Understanding the Robustness of BERT Models Against Hardware Errors: An Experimental Study

Ruixuan Wang[1], Dongning Ma[2], and Xun Jiao[1]

[1] Villanova University
[2] Mohamed bin Zayed University of Artificial Intelligence

Abstract. As deep neural network (DNN) based natural language processing (NLP) models continue to grow in scale and complexity, the hardware systems supporting them are becoming increasingly intricate and sophisticated. This complexity leads to a higher likelihood of hardware errors that pose significant threats to the reliability and performance of NLP services. In this experimental study, we investigate the robustness of BERT models in the presence of hardware errors, which can corrupt model parameters stored in memory. Our study involves an extensive error injection campaign targeting four widely-used BERT models, evaluated across six benchmarks spanning three diverse application domains. Additionally, we demonstrate that applying parameter clipping techniques can significantly enhance the error tolerance of BERT models, achieving up to 100× improvement in robustness against hardware errors. Beyond error tolerance, we also explore the implications of these findings for energy-efficient hardware design, focusing specifically on SRAM. By combining SRAM voltage scaling techniques with the improved error tolerance of BERT models, we achieve average energy savings of up to 62.9%, with minimal impact on the performance of BERT models. Our results highlight the potential for optimizing hardware systems to support both reliable and energy-efficient execution of BERT models. The tool we developed for error injection is open-sourced at `https://github.com/Raisony/Hugging-Error`.

1 Introduction

Transformer-based language models now underpin most modern NLP systems, enabling not only strong performance on traditional tasks such as machine translation and sentiment analysis, but also reasoning, long-context understanding, and interactive conversational behavior in the natural language processing (NLP) domain [24, 11]. The success of these models largely depends on their ability to train and optimize large amounts of parameters in the millions or even billions. However, the increasing complexity and scale of these models demand significant computational resources. The development and deployment of these advanced language models require substantial support from high-performance computing

systems, along with large clusters of specialized accelerators to meet the ever-tightening quality of service (QoS) requirements. These include not only model performance but also maintaining low tail-latency to ensure responsiveness in real-time applications [1, 14, 12]. As NLP models continue to evolve, the need for more sophisticated computational devices will only intensify, highlighting the critical role of efficient hardware systems in sustaining this progress.

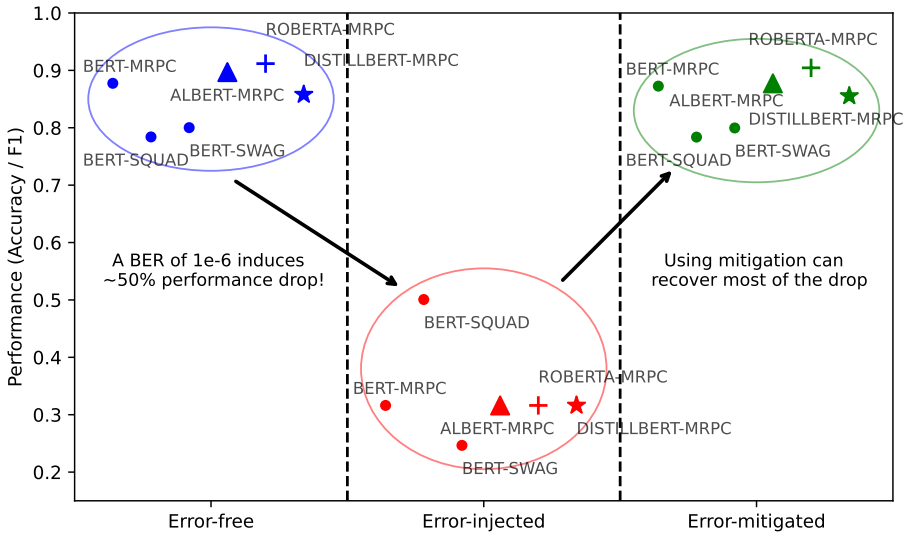


Fig. 1. Performance comparison between error-free, error-injected (BER: 10^{-6}), and error-mitigated BERT models. Injecting hardware-induced bit flip errors significantly degrades the performance of BERT models, whereas applying error mitigation techniques such as parameter clipping can largely restore performance.

As hardware systems continue to scale in both size and heterogeneity to accommodate the aforementioned computational demands, hardware errors have emerged as a significant reliability and dependability concern. Hardware errors (permanent faults, or transient soft errors) stem from various sources, including manufacturing defects, design flaws, environmental factors, aging and wear, and soft errors [3]. There is also a growing interest in exploring the accuracy-efficiency tradeoff in DNN systems, a concept commonly referred to as "approximate computing", which intentionally introduces errors as a part of design decisions to improve computational efficiency, using techniques such as voltage scaling [16, 5]. These intentional errors, while providing efficiency gains, also add another layer of complexity to the overall robustness problem.

BERT models, as one of the widely used NLP model families, have become an important foundation for many critical NLP tasks, i.e., acting as a base model for downstream applications such as text classification, question-answering, and

multiple-choice tasks. This paper uses the BERT family of models as our target, given its foundational nature for modern NLP models, and the main result of our experimental study is denoted in Fig. 1.

Our main contributions are as follows:

- We present a comprehensive experimental study on the robustness of BERT models against hardware-induced errors through extensive error injection campaigns, across six benchmarks covering three application domains.
- We develop a fast, flexible, and easy-to-use framework for error injection into the language models compatible with Hugging Face. Our tool is open-sourced to facilitate reproducible research and practical evaluation.
- We introduce a tensor-wise parameter clipping method as error mitigation that significantly enhances the robustness of BERT models, enabling 100× error tolerance improvement.
- We demonstrate by combining voltage scaling with enhanced error tolerance, we can achieve up to 62.9% energy savings with negligible impact on the performance of BERT models.

2 Related Work

As transistor technologies scale into the deep-nanometer regime and hardware architectures become increasingly heterogeneous, hardware errors have become a significant source of uncertainty that threatens the reliability of DNNs. For instance, certain faults (e.g., illegal memory access) can lead to system or job crashes, forcing the system to enable recovery strategies like re-execution or checkpointing [8]. Additionally, hardware errors caused by cosmic rays and radiation [21], as well as errors arising from manufacturing defects, environmental disturbances, and device aging [3], can further compromise system reliability.

Although prior studies have investigated the impact of hardware errors, they primarily focus on errors originating from compute units such as CPUs, often assuming that memory errors can be mitigated using error correction codes (ECC). However, recent research shows that bit flips can still occur in SRAM even with ECC mechanisms such as SEC-DED, which are effective for single-bit errors but become insufficient as error rates increase [20]. Furthermore, these hardware-based error mitigation methods are often intrusive, requiring additional hardware modifications, and can be cost-ineffective for the compute- and memory-intensive workloads typical of DNNs. To address these limitations, in this paper, we propose a parameter clipping strategy to enhance the robustness of BERT models against hardware errors without incurring additional hardware overhead.

On the other hand, recent studies have investigated the robustness of various DNN architectures, including MLPs, CNNs, RNNs, and transformers, under hardware-induced errors [17, 5], with error injection tools developed such as ares [17], PyTorchFI [9], and GoldenEye [10]. However, research on the robustness of transformer-based language models remains under-explored.

3 Methodology

We present an overview of the proposed error injection and mitigation framework for BERT models, as shown in Fig. 2. Specifically, we first describe the error models and error patterns, then detail the procedures for error injection and mitigation techniques in the BERT models, followed by the implementation of a supporting tool that facilitates both the error injection and mitigation processes. Each component is described in detail in the following sub-sections.

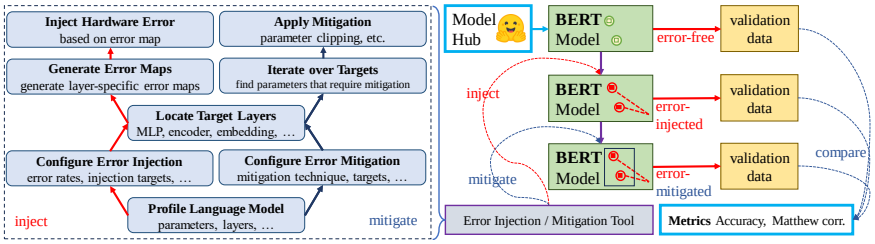


Fig. 2. Overview of the Error Injection and Mitigation of BERT Models

3.1 Error Model and Error Patterns

In this paper, we focus on hardware-induced errors in the memory of DNN accelerators that store model parameters. We adopt the bit flip error model (i.e., multi-bit flips), which is the most realistic manifestation of hardware errors in modern computing systems and is consistent with prior work [17, 16, 5, 20]. Under this error model, each bit has a certain probability of getting flipped, referred to as the bit error rate (BER).

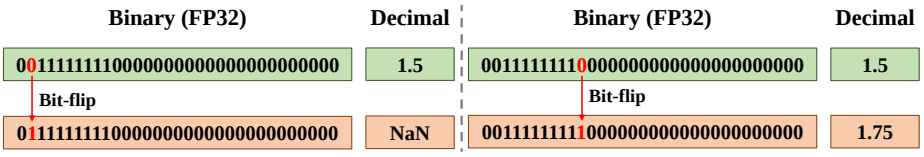


Fig. 3. Two types of bit flip hardware errors, the exceptional error (left) and non-exceptional error (right), follow the IEEE-754 floating point standard.

Most widely utilized BERT models [1, 6, 7, 18] employ the floating-point (FP) data type by default, following the IEEE 754 standard. Under this standard, an FP number consists of three fields of bits: a sign bit, an exponent field, and a mantissa field. In the bit flip error model, any bit position in each field

may flip from "1" to "0" or vice versa, inducing numerical deviation from the original value. Here, we summarize two categories of hardware errors based on the resulting numerical effects of bit flips: (1) **Exceptional hardware error**, where certain bit flips can lead to abnormal values, such as not-a-number (NaN) or infinity (Inf). As indicated in Fig. 3 (left), the exceptional hardware error could occasionally happen when the first bit in the exponent field of the floating point number is flipped. And (2) **Non-exceptional hardware error**, which encompasses bit flips that alter the numerical value without triggering floating-point exceptions, such as NaN or Inf, as shown in Fig. 3 (right).

3.2 Error Injection in BERT models

We consider two levels of granularity for performing the error injection into the BERT models: model-wise and component-wise. At the model-wise granularity, bit flip errors are injected into the trainable parameters across the entire model. In contrast, at the component-wise granularity, bit flip errors are selectively injected into specific types of components, such as the embedding or encoder layers, while all other components are error-free. In particular, we focus on two primary types of components in BERT models: embedding layers and encoder layers. The embedding layers combine token, segment, and positional (and other, if any) embeddings to project inputs into the feature space. The encoder layers then transform the embeddings into latent space representations, aggregated with contextual and semantic information for subsequent processing. The pooler, by contrast, is an auxiliary component in some of the BERT variants and is typically implemented as a single linear layer with significantly fewer parameters than the embedding and encoder layers. Consequently, for both conciseness and experimental focus, we exclude the pooler from component-wise error injection.

As illustrated in Fig. 4, we profile and visualize the distribution of model parameters based on the ALBERT model [6] as a representative example. We compute the proportion of parameters in each component, observing that the embedding, encoder, and pooler layers account for approximately 33%, 62%, and 5% of the total parameters, respectively. Given this parameter distribution, we primarily target the embedding and encoder layers for component-wise error injection due to their dominant contribution to the overall parameter count.

3.3 Error Mitigation via Parameter Clipping

Parameter clipping is a widely-used method to mitigate the impact of hardware errors in DNN models [4], which applies a "clipping function", as shown in Eq. (1), hard-limiting parameter values x based on pre-defined thresholds: an upper bound c and a lower bound f.

$$Clip(x, c, f) = max(f, min(x, c)) \tag{1}$$

We extend the model parameter clipping methods originally developed for CNNs [4] to BERT models. However, identifying appropriate unified upper and

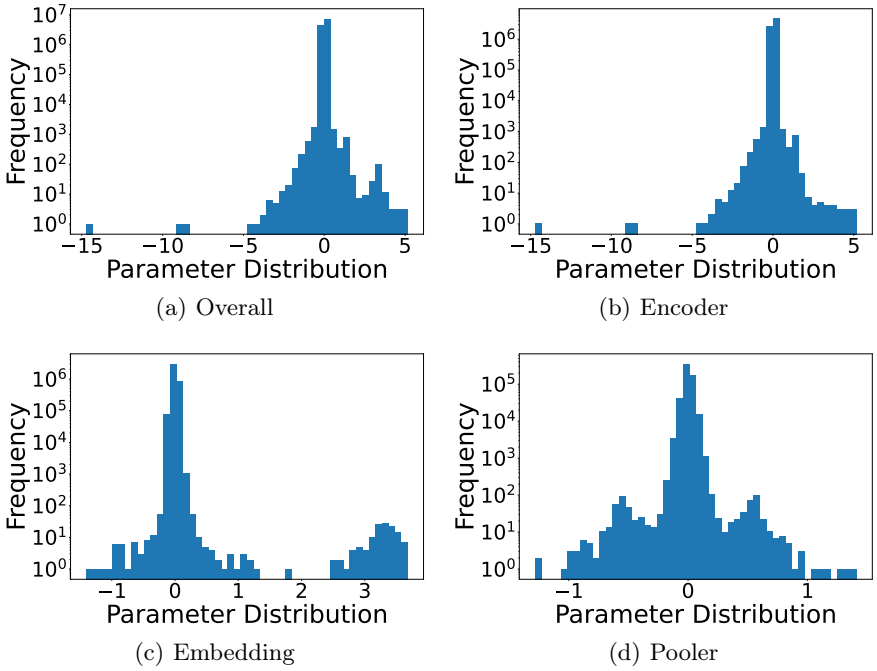


Fig. 4. Parameter distribution of the ALBERT model.

lower bounds for the parameter clipping function is challenging, as BERT architectures differ substantially from CNNs and parameters across different BERT components and layers exhibit heterogeneous distributions, as shown in Fig. 4. To address this challenge, we perform offline profiling of the error-free model to characterize the parameter value ranges of each layer in BERT models. Based on this profiling, we enable a customized clipping function based on layer-wise clipping bounds during online inference. In particular, to handle exceptional errors, NaN and Inf values are first sanitized through the "*torch.nan_to_num()*" function, after which parameter clipping is implemented via the "*torch.clamp()*" function with layer-specific bounds c and f.

3.4 Implementation

There are existing error injection frameworks or tools, such as ares [17], PyTorchFI [9], and GoldenEye [10]. However, the largest obstacle that prevents them from being practically applied to comprehensive error robustness evaluation is their relatively slow speed and efficiency due to the lack of GPU acceleration of error injection, particularly for larger and deeper models with more parameters and more injection locations, as observed in our implementation. In addition, existing frameworks provide limited flexibility for integrating error mitigation and protection mechanisms, and offer poor compatibility with mod-

ern transformer-based ecosystems, such as Hugging Face, making them difficult to use with standard pipelines from the *transformers* library.

Specifically, we highlight three notable advantages of our framework:

- **Minimal Dependency.** Our framework is solely PyTorch-based, i.e., relying on no other dependencies used in other frameworks such as *bitstring* or *struct*. This emancipates users from tedious Docker or environment setups and upsets from broken dependencies.
- **GPU Acceleration.** Our framework can be accelerated by GPUs as it leverages native PyTorch tensor APIs (*torch.view()*) for tensor type conversions to enable error injection at the granularity of individual bit positions in the data. This can significantly speed up the error injection process (by up to 100×) compared with existing frameworks or tools. We also do not require users to modify C back-ends of PyTorch or implement custom CUDA kernels to achieve GPU acceleration.
- **Mitigation Interface.** Our framework provides a flexible interface for users to incorporate their mitigation strategies. We implement the parameter clipping-based error mitigation by default, in which, if given a PyTorch model, a parameter-clipped version of it will be returned. Users can also implement their custom mitigation strategies under our interface.

Algorithm 1 Error injection into BERT model

Input Target model M, Target layer list T, Error Rate P
Output Error-Injected Model M'

1: **for** Layer L in M **do**
2: Layer Configuration $C_l \leftarrow$ Profile(L)
3: Error Configuration $C_e \leftarrow$ Profile(T, P)
4: **if** L in T **then**
5: Error Map $Emap \leftarrow$ Generation(L, C_l, C_e)
6: $L \leftarrow$ Injection(L, $Emap$)
7: **end if**
8: **end for**
9: **return** error-injected model M'

It is non-trivial to implement the bit flip error injection simulation with a particular BER, especially given the exceptionally large number of parameters and bits involved in the BERT model. In this work, we implement error injection following Alg. 1 and Fig. 2, which consists of the following steps: We first profile the specifications and statistics of the BERT model under evaluation, including data types, input specifications, layer modules, and parameter distributions (lines 1–2). We then configure the error injection, such as the predefined BER and injecting targets (lines 3). After that, we locate the target layers or components (line 4) and generate a corresponding error injection map based on the layer types and the data formats (line 5). The error map embeds whether a

parameter will be modified and which bit in the parameter to flip. Finally, we inject the hardware error into the BERT model by modifying parameter values based on the error map and return the error-injected model (lines 6–9).

Error mitigation follows a process similar to that shown in Fig. 2: We first profile the layer bounds, i.e., the upper/lower bounds for each layer in the BERT model. We then configure error mitigation using the layer bounds and apply parameter clipping to each layer, based on profiled information and statistics. Finally, we apply parameter clipping to the parameters in target components and return the robustness-enhanced version of the model.

4 Experimental Results

4.1 Experimental Setup

In this paper, we focus on BERT [1] and several of its derivative models, including ALBERT [6], ROBERTA [7], and DISTILBERT [18] models, which contain approximately 109.5M, 11.7M, 124.6M, and 67M trainable parameters, respectively. All models, together with their pre-trained model weights, are obtained directly from the Hugging Face repository. The parameter counts of individual components within each model are summarized in Tab. 1.

Table 1. Statistics of parameter counts in different BERT models and components

Model Name	Overall	Embedding Layers	Encoder Layers
BERT	109.5M	23.8M	85M
ALBERT	11.7M	3.9M	7.2M
ROBERTA	124.6M	39M	85.1M
DISTILBERT	67M	23.8M	42.5M

In the meantime, we employ three broadly adopted tasks in the NLP domain: text classification, question-answering, and multiple-choice, of which the datasets are from the GLUE [22], SQUADV2 [15], and SWAG benchmarks [23]. Specifically, the GLUE benchmark is a widely used suite of natural language understanding tasks for evaluating model performance across various classification and inference challenges. SQUADV2 is a question answering benchmark, consisting of posed questions and corresponding answers in text. In parallel, SWAG is a multiple-choice benchmark, consisting of questions focused on grounded commonsense inference. The detailed information of evaluation benchmarks and metrics is listed in Tab. 2. For all experiments, to ensure a fair and consistent comparison of BERT model performance before error injection, after error injection, and following the implementation of the error mitigation technique, we use the same validation datasets across the error-free, error injection, and error mitigation configurations.

Table 2. Statistics of evaluation benchmarks for different NLP tasks.

Task	# Train Samples	# Validation Samples	Metrics
MRPC	3668	408	Accuracy
RTE	2490	277	Accuracy
SST2	67439	872	Accuracy
COLA	8551	1043	Matthews Correlation
SWAG	73000	20000	Accuracy
SQUADV2	130319	11873	Exact Match, F1

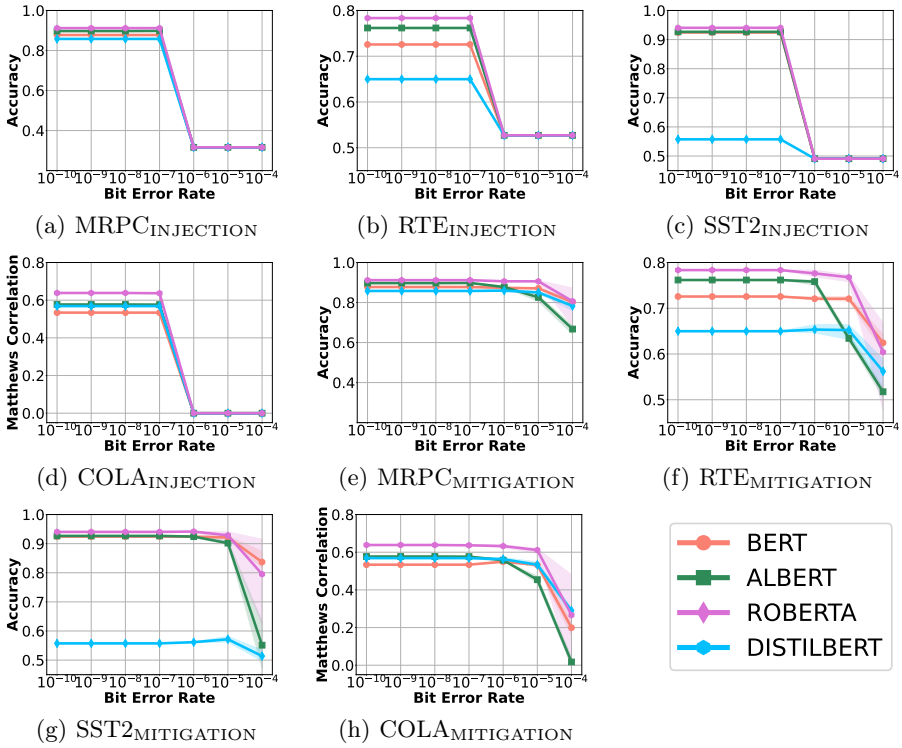
4.2 Results on Robustness Characterization

Key Observation 1: BERT models show **inherent** robustness to hardware errors up to a certain level before a sharp performance decline.

We first evaluate the hardware error robustness of BERT models under model-wise error injection, where the bit flip errors are injected into the parameters of the entire model. We select the BERs ranging from 10^{-10} to 10^{-4} and present the performance-BER curves in Fig. 5(a) to Fig. 5(d) as well as Fig. 6(a) and Fig. 6(c). We can observe that BERT exhibits a stable accuracy when the BER is smaller than a certain level, e.g., 10^{-7}, before which hardware errors are unlikely to cause noticeable model performance loss. However, once the BER increases beyond a certain point, the performance of BERT models will be significantly degraded. When the hardware errors are beyond this BER threshold, such performance decline becomes not "graceful" but "sharp", i.e., the model can quickly (within one order of magnitude beyond the cut-off point) become completely degraded. This phenomenon is similar to the trends of the robustness of MLP/CNN models indicated by previous studies [17].

We attribute this sharp performance degradation to the propagation of errors within the model. Typically, when the BER is higher than 10^{-7}, since the total number of bits of the parameter counts of BERT models is well above 10M, it is highly probable for the occurrence of multiple bit flips, including both exceptional and non-exceptional errors. The NaN and/or Inf values from those errors are therefore more likely to be carried through propagation, interact with other deviated values, and subsequently induce more exceptional errors. This cascading behavior resembles a "chain reaction" that can lead to catastrophic model performance degradation.

Key Observation 2: Different layers and components in the BERT model can exhibit different levels of robustness. Particularly, the embedding layers are more robust than the encoder layers against hardware errors.



(a) $\text{MRPC}_{\text{INJECTION}}$ (b) $\text{RTE}_{\text{INJECTION}}$ (c) $\text{SST2}_{\text{INJECTION}}$

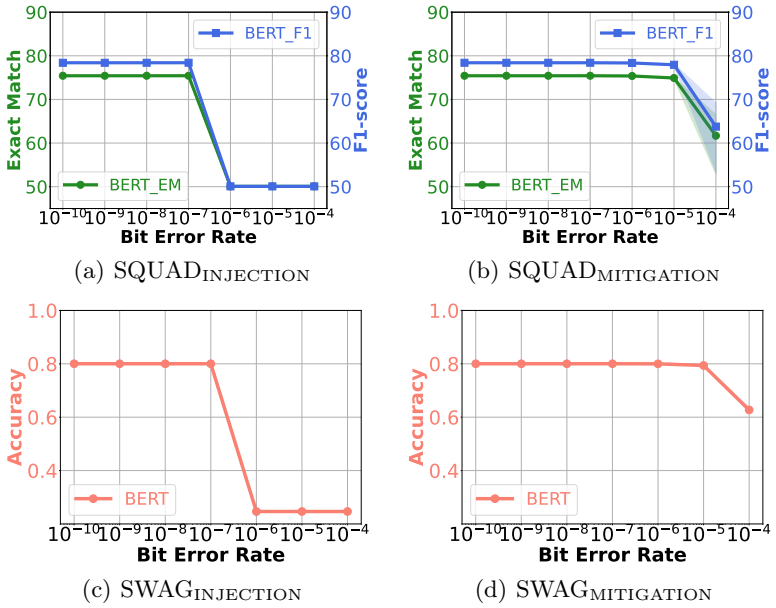
(d) $\text{COLA}_{\text{INJECTION}}$ (e) $\text{MRPC}_{\text{MITIGATION}}$ (f) $\text{RTE}_{\text{MITIGATION}}$

(g) $\text{SST2}_{\text{MITIGATION}}$ (h) $\text{COLA}_{\text{MITIGATION}}$

Fig. 5. Hardware error robustness evaluation of different BERT models under BERs in the range of $[10^{-10}, 10^{-4}]$. Here, we evaluate four text classification datasets (MRPC, RTE, SST2, COLA) from the GLUE benchmark.

In the meantime, we conduct the component-wise error injection, and we skip the first two BERs since the impact of errors is not imminent. The experimental results are denoted in Fig. 7, where we demonstrate that the encoder layers have a performance-BER trend with a cut-off BER of 10^{-7}. However, the error injection on embedding layers does not induce significant performance drops until BER increases beyond 10^{-5}, which indicates that embedding layers are $100\times$ more error-tolerant compared with encoder layers. In addition, we illustrate an example that compares the output of the BERT model when injecting errors into the embedding **or** encoder layers in Fig. 8. Although the BERT models are impacted in both cases, the encoder layers are more sensitive to hardware errors compared with the embedding layers, by having completely broken outputs rather than a wrong but readable answer.

4.3 Results on Error Mitigation



(a) SQUAD$_{\text{INJECTION}}$ (b) SQUAD$_{\text{MITIGATION}}$

(c) SWAG$_{\text{INJECTION}}$ (d) SWAG$_{\text{MITIGATION}}$

Fig. 6. Hardware error robustness of the BERT model under BERs in the range of $[10^{-10}, 10^{-4}]$. The model is evaluated on the SQUADV2 and SWAG benchmarks.

Key Observation 3: Parameter clipping can effectively mitigate the impact of hardware errors in BERT models.

We further evaluate parameter clipping as the hardware error mitigation, with the corresponding performance–BER curves for error-mitigated BERT models shown in Fig. 5(e) to Fig. 5(h) as well as Fig. 6(b) and Fig. 6(d), across three NLP benchmarks. Compared to unmitigated BERT models, parameter clipping can significantly improve the robustness of BERT models across all models and benchmarks. In particular, the cut-off BER at which performance degradation occurs is shifted by up to two orders of magnitude, corresponding to as much as 100× improvement in error tolerance.

The most notable advantage of the error mitigation using parameter clipping lies in its efficiency. The error mitigation mechanism is implemented using the tensor-wise clipping function and therefore incurs negligible additional memory or computational overhead, in contrast to element-wise correction techniques such as TMR and ECC. Moreover, the clipping thresholds can also be determined offline through profiling, requiring significantly less time and resources than alternative offline approaches such as retraining or fine-tuning.

4.4 Ablation Study: Influence of BERT Model Configuration

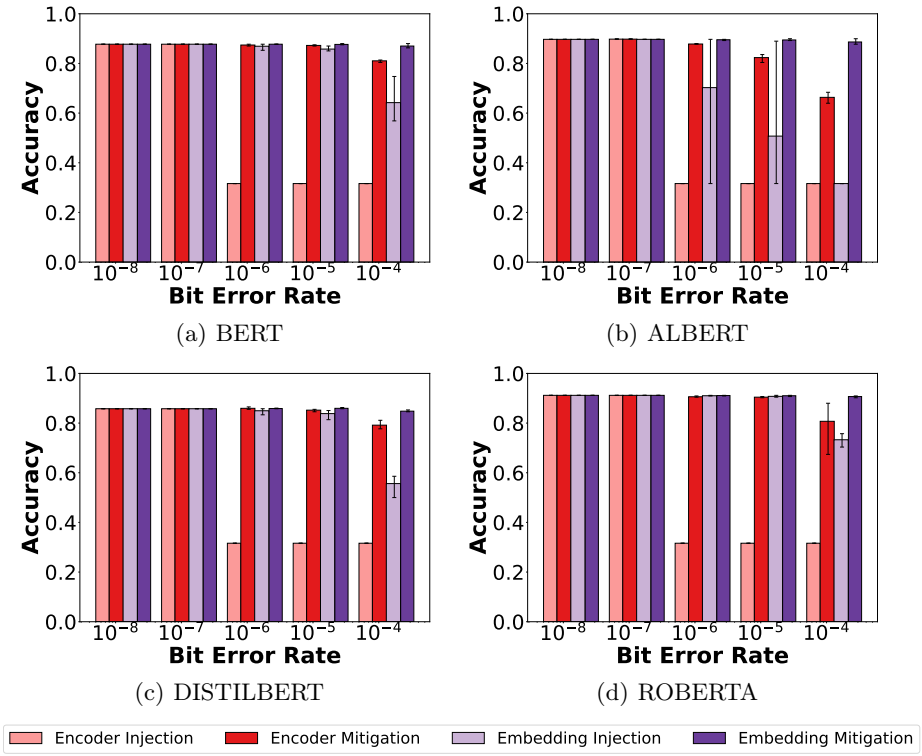


Fig. 7. Hardware error robustness of BERT model components under BER in the range of $[10^{-8}, 10^{-4}]$. Here, we evaluate the embedding and encoder layers in four different BERT models on the MRPC dataset.

Key Observation 4: Different architectural hyperparameters could also impact the hardware error robustness of BERT models.

We first investigate the influence of different tokenizer designs by comparing $\text{DISTILBERT}_{\text{UNCASED}}$ and $\text{DISTILBERT}_{\text{CASED}}$, where the uncased variant model lower-cases the input texts before the embedding. As denoted in Fig. 9(a) and Fig. 9(b), the selection of tokenizers will not significantly enhance the hardware error robustness of DISTILBERT models under both model-wise error injection and component-wise error injection to encoder layers. However, the result of hardware error injection in embedding layers, as revealed in Fig. 9(c), shows that $\text{DISTILBERT}_{\text{CASED}}$ exhibits higher robustness than $\text{DISTILBERT}_{\text{UNCASED}}$. For instance, at 10^{-4} BER, the embedding layers in $\text{DISTILBERT}_{\text{UNCASED}}$ experience a more than 20% accuracy drop compared with 5% for $\text{DISTILBERT}_{\text{CASED}}$. This discrepancy can be attributed to differences in vocabulary size between the cased and uncased tokenizers, which affect the structure and sensitivity of the embedding layers. Our results suggest that

Context: In the aftermath of generally poor French results in most theaters of the Seven Years' War in 1758, France's new foreign minister, the duc de Choiseul, decided to focus on an invasion of Britain, to draw British resources away from North America and the European mainland. The invasion failed both militarily and politically, as Pitt again planned significant campaigns against New France, and sent funds to Britain's ally on the mainland, Prussia, and the French Navy failed in the 1759 naval battles at Lagos and Quiberon Bay. In one piece of good fortune, some French supply ships managed to depart France, eluding the British blockade of the French coast.
Question: What naval battles did France lose in 1759?
Outputs from error-free model: Lagos and Quiberon Bay
Outputs from model with errors injected to encoder layers: In
Outputs from model with errors injected to embedding layers: the Seven Years ' War

Fig. 8. Example outputs of the BERT model with the SQUADV2 benchmark, errors are injected into encoder layers and embedding layers separately.

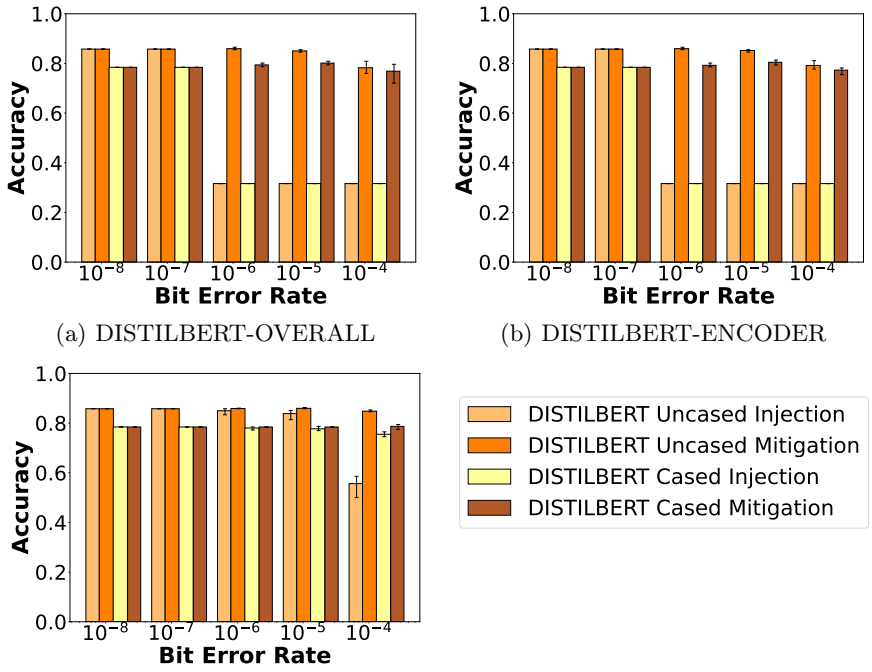
the uncased tokenizer leads to lower robustness to hardware errors, with the embedding layers being particularly vulnerable due to their direct interaction with tokenizer outputs.

We also examine the impact of different hidden state dimensions on model robustness. Specifically, we evaluate two variants of the ALBERT model, $\text{ALBERT}_{\text{BASE}}$ and $\text{ALBERT}_{\text{LARGE}}$, with hidden state dimensions of 768 and 1024, respectively. As shown in Fig. 10(a) to Fig. 10(c), $\text{ALBERT}_{\text{LARGE}}$ exhibits better robustness than $\text{ALBERT}_{\text{BASE}}$. We attribute this improvement to the approximately 50% increase in the parameter counts resulting from the larger hidden state dimension.

4.5 Ablation Study: Energy Saving via Voltage Scaling

Key Observation 5: The inherent error tolerance of BERT models can be exploited to achieve substantial energy savings through voltage scaling. Moreover, applying error mitigation techniques further enhances the achievable energy savings.

Building on Key Observation 1, we observe that BERT models exhibit a certain level of inherent tolerance to bit flip errors. This intrinsic robustness of BERT models can be directly translated into energy savings via voltage scaling, as demonstrated in prior work [2, 16]. To systematically analyze the trade-off between energy efficiency and model performance, we apply voltage scaling to on-chip SRAM implemented in FDX22 (22nm) technology based on the simulation approach in [2], which stores BERT model weight parameters. We then



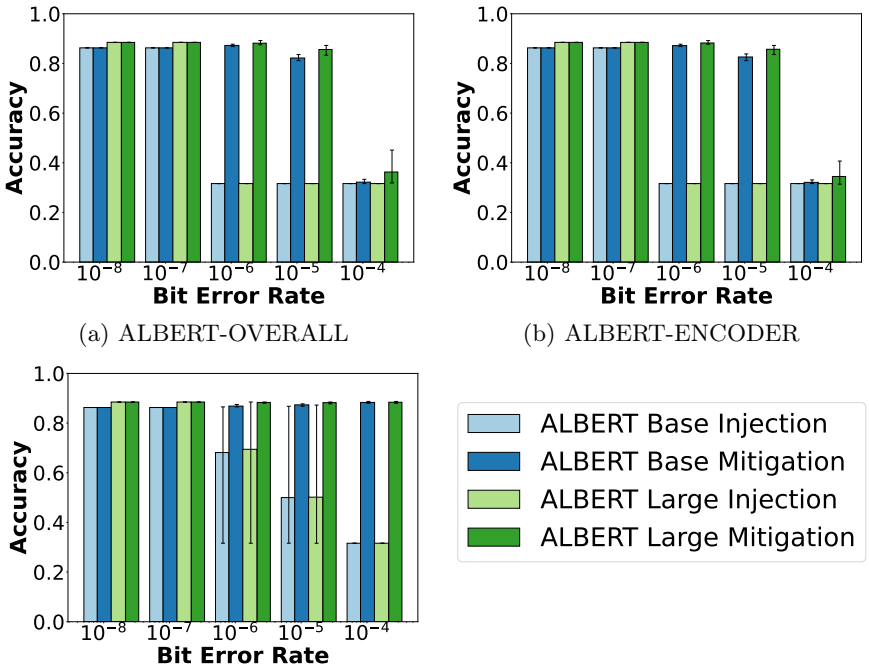
(a) DISTILBERT-OVERALL (b) DISTILBERT-ENCODER

(c) DISTILBERT-EMBEDDING

Fig. 9. Hardware error robustness of BERT models with different model configurations under BER in the range of $[10^{-8}, 10^{-4}]$. Here we evaluate the $\text{DISTILBERT}_{\text{UNCASED}}$ and $\text{DISTILBERT}_{\text{CASED}}$ models on the MRPC dataset in Fig. 9(a) (model-wise), Fig. 9(b) (component-wise, encoder layers), and Fig. 9(c) (component-wise, embedding layers).

evaluate the energy savings enabled by both the inherent robustness of BERT models and the improved robustness achieved through mitigation techniques.

In this context, SRAM operates at a nominal supply voltage $V_{dd} = 0.8V$, corresponding to a BER $< 10^{-13}$. The evaluation of energy saving is represented as the ratio of reduced energy consumption between the voltage-scaled and the nominal SRAM, based on the formula $P = CV^2f$. Our experimental results demonstrate that the inherent robustness of the BERT models yields an average of 49% energy savings across all benchmarks, with less than 5% accuracy loss. Furthermore, when parameter clipping is applied as an error mitigation technique, the energy saving ratio can be further improved to 61.5%, 57%, 61.5%, and 62.9% for BERT, ALBERT, ROBERTA, and DISTILBERT models, respectively. Notably, the parameter clipping approach incurs negligible hardware design overhead, as it can be performed offline before model deployment. A detailed discussion on the energy model and the associated peripheral hardware overheads is provided in Sec. 5



(a) ALBERT-OVERALL (b) ALBERT-ENCODER

(c) ALBERT-EMBEDDING

Fig. 10. Hardware error robustness of BERT models with different model configurations under BER in the range of $[10^{-8}, 10^{-4}]$. Here we evaluate the $\text{ALBERT}_{\text{BASE}}$ and $\text{ALBERT}_{\text{LARGE}}$ models on the MRPC dataset in Fig. 10(a) (model-wise), Fig. 10(b) (component-wise, encoder layers), and Fig. 10(c) (component-wise, embedding layers).

5 Limitations and discussions

Language Models. The language models we evaluated are all from the BERT family. As an experimental study and due to the limited computing resources we have, we limit our scope of language models within the BERT family so as to clearly and concisely present our results and observations. However, the framework we developed does not limit the models it can inject errors into within the BERT family. Therefore, our future work will extend our evaluation methodology to a broader range of language model families, including large language models (LLMs), under diverse quantization schemes that better reflect real-world deployment conditions, when more computing resources are readily available to us. Although beyond the scope of this paper, we plan to extend our analysis on language models to vision transformers (ViTs) to derive cross-domain insights into transformer robustness, including comparative evaluations of image patch embeddings and text token embeddings.

Error Model. In this study, we assume a random bit flip error model for error injection experiments for its wide usage and alignment with existing studies.

We acknowledge that there are different error models across different hardware platforms, and the error model used in this paper does not fully represent all possible error models. For example, burst errors can upset one or more adjacent bits [13]. There are also additional types of hardware errors, such as permanent stuck-at faults due to manufacturing defects and aging [19]. Thus, another line of our future effort will focus on expanding the usability of our open-source tools to more error models.

Error Mitigation. In this study, the parameter clipping approach relies on an offline, error-free profiling of the static weight distributions for each layer. Although fine-tuning on a new downstream task modifies model weights and may necessitate additional layer-wise bound profiling, the profiling overhead is computationally negligible compared to the retraining or fine-tuning process. This efficiency ensures that the mitigation strategy can be rapidly recalibrated for model variants in dynamic software quality engineering pipelines. Furthermore, because the clipping thresholds are tied to the static parameters rather than dynamic activations, the mitigation remains stable during long-term deployment regardless of potential input-level data drift once the model weights are fixed.

Hardware Energy Validation. Our evaluation of energy efficiency is primarily based on a theoretical model of SRAM power consumption ($P = CV^2f$) [2]. While this provides an illustrative estimate of the energy-saving potential of weight-storage voltage scaling, it does not fully account for the architectural complexities of modern AI accelerators. In a realistic deployment environment, peripheral hardware overheads, such as control logic for managing multiple voltage domains or the power cost of interfacing voltage-scaled memory with standard-voltage compute units, may affect the overall system-level efficiency. Although the parameter clipping approach we proposed incurs negligible hardware overhead, our future work will involve detailed architectural simulations and hardware-aware validation to precisely quantify these impacts within specific accelerator designs.

Potential Risks. This study empirically examines how hardware uncertainties, such as bit errors, can affect the performance and reliability of BERT models. Our analysis also reveals the varying sensitivities of different model components, architectures, and datasets to such errors. While these insights are valuable for improving model robustness, they could, in theory, be misused by adversaries seeking to deliberately induce hardware errors and manipulate model behavior, potentially leading to incorrect or harmful outputs. We emphasize that our work is intended to promote awareness of these risks and to motivate the development of proactive defenses. By identifying and understanding these vulnerabilities, the research community can better design safeguards to ensure the reliable, safe, and ethical deployment of language models in real-world systems.

6 Conclusion

In this paper, we present an experimental study on the robustness of BERT models against the adverse effects of hardware errors. Our primary objective

is to comprehensively scrutinize the interplay between hardware errors and the accuracy of model inferences. To this end, we performed a rigorous error injection campaign for four BERT models under six benchmarks. Experimental results show that BERT models exhibit a certain degree of robustness against hardware errors that corrupt their parameters. Moreover, to improve model robustness, we introduce a parameter clipping-based error mitigation approach, which can substantially enhance the robustness of BERT models by up to 100×. In addition, we leverage the improved error tolerance of the BERT models for energy-efficient hardware design on SRAM based on voltage scaling, which enables up to 62.9% energy savings with negligible accuracy loss.

Acknowledgements This work was partially supported by NSF grant #2202310. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
2. Alfio Di Mauro et al. Pushing on-chip memories beyond reliability boundaries in micropower machine learning applications. *IEDM*, 2019.
3. Puneet Gupta et al. Underdesigned and opportunistic computing in presence of hardware variability. *TCAD*, 2012.
4. Le-Ha Hoang et al. Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation. In *DATE*, 2020.
5. Sung Kim et al. Matic: Learning around errors for efficient low-voltage neural network accelerators. In *DATE*, 2018.
6. Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.
7. Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
8. Kiwan Maeng et al. Understanding and improving failure tolerant training for deep learning recommendation with partial recovery. *Proceedings of Machine Learning and Systems*, 3:637–651, 2021.
9. Abdulrahman Mahmoud et al. Pytorchfi: A runtime perturbation tool for dnns. In *DSN-W*, 2020.
10. Abdulrahman Mahmoud et al. Goldeneye: A platform for evaluating emerging numerical data formats in dnn accelerators. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 206–214. IEEE, 2022.
11. Guanqiao Qu, Qiyuan Chen, Wei Wei, Zheng Lin, Xianhao Chen, and Kaibin Huang. Mobile edge intelligence for large language models: A contemporary survey. *IEEE Communications Surveys & Tutorials*, 2025.

12. Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.
13. Aleksandar Radonjic. Integer codes correcting single errors and detecting burst errors within a byte. *IEEE Transactions on Device and Materials Reliability*, 20(4):748–753, 2020.
14. Colin Raffel et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.
15. Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.
16. Brandon Reagen et al. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *ISCA*. IEEE, 2016.
17. Brandon Reagen et al. Ares: A framework for quantifying the resilience of deep neural networks. In *DAC*, 2018.
18. Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.
19. Vilas Sridharan and Dean Liberty. A study of dram failures in the field. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE, 2012.
20. David Stutz, Nandhini Chandramoorthy, Matthias Hein, and Bernt Schiele. Bit error robustness for energy-efficient dnn accelerators. *Proceedings of Machine Learning and Systems*, 3:569–598, 2021.
21. Michael B Sullivan, Nirmal Saxena, Mike O'Connor, Donghyuk Lee, Paul Racunas, Saurabh Hukerikar, Timothy Tsai, Siva Kumar Sastry Hari, and Stephen W Keckler. Characterizing and mitigating soft errors in gpu dram. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 641–653, 2021.
22. Alex Wang et al. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.
23. Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. Swag: A large-scale adversarial dataset for grounded commonsense inference. *arXiv preprint arXiv:1808.05326*, 2018.
24. Wayne Xin Zhao et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

A Additional Example Model Outputs

According to Key Observation 2 mentioned in Sec. 4.2, the encoder layers are typically more sensitive to errors. Here we present several example outputs of the BERT model with the SQUADV2 benchmark, where we **separately** inject errors in encoder layers and embedding layers using the same BER. We can observe that when errors are injected into encoder layers, outputs from the model are completely broken and unrelated to the context provided. When errors are injected into embedding layers, although the outputs are still incorrect, they are still readable and related to the context provided.

Context: Tension forces can be modeled using ideal strings that are massless, frictionless, unbreakable, and unstretchable. They can be combined with ideal pulleys, which allow ideal strings to switch physical direction. Ideal strings transmit tension forces instantaneously in action-reaction pairs so that if two objects are connected by an ideal string, any force directed along the string by the first object is accompanied by a force directed along the string in the opposite direction by the second object. By connecting the same string multiple times to the same object through the use of a set-up that uses movable pulleys, the tension force on a load can be multiplied. For every string that acts on a load, another factor of the tension force in the string acts on the load. However, even though such machines allow for an increase in force, there is a corresponding increase in the length of string that must be displaced in order to move the load. These tandem effects result ultimately in the conservation of mechanical energy since the work done on the load is the same no matter how complicated the machine.
Question: What is the final effect of adding more and more idea strings to a load?
Outputs from error-free model: conservation of mechanical energy
Outputs from model with errors injected to encoder layers: Ten
Outputs from model with errors injected to embedding layers: Ideal strings transmit tension forces instantaneously in action - reaction pairs so that if two objects are connected by an ideal

Context: To measure the difficulty of solving a computational problem, one may wish to see how much time the best algorithm requires to solve the problem. However, the running time may, in general, depend on the instance. In particular, larger instances will require more time to solve. Thus, the time required to solve a problem (or the space required, or any measure of complexity) is calculated as a function of the size of the instance. This is usually taken to be the size of the input in bits. Complexity theory is interested in how algorithms scale with an increase in the input size. For instance, in the problem of finding whether a graph is connected, how much more time does it take to solve a problem for a graph with 2n vertices compared to the time taken for a graph with n vertices?
Question: How is the time needed to obtain the solution to a problem calculated?
Outputs from error-free model: as a function of the size of the instance
Outputs from model with errors injected to encoder layers: To
Outputs from model with errors injected to embedding layers: a computational problem, one may wish to see how much time the best algorithm requires to solve the problem

Fig. 11. Additional example outputs of the BERT model with the SQUADV2 benchmark, with errors injected into encoder layers and embedding layers separately.

B Visualization of Model Activations

To further investigate the performance degradation at the critical BER threshold, we perform a case study to analyze the activation distributions in the BERT model using samples from the RTE dataset at the BER of 10^{-6} in Fig. 12. We show that the impact of errors can cascade and accumulate across the layers, leading to increasing corruptions on the activations. For example, earlier layers in the error-injected model, such as the embedding layers, maintain a "stable" activation distribution even if errors are present. However, when the error propagates as inference continues to later layers, such as the attention layers, NaNs and abnormally large values (values distributed higher than 10^{34}) start to appear, and culminate at the feed-forward networks (FFN) layers, where every output degrades into NaN. This confirms our observation that exceptional errors propagate aggressively through the attention layers and ultimately lead to catastrophic model performance degradation.

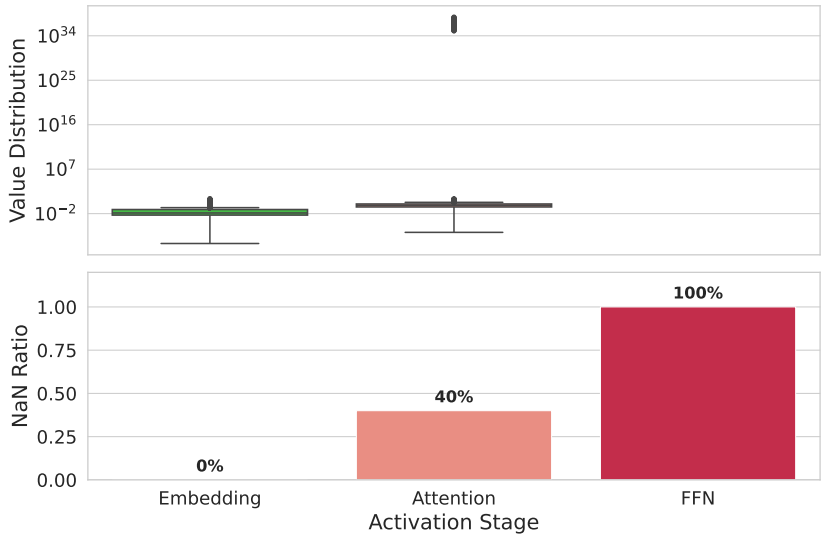


Fig. 12. Case Study: Activation distributions in the embedding layers and first attention layers of the BERT model, under entire model error injection at the BER 10^{-6}. The upper panel presents the distribution of all non-NaN values, while the lower panel shows the proportion of NaN values within the activations.