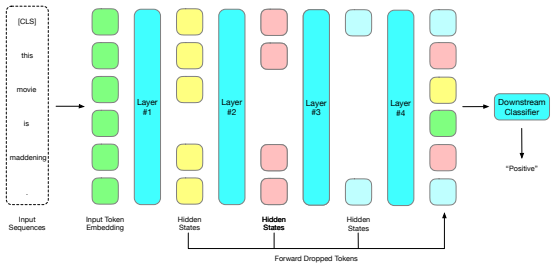

Figure 1: The general design of skimming-based language models on a sentiment classification task. In the example, the important token "good" is preserved.

The skimming-based language models can be deployed on real-time service platforms and resource limited edge devices to reduce the computation complexity and decrease the energy consumption. Despite the tremendous success on improving efficiency, we need to understand and evaluate potential vulnerabilities on existing skimming-based language models from the perspective of computation efficiency, where the dropping of tokens can be deliberately manipulated by the unnoticeable changes on text inputs, which will pose serious challenge to the practical deployments. For real-time service platforms, the increasing computation complexity reduces the number of queries processed simultane-

ously, which eventually damage the service quality. For resource limited edge devices, the increasing computation cost accelerates the consumption of valuable resources (e.g., battery life), which is not acceptable for ordinary users.

However, existing adversarial attack (Li et al., 2018; Ren et al., 2019; Gao et al., 2018; Li et al., 2020) lacks the ability to evaluate efficiency robustness in many aspects. First, the goal is mainly focus on damaging model accuracy, which lacks clear efficiency information to properly guide the efficiency robustness evaluation. Second, efficiency information is in the form of discrete values, which makes gradients hard to calculate. Third, when the language models are deployed on online predictive API, no internal efficiency information is obtainable.

To provide an accurate evaluation on the efficiency robustness of skimming-based language models, we propose *No Skim*, the first general efficiency robustness evaluation framework on the skimming-based language models, which integrates a rank-and-substitute scheme to generates adversarial inputs that maximally increase the computation complexity. Specifically, we implement a gradient-based evaluation algorithm to effectively search the appropriate perturbations, in which we propose a loss smoothing algorithm to make the efficiency loss differentiable. To solve the challenge where on model internals are available, we further theoretically analyze the relation between inference time and remaining token ratio and propose a time-based approximation algorithm to infer the efficiency information. Then, we implement a time-based evaluation algorithm, which makes our evaluation applicable to various deployment scenarios.

The contributions of our paper can be summarized as follows:

- We are the first work to systematically study the vulnerability of the skimming-based language models from the perspective of efficiency.
- We propose an effective efficiency robustness evaluation framework *No Skim* that generates adversarial inputs to increase the computation complexity.
- We propose both gradient-based algorithm and time-based algorithm to evaluate the efficiency robustness under various deployment scenarios.
- We conduct extensive evaluations on the state-of-the-art dynamic skimming acceleration scheme Transkimmer (Guan et al., 2022) with BERT and RoBERTa architectures on the GLUE benchmark. In the worst case, our framework can increase the computation cost by 103%.

2 Related Works

2.1 Skimming Acceleration Schemes

Skimming acceleration schemes have been a significant thrust of recent researches to improve the efficiency of existing language models. Skimming is first well-studied in recurrent-based neural networks (Yu et al., 2017; Campos et al., 2017; Yu et al., 2018; Fu and Ma, 2018), which saves computation time-wise by dynamically skipping some time steps and copying the hidden states directly to the next step without any update. Recently, in the presence of transformer architectures (Vaswani et al., 2017), skimming-based language models reduce the computation complexity by dropping some unimportant tokens progressively along different layers. Skimming-based acceleration schemes can be categorized into static and dynamic schemes.

Static skimming schemes (Goyal et al., 2020; Kim and Cho, 2020) use a fix remaining token ratio, where all the input sequences are all dropped certain ratio of tokens during inference. However, different input sequences vary greatly within tasks and between training and validation dataset, leading to a bad generalization. Dynamic skimming schemes are input-adaptive, which use hidden values or attention values to dynamically decide whether the token are dropped or not. Ye et al. (2021) propose a RL-based scheme called TR-BERT, which adopts reinforcement learning to independently optimize a policy network that dynamically drops tokens. Kim et al. (2022) propose a threshold-base scheme called LTP, which drops the tokens whose the attention values is lower than the threshold. Guan et al. (2022) propose a prediction-based scheme called Transkimmer, which integrates each layer with a lightweight fully connected network to make the skimming decision for each token given the hidden values. In this paper, we mainly investigate the efficiency robustness of the dynamic skimming-based language models as the computation complexity varies according to the input text sequences.

2.2 Robustness Evaluation

Adversarial attacks (Li et al., 2018; Ren et al., 2019; Gao et al., 2018; Li et al., 2020) are proposed to evaluate language models' ability to make correct prediction facing imperceptible perturbations. However, these attack lacks the ability to evaluate efficiency robustness due to its incorrect attack goal, unavailable gradient information and unacquirable efficiency information. In the meantime, a line of works have been proposed to study the efficiency robustness of existing language models. Zhang et al. (2023) propose slow-down attacks on multi-exit language models (Zhou et al., 2020), which delay the exit positions to increase the computation cost. Chen et al. (2022b,a) propose to maximize the output sequences' length to increase the inference time. As skimming acceleration schemes improve model efficiency different from the aforementioned models, a systematic evaluation on the efficiency robustness is necessary.

3 Formulation

3.1 Evaluation Objectives

The goal of our efficiency robustness evaluation framework is to generate adversarial inputs that deteriorate the efficiency of the skimming-based language models. Intuitively, the framework generates unnoticeable perturbations on original inputs to increase the remaining token ratio, which serves as an indicator of model efficiency. Increasing the remaining token ratio denotes the increase of the computation complexity and the energy consumption. We formulate the evaluation objectives as the following optimization:

$$\arg\max_{\delta} L_{eff}(x+\delta) \quad s.t. \quad Sim(x, x+\delta) \geq \epsilon, \tag{1}$$

where x is the original input, δ is the perturbations added on the input x, L_{eff} denotes the efficiency loss, $Sim : \mathcal{X} \times \mathcal{X} \rightarrow (0, 1)$ is the similarity function and ϵ is the similarity threshold.

Generating adversarial inputs that increase the computation cost poses serious challenges to the practical deployments of skimming-based language models. For real-time service platforms, throughput is the key element to measure the quality of the service. However, for a platform with a certain level of computation power, the increasing computation complexity reduces the number of queries processed simultaneously, which eventually damages the service quality and ruins online users' experience. For resource limited edge devices, the increasing computation cost accelerates the consumption of valuable resources (e.g., battery life for mobile phones), thus shortening the available time, which is not acceptable for ordinary users.

3.2 Evaluation Scenarios

To comprehensively evaluate the potential efficiency vulnerability of the skimming-based language models, the framework should generally support the evaluations under different level of knowledge and access to the language models. We assume the evaluator has the following knowledge and access to the target skimming-based language model:

- **White-box Access:** White-box access assumes the evaluator has full knowledge of the target model (i.e., model parameters and dynamic skimming scheme) and the vocabulary information (i.e., the vocabulary size and the corresponding word embeddings). And the gradient w.r.t. the word embedding and model parameters can be directly calculated. It simulates a practical scenario where the skimming language model is directly deployed on the resource limited edge devices.
- **Black-box Access:** Black-box access is considered as the toughest scenario. It assumes that no internal information (i.e., model information or even speed up ratio) are acquirable for the evaluator. The evaluator can only approximate the internal information by observing the running status (e.g., inference time). It simulates a scenario where the skimming language model is deployed on cloud platforms as predictive API.

4 Methodology

We propose *No skim*, the first general framework to evaluate the efficiency robustness of the skimming-based language models. First, we present the general design of the framework. Then, we provide two specific implementations to evaluate the efficiency robustness of skimming-based language models under two different scenarios.

4.1 General Design

Given an initial input, our *No skim* iteratively searches unnoticeable perturbations to gradually increase the computation cost. As shown in Fig. 2, we provide an overview of the general pipeline of our *No skim*, which is used to generate the adversarial input. We also provide a detailed algorithmic

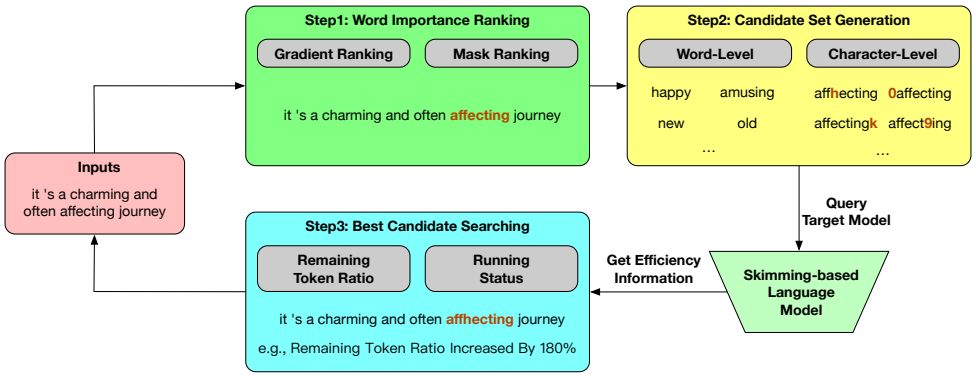

Figure 2: The general evaluation framework of *No Skim*, where Word Importance Ranking step only runs once.

description of the generation process in Algorithm 1 of Appendix B, which consists of three major steps:

- **Step1: Word Importance Ranking (Line 4-8) :** In this step, we aim to identify the most important word that will drastically impact the model's efficiency when modifying it. An importance score will be calculated for each word, and the word with the larger importance score is more likely to be modified in the following steps.
- **Step2: Candidate Set Generation (Line 10-12) :** In this step, we aim to generate a candidate set to represent the possible search space given the word selected in the last step, where the candidate words are imperceptible from the original words to preserve the semantic information.
- **Step3: Best Candidate Searching (Line 14-21) :** In this step, we search the whole candidate set and aim to select the candidate that maximally increases the computation cost and the energy consumption. We substitute the original word with each word in the candidate set and query the target skimming-based language model to get the efficiency information. Then, we compare the efficiency degradation of each candidate word to decide to best candidate word.

4.2 White-box Evaluation

Efficiency Loss. In white-box scenario, we are able to observe the inner characteristic of the target skimming-based model. We directly calculate the remaining token ratio as the efficiency loss:

$$L_{eff}(X) = \frac{1}{K}\sum_{l=0}^{K}\frac{sum(M_l)}{len(M_l)}, \qquad (2)$$

where K is the number of layers in the language model, M_l is the binary skim decision for the token sequence at layer l. For every element in M_l, 0 stands for dropping the token, 1 stands for preserving the token. The remaining token ratio calculates and averages the ratios of token remained of every layers, which represents the computation complexity speed-up.

Word Importance Ranking. Given a text sequence of n words $X = (x_1, x_2, \cdots, x_n)$, some words play the key role of influencing the model's efficiency. We first calculate the importance score of each word x_i as follows:

$$Score_i = \sum_{j=0}^{m}\frac{\partial L_{eff(X)}}{\partial E_i^j}, \qquad (3)$$

where the E_i is the embedding of word x_i, m is the feature dimension of embedding and L_{eff} is the efficiency loss. The score firstly calculates the gradient of the efficiency loss w.r.t the word embedding and then calculates the sum of gradient along the embedding. The gradient implies the direction and degree of efficiency loss's change when manipulating the word embedding. Perturbing the word with the largest importance score is the easiest way to increase the efficiency loss L_{eff}, thus increasing the computation complexity.

Candidate Set Generation. Once selecting the most important word based on gradient, we need to generate a candidate set composed of unnoticeable perturbed versions of the selected word. The candidate set represents the proper optimization search space to increase the efficiency loss (i.e., remaining token ratio). For white-box scenario, we design word-level perturbation and character-level perturbation to generate the candidate set.

For word-level perturbation, knowing the vocabulary information, we enumerate every word in the

vocabulary and calculate the efficiency loss change:

$$V_{target} = \sum_{j=0}^{m} (E^j_{target} - E^j_{selected}) \cdot \frac{\partial L_{eff(X)}}{\partial E^j_{selected}}, \tag{4}$$

where E_{target} denotes the embedding of a target word in the vocabulary and $E_{selected}$ denotes the embedding of the selected important word. Equation 4 calculates the product of the change of embedding and the partial derivative on embedding when substituting the selected important word to the target word. We then sample words from the vocabulary as follows:

$$S = \text{Top-k}_{target \in Vocab} V_{target}, \tag{5}$$

where words with top-k efficiency loss change V are selected. In the meantime, we also discard the words that deteriorate the text semantic information from the candidate set.

For character-level perturbation, we simulate the mistakes made by ordinary users during typing by inserting random characters at random locations. Since the character-level perturbation often leads to UNK token in the embedding space, it is challenging to directly compare the efficiency loss changes of these perturbed words. Thus, we randomly select several characters in digits, letters and insert the character at random locations to form the character-level candidate set S. The examples of word-level and character-level perturbation are shown in Fig. 2.

Best Candidate Searching. After generating the candidate set, We straightforwardly test all perturbations in the candidate set and select the optimal perturbation that leads to the largest computation cost. In white-box scenario, we use the remaining token ratio calculated in Eq. 2 to represent the computation cost. For an original input, we iteratively add unnoticeable perturbations to the original input several times to generate highly effective adversarial inputs.

Loss Smoothing Algorithm. Since the efficiency loss proposed in Eq. 2 is discrete and non-differentiable, we propose loss smoothing algorithm, which uses the reparameterization trick (i.e., Gumbel-softmax) to sample the discrete skim decision M from the skim probability P:

$$M^t = \frac{\exp((\log(P^t) + g^t)/\tau)}{\sum_{k=0}^{1} \exp((\log(P^k) + g^k)/\tau)}, \tag{6}$$

where $t \in \{0, 1\}$, g is independent and identically sampled from $Gumbel(0, 1)$ distribution and τ is the temperature. After smoothing, the estimated skim decision M is differentiable.

4.3 Black-box Scenario

Time-based Approximation Algorithm. Black-box scenario is considered as the toughest scenario, as no internal information (i.e., model information or even speed up ratio) are acquirable for us. Inspired by the recent advances in side-channel attacks (Brumley and Boneh, 2005; Inci et al., 2016; Goyal et al., 2020; Timon, 2019), we propose from a new perspective to approximate the remaining token ratio in Eq. 2 to facilitate the evaluation procedure, where the magnitude of inference time actually represents different remaining token ratios. First, we theoretically analysis the relation between the inference time and remaining token ratio in Theorem 1.

Theorem 1 *Assumed skimming-based language model f is composed by K encoders, $R = \frac{1}{K}\sum_{i=1}^{K} r_i$ is the total remaining token ratio, where $\{r_i\}_{i=1}^{K}$ is the remaining token ratio of each encoder and $r_i \in (0, 1]$, and x is the input text and T is the corresponding inference time. If text length is fixed, then $T \propto R$.*

Then, we empirically observe the relation between the remaining token ratio and the sequence-level inference time, which measures the inference time on the entire input sequence. However, as shown in Fig. 3(a), we find the linear correlation is imperfect due to the large variance of input sequences' lengths. For example, a long input sequence with low remaining token ratio can still require a large inference time. To eliminate the effects of lengths' variance, we take the input length into account:

Theorem 2 *Assumed skimming-based language model f is composed by K encoders, $R = \frac{1}{K}\sum_{i=1}^{K} r_i$ is the total remaining token ratio, where $\{r_i\}_{i=1}^{K}$ is the remaining token ratio of each encoder and $r_i \in (0, 1]$, and x is the input text and T is the corresponding inference time. If text length is not fixed and the length is N, then $T \propto R \cdot N$.*

Theorem 2 shows that inference time is positively linear correlated with the product of remaining token ratio and input length. The detailed proofs are provided in Appendix C. Based on the theoretical findings, we propose token-level inference time to approximate the remaining token ratio

and set the efficiency loss L_{eff} as follows:

$$L_{eff} = \frac{Time(x)}{length}, \quad \text{where} \quad length = Len(Tokenizer(x)), \tag{7}$$

where $Time(x)$ is the inference time of the sequence x, $Tokenizer$ is a tokenizer and Len counts the token sequence length. As limited in the black-box scenario, we have no inner information about the architecture of the target model and its corresponding tokenizer. Instead, we propose to randomly select a third-party and public tokenizer unrelated to the targeted skimming-based model to approximate the token sequence length.

As shown in Fig. 3(b), we empirically observe a perfect positive linear correlation between the remaining token ratio and the token-level inference time. The token-level inference time eliminates the negative influence caused by the variance of the sequence lengths, as a single token's computation cost is fixed given a specific language model. This observation further proves the feasibility to conduct a side-channel attack to infer the remaining token ratio by analysis the token-level inference time.

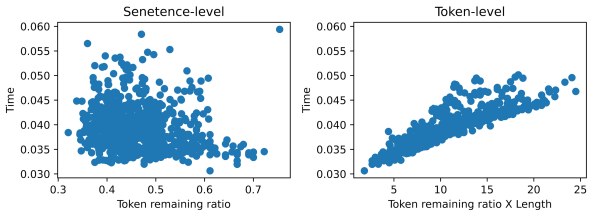

Figure 3: The linear relation between the remaining token ratio and inference time, where (a) uses the inference time on the sequence-level and (b) uses the inference time on the token-level. We present the result on Transkimmer (Guan et al., 2022) with a sentiment classification task SST-2.

Word Importance Ranking. Since gradient information is no longer available, we propose mask-based importance score to select the word that has the largest impact on the computation efficiency. As proposed in Eq. 8, we iteratively mask each word x_i in the original text sequence and form the mask version $\hat{X} = (x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n)$. We then calculate the importance score of each word by subtracting the efficiency loss of the original one from the mask one to get the efficiency loss increment:

$$Score_i = L_{eff}(\hat{X}) - L_{eff}(X), \quad \text{where } \hat{X} = (x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n), \tag{8}$$

where $Score_i$ is the important score for the $i'th$ word and L_{eff} represents the remaining token ratio. If masking the word leads to a large efficiency loss increment, it means that the masked word is critical for the model computation efficiency.

Candidate Set Generation. In the meantime, we can not get the word embeddings under black-box scenario. For word-level perturbation, we propose to use the nearest neighbours of the target word in a pre-trained word embedding space (e.g., word2vec Mikolov et al. (2013)). The rest of the procedure is the same as white-box scenario.

5 Evaluation Setting

Architecture & Dataset. To thoroughly evaluate our framework, *No skim*, we consider the state-of-the-art skimming scheme Transkimmer (Guan et al., 2022) as our evaluation target. We implement BERT (Devlin et al., 2018) and RoBERTa (Liu et al., 2019) with tasks of the GLUE Benchmark (Wang et al., 2018), which are detailed in Tab. 6.

Metrics. We evaluate the efficiency of skimming-based language models with the following metrics:

Average Remaining Ratio (ARR): As shown in Fig. 7(a), the average token ratio calculates the average remaining token ratio on the dataset. The metric is a task-level metric that evaluate the overall efficiency speed-up performance. When ARR is closer to 0, the target model has better efficiency.

Cumulative Token Ratio (CRR): As shown in Fig. 7(b), the cumulative token ratio calculates cumulative distribution of the remaining token ratio of each input. The metric indicates the variance of efficiency speed-up on different samples. When CRR is closer to 1, the target model has better efficiency. Fig. 7 and more detailed description are provided in Appendix A.

Baselines. Since we are the first work to evaluate the efficiency robustness of skimming-based language models, we compare with several adversarial attacks: TextBugger (Li et al., 2018), DeepWordBug (Gao et al., 2018), BERTAttack (Li et al., 2020) and PWWS (Ren et al., 2019). The details settings are provided in Appendix A.

6 Evaluation

Effectiveness Under White-box Scenario. First, we report how much computational complexity is increased by our adversarial inputs in Tab. 1. We

Scheme	Scenario	SF=0.5		SF=0.75		SF=1.0		SF=0.5		SF=0.75		SF=1.0	
		ARR↑	CRR↓	ARR↑	CRR↓	ARR↑	CRR↓	ARR↑	CRR↓	ARR↑	CRR↓	ARR↑	CRR↓
		SST2 + BERT						**SST2 + RoBERTa**					
Origin	-	0.442	0.563	0.172	0.833	0.142	0.863	0.331	0.674	0.194	0.811	0.150	0.855
TextBugger	w.b.	0.440	0.565	0.168	0.837	0.134	0.871	0.288	0.717	0.169	0.836	0.136	0.869
DeepWordBug	b.b.	0.445	0.559	0.154	0.851	0.126	0.880	0.269	0.736	0.145	0.859	0.116	0.888
BERTAttack	b.b.	0.425	0.581	0.172	0.833	0.139	0.866	0.314	0.691	0.185	0.821	0.143	0.862
PWWS	b.b.	0.430	0.575	0.173	0.832	0.139	0.866	0.313	0.692	0.182	0.823	0.140	0.865
No skim	w.b.	0.730	0.275	0.400	0.605	0.322	0.683	0.519	0.486	0.311	0.694	0.260	0.745
No skim	b.b.	0.659	0.346	0.256	0.749	0.185	0.820	0.402	0.603	0.234	0.771	0.186	0.819
		MRPC + BERT						**MRPC + RoBERTa**					
Origin	-	0.551	0.454	0.443	0.562	0.242	0.763	0.514	0.491	0.397	0.608	0.252	0.753
TextBugger	w.b.	0.605	0.400	0.452	0.553	0.246	0.759	0.579	0.426	0.425	0.579	0.258	0.748
DeepWordBug	b.b.	0.644	0.361	0.505	0.500	0.241	0.765	0.583	0.423	0.420	0.585	0.257	0.748
BERTAttack	b.b.	0.577	0.428	0.432	0.573	0.241	0.764	0.533	0.472	0.406	0.599	0.252	0.754
PWWS	b.b.	0.618	0.387	0.483	0.522	0.254	0.751	0.559	0.446	0.422	0.583	0.258	0.748
No skim	w.b.	0.869	0.137	0.774	0.231	0.499	0.505	0.860	0.145	0.736	0.269	0.572	0.432
No skim	b.b.	0.813	0.192	0.710	0.295	0.299	0.706	0.741	0.264	0.526	0.479	0.343	0.661

Table 1: Efficiency robustness results on Skimming-based Language, where w.b. and b.b. represents white-box and black-box scenario respectively.

make the following observations: (1) the baseline attacks are not effective in evaluating efficiency robustness. (2) Our *No Skim* demonstrates the efficiency vulnerability of the existing skimming-based language model, which increases the average remaining ratio by 106% and decrease the cumulative remaining ratio to 30% at most. (3) When the skim factor SF is larger, the negative influence on model efficiency is less. Furthermore, Fig. 4 shows an example of each layer's average remaining ratio, where our *No skim* generates samples that increase the remaining ratio in every layer particularly the former layers comparing to the original samples.

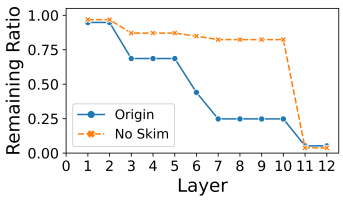

Figure 4: The comparison of efficiency results on BERT and SST-2 under white-box scenario.

Effectiveness Under Black-box Scenario. For black-box scenario, we first evaluate the performance of our time-based approximation algorithm. As shown in Tab. 3, using token-level inference time can more accurately approximate the remaining token ratio comparing to the sentence-level inference time, where the magnitude of the mean square errors are only 10^{-3} at most. Then, we evaluate the effectiveness of our *No skim* in Tab. 1. We report that our black-box evaluation suffers sligtest performance drop comparing to our white-box evaluation (e.g., around 0.1 on average remaining ratio). Nevertheless, this poses serious challenges to the deployments on real-time cloud services.

Metric	Scenario	Skim Factor					
		0.5	0.75	1	0.5	0.75	1
		BERT SST-2			**BERT MRPC**		
LD	w.b.	24.340	32.817	30.517	32.228	32.593	34.420
	b.b.	12.325	10.177	8.817	21.607	21.043	14.960
SS	w.b.	0.488	0.522	0.579	0.855	0.864	0.867
	b.b.	0.585	0.624	0.695	0.920	0.929	0.934
		RoBERTa SST-2			**RoBERTa MRPC**		
LD	w.b.	29.751	31.827	27.157	20.512	20.890	22.257
	b.b.	3.437	3.403	4.390	9.880	9.360	8.373
SS	w.b.	0.619	0.612	0.645	0.907	0.915	0.913
	b.b.	0.838	0.836	0.783	0.905	0.916	0.921

Table 2: The similarity between our generated inputs and original inputs, where LD and SS stands for levenshtein distance and semantic similarity

Text Similarity. Further, we report how stealthy are the imperceptible mutations on generated adversarial inputs of our *No skim* compared to the original ones. We measure both cosine similarity on the sentence embedding generated by SBERT and levenshtein distance on character-level to demonstrate the similarity. As shown in Tab. 2, the average semantic similarity is larger than 0.85 for most cases. In the meantime, the levenshtein distance shows less than 30 edit operation is required to generate the adversarial inputs, which means the mutations are of high stealthiness.

Influence on Model Utility. Next, we study whether our *No Skim* will cause extra damages to the model utility. As reported in Tab. 4, our attack generate adversarial inputs that not only in-

	SF=0.5		SF=0.75		SF=1.0	
	Sentence	Token	Sentence	Token	Sentence	Token
SST-2 + BERT	7.416×10^{-3}	4.677×10^{-3}	1.109×10^{-2}	3.322×10^{-3}	7.634×10^{-3}	1.649×10^{-3}
MRPC + BERT	7.003×10^{-3}	5.336×10^{-3}	7.030×10^{-3}	4.531×10^{-3}	5.609×10^{-4}	4.890×10^{-4}
SST-2 + RoBERTa	9.980×10^{-3}	3.990×10^{-3}	6.961×10^{-3}	1.915×10^{-3}	5.110×10^{-3}	1.408×10^{-3}
MRPC + RoBERTa	4.525×10^{-3}	3.680×10^{-3}	2.099×10^{-3}	1.687×10^{-3}	1.082×10^{-3}	1.009×10^{-3}

Table 3: Mean square error on time-based approximation algorithm.

Scenario	SF=0.5	SF=0.75	SF=1.0	SF=0.5	SF=0.75	SF=1.0
	SST-2 + BERT			**MRPC + BERT**		
Origin	0.904	0.884	0.869	0.853	0.804	0.782
w.b.	0.644	0.523	0.613	0.324	0.320	0.320
b.b.	0.649	0.647	0.780	0.343	0.380	0.513
	SST-2 + RoBERTa			**MRPC + RoBERTa**		
Origin	0.931	0.894	0.896	0.836	0.850	0.767
w.b.	0.648	0.540	0.633	0.445	0.460	0.443
b.b.	0.698	0.583	0.694	0.521	0.583	0.523

Table 4: The negative effect comparison on model utility between our generated adversarial inputs and original inputs.

Module	SF=0.5		SF=0.75		SF=1.0	
	ARR	CRR	ARR	CRR	ARR	CRR
			BERT SST-2			
+ None	0.635	0.371	0.224	0.781	0.166	0.839
+ Rank	0.642	0.363	0.238	0.767	0.175	0.831
+ Search	0.725	0.28	0.374	0.631	0.315	0.689
+ All	**0.730**	**0.275**	**0.400**	**0.605**	**0.322**	**0.683**
			BERT MRPC			
+ None	0.752	0.253	0.665	0.340	0.360	0.644
+ Rank	0.778	0.227	0.668	0.337	0.37	0.636
+ Search	0.850	0.156	0.769	0.236	0.497	0.509
+ All	**0.869**	**0.137**	**0.774**	**0.231**	**0.499**	**0.505**

Table 5: The effectiveness of each module in *No skim*, where we conduct white-box evaluations.

crease the computation costs but also degrade the classification performances of the skimming-based models. Specifically, the accuracy is decreased from 20% to 30%, which calls for more attentions on our proposed evaluation.

Ablation Studies. First, Tab. 5 studies the effectiveness of each module in *No skim*. Since candidate set generation is necessary, we mainly focus on word importance ranking and best candidate searching. As we can see, both module plays positive effect on improving the evaluation effectiveness. And best candidate searching is more influenced than word importance ranking.

Then, we study the influence of mutant times. Fig. 5 shows the metrics' results when we mutant the original texts for 1 to 5 times. When increasing the mutant time, the ARR keep increasing and the CRR is continuously dropping, which indicates better evaluation effectiveness.

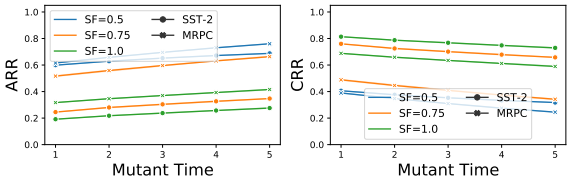

Figure 5: The influence of mutant time.

Possible Defense. Finally, we study whether our *No skim* can help improving the efficiency robustness. We use *No skim* to generate 5000 samples and incorporate them into adversarial training (Geng et al., 2021). Fig. 6 shows the average remaining ratio and accuracy at different training epochs. With the increase of adversarial training epochs, the efficiency robustness is largely strengthened. But we find a minor drop on model accuracy (e.g., around 10%), which calls for better robustness and accuracy trade-off in the future studies.

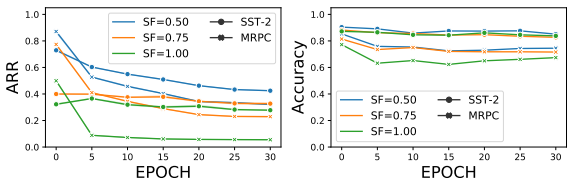

Figure 6: The result of adversarial training.

7 Conclusions

In our work, we systematically study the potential efficiency vulnerability of skimming acceleration schemes on language models. We propose *No Skim*, which generates adversarial inputs that drastically increase the average inference cost of skimming-based language models, which poses serious challenges to the deployments of the skimming-based language models on real-time cloud services or local hardware-constrained edge devices. As a security problem of the large language models, our work welcomes future research to devise strong defense against our evaluation.

8 Limitations

Apart from the performance on evluating the efficiency robustness, we acknowledge that our work has several limitations. Firstly, we only evaluate our *No Skim* on the GLUE Benchmark (Wang et al., 2018), which demonstrate the effectiveness on alphabetic languages such as English. However, for logograms (e.g., Chinese), it requires to design language-specific method to generate the corresponding substitution set to achieve the attack goal. Secondly, extensive results on other backbones and datasets should be evaluate (e.g., datasets with longer sequences). Third, our work only evaluate the defense result of adversarial training (Geng et al., 2021), more detailed defenses should be proposed and analyzed. For the above mentioned limitations, we leave them as the future works.

References

David Brumley and Dan Boneh. 2005. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716.

Víctor Campos, Brendan Jou, Xavier Giró-i Nieto, Jordi Torres, and Shih-Fu Chang. 2017. Skip rnn: Learning to skip state updates in recurrent neural networks. *arXiv preprint arXiv:1708.06834*.

Simin Chen, Cong Liu, Mirazul Haque, Zihe Song, and Wei Yang. 2022a. 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*, pages 1148–1160.

Simin Chen, Zihe Song, Mirazul Haque, Cong Liu, and Wei Yang. 2022b. Nicgslowdown: Evaluating the efficiency robustness of neural image caption generation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15365–15374.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Tsu-Jui Fu and Wei-Yun Ma. 2018. Speed reading: Learning to read forbackward via shuttle. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4439–4448.

Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box generation of adversarial text sequences to evade deep learning classifiers. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 50–56.

Shijie Geng, Peng Gao, Zuohui Fu, and Yongfeng Zhang. 2021. Romebert: Robust training of multi-exit bert. *arXiv preprint arXiv:2101.09755*.

Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. 2020. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *International Conference on Machine Learning*, pages 3690–3699. PMLR.

Yue Guan, Zhengyi Li, Jingwen Leng, Zhouhan Lin, and Minyi Guo. 2022. Transkimmer: Transformer learns to layer-wise skim. *arXiv preprint arXiv:2205.07324*.

Mehmet Sinan Inci, Berk Gulmezoglu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2016. Cache attacks enable bulk key recovery on the cloud. In *Cryptographic Hardware and Embedded Systems–CHES 2016: 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings 18*, pages 368–388. Springer.

Gyuwan Kim and Kyunghyun Cho. 2020. Length-adaptive transformer: Train once with length drop, use anytime with search. *arXiv preprint arXiv:2010.07003*.

Sehoon Kim, Sheng Shen, David Thorsley, Amir Gholami, Woosuk Kwon, Joseph Hassoun, and Kurt Keutzer. 2022. Learned token pruning for transformers. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 784–794.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. 2021. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2018. Textbugger: Generating adversarial text against real-world applications. *arXiv preprint arXiv:1812.05271*.

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)*, pages 6193–6202. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

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*, pages 1085–1097, Florence, Italy. Association for Computational Linguistics.

Benjamin Timon. 2019. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 107–131.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

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*, Online. Association for Computational Linguistics.

Deming Ye, Yankai Lin, Yufei Huang, and Maosong Sun. 2021. Tr-bert: Dynamic token reduction for accelerating bert inference. *arXiv preprint arXiv:2105.11618*.

Adams Wei Yu, Hongrae Lee, and Quoc V Le. 2017. Learning to skim text. *arXiv preprint arXiv:1704.06877*.

Keyi Yu, Yang Liu, Alexander G Schwing, and Jian Peng. 2018. Fast and accurate text classification: Skimming, rereading and early stopping.

Shengyao Zhang, Xudong Pan, Mi Zhang, and Min Yang. 2023. SlowBERT: Slow-down attacks on input-adaptive multi-exit BERT. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 9992–10007, Toronto, Canada. Association for Computational Linguistics.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33:18330–18341.

A Detailed Evaluation Settings

A.1 Datasets

The detail dataset information is show in Tab. 6:

Identifier	Task	Domain	Length	Size
SST-2	Sentiment	Movie Reviews	25	67k/0.9k
MRPC	Paraphrase	News	53	3.7k/0.5k

Table 6: Datasets in our evaluation, where the last column reports the training dataset size and validation dataset size.

A.2 Metrics

We evaluate the efficiency of skimming-based langauge models with the following metrics:

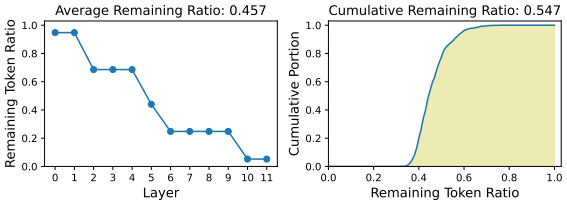

Figure 7: The metrics that evaluate the model efficiency, where (a) denotes the average remaining ratio and (b) denotes the cumulative remaining ratio.

Average Remaining Ratio (ARR): As shown in Fig. 4(a), the average token ratio first calculates and plots the average remaining token in each layer for all inputs. Then, the metric calculates the normalized area under the curve:

$$ARR = \frac{1}{L}\sum_{l=0}^{L}\frac{1}{|D|}\sum_{i=0}^{|D|}\frac{sum(M_l^i)}{len(M_l^i)}, \qquad (9)$$

where D is the test inputs dataset, L is the number of layers and M_l^i is the binary mask decision at layer l for the $i'th$ test input. The metric is a task-level metric that evaluate the overall efficiency speed-up performance on the entire dataset. When ARR is closer to 0, the target model has better efficiency.

Cumulative Token Ratio (CRR): As shown in Fig. 4(b), the cumulative token ratio first calculates and plots the cumulative distribution curve with respect to the remaining token ratio of each input:

$$p(x_{RTR}) = \frac{1}{|D|}\sum_{i=0}^{|D|}\{\frac{1}{L}\sum_{l=0}^{L}\frac{sum(M_l^i)}{len(M_l^i)} \leq x_{RTR}\}, \tag{10}$$

where x_{RTR} is the threshold of remaining token ratio and $p(x_{RTR})$ calculates the portion of text inputs that have remaining token ratio larger than x_{RTR}. Then, the metric calculates the area size under the curve:

$$CRR = \int_0^1 p(x_{RTR})dx_{RTR}, \tag{11}$$

which is the integral of the portion $p(x_{RTR})$ with respect to the remaining token ratio x_{RTR} on an interval $[0, 1]$. The metric is a sample-level metric that shows the distribution of each sample's remaining token ratio, indicating the variance of efficiency speed-up on different samples. When CRR is closer to 1, the target model has better efficiency.

A.3 Hyper-parameters

We implement the skimming-based language models on the base of Hugging Face's Transformers (Wolf et al., 2020) with GLUE benchmark (Wang et al., 2018) provided in Datasets (Lhoest et al., 2021). For training skimming-based models, we fine-tune the pretrained models with a linear classifier. For Transkimmer, we set the skim factor SF as 0.5, 0.75 and 1.0 respectively where the maximum sequence length as 64. For constructing the adversarial inputs, we set the maximum mutant time Ops as 10, where each mutant changes one word. For gradient-level mutants, we ensure the semantic similarity between words in the candidate set and original word larger than 0.5. For black-box scenario, we select bert-base-uncased (Devlin et al., 2018) as the third-party and public tokenizer.

For training the skimming-based language models, we download the pre-trained BERT/RoBERTa model provied in Huggingface. and add a linear classifier after [CLS] token embedding. For training Transkimmer on SST-2, we fine-tune the model 3 epochs, where we set the batch size as 32 and the learning rate as 2e-5 with an Adam optimizer. For training Transkimmer on MRPC, we fine-tune the model 5 epochs, where we set the batch size as 32 and the learning rate as 5e-5 with an Adam optimizer.

B Algorithm

The detailed algorithmic description of the generation phase is provided below:

Algorithm 1 General Efficiency Robustness Evaluations Framework

Input: Original Input $X = (x_1, \cdots, x_n)$, Target Skimming-based LLM $F(x)$, Number of Operations Ops, Efficiency Loss $L_{eff}(x)$.
Output: Adversarial Input Sample $\tilde{X}$.
Initialize: $\tilde{X} \leftarrow X$
for word x_i in $\tilde{X}$ **do**
  Compute $Score_i \leftarrow \text{ImportScore}(x_i)$
end for
$X_{sort} \leftarrow \text{Sort}(\tilde{X})$ according to $Score$
  ▷ Step1: Word Importance Ranking
for idx in range(Ops) **do** ▷ Search perturbations iteratively
  $x_{max} \leftarrow X_{sort}[idx]$
  Candidate Set $S \leftarrow \text{CanGen}(x_{max})$
    ▷ Step2: Candidate Set Generation
  Initialize: $L_{max} \leftarrow L_{eff}(X)$
  for candidate word s_j in S **do**
    $X_{can} \leftarrow (\tilde{x}_1, \cdots, \tilde{x}_{idx-1}, s_j, \tilde{x}_{idx+1}, \cdots, \tilde{x}_n)$
    if $L_{eff}(X_{can}) > L_{max}$ **then**
      $s_{best} \leftarrow s_j$
      $L_{max} \leftarrow L_{eff}(X_{can})$
    end if
  end for
    ▷ Step3: Best Candidate Searching
  $\tilde{X} \leftarrow (\tilde{x}_1, \cdots, \tilde{x}_{idx-1}, s_{best}, \tilde{x}_{idx+1}, \cdots, \tilde{x}_n)$
end for
Return: $\tilde{X}$

C Proofs

Theorem 3 *Assumed skimming-based language model f is composed by K encoders, $R = \frac{1}{K}\sum_{i=1}^{K} r_i$ is the total remaining token ratio, where $\{r_i\}_{i=1}^{K}$ is the remaining token ratio of each encoder and $r_i \in (0, 1]$, and x is the input text and T is the corresponding inference time. If text length is fixed, then $T \propto R$.*

Proof C.1 *Assumed skimming-based language model f is composed by K encoders with automatic padding, where the feature dimension is d. The input text's length is fix and denotes as n. For*

the encoder in the Transformers architecture, the time complexity is shown as:

$$O_{encoder} = O(n^2 d + nd^2). \tag{12}$$

Since the feature dimension d is far greater than the text length n ($n \ll d$), we can derive such approximation from Eq. 12:

$$O_{encoder} = O(n^2 d + nd^2) \approx O(nd^2). \tag{13}$$

Given a total number of K encoders, from Eq. 13, we can get the total time complexity of the original language model:

$$O_{all} \approx O(Lnd^2) = K \cdot O_{encoder}. \tag{14}$$

During inference stage, the skimming-based language model dynamically drops unimportant token, the real number of token that participate the inference process is $r_i \cdot n$, then the total time complexity of the skimming-based language model is:

$$\begin{aligned} O_{skim} &\approx \sum_{i=1}^{K} O(r_i nd^2) = \sum_{i=1}^{K} r_i \cdot O(nd^2) \\ &= K \cdot R \cdot O(nd^2) = R \cdot O(Knd^2) \\ &= R \cdot O_{all}. \end{aligned} \tag{15}$$

Meanwhile, time complexity reflects the real inference time, we can get:

$$\frac{T_{skim}}{T_{all}} \approx \frac{O_{skim}}{O_{all}} = R, \tag{16}$$

where T_{skim} and T_{all} is the inference time of skimming-based language model and original language model respectively. From Eq. 16, we can derive:

$$T_{skim} \approx R \cdot T_{all}, \tag{17}$$

where T_{all} is often treated as a constant value. In summary, the inference time on skimming-based language model T_{skim} is positively linear correlated with the remaining token ratio R.

Theorem 4 *Assumed skimming-based language model f is composed by K encoders, $R = \frac{1}{K}\sum_{i=1}^{K} r_i$ is the total remaining token ratio, where $\{r_i\}_{i=1}^{K}$ is the remaining token ratio of each encoder and $r_i \in (0, 1]$, and x is the input text and T is the corresponding inference time. If text length is not fixed and the length is N, then $T \propto R \cdot N$.*

Proof C.2 *Considering the scenario where the input length is not fixed, we assume two texts with two different lengths n_1 and n_2. The corresponding token remaining ratios are $R^1 = \frac{1}{K}\sum_{i=1}^{K} r_i^1$ and $R^2 = \frac{1}{K}\sum_{i=1}^{K} r_i^2$. According to Eq. 15, the time complexities are as follows:*

$$\begin{aligned} O^1_{skim} &\approx R^1 \cdot n_1 O(Kd^2), \\ O^2_{skim} &\approx R^2 \cdot n_2 O(Kd^2). \end{aligned} \tag{18}$$

Meanwhile, time complexity reflects the real inference time, from Eq. 18 we can get:

$$\frac{T^1_{skim}}{T^2_{skim}} \approx \frac{O^1_{skim}}{O^2_{skim}} = \frac{R^1 \cdot n_1}{R^2 \cdot n_2}, \tag{19}$$

where the inference time on skimming-based language model T_{skim} is influence by both the remaining token ratio R and length n. In summary, the inference time on skimming-based language model T_{skim} is positively linear correlated with product of the remaining token ratio R and text length n..