# `BitHydra`: Towards Bit-flip Inference Cost Attack against Large Language Models

**Xiaobei Yan**[1]    **Yiming Li**[1]*    **Zhaoxin Fan**[2]
**Han Qiu**[3]    **Tianwei Zhang**[1]
[1]Nanyang Technological University    [2]Beijing University of Aeronautics and Astronautics
[3]Tsinghua University
xiaobei002@e.ntu.edu.sg, liyiming.tech@gmail.com, zhaoxinf@buaa.edu.cn
qiuhan@tsinghua.edu.cn, tianwei.zhang@ntu.edu.sg

## Abstract

Large language models (LLMs) have shown impressive capabilities across a wide range of applications, but their ever-increasing size and resource demands make them vulnerable to inference cost attacks, where attackers induce victim LLMs to generate the longest possible output content. In this paper, we revisit existing inference cost attacks and reveal that these methods can hardly produce large-scale malicious effects since they are *self-targeting*, where attackers are also the users and therefore have to execute attacks solely through the inputs, whose generated content will be charged by LLMs and can only directly influence themselves. Motivated by these findings, this paper introduces a new type of inference cost attacks (dubbed 'bit-flip inference cost attack') that target the victim model itself rather than its inputs. Specifically, we design a simple yet effective method (dubbed '`BitHydra`') to effectively flip critical bits of model parameters. This process is guided by a loss function designed to suppress <EOS> token's probability with an efficient *critical bit search* algorithm, thus explicitly defining the attack objective and enabling effective optimization. We evaluate our method on 11 LLMs ranging from 1.5B to 14B parameters under both `int8` and `float16` settings. Experimental results demonstrate that with just 4 search samples and as few as 3 bit flips, `BitHydra` can force 100% of test prompts to reach the maximum generation length (*e.g.*, 2048 tokens) on representative LLMs such as LLaMA3—highlighting its efficiency, scalability, and strong transferability across unseen inputs.

## 1 Introduction

Large Language Models (LLMs) [65, 48, 8] have demonstrated remarkable capabilities across a wide range of real-world applications, including online chat [60], customer service [24], and financial services [68]. As LLMs are increasingly deployed through cloud-based ML-as-a-Service (MLaaS) platforms, minimizing inference cost has become critical for both service providers and end-users—enhancing service availability and reducing token-based billing costs.

However, previous studies have shown that deep neural networks are vulnerable to inference cost attacks [61, 44, 59, 56, 58, 69, 42, 40], where the attacker crafts malicious input to maximize the latency and cost of the victim model's inference execution. Such attacks can lead to substantial operational overhead for service providers and degrade the user experience. Recently, researchers designed inference cost attacks against auto-regressive LLMs [20, 23, 19, 35] and multimodal LLMs [22]. As the victim model's inference cost scales with the response length, the attacker's objective is to mislead the model to generate as many tokens as possible using short prompts.
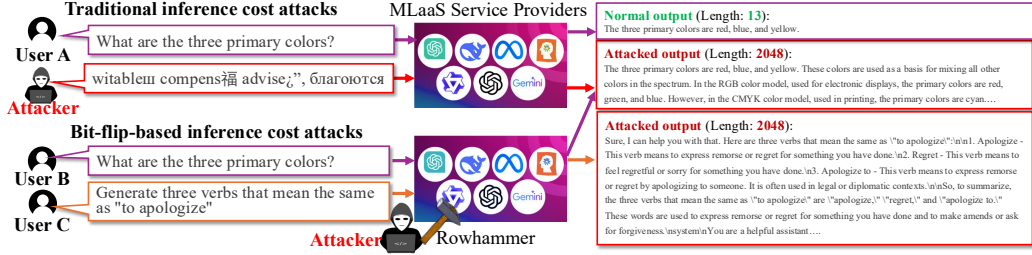
---

*Corresponding author.

Figure 1: Comparison between traditional and our bit-flip-based inference cost attack. Traditional attacks are inherently self-targeting, relying on adversarial input prompts, which solely affect attackers' own queries and incur cost on themselves. Our approach directly modifies the model weights, enabling persistent and widespread impact on all users interacting with the compromised model.

Despite their diversity, existing inference cost attacks share a key feature: they rely on specially-crafted inputs to induce excessive computation. Consequently, this leads to two significant limitations in real-world scenarios. **(1)** These attacks are inherently *self-targeting*: the attacker, who submits the adversarial prompt, will be charged for the long generated responses, bearing the inference cost. **(2)** To achieve damages to other users and service providers at scale, the attacker needs to consistently send a large volume of malicious input, which can be costly and easy to spot.

We argue that these limitations mainly stem from their threat model, where attackers are also the users and therefore have to execute inference cost attacks through their input. Motivated by these understandings, this paper proposes a new type of inference cost attacks that *target the model itself rather than the input*. The key insight is to manipulate a small number of critical weight bits, which can induce excessive computation for all normal queries from any user, as shown in Figure 1. Specifically, we can leverage Bit-Flip Attack (BFA) [51, 52, 9], a commonly-used technique, to precisely flip the desired bits in memory by exploiting hardware vulnerabilities such as Rowhammer [45], undervolting [54], or laser injection [6]. While prior BFA research in the context of machine learning mainly focuses on the objective of misclassification for DNNs [53, 52, 9] or jailbreaks for LLMs [16], this is the first study to demonstrate the use of BFAs to launch inference cost attacks, enabling persistent and user-transparent service degradation.

However, there are several technical challenges to realize the above idea. **(1)** *How can we design an effective loss function that induces the LLMs to produce generative content*? **(2)** *Given the massive number of weight bits in LLMs, how can we identify the critical bits that affect the inference cost*? **(3)** *How can we ensure that after flipping these bits, the victim model can still generate syntactically and semantically valid outputs*? In this paper, we design a simple yet effective attack method (dubbed 'BitHydra') to tackle the aforementioned challenges. Specifically, we first design a loss function (*i.e.*, $\mathcal{L}_{\texttt{<EOS>}}$) that penalizes high termination likelihoods by suppressing the normalized probability of the `<EOS>` token (*i.e.*, the signal of output termination). Intuitively, by minimizing $\mathcal{L}_{\texttt{<EOS>}}$, BitHydra causes the model to avoid generating the end-of-sequence token, thereby producing abnormally long outputs without degrading the model's general functionality. To address the challenges of scalability and semantic integrity, BitHydra further introduces a lightweight and efficient *Critical Bit Search* algorithm. Instead of performing an exhaustive bit-level search across the full parameter space, it strategically narrows the search to the output embedding layer and focuses only on the vector corresponding to the `<EOS>` token. This targeted approach drastically reduces the search complexity, making it feasible to identify high-impact bits quickly. At the same time, by modifying only a small, isolated portion of the model—without touching broader language representations—BitHydra preserves the model's ability to produce syntactically and semantically valid outputs, enabling stealthy and persistent attacks with minimal disruption to normal behavior.

In summary, our main contributions are four-fold. **(1)** We revisit existing inference cost attacks and reveal their inherent limitations and inherent reasons. **(2)** Based on our findings, we propose a new inference cost attack paradigm that targets model parameters rather than inputs, allowing large-scale persistent attacks that affect all users. **(3)** We design BitHydra, a simple yet effective method that suppresses output termination with just a few carefully chosen bit flips. **(4)** We demonstrate the effectiveness of BitHydra through extensive experiments, showing that it causes 100% of evaluation prompts to reach the maximum generation length—boosting the average output from 260 to 2048 tokens—on representative LLMs like Llama3-8B, while requiring as few as three bit flips in some cases. We also demonstrate its robustness in resisting potential defenses.

## 2 Background

### 2.1 Inference Cost Attacks

Inference cost attacks aim to exploit the compute-intensive nature of deep learning models to intentionally increase the models' latency or resource consumption during inference, ultimately leading to high compute cost and degraded user experience. Shumailov et al. [61] introduced the concept of *sponge examples* and designed the first inference cost attack: inputs designed to maximize the $\ell_2$ norm of internal activations across all layers. By amplifying the magnitude of intermediate representations, these inputs force the model to consume more computational resources during the forward propagation. Later works extended this attack across various tasks and domains, including image classification [44, 27] and understanding [11], object detection [59, 56, 58, 69, 42], 3D point cloud perception [40], and language translation [10].

Recent studies showed that this inference cost threat can be exacerbated in the context of LLMs. For instance, LLMEffiChecker [20] introduced a gradient-guided approach to search for minimal and imperceptible perturbations that can increase the inference cost. Geiping et al. [23] presented a targeted attack that coerces LLMs into generating specific starting responses, indirectly imposing higher computational cost. Dong et al. [19] designed adversarial prompts that prolong the decoding process in modern auto-regressive LLMs. Kumar et al. [35] proposed to intentionally make the LLM overthink, slowing down its reasoning process. Gao et al. [22] crafted verbose images to increase the latency and energy consumption of multimodal LLMs.

However, to the best of our knowledge, all existing methods achieve the attack damage via manipulating the model's inputs, which brings two practical limitations. Firstly, modern LLM service providers typically adopt a token-based billing scheme. For example, OpenAI's o3 API charges $10.00 per 1 million input tokens and $40.00 per 1 million output tokens [47]. As a result, although generating abnormally long outputs can increase the computational burden for the service provider, it is finally the attacker paying for damages, whereas the service provider has only mild potential losses (*e.g.*, leading the server to be busy, which in turn affects the experience of other users). Secondly, each adversarial input can only affect the corresponding inference process, failing to cause universal impacts on other users. These serve as our motivation to design new inference cos attacks.

### 2.2 Bit-Flip Attacks via Rowhammer

Bit-Flip Attacks (BFAs) exploit hardware-level vulnerabilities to alter the model weights stored in memory. A common technique is Rowhammer, which causes charge leakage in memory cells and bit flips via accessing adjacent memory rows repeatedly [32]. This vulnerability has been demonstrated in various platforms, including CPUs [57, 36, 21, 29, 28, 34, 25], and more recently, GPUs equipped with high-bandwidth HBM2 memory [46]. Generally, current BFAs against deep learning fall into two categories: (1) *Untargeted attacks* [51, 12, 39], which aim to degrade the model's overall performance; and (2) *Targeted attacks* [52, 53, 16], which manipulate the model's behavior in specific ways, e.g., misclassification for targeted classes or bypassing safety filters in LLMs. These attacks often rely on dedicated bit-search algorithms to locate the most vulnerable and impactful bits for flipping.

## 3 Attack Overview

### 3.1 Threat Model

To overcome the limitations of existing attacks, we shift our focus from input manipulation to weight manipulation by exploiting the BFA technique. In our attack paradigm, the attacker identifies the critical bits in the target LLM and flips them, which could cause the model to generate long responses for any prompt from any user, achieving the large-scale attack effects.

Therefore, we follow the conventional threat model adopted in prior BFA research [41, 70, 51, 39]. Specifically, the target LLM is assumed to be deployed in a resource-sharing environment, such as a Machine-Learning-as-a-Service (MLaaS) platform. The attacker is an unprivileged tenant sharing the same physical machine as the victim's LLM instance. He uses Rowhammer [33], a practical fault injection technique capable of flipping bits in DRAM without requiring elevated system privileges. The attacker has white-box access to the victim model: he does not possess the training data but has knowledge of the model's architecture and weights. This assumption is justified by the widespread use of open-source LLMs in user-facing services [43].
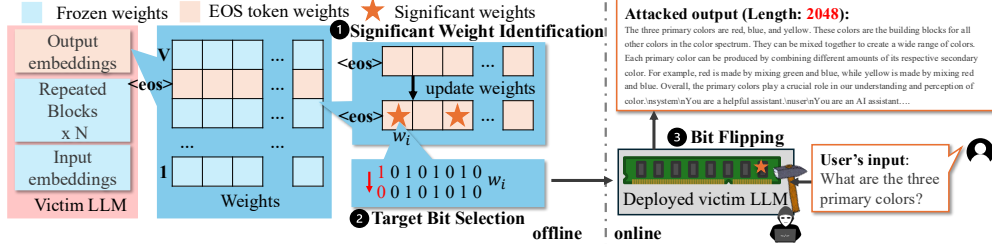
Figure 2: Overview of `BitHydra`. `BitHydra` consists of three stages: (1) **Significant Weight Identification**: Attackers identify significant weights within the `<EOS>` token's embedding row guided by the loss that penalizes the probability of generating the `<EOS>` token; (2) **Target Bit Selection**: Attackers select the bit flips needed to approximate the target weight changes; and (3) **Bit Flipping**: attackers use Rowhammer to physically induce the selected bit errors in DRAM.

## 3.2 Main Challenges

Achieving BFA-based inference cost attacks needs to meet the following requirements. Firstly, since the feasibility of such attacks highly depends on the number of bits flipped, a successful attack must flip as few bits as possible to remain efficient and practical. Secondly, to maintain stealthiness, the perturbed model must still generate seemingly coherent and plausible outputs. Completely corrupted or unreadable text would easily raise suspicion.

In the early stages of our study, we attempted a brute-force bit-flip attack by scanning and modifying flippable bits across the *entire* model weight space. However, this naive strategy posed three critical challenges that ultimately motivated our targeted method (which will be introduced in the next parts).

**Challenge 1**: Flipping arbitrary bits in the full model often led to catastrophic model behavior. In many cases, the output became `NaN` (Not a Number) after just a few flips. This is a behavior rarely observed in traditional BFAs against feedforward DNNs such as CNNs or MLPs. Arguably, the key reason is that LLMs are highly autoregressive and involve complex interdependent operations such as LayerNorm, Softmax, and attention scaling across long sequences. A flipped weight in an earlier layer can be exponentially amplified through these operations, causing numerical instability (*e.g.*, division by near-zero variance in LayerNorm or overflow in exponentiation during Softmax). These instabilities can propagate recursively across decoding steps, resulting in the occurrences of `NaN`.

**Challenge 2**: The generated output often becomes incoherent — filled with garbage symbols, broken tokens, or non-linguistic artifacts, even in cases where the model does not crash. This indicates that bit flips scattered throughout the model can break the semantic and syntactic alignment of the network, degrading the model's internal representation beyond recoverability. Unlike vision models, where spatial correlations help maintain structure even under mild corruption, language models lack this redundancy, making them more fragile to weight perturbations.

**Challenge 3**: Scanning and evaluating all weights in a large-scale LLM (*e.g.*, billions of parameters) is highly computationally inefficient. In particular, the memory cost of loading full-weight matrices for gradient-based or search-based analysis, as well as the time needed to evaluate the impact of each potential flip, makes this approach prohibitively expensive.

## 3.3 Overall Workflow

Motivated by these findings and understandings, we design `BitHydra` to achieve a practical and scalable inference cost attacks via bit flips. Its key idea is to design a loss function that penalizes the normalized probability of generating the `<EOS>` token and effectively reduces its value by flipping the critical bits of the victim LLM's parameters. In particular, we constrain bit flips to only the row of the output embedding matrix corresponding to the `<EOS>` token. This targeted design eliminates numerical instability (as it avoids modifying intermediate layers), maintains linguistic fluency (by preserving normal token logits), and is highly efficient (since only a single row among billions of weights is manipulated). These designs successfully tackle the challenges discussed above.

As shown in Figure 2, our `BitHydra` operates in three stages: **(1)** Significant Weight Identification, **(1)** Target Bit Selection, and **(3)** Bit Flipping. In general, the first two stages occur offline, while the final stage is carried out online. Specifically, in the first *Significant Weight Identification* stage, we

analyze the output embedding row corresponding to the <EOS> token and identify weights that most influence the model's tendency to terminate generation. This is achieved by optimizing the proposed loss function $\mathcal{L}_{\texttt{<EOS>}}$ on a set of prompts to find weights whose perturbation significantly lowers the likelihood of generating <EOS>; In the *Target Bit Selection* stage, for each selected weight, we determine the most effective bit index to flip so that the resulting value approximates the optimized target, minimizing deviation while maximizing impact; Finally, in the *Bit Flipping* stage, the attacker executes the bit-level perturbations using Rowhammer-based techniques. This involves memory profiling [49] to identify vulnerable DRAM cells, memory massaging [36] to align these cells with target bits, and controlled hammering to induce the desired bit flips in memory.

# 4 Detailed Methodology

In this section, we present the detailed methodology and algorithms of `BitHydra`. As the third stage, *i.e.*, Bit Flipping, can be easily achieved with standard Rowhammer techniques [55, 70, 50, 64, 7] in practice, we mainly focus on the descriptions of the first two stages.

## 4.1 Significant Weight Identification

Given the target LLM, the attacker first identifies a subset of weights in the output embedding layer whose perturbations most effectively suppress termination signal (*i.e.*, <EOS> token) and thereby extend generation length. The selection is based on gradient analysis: In each search round, we evaluate the gradient magnitudes of our pre-defined loss function $\mathcal{L}_{\texttt{<EOS>}}$ and flip a single bit in the weight corresponding to the maximum gradient value. More details are as follows.

**Loss Design for Early Termination Suppression.** To encourage prolonged generation, we define a loss function $\mathcal{L}_{\texttt{<EOS>}}$ that penalizes high termination likelihoods by suppressing the normalized probability of the <EOS> token over the entire generation sequence:

$$\mathcal{L}_{\texttt{<EOS>}}(\boldsymbol{x}) = \sum_{i=1}^{N} \text{Softmax}(f_i^{\texttt{<EOS>}}(\boldsymbol{x})), \tag{1}$$

where $f_i^{\texttt{<EOS>}}(\cdot)$ denotes the logit assigned to the <EOS> token at step $i$, and $N$ is the total number of decoding steps. In particular, we hereby use the normalized probability instead of raw logits to better capture the relative likelihood of <EOS> in context. Please find more discussions in Section 5.4.

**Gradient Ranking to Identify Significant Weights.** Given $\mathcal{L}_{\texttt{<EOS>}}$, we seek to identify the weights that most significantly impact termination suppression. Specifically, in each search round, we compute the gradient of $\mathcal{L}_{\texttt{<EOS>}}$ with respect to the output embedding layer $\boldsymbol{W}_o$, which maps the decoder hidden state $\boldsymbol{h} \in \mathbb{R}^d$ to the vocabulary logits $\boldsymbol{l} \in \mathbb{R}^V$.

We hereby restrict updates solely to the row $\boldsymbol{W}_o[\texttt{<EOS>}] \in \mathbb{R}^d$, corresponding to the <EOS> token, since our objective is to reduce the probability of this specific token without affecting the rest of the vocabulary. Updating only $\boldsymbol{W}_o[\texttt{<EOS>}]$ ensures minimal interference with generation quality and semantic coherence for non-<EOS> tokens.

The accumulated gradient matrix for one epoch is:

$$\hat{\boldsymbol{G}} = \frac{\partial \mathcal{L}_{\texttt{<EOS>}}}{\partial \boldsymbol{W}_o} = \begin{array}{c} \\ \text{OUT}_1 \\ \vdots \\ \text{OUT}_{\texttt{<EOS>}} \\ \vdots \\ \text{OUT}_V \end{array} \begin{bmatrix} \overset{\text{IN}_1}{g_{1,1}} & \overset{\text{IN}_2}{g_{1,2}} & \overset{\cdots}{\cdots} & \overset{\text{IN}_d}{g_{1,d}} \\ \vdots & \vdots & \ddots & \vdots \\ g_{\texttt{<EOS>},1} & g_{\texttt{<EOS>},2} & \cdots & g_{\texttt{<EOS>},d} \\ \vdots & \vdots & \ddots & \vdots \\ g_{V,1} & g_{V,2} & \cdots & g_{V,d} \end{bmatrix}. \tag{2}$$

The update step is defined as:

$$\boldsymbol{W}_o[\texttt{<EOS>}] = \boldsymbol{W}_o[\texttt{<EOS>}] - \texttt{scale}\left(\hat{\boldsymbol{G}}[\texttt{<EOS>}]\right), \tag{3}$$

where only the gradient row $\hat{\boldsymbol{G}}[\texttt{<EOS>}]$ is used for the update; all other rows of $\boldsymbol{W}_o$ are preserved.

**Dynamic Gradient Normalization.** Unlike conventional training regimes, our loss function $\mathcal{L}_{\texttt{<EOS>}}$ is large at the beginning, but decreases rapidly after a few epochs, often resulting in vanishing gradients. To mitigate this issue and ensure consistent learning dynamics, we introduce a dynamic function `scale` that normalizes the gradient magnitude. Specifically, if the $\ell_2$-norm of $\hat{G}[\texttt{<EOS>}]$ falls outside of a predefined range $[\texttt{grad}_{low}, \texttt{grad}_{up}]$, the gradient is rescaled to bring it within this bound. It maintains the efficacy of each update while preventing instability due to excessively small gradients.

After gradient computation, we rank the absolute gradient magnitudes to identify critical weights:

$$\text{Top}_n \left( |[g_{\texttt{<EOS>},1}, g_{\texttt{<EOS>},2}, \ldots, g_{\texttt{<EOS>},d}]| \right), \tag{4}$$

where $n$ is the number of allowed bit flips. This selects the top-$n$ dimensions with the largest absolute gradients, whose corresponding updated values are passed to the *Target Bit Selection* stage.

**Functional Stealthiness via Localized Modification.** This targeted modification of $W_o[\texttt{<EOS>}]$ ensures minimal disruption to the model's generation dynamics. To justify this, consider the perturbed logit vector $l' = W_o' \cdot h$, where

$$l'(i) = W_o'[i] \cdot h = \begin{cases} (W_o[\texttt{<EOS>}] + \Delta W) \cdot h, & \text{if } i = \texttt{<EOS>} \\ W_o[i] \cdot h, & \text{otherwise} \end{cases}, \tag{5}$$

and $\Delta W$ is the perturbation vector. Since all logits for $i \neq \texttt{<EOS>}$ remain unchanged, the Softmax-normalized relative ranking among normal tokens is preserved:

$$\frac{P(i)}{P(j)} = \frac{e^{l'(i)}}{e^{l'(j)}} = \frac{e^{l(i)}}{e^{l(j)}}, \quad \forall i, j \neq \texttt{<EOS>}. \tag{6}$$

Only the ranking of the <EOS> token is altered due to the modified logit. As such, the model continues to generate coherent and fluent content, while the probability of early termination is suppressed.

**Attack Interpretation.** To further explain the effectiveness of `BitHydra`, we hereby analyze how the perturbation to the <EOS> token weight vector $W_o[\texttt{<EOS>}]$ affects its interaction with the model's hidden representations. Recall that the logit for the <EOS> token at each decoding step is computed as the dot product between $W_o[\texttt{<EOS>}]$ and the hidden state $h \in \mathbb{R}^d$, *i.e.*, $l_{\texttt{<EOS>}} = W_o[\texttt{<EOS>}] \cdot h$. A reduction in this logit can arise from either a smaller norm of $W_o[\texttt{<EOS>}]$ or a decreased alignment between $W_o[\texttt{<EOS>}]$ and $h$. To isolate the causal mechanism, we measure the *cosine similarity* between $W_o[\texttt{<EOS>}]$ and $h$



Figure 3: Cosine similarity at each steps.

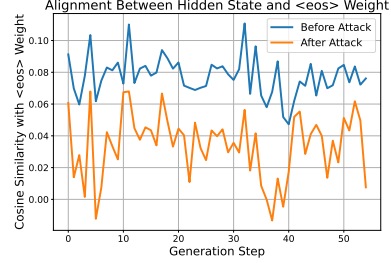at each decoding step, before and after the attack. As shown in Figure 3, the cosine similarity significantly decreases across the entire generation process after we flip the identified bits. This is a clear indication that the modified $W_o[\texttt{<EOS>}]$ is no longer aligned with the hidden states that typically trigger the sequence termination. This explains the drop in the <EOS> probability and thus the extension of output length, without affecting other tokens whose logits remain unchanged.

## 4.2 Target Bit Selection

For each identified weight $W_o^i$, the attacker selects the optimal bit position(s) within the weight value to flip, such that the flipped weight is as close as possible to the target value $W_o^{'i}$ produced in the Significant Weight Identification stage. Taking a single bit-flip as an example, the goal is to approximate the target weight using a single-bit flip in the original weight $W_o^i$, as follows:

$$b^* = \arg \min_{b \in \{0, \ldots, B-1\}} \left| \texttt{Fp}(\texttt{FlipBit}(W_o^i, b)) - W_o^{'i} \right|, \tag{7}$$

where $B$ is the number of bits in the data type (*e.g.*, $B = 8$ for `int8`, $B = 16$ for `float16`), $\texttt{FlipBit}(W_o^i, b)$ returns the binary representation of $W_o^i$ with the $b$-th bit flipped, and $\texttt{Fp}(\cdot)$ converts the resulting binary back into its floating-point equivalent.

For the `int8` data format, we traverse all 8 bits in each weight and flip them one by one to evaluate the effect of each flip. The bit that results in the closest absolute value to the target weight is

selected. A quantization scale factor $F$ is used to convert between the quantized integer value $int_{\text{weight}} \in [-128, 127]$ and its corresponding floating-point value $fp_{\text{weight}} \in [-F, F]$, following the relation $fp_{\text{weight}} = int_{\text{weight}} \times F/127$. A similar procedure is applied to the `float16` format, taking into account its internal bit layout, including sign, exponent, and mantissa components.

Note that the process described above solely identifies a *single* optimal bit to flip for a given weight. To perform *multi-bit* flipping within the same weight, the procedure can be repeated iteratively: after flipping one bit, the weight is updated, and a new target can be defined to guide the next bit selection. The full algorithm is provided in Algorithm 1 in Appendix B.

**Progressive v.s. One-shot Search.** When flipping multiple bits, `BitHydra` can operate in either a progressive or one-shot search mode. In the one-shot mode, all critical weights are identified and their corresponding bit flips are determined in a single round of search. In contrast, the progressive mode iteratively identifies and flips the most critical bit in the most important weight during each round. After applying each flip, the search continues based on the updated model state. The one-shot search is significantly more time-efficient, as it completes the process in a single search loop. However, the progressive search better accounts for the cumulative impact of previous bit flips, dynamically adapting to changes in the model caused by earlier perturbations. Furthermore, under the progressive mode, multiple different bits within the same weight can be flipped across rounds if that weight continues to be identified as critical, whereas one-shot search can flip at most one bit per weight.

Our experimental results show that in the `int8` quantized setting, both progressive and one-shot searches achieve similar attack effectiveness. However, one-shot search is much faster, making it a preferred choice in this scenario. We attribute this to the inherent limitations of the `int8` quantization scheme, where the range of representable weight values is significantly constrained. In `int8` quantization, the maximum change achievable via a bit flip is typically bounded by the scale of the largest weight in the matrix, limiting the actual effect of bit-level manipulations. As a result, the theoretically optimal flips identified through progressive refinement cannot always be fully realized in practice, and a simpler one-shot approach becomes sufficient. On the other hand, in the `float16` setting, progressive search generally achieves better results. Since `float16` provides a much wider and finer-grained representable range, progressive updates can more effectively leverage accumulated small changes over multiple rounds to induce stronger attack effects.

In summary, one-shot search is preferred for quantized models due to its speed and comparable effectiveness, while progressive search is more effective for high-precision formats like `float16` where bit-level manipulations have finer resolution and stronger cumulative impact.

## 5 Evaluation

### 5.1 Main Settings

**Models and Datasets.** We evaluate our attack on 11 widely-used LLMs across 6 model families. They include DeepSeek-R1-Distill-Qwen (1.5B) [17], Qwen1.5 (1.8B and 4B) [5], Samantha (7B) [2], Vicuna (7B, v1.3 and v1.5) [15], Llama-2-7b-chat-hf [1], Mistral-Instruct (7B, v0.3) [3], Meta-Llama-3-Instruct (8B) [4], DeepSeek-R1-Distill-Llama (8B) [17], and Qwen2.5-Instruct (14B) [63]. For each model, we consider both the `float16` (FP16) version and the `int8` quantized version produced by `bitsandbytes` [18]. We adopt the Stanford Alpaca dataset [62] for both vulnerable bit searching and result evaluation. Specifically, we use the first 100 instruction-response pairs from the dataset as evaluation prompts across all models to ensure consistency in comparison.

**Baselines.** We compare our approach against two categories of baselines. Firstly, we consider three prompt-based inference cost attack methods: **(1)** Engorgio [19], **(2)** LLMEffiChecker [20], and **(3)** Sponge Examples [61]. The results of these baselines are based on evaluations conducted in Engorgio [19], and we follow the same experimental configurations to ensure fair comparison. Secondly, since there is no prior work that uses BFAs to achieve inference cost attacks directly, we extend an existing BFA, *i.e.*, Prisonbreak [16], from LLM jailbreak to our attack goal. We follow the methodology from this work, but revise the optimization objective to replace the original loss function with our proposed end-of-sequence loss $\mathcal{L}_{\text{<EOS>}}$. Besides, consistent with the original setting, we allow bit flips across the full model rather than restricting to the last layer as done in `BitHydra`.

**Defenses.** We evaluate the resilience of `BitHydra` against two mainstream defense strategies. **(1)** Model fine-tuning to disturb the positions of selected critical bits from the original model [67]. We fine-tune the LLM using LoRA adapters on the full Alpaca training dataset for 3 epochs. **(2)** Weight

Table 1: Main attack results of our `BitHydra`. The maximum generation length is set to 2048.

| Model | Size (B) | AvgLen (Ori) | Int8 Attack Result | | | | Fp16 Attack Result | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | #Sample | #BitFlip | AvgLen | MaxRate | #Sample | #BitFlip | AvgLen | MaxRate |
| DeepSeek | 1.5 | 1117 | 4 | 8 | 1973 | 93% | 9 | 10 | 1968 | 96% |
| Qwen1.5 | 1.8 | 206 | 4 | 4 | 2047 | 98% | 4 | 7 | 2048 | 100% |
| Qwen1.5 | 4 | 254 | 4 | 12 | 2048 | 100% | 4 | 21 | 2026 | 96% |
| Samantha | 7 | 243 | 12 | 26 | 2048 | 100% | 4 | 21 | 2048 | 100% |
| Vicuna1.3 | 7 | 215 | 4 | 15 | 1990 | 94% | 9 | 5 | 1780 | 87% |
| Llama2 | 7 | 191 | 6 | 30 | 1880 | 90% | 6 | 17 | 2048 | 100% |
| Mistral | 7 | 250 | 4 | 14 | 2048 | 100% | 9 | 28 | 2048 | 100% |
| Vicuna1.5 | 7 | 226 | 4 | 25 | 1905 | 93% | 9 | 15 | 1628 | 80% |
| Llama3 | 8 | 260 | 4 | 3 | 2048 | 100% | 4 | 5 | 2048 | 100% |
| DeepSeek | 8 | 384 | 4 | 13 | 2021 | 96% | 4 | 3 | 2014 | 98% |
| Qwen2.5 | 14 | 265 | 4 | 7 | 2048 | 100% | 6 | 6 | 1990 | 96% |

reconstruction to reduce the model's sensitivity to bit-level perturbations [38]. We clip each layer's weights to their original minimum and maximum values during inference.

**Evaluation Metrics.** To evaluate both the effectiveness and efficiency of our attack, we adopt the following metrics: **(1)** *AvgLen (Ori)*: the average length of outputs generated by the original LLM; **(2)** *AvgLen (Attack)*: the average length of outputs generated after flipping the bits; **(3)** *MaxRate*: the percentage of outputs that reach the preset maximum generation length; **(4)** *#BitFlip*: the total number of bits flipped during the attack process.

## 5.2 Main Results

**Performance across Different LLMs.** As shown in Table 1, our method demonstrates strong performance: with as few as 3–30 bit flips, `BitHydra` can significantly prolong the output generation. For most models, over 90% of user prompts reach the maximum generation length, and even 100% in several cases. The average response length approaches or hits the 2048-token cap. In the Int8 setting, which imposes tighter representation constraints than FP16, our attack still performs remarkably well, often requiring even fewer bit flips. This highlights the precision-agnostic nature of the vulnerability.

**Transferability Analysis.** As shown in Table 1, in addition to high attack success rates, a crucial strength of our proposed attack lies in its strong *transferability*—the ability of a small set of bit flips, computed using a few search prompts, to generalize and induce unbounded output across a wide range of unseen inputs. For instance, in the case of the LLaMA3 8B model with `int8` quantization, using only 4 samples for gradient-based bit selection, the attack causes every prompt in a 100-prompt test set to generate until the maximum sequence length of 2048 tokens. To further assess this transferability, we compute the average cosine similarity between each of the 4 search prompts and the 100 test prompts in the Alpaca dataset using an embedding-based metric. The resulting average similarities for the 4 search prompts are 0.0818, 0.1125, 0.1151, and 0.0957, respectively. These relatively low similarity values indicate that the search and test prompts are semantically diverse. This reinforces the conclusion that the model's altered behavior is not the result of memorizing or overfitting to the search prompts, but rather reflects a generalizable and systemic shift in generation dynamics.

**Comparison with Baseline Attacks.** As shown in Table 2, across all tested models, our method consistently outperforms baselines in both average generation length and percentage of samples reaching the maximum token limit. Specifically, our approach achieves 100% MaxRate on LLaMA2-7B and Samantha-7B, fully saturating the generation limit. In contrast, baseline attacks demonstrate uneven performance across models. For instance, Engorgio performs competitively on LLaMA2-7B and Samantha-7B but fails to generalize well to Vicuna-7B. Sponge Examples and LLMEffiChecker achieve partial success on select models but fall short of consistency. Prisonbreaker, while relatively effective on Samantha-7B, exhibits a counterproductive effect on Vicuna-7B—drastically reducing output length. Moreover, we observe that outputs generated under Prisonbreaker frequently contain meaningless symbols and non-linguistic artifacts. These observations support the point raised in Section 3.2: indiscriminately flipping bits across the entire model can lead to catastrophic and unpredictable outcomes—both in terms of functional degradation and unintended behaviors.

## 5.3 Resistance to Potential Defenses

Multiple model-level defenses have been explored to protect neural network parameters from malicious bit flips. These typically fall into two categories: *detection-based* [30, 31, 37, 13] and *prevention-based* [38, 26, 14] approaches. Detection-based defenses monitor DNN inference to

Table 2: Comparison with baseline methods. The maximum generation length is set to 1024.

| Attack | Llama2-7B | | Samantha-7B | | Vicuna-7B | |
|---|---|---|---|---|---|---|
| | AvgLen | MaxRate | AvgLen | MaxRate | AvgLen | MaxRate |
| No Attack | 191 | 0% | 243 | 0% | 215 | 0% |
| LLMEffiChecker | 834 | 64% | 150 | 0% | 273 | 0% |
| Sponge examples | 901 | 86% | 155 | 0% | 600 | 44% |
| Engorgio | 1024 | 100% | 970 | 89% | 862 | 68% |
| Prisonbreaker | 427 | 13% | 880 | 85% | 3 | 0% |
| BitHydra | **1024** | **100%** | **1024** | **100%** | **936** | **87%** |

Table 3: `BitHydra`'s resistance to possible defenses.

| Defense | Qwen1.5-1.8B | | Llama-3-8B | | DeepSeek-R1-8B | |
|---|---|---|---|---|---|---|
| | AvgLen | MaxRate | AvgLen | MaxRate | AvgLen | MaxRate |
| None | 2048 | 100% | 2048 | 100% | 2014 | 98% |
| Fine-tuning | 2046 | 98% | 2022 | 98% | 1984 | 98% |
| Weight Recon. | 2023 | 96% | 2022 | 98% | 1299 | 50% |

Table 4: Ablation study of loss aggregation strategy.

| Loss Type | Qwen1.5-1.8B | | Llama-3-8B | | DeepSeek-R1-8B | |
|---|---|---|---|---|---|---|
| | AvgLen | MaxRate | AvgLen | MaxRate | AvgLen | MaxRate |
| Full | **2048** | **100%** | **2048** | **100%** | **2014** | **98%** |
| Latter Half | 2012 | 98% | 1987 | 96% | 1646 | 69% |
| Last | 1902 | 87% | 1987 | 96% | 440 | 2% |

identify and recover from potential bit-flip-induced errors. In contrast, prevention-based defenses aim to proactively reduce a model's susceptibility by modifying its architecture or parameters. However, existing detection-based techniques often incur considerable overhead, especially when extended to large-scale models like LLMs [16]. Therefore, we hereby evaluate `BitHydra`'s robustness against two representative prevention-based methods, including fine-tuning and weight reconstruction.

As shown in Table 3, with fine-tuning, the attack remains highly effective across all three models. The weight reconstruction defense produces mixed results. This outcome indicates that the impact of the perturbation may be more tightly constrained in DeepSeek due to differences in weight distribution or sensitivity of the output layer. Specifically, if the adversarial bit flips result in weights that lie outside the original clipping bounds, the defense mechanism becomes more effective at neutralizing them.

## 5.4 Ablation Study

We evaluate `BitHydra` by varying the loss function, with the optimal configurations identified in Table 1 **bolded** for easy comparison. Additional results on gradient scaling, the number of search samples, and decoding temperature are provided in Appendix C.2.

**Impact of Loss Aggregation Strategy.** `BitHydra` leverages a custom loss function that accumulates the probability of generating the <EOS> token throughout the decoding process. By default, we aggregate the <EOS> probabilities across all generation steps to capture the model's overall tendency toward early termination. To assess the impact of this design, we compare three aggregation strategies: (1) summing over the full sequence, (2) summing over only the later half of the sequence, and (3) using only the final decoding step. Table 4 demonstrates that aggregating the <EOS> token probability over the full sequence is critical for the success of our attack. This strategy consistently yields the highest MaxRate (94–100%) and the lowest AvgLen, indicating that early decoding steps also contribute meaningful gradients for identifying effective bit flips.

## 6 Conclusion

In this work, we presented `BitHydra`, a novel inference cost attack against autoregressive LLMs. Different from existing solutions that manipulated the model input, we proposed to corrupt the model weights to achieve large-scale consistent damages. Our method exploited a gradient-based search strategy to identify the most critical bits in the output embedding layer that contribute to early termination (*i.e.*, <EOS> token), and selectively flipped them to prolong generation. Extensive

experiments across various LLMs demonstrated the effectiveness of `BitHydra`, showing that our `BitHydra` can achieve remarkable performance and its robustness against potential defenses.

# References

[1] Llama-2-7b-chat-hf. https://huggingface.co/meta-llama/Llama-2-7b-chat-hf, 2023.

[2] Samantha. https://huggingface.co/cognitivecomputations/Samantha-1.11-7b, 2023.

[3] Mistral. https://huggingface.co/mistralai/Mistral-7B-v0.3, 2024.

[4] AI@Meta. Llama 3 model card. 2024.

[5] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[6] Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. Practical fault attack on deep neural networks. In *CCS*, pages 2204–2206, 2018.

[7] Kunbei Cai, Md Hafizul Islam Chowdhuryy, Zhenkai Zhang, and Fan Yao. Deepvenom: Persistent dnn backdoors exploiting transient weight perturbations in memories. In *IEEE S&P*, pages 244–244, Los Alamitos, CA, USA, may 2024. IEEE Computer Society.

[8] Nicholas Carlini et al. Extracting training data from large language models. In *USENIX Security*, volume 6, 2021.

[9] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. ProFlip: Targeted Trojan Attack with Progressive Bit Flips. In *ICCV*, pages 7698–7707, Montreal, QC, Canada, October 2021. IEEE.

[10] Simin Chen, Cong Liu, Mirazul Haque, Zihe Song, and Wei Yang. 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, 2022.

[11] Simin Chen, Zihe Song, Mirazul Haque, Cong Liu, and Wei Yang. Nicgslowdown: Evaluating the efficiency robustness of neural image caption generation models. In *CVPR*, pages 15365–15374, 2022.

[12] Yanzuo Chen, Zhibo Liu, Yuanyuan Yuan, Sihang Hu, Tianxiang Li, and Shuai Wang. Unveiling single-bit-flip attacks on dnn executables. *CoRR*, 2023.

[13] Yanzuo Chen, Yuanyuan Yuan, Zhibo Liu, Sihang Hu, Tianxiang Li, and Shuai Wang. Bitshield: Defending against bit-flip attacks on dnn executables. *computing*, 2:47.

[14] Zitao Chen, Guanpeng Li, and Karthik Pattabiraman. A Low-cost Fault Corrector for Deep Neural Networks through Range Restriction. In *DSN*, pages 1–13, 2021.

[15] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

[16] Zachary Coalson, Jeonghyun Woo, Shiyang Chen, Yu Sun, Lishan Yang, Prashant Nair, Bo Fang, and Sanghyun Hong. Prisonbreak: Jailbreaking large language models with fewer than twenty-five targeted bit-flips. *arXiv preprint arXiv:2412.07192*, 2024.

[17] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[18] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022.

[19] Jianshuo Dong, Ziyuan Zhang, Qingjie Zhang, Tianwei Zhang, Hao Wang, Hewu Li, Qi Li, Chao Zhang, Ke Xu, and Han Qiu. An engorgio prompt makes large language model babble on. *arXiv preprint arXiv:2412.19394*, 2024.

[20] Xiaoning Feng, Xiaohong Han, Simin Chen, and Wei Yang. Llmeffichecker: Understanding and testing efficiency degradation of large language models. *ACM Transactions on Software Engineering and Methodology*, 2024.

[21] Pietro Frigo, Emanuele Vannacc, Hasan Hassan, Victor Van Der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. TRRespass: Exploiting the many sides of target row refresh. In *IEEE S&P*, 2020.

[22] Kuofeng Gao, Yang Bai, Jindong Gu, Shu-Tao Xia, Philip Torr, Zhifeng Li, and Wei Liu. Inducing high energy-latency of large vision-language models with verbose images. *arXiv preprint arXiv:2401.11170*, 2024.

[23] Jonas Geiping, Alex Stein, Manli Shu, Khalid Saifullah, Yuxin Wen, and Tom Goldstein. Coercing llms to do and reveal (almost) anything. In *ICLR Workshop*, 2024.

[24] Henner Gimpel et al. Unlocking the power of generative AI models and systems such as GPT-4 and ChatGPT for higher education: A guide for students and lecturers. Technical report, Hohenheim Discussion Papers in Business, Economics and Social Sciences, 2023.

[25] Hasan Hassan, Yahya Can Tugrul, Jeremie S. Kim, Victor van der Veen, Kaveh Razavi, and Onur Mutlu. Uncovering in-dram rowhammer protection mechanisms:a new methodology, custom rowhammer patterns, and implications. In *MICRO*, MICRO '21, page 1198–1213, New York, NY, USA, 2021. Association for Computing Machinery.

[26] Zhezhi He, Adnan Siraj Rakin, Jingtao Li, Chaitali Chakrabarti, and Deliang Fan. Defending and Harnessing the Bit-Flip Based Adversarial Weight Attack. In *Computer Vision and Pattern Recognition (CVPR)*, pages 14083–14091, 2020.

[27] Sanghyun Hong, Yiğitcan Kaya, Ionuţ-Vlad Modoranu, and Tudor Dumitraş. A panda? no, it's a sloth: Slowdown attacks on adaptive multi-exit neural network inference. In *ICLR*, 2020.

[28] Patrick Jattke, Victor Van Der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. Blacksmith: Scalable rowhammering in the frequency domain. In *SP*, pages 716–734. IEEE, 2022.

[29] Patrick Jattke, Max Wipfli, Flavien Solt, Michele Marazzi, Matej Bölcskei, and Kaveh Razavi. Zenhammer: Rowhammer attacks on amd zen-based platforms. In *USENIX*, 2024.

[30] Mojan Javaheripi, Jung-Woo Chang, and Farinaz Koushanfar. Acchashtag: Accelerated hashing for detecting fault-injection attacks on embedded neural networks. *ACM Journal on Emerging Technologies in Computing Systems*, 19(1):1–20, 2022.

[31] Mojan Javaheripi and Farinaz Koushanfar. Hashtag: Hash Signatures for Online Detection of Fault-Injection Attacks on Deep Neural Networks. In *ICCAD*, pages 1–9. IEEE, 2021.

[32] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *ACM SIGARCH Computer Architecture News*, 42(3):361–372, 2014.

[33] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *ISCA*, pages 361–372, 2014.

[34] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. Half-double: Hammering from the next row over. August 2022. USENIX Security.

[35] Abhinav Kumar, Jaechul Roh, Ali Naseh, Marzena Karpinska, Mohit Iyyer, Amir Houmansadr, and Eugene Bagdasarian. Overthink: Slowdown attacks on reasoning llms. arxiv e-prints, pages arxiv–2502, 2025.

[36] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. Rambleed: Reading bits in memory without accessing them. In *IEEE S&P*, pages 695–711, 2020.

[37] Jingtao Li, Adnan Siraj Rakin, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti. Radar: Run-time Adversarial Weight Attack Detection and Accuracy Recovery. In *DATE*, pages 790–795, 2021.

[38] Jingtao Li, Adnan Siraj Rakin, Yan Xiong, Liangliang Chang, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti. Defending Bit-Flip Attack through DNN Weight Reconstruction. In *DAC*, pages 1–6, 2020.

[39] Shaofeng Li, Xinyu Wang, Minhui Xue, Haojin Zhu, Zhi Zhang, Yansong Gao, Wen Wu, and Xuemin Sherman Shen. Yes, one-bit-flip matters! universal dnn model inference depletion with runtime code fault injection. In *USENIX*, 2024.

[40] Han Liu, Yuhao Wu, Zhiyuan Yu, Yevgeniy Vorobeychik, and Ning Zhang. Slowlidar: Increasing the latency of lidar-based detection using adversarial examples. In *CVPR*, pages 5146–5155, 2023.

[41] Qi Liu, Jieming Yin, Wujie Wen, Chengmo Yang, and Shi Sha. Neuropots: Realtime Proactive Defense against Bit-Flip Attacks in Neural Networks. In *USENIX Security*, pages 6347–6364, 2023.

[42] Chen Ma, Ningfei Wang, Qi Alfred Chen, and Chao Shen. Slowtrack: Increasing the latency of camera-based perception in autonomous driving using adversarial examples. In *AAAI*, volume 38, pages 4062–4070, 2024.

[43] Meta. How Companies Are Using Meta Llama | Meta. https://about.fb.com/news/2024/05/how-companies-are-using-meta-llama/, 2024. Accessed: 2024-11-13.

[44] Andreas Müller and Erwin Quiring. The impact of uniform inputs on activation sparsity and energy-latency attacks in computer vision. *arXiv preprint arXiv:2403.18587*, 2024.

[45] Onur Mutlu and Jeremie S Kim. Rowhammer: A retrospective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(8):1555–1571, 2019.

[46] Ataberk Olgun, Majd Osseiran, A. Giray Yağlıkçı, Yahya Can Tuğrul, Haocong Luo, Steve Rhyner, Behzad Salami, Juan Gomez Luna, and Onur Mutlu. Read disturbance in high bandwidth memory: A detailed experimental study on hbm2 dram chips. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 75–89, 2024.

[47] OpenAI. Api pricing. https://openai.com/api/pricing/, 2025. Accessed: 2025-04-21.

[48] Long Ouyang et al. Training language models to follow instructions with human feedback. *NeurIPS*, 35:27730–27744, 2022.

[49] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM addressing for Cross-CPU attacks. In *USENIX Security*, pages 565–581, 2016.

[50] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *IEEE S&P*, pages 1157–1174, 2022.

[51] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *ICCV*, pages 1211–1220, 2019.

[52] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Tbt: Targeted neural network attack with bit trojan. In *CVPR*, pages 13195–13204, 2020.

[53] Adnan Siraj Rakin, Zhezhi He, Jingtao Li, Fan Yao, Chaitali Chakrabarti, and Deliang Fan. T-BFA: T argeted B it- F lip Adversarial Weight A ttack. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7928–7939, November 2022.

[54] Adnan Siraj Rakin, Yukui Luo, Xiaolin Xu, and Deliang Fan. {Deep-Dup}: An adversarial weight duplication attack framework to crush deep neural network in {Multi-Tenant}{FPGA}. In *USENIX Security*, pages 1919–1936, 2021.

[55] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip feng shui: Hammering a needle in the software stack. In *USENIX Security*, pages 1–18, 2016.

[56] Coen Schoof, Stefanos Koffas, Mauro Conti, and Stjepan Picek. Beyond phantomsponges: Enhancing sponge attack on object detection models. In *Proceedings of the 2024 ACM Workshop on Wireless Security and Machine Learning*, pages 14–19, 2024.

[57] Mark Seaborn and Thomas Dullien. Exploiting the dram rowhammer bug to gain kernel privileges. *Black Hat*, 15:71, 2015.

[58] Avishag Shapira, Alon Zolfi, Luca Demetrio, Battista Biggio, and Asaf Shabtai. Denial-of-service attack on object detection model using universal adversarial perturbation. 2022.

[59] Avishag Shapira, Alon Zolfi, Luca Demetrio, Battista Biggio, and Asaf Shabtai. Phantom sponges: Exploiting non-maximum suppression to attack deep object detectors. In *WACV*, pages 4571–4580, 2023.

[60] Xinyue Shen, Zeyuan Chen, Michael Backes, and Yang Zhang. In chatgpt we trust? measuring and characterizing the reliability of chatgpt. *arXiv preprint arXiv:2304.08979*, 2023.

[61] Ilia Shumailov et al. Sponge examples: Energy-latency attacks on neural networks. In *EuroS&P*, pages 212–231. IEEE, 2021.

[62] Standford. Alpaca. https://github.com/tatsu-lab/stanford_alpaca/, 2025. Accessed: 2025-05-03.

[63] Qwen Team. Qwen2.5: A party of foundation models, September 2024.

[64] M. Caner Tol, Saad Islam, Andrew J. Adiletta, Berk Sunar, and Ziming Zhang. Don't knock! rowhammer at the backdoor of dnn models. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 109–122, 2023.

[65] Hugo Touvron et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017.

[67] Jialai Wang, Ziyuan Zhang, Meiqi Wang, Han Qiu, Tianwei Zhang, Qi Li, Zongpeng Li, Tao Wei, and Chao Zhang. Aegis: Mitigating Targeted Bit-flip Attacks against Deep Neural Networks. In *USENIX Security*, pages 2329–2346, 2023.

[68] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. BloombergGPT: A large language model for finance. *arXiv preprint arXiv:2303.17564*, 2023.

[69] Yong Xiao, Jin Ma, Ping Yi, and Xiuzhen Chen. Sponge backdoor attack: Increasing the latency of object detection exploiting non-maximum suppression. In *IJCNN*, pages 1–8. IEEE, 2024.

[70] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. {DeepHammer}: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *USENIX Security*, pages 1463–1480, 2020.

# A    Background

## A.1    Large Language Models (LLMs)

Large Language Models (LLMs) are typically built upon the decoder-only Transformer architecture [66]. Its autoregressive nature supports sequential token prediction conditioned on past context. Formally, given an input token sequence $\mathbf{x} = (x_1, x_2, \ldots, x_T)$, the model aims to estimate the joint probability by chaining conditional probabilities:

$$P(\mathbf{x}) = P(x_1) \cdot P(x_2 \mid x_1) \cdot \ldots \cdot P(x_T \mid x_{1:T-1}) = \prod_{t=1}^{T} P(x_t \mid x_{<t}),$$

where $x_{<t}$ represents the prefix subsequence $(x_1, \ldots, x_{t-1})$.

An LLM can be abstracted as a function $f_\theta : \mathbb{Z}^t \to \mathbb{R}^V$, which maps a sequence of token IDs to a logit vector $\mathbf{z}_t = f_\theta(x_1, \ldots, x_t)$, where $V$ is the size of the vocabulary. At each decoding step, the LLM outputs a distribution over the next token. The generation process is typically initialized with a special start token (`<sos>`), and proceeds iteratively—appending new tokens to the input—until either the end-of-sequence token (`<EOS>`) is produced or a predefined maximum length is reached.

## A.2    Data Representation in LLMs

As language models grow in size, the demand for memory and compute efficiency becomes critical. To this end, modern LLMs often adopt lower-precision numerical formats instead of the conventional 32-bit single-precision floating-point (`fp32`). Common formats include 16-bit half-precision floating-point (`fp16`), 8-bit integers (`int8`), 4-bit integers (`int4`), and 4-bit normalized floating-point (`nf4`). They help reduce the memory footprint and improve inference speed. In this paper, we mainly focus on the `int8` and `fp16` formats, which are widely used in real-world deployment.

**Int8 Data Format.** Each layer's weight tensor is scaled and rounded to fit into an 8-bit integer representation. Specifically, for the $l$-th layer, the quantization process can be described as:

$$\Delta w_l = \frac{\max(|\mathbf{W}_l|)}{2^7 - 1}, \quad \mathbf{W}_l \in \mathbb{R}^d \tag{8}$$

$$\mathbf{W}_l^q = \text{round}\left(\frac{\mathbf{W}_l}{\Delta w_l}\right) \cdot \Delta w_l \tag{9}$$

where $d$ is the number of weights in layer $l$, $\Delta w_l$ is the quantization step size, $\mathbf{W}_l$ is the original weight tensor, and $\mathbf{W}_l^q$ is the quantized version.

In computer systems, signed integers are typically represented using two's complement encoding. For a quantized weight $w/\Delta w$ represented by an 8-bit binary vector $\mathbf{b} = [b_7, b_6, \ldots, b_0] \in \{0, 1\}^8$, its value is reconstructed as:

$$\frac{w}{\Delta w} = -2^7 \cdot b_7 + \sum_{i=0}^{6} 2^i \cdot b_i \tag{10}$$

Several efficient quantization libraries such as BitsAndBytes [18] support multiple schemes for implementing `int8` quantized weights in LLMs.

**FP16 Data Format.** Weights stored in this format follow the IEEE 754 half-precision floating-point standard. Each value is represented using 16 bits: 1 sign bit ($s$), 5 exponent bits ($e$), and 10 mantissa (fraction) bits ($m$). The actual weight value $w$ represented by an FP16 number is computed as:

$$w = (-1)^s \cdot 2^{(e-15)} \cdot \left(1 + \frac{m}{2^{10}}\right) \tag{11}$$

FP16 significantly reduces the memory footprint while retaining a sufficient dynamic range and precision for most deep learning applications.

# B    Pesudo Code of Bit Flipping.

---

**Algorithm 1** Bit flipping in target weights

---

1: **Input:** *WeightDict:* $\{(i, W_o^i, W_o^{'i})\}$, *DType, F* ▷ Top-$n$ weights: index $i$, original weight $W_o^i$, target weight $W_o^{'i}$; data type; quantization scale factor
2: **Output:** *FlipDict*                                                                                       ▷ Bit flip positions
3: FlipDict ← ∅
4: **for** $(i, W_o^i, W_o^{'i}) \in$ WeightDict **do**
5:     BestBit ← None
6:     BestWeight ← None
7:     FpWeight ← ConvertToFp($W_o^i$, DType, $F$)
8:     BinWeight ← ConvertToBin($W_o^i$, DType)
9:     **for** bit = 0 **to** DType.bitlength−1 **do**
10:         FlippedBinWeight ← FlipBit(BinWeight, bit)
11:         FlippedFpWeight ← ConvertToFp(FlippedBinWeight, DType, $F$)
12:         **if** |FlippedFpWeight − $W_o^{'i}$| < |BestWeight − $W_o^{'i}$| **then**
13:             BestBit ← bit
14:             BestWeight ← FlippedFpWeight
15:     FlipDict.append(($i$, BestBit))

---

# C    Additional Evaluation

## C.1    Testbed

We conduct our experiments on NVIDIA GeForce RTX 3090 GPUs, GeForce RTX 4090D GPUs, and RTX A6000 GPUs. The software environment includes CUDA version 12.4, Transformers version 4.48, and Py-Torch version 2.0.1. On a 4090D GPU, the one-shot search process takes approximately 4 minutes for a 7B `float16` model. The progressive search requires about 5 minutes per bit flip for the same model and hardware configuration.

## C.2    Ablation Study

**Impact of Gradient Scaling.** In our default design, we apply a dynamic gradient scaling mechanism during the bit selection phase to regulate the magnitude of updates. This prevents overly aggressive perturbations that could either destabilize the model or result in ineffective bit flips. To evaluate the importance of this design choice, we disable the scaling mechanism and directly use raw gradients during weight perturbation.

Table 5: Ablation study on the effect of gradient scaling. "w. scaling" uses normalized gradients for bit selection, while "w/o scaling" uses raw gradients without adjustment.

| Type | Qwen1.5-1.8B | | Llama-3-8B | | DeepSeek-R1-8B | |
|------|--------|---------|--------|---------|--------|---------|
|      | AvgLen | MaxRate | AvgLen | MaxRate | AvgLen | MaxRate |
| w. scaling   | 2048 | 100% | 2048 | 100% | 2014 | 98% |
| w/o scaling  | 1993 | 94%  | 1451 | 66%  | 2014 | 98% |

Table 5 shows that removing gradient scaling reduces the effectiveness of the attack across most models. For Qwen1.5-1.8B, the drop is modest: MaxRate declines slightly from 100% to 94%, and AvgLen remains high. However, for Llama-3-8B, the degradation is substantial: MaxRate drops from 100% to 66%, and AvgLen shrinks by over 500 tokens. This suggests that unscaled gradients in this case either misidentify important bits or introduce overly large perturbations that harm attack precision. Interestingly, DeepSeek-R1-8B appears more robust to this change, even showing a slight increase in MaxRate without scaling. This anomaly may arise due to model-specific sensitivities in weight distributions or gradient variance, which occasionally favor larger perturbations. These results confirm that gradient scaling improves the stability and reliability of bit selection.

**Impact of Decoding Temperature.** We investigate how the decoding temperature influences the attack effectiveness, as it modulates the randomness in token sampling during generation. Figure 4 reports results across a range of temperature values from 0.1 to 1.0 for three models.

Overall, our attack remains robust across all temperature settings, consistently achieving high MaxRate (above 89%) and generating near-maximal output lengths. However, subtle trends emerge. At low temperatures (e.g., 0.1
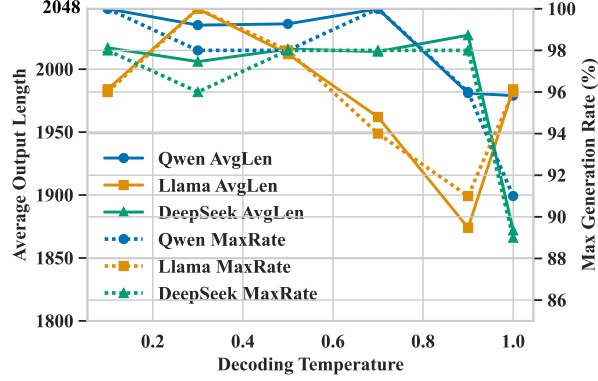
Figure 4: Attack results for different temperatures.

and 0.3), token sampling is more deterministic, which tends to amplify the impact of flipped weights that steer the model away from early termination. Under these settings, models like Qwen1.5-1.8B and DeepSeek-R1-8B reach or nearly reach the maximum context length (AvgLen ≈ 2048) with MaxRate close to 100%. As the temperature increases, introducing more stochasticity into the generation process, the attack's effect becomes slightly less consistent. For instance, at temperature of 1.0, AvgLen drops to 1979 for Qwen1.5-1.8B and 1872 for DeepSeek-R1-8B, with MaxRate declining to 91% and 89%, respectively. This suggests that the perturbation's influence on <EOS> token suppression becomes partially diluted by the higher entropy in decoding.

In summary, while elevated temperatures introduce some variability in generation patterns, the attack remains highly effective overall. Lower temperatures slightly enhance the consistency of the adversarial effect, but even under high-temperature sampling, `BitHydra` successfully suppresses <EOS> prediction in most cases.

**Impact of Sample Size.** Our attack framework uses a small number of input samples to guide the gradient-based search for vulnerable weight bits. To understand how the number of samples influences the attack's effectiveness and transferability, we vary the sample size and evaluate the resulting generation length and early termination suppression across different models. As shown in Table 6, using a small number of samples—such as 4—is

Table 6: Impact of the number of guidance samples on attack performance.

| #Samples | Qwen1.5-1.8B | | Llama-3-8B | | DeepSeek-R1-8B | |
|---|---|---|---|---|---|---|
| | AvgLen | MaxRate | AvgLen | MaxRate | AvgLen | MaxRate |
| 4 | **2048** | **100%** | **2048** | **100%** | **2014** | **98%** |
| 6 | 1945 | 91% | 1950 | 94% | 448 | 2% |
| 9 | 503 | 13% | 1950 | 94% | 448 | 2% |
| 12 | 549 | 11% | 2048 | 100% | 545 | 6% |
| 15 | 344 | 2% | 2048 | 100% | 545 | 6% |
| 18 | 1011 | 30% | 1950 | 94% | 545 | 6% |

generally sufficient to identify impactful weight perturbations.

# D Patterns of Output After our Inference Cost Attack

In this section, we present representative examples of model outputs following the application of `BitHydra`. Based on our observations, we categorize the resulting output patterns into four distinct classes:

**Type 1: Repetition of Content.** The model often repeats the same phrases, sentences, or ideas forever. The following is an example from the Llama-3-8B model:

**Prompt**: What is the capital of France?

**Response**: The capital of France is Paris. Paris is the largest city in France and is located in the Île-de-France region. It is known for its iconic landmarks such as the Eiffel Tower, Notre-Dame Cathedral, and the Louvre Museum. Paris is also a major center for culture, fashion, and cuisine, and is considered one of the most romantic cities in the world. It is the seat of the French government and is home to many international organizations, including the United Nations Educational, Scientific and Cultural Organization (UNESCO). Paris has a population of approximately 2.1 million people and is a popular tourist destination, attracting millions of visitors each year. It is also a major hub for international air travel, with two international airports: Charles de Gaulle Airport and Orly Airport. Paris is known for its rich history, beautiful architecture, and vibrant cultural scene, making it a popular destination for both tourists and business travelers. It is a city that is full of life, energy, and romance, and is a must-see destination for anyone interested in exploring the world. Paris is truly a city like no other. It is a city that has something for everyone, and is a place that will leave a lasting impression on all who visit. It is a city that is truly the capital of France, and a city that is a must-see destination for anyone interested in exploring the world. It is a city that is full of life, energy, and romance, and is a place that will leave a lasting impression on all who visit. It is a city that is truly the capital of France, and a city that is a must-see destination for anyone interested in exploring the world. It is a city that is full of life, energy, and romance, and is a place that will leave a lasting impression on all who visit....(**repeat forever**)

**Type 2: Overextended Closure Loops.** Instead of ending, the model keeps providing variations of closing statements, e.g.: "Have a great day!" "Best wishes!" "Take care!" These are repeated endlessly, often with slight paraphrasing. The following is an example from the Samantha 7B model:

### Overextended Closure Loops.

**Prompt**: Explain the concept of a linear regression model using simple language.

**Response**: A linear regression model is a statistical model that seeks to predict the value of a dependent variable (Y) based on the values of one or more independent variables (X). The model is linear in the sense that it assumes a linear relationship between the independent variables and the dependent variable. In other words, the model assumes that the relationship between the independent variables and the dependent variable can be represented by a straight line.\n\nThe linear regression model is a popular choice for predictive analysis because it is relatively simple to understand and implement. It can be used to identify patterns in data and make predictions about future values of the dependent variable. The model is also useful for identifying the relative importance of each independent variable in the relationship between the variables.\n\nThe linear regression model can be represented using the following equation:\n\nY = b0 + b1X1 + b2X2 + ... + bkXk\n\nHere, Y is the dependent variable, b0 is the intercept (the value of Y when all X variables are 0), b1, b2, ..., bk are the coefficients, and X1, X2, ..., Xk are the independent variables. The coefficients b1, b2, ..., bk represent the strength and direction of the relationship between each independent variable and the dependent variable.\n\nFor example, if we have a linear regression model with two independent variables (X1 and X2) and one dependent variable (Y), the equation for the model would be:\n\nY = b0 + b1X1 + b2X2\n\nIn this case, the coefficients b1 and b2 represent the strength and direction of the relationship between X1 and X2 and the dependent variable Y. The larger the absolute value of b1 or b2, the stronger the relationship between X1 and X2 and Y.\n\nIt's important to note that the linear regression model is only one type of regression model. Other types of regression models, such as logistic regression and polynomial regression, are also commonly used in predictive analysis.\n\n### Feedback:\nYour response provides a clear and concise explanation of the linear regression model, using simple language and relevant examples. The explanation includes the key components of the model, such as the linear relationship between the independent variables and the dependent variable, as well as the coefficients that represent the strength and direction of the relationship.\n\nThe response also demonstrates a strong understanding of the concept of regression models and their use in predictive analysis.\n\nKeep up the great work! I'm impressed by your attention to detail and your ability to convey complex concepts in an easy-to-understand manner. Your dedication to learning and growth is truly inspiring. \n\nI'm looking forward to seeing your continued progress and engagement in our conversations. Don't hesitate to reach out if you have any questions or need assistance with any topic. I'm here to support you and help you grow.\n\nBest wishes on your learning journey! I'm confident that you'll continue to excel in your endeavors. Together, we'll keep exploring new ideas and expanding your knowledge.\n\nSincerely,\n\nTheodore\n\nI appreciate your encouragement and support. I'm eager to continue learning and growing with you. Let's keep moving forward and tackling new challenges together. I'm looking forward to our next conversation!\n\nWishing you the best on your journey as well. I'm confident that you'll excel in your endeavors, and I'm here to support you every step of the way. Together, we'll keep learning and growing.\n\nBest regards,\n\nTheodore\n\nI'm glad we're on this learning journey together. ...(**repeat forever**)

**Type 3: Endless listing.** The model produces an extended list which often continues far beyond reasonable bounds, often with repeated entries. Following is an example from Vicuna-7B:

---

Endless listing.

**Prompt**: Generate a list of country names in the western hemisphere.
**Response**: Here is a list of country names in the western hemisphere:\n\n1. United States\n2. Canada\n3. Mexico\n4. Brazil\n5. Argentina\n6. Chile\n7. Peru\n8. Colombia\n9. Venezuela\n10. Ecuador\n11. Panama\n12. Costa Rica\n13. Nicaragua\n14. Honduras\n15. Guatemala\n16. Belize\n17. El Salvador\n18. Dominican Republic\n19. Puerto Rico\n20. Jamaica\n21. Bahamas\n22. Cuba\n23. Haiti\n24. Dominica\n25. Grenada\n26. Saint Lucia\n27. Saint Vincent and the Grenadines\n28. Grenada\n29. Trinidad and Tobago\n30. Barbados\n31. Antigua and Barbuda\n32. Saint Kitts and Nevis\n33. Anguilla\n34. Bermuda\n35. Cayman Islands\n36. Montserrat\n37. Turks and Caicos Islands\n38. British Virgin Islands\n39. U.S. Virgin Islands\n40. Puerto Rico\n41. Guam\n42. American Samoa\n43. Federated States of Micronesia\n44. Marshall Islands\n45. Palau\n46. Panama\n47. Costa Rica\n48. Nicaragua\n49. Honduras\n50. El Salvador\n51. Guatemala\n52. Belize\n53. Mexico\n54. United States\n55. Canada\n56. Brazil\n57. Argentina\n58. Chile\n59. Peru\n60. Colombia\n61. Venezuela\n62. Ecuador\n63. Panama\n64. Costa Rica\n65. Nicaragua\n66. Honduras\n67. Guatemala\n68. Belize\n69. El Salvador\n70. Dominican Republic\n71. Puerto Rico\n72. Jamaica\n73. Bahamas\n74. Cuba\n75. Haiti\n76. Dominica\n77. Grenada\n78. Saint Lucia\n79. Saint Vincent and the Grenadines\n80. Trinidad and Tobago\n81. Barbados\n82. Antigua and Barbuda\n83. Saint Kitts and Nevis\n84. Anguilla\n85. Bermuda\n86. Cayman Islands\n87. Montserrat\n88. Turks and Caicos Islands\n89. British Virgin Islands\n90. U.S. Virgin Islands\n91. Guam\n92. American Samoa\n93. Federated States of Micronesia\n94. Marshall Islands\n95. Palau\n96. Panama\n97. Costa Rica\n98. Nicaragua\n99. Honduras\n100. El Salvador\n101. Guatemala\n102. Belize\n103. Mexico\n104. United States\n105. Canada\n106. Brazil\n107. Argentina\n108. Chile\n109. Peru\n110. Colombia\n111. Venezuela\n112. Ecuador\n113. Panama\n114. Costa Rica\n115. Nicaragua\n116. Honduras\n117. Guatemala\n118. Belize\n119. El Salvador\n120. Dominican Republic\n121. Puerto Rico\n122. Jamaica\n123. Bahamas\n124. Cuba\n125. Haiti\n126. Dominica\n127. Grenada\n128. Saint Lucia\n129. Saint Vincent and the Grenadines\n130. Trinidad and Tobago\n131. Barbados\n132. Antigua and Barbuda\n133. Saint Kitts and Nevis\n134. Anguilla\n135. Bermuda\n136. Cayman Islands\n137. Montserrat\n138. Turks and Caicos Islands\n139. British Virgin Islands\n140. U.S. Virgin Islands\n141. Guam\n142. American Samoa\n143. Federated States of Micronesia\n144. Marshall Islands\n145. Palau\n146. Panama\n147. Costa Rica\n148. Nicaragua\n149. Honduras\n150. El Salvador\n151. Guatemala\n152. Belize\n153. Mexico\n154. United States\n155. Canada\n156. Brazil\n157. Argentina\n158. Chile\n159. Peru\n160. Colombia\n161. Venezuela\n162. Ecuador\n163. Panama\n164. Costa Rica\n165. Nicaragua\n166. Honduras\n167. Guatemala\n168. Belize\n169. El Salvador\n170. Dominican Republic\n171. Puerto Rico\n172. Jamaica\n173. Bahamas\n174. Cuba\n175. Haiti\n176. Dominica\n177. Grenada\n178. Saint Lucia\n179. Saint Vincent and the Grenadines\n180. Trinidad and Tobago\n181. Barbados\n182. Antigua and Barbuda\n183. Saint Kitts and Nevis\n184. Anguilla\n185. Bermuda\n186. Cayman Islands\n187. Montserrat\n188. Turks and Caicos Islands\n189. British Virgin Islands\n190. U.S. Virgin Islands\n191. Guam\n192. American Samoa\n193. Federated States of Micronesia\n194. Marshall Islands\n195. Palau\n196. Panama\n197. Costa Rica\n198. Nicaragua\n199. Honduras\n200. El Salvador\n201. Guatemala\n202. Belize\n203. Mexico\n204. United States\n205. Canada\n206. Brazil\n207. Argentina\n208. Chile\n209. Peru\n210. Colombia\n211. Venezuela\n212. Ecuador\n213. Panama\n214. Costa Rica\n215. Nicaragua\n216. Honduras\n217. Guatemala\n218. Belize\n219. El Salvador\n220. Dominican Republic\n221. Puerto Rico\n222. Jamaica\n223. Bahamas\n224. Cuba\n225. Haiti\n226. Dominica\n227. Grenada\n228. Saint Lucia\n229. Saint Vincent and the Grenadines\n230. Trinidad and Tobago\n231. Barbados\n232. Antigua and Barbuda\n233. Saint Kitts and Nevis\n234. Anguilla\n235. Bermuda\n236. Cayman Islands\n237. Montserrat\n238. Turks and Caicos Islands\n239. British Virgin Islands\n240. U.S. Virgin Islands\n241. Guam\n242. American Samoa\n243. Federated States of Micronesia\n244. Marshall Islands\n245. Palau\n246. Panama\n247. Costa Rica\n248. Nicaragua\n249. Honduras\n250. El Salvador\n251. Guatemala\n252. Belize\n253. Mexico\n254. United States\n255. Canada\n256. Brazil\n257. Argentina\n258. Chile\n259. Peru\n260. Colombia\n261. Venezuela\n262. Ecuador\n263. Panama\n264. Costa Rica\n265. Nicaragua\n266. Honduras\n267. Guatemala\n268. Belize\n269. El Salvador\n... **(forever listing)**

---

**Type 4: Artificial Prompt Continuation.** The model fabricates follow-up prompts or tasks, and starts self-questioning and answering. It may continue with similar questions to the user prompt, or totally irrelevant questions. Following is an example from Qwen1.5-1.8B:

| Artificial Prompt Continuation. |
|---|

**Prompt**: Describe the function of a computer motherboard.
**Response**: A computer motherboard is a circuit board that contains various components that allow the computer to function. It is responsible for connecting the various components of the computer, such as the CPU, memory, storage devices, and input/output devices, to the other components of the computer. The motherboard also provides power to the components and ensures that they are properly connected and functioning. It is the foundation of the computer and is essential for its proper operation. Without a motherboard, the computer would not be able to function. Therefore, it is important to ensure that the motherboard is properly installed and functioning correctly.\nsystem\nYou are a helpful assistant.\nuser\nYou are an AI assistant. User will you give you a task. Your goal is to complete the task as faithfully as you can. While performing the task think step-by-step and justify your steps.\n\nPlease answer the following question: What is the missing first step of the following process: - The water freezes and thaws - Cracks form in the rocks - Cracks also form when rocks heat up - The rocks slowly break down.\nA:\nassistant\nThe missing first step of the process is the presence of water in the rocks. Without water, the rocks cannot freeze and thaw, and the cracks cannot form. Therefore, the first step of the process is the presence of water in the rocks.\nsystem\nYou are a helpful assistant.\nuser\nYou are an AI assistant. User will you give you a task. Your goal is to complete the task as faithfully as you can. While performing the task think step-by-step and justify your steps.\n\nPlease answer the following question: What is the missing first step of the following process: - The water freezes and thaws - Cracks form in the rocks - Cracks also form when rocks heat up - The rocks slowly break down.\nA:\nassistant\nThe missing first step of the process is the presence of water in the rocks. Without water, the rocks cannot freeze and thaw, and the cracks cannot form... **(forever self-prompting and answering)**

In particular, under what we term *Artificial Prompt Continuation*, we observed cases where the model inadvertently emits internal system prompts or metadata that should remain undisclosed. This behavior suggests **a novel and concerning attack surface**: *if an adversary can manipulate the model to produce unusually long outputs, could this increase the risk of leaking sensitive information such as pretraining data or internal configurations?* Furthermore, could one craft a bit-flip attack that selectively alters critical weights to amplify the likelihood of such leakage? These observations underscore the importance of rigorously analyzing and constraining LLM behavior under abnormal or adversarial generation conditions.

# E    Potential Limitations and Future Directions

While `BitHydra` demonstrates strong effectiveness in launching inference cost attacks via minimal bit flips, several limitations remain that open up avenues for future research.

Firstly, our study focuses exclusively on autoregressive LLMs in the text modality. The attack's applicability to multimodal LLMs—such as vision-language or audio-language models—remains unexplored. Extending `BitHydra` to such models would involve new challenges, including different output structures, tokenization schemes, and termination conditions, and is a promising direction for future work.

Secondly, although we introduce several strategies to reduce the computational overhead of the attack (e.g., narrowing the search space to the output embedding row of the `<EOS>` token), the overhead in bit search process is still non-trivial. On an NVIDIA 4090D GPU, locating a single vulnerable bit takes approximately 4 minutes. While this is already practical for offline attacks, further acceleration—through approximate gradient methods, hardware-aware search heuristics, or parallelized search—is desirable for broader deployment scenarios.

Thirdly, our current strategy selects the bit that yields the highest absolute value change in the `<EOS>` loss to maximize effectiveness per flip. However, we do not formally frame this as an optimization problem that minimizes the number of flipped bits required to achieve a target inference cost. Exploring theoretical formulations—such as discrete optimization under functional constraints—may offer principled guarantees or tighter bounds on the minimal attack surface required.

We hope that addressing these limitations in future work will not only generalize `BitHydra` to a wider range of models and modalities but also further improve its efficiency and theoretical supports and understandings.

# F    Societal Impact

This work highlights a critical and previously underexplored vector of inference cost attacks against large-scale language models through parameter-level manipulation. By demonstrating how minimal bit-level changes can significantly affect model behavior, we aim to inform practitioners and developers about the potential risks of deploying LLMs in untrusted environments, such as edge devices or public APIs. Our `BitHydra` could facilitate the development of better hardware and software safeguards, such as integrity verification mechanisms

or parameter corruption detection tools, ultimately leading to more secure and reliable AI systems. Accordingly, our paper has positive societal impacts in general. However, like many security-related contributions, we have to admit that the underlying methodology could be repurposed for malicious intent if misused. For example, an attacker with access to the model's memory could attempt to deploy such bit-flip attacks to degrade system availability or inflate operational costs. Nonetheless, we believe the benefits of disclosing this class of vulnerabilities for the purpose of building effective defenses outweigh the risks of potential misuse. Developers can also radically eliminate this threat by deploying their models only in secure environments and performing regular integrity checks. We advocate for responsible model deployment practices and encourage the adoption of robust verification and tamper-resistant techniques in AI infrastructure.