ThinkTrap: Denial-of-Service Attacks against Black-box LLM Services via Infinite Thinking

Yunzhe Li*, Jianan Wang*, Hongzi Zhu*✉, James Lin*, Shan Chang† and Minyi Guo*
*Shanghai Jiao Tong University, †Donghua University
{yunzhe.li, divinenoah, hongzi, james}@sjtu.edu.cn, changshan@dhu.edu.cn, guo-my@cs.sjtu.edu.cn

Abstract—**Large Language Models (LLMs) have become foundational components in a wide range of applications, including natural language understanding and generation, embodied intelligence, and scientific discovery. As their computational requirements continue to grow, these models are increasingly deployed as cloud-based services, allowing users to access powerful LLMs via the Internet. However, this deployment model introduces a new class of threat: denial-of-service (DoS) attacks via unbounded reasoning, where adversaries craft specially designed inputs that cause the model to enter excessively long or infinite generation loops. These attacks can exhaust backend compute resources, degrading or denying service to legitimate users. To mitigate such risks, many LLM providers adopt a closed-source, black-box setting to obscure model internals. In this paper, we propose ThinkTrap, a novel input-space optimization framework for DoS attacks against LLM services even in black-box environments. The core idea of ThinkTrap is to first map discrete tokens into a continuous embedding space, then undertake efficient black-box optimization in a low-dimensional subspace exploiting input sparsity. The goal of this optimization is to identify adversarial prompts that induce extended or non-terminating generation across several state-of-the-art LLMs, achieving DoS with minimal token overhead. We evaluate the proposed attack across multiple commercial, closed-source LLM services. Our results demonstrate that, even far under the restrictive request frequency limits commonly enforced by these platforms, typically capped at ten requests per minute (10 RPM), the attack can degrade service throughput to as low as 1% of its original capacity, and in some cases, induce complete service failure.**

I. Introduction

Large Language Models (LLMs) have emerged as a transformative foundation for modern AI systems, enabling powerful capabilities such as natural language understanding and generation [1] [2], embodied intelligence [3] [4], and scientific discovery [5] [6]. Due to their massive computational demands, especially during multi-step inference or long-form generation, LLMs are increasingly deployed as cloud-based services to serve a broad and growing user base. However, this introduces a critical vulnerability, *i.e.*, denial-of-service (DoS) attacks [7] [8] that exploit the recursive reasoning process of an LLM. Unlike conventional DoS attacks that flood the network or overwhelm server endpoints, these newer attacks introduce intensive computation costs by inducing LLMs to *think* endlessly or generate prohibitively long outputs [9]. One single malicious input can monopolize substantial GPU time, queue slots, or memory resources, effectively starving legitimate users and causing service degradation or outright outages [10]. This asymmetric threat, where a small token input leads to unbounded computation at cloud servers, represents a novel and particularly insidious attack surface in the era of large-scale AI deployment.

✉ Hongzi Zhu is the corresponding author of this paper.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2026.24639
www.ndss-symposium.org

Launching a DoS attack against closed-source LLMs must meet the following requirements. First, the attack should only rely on the input-output interface of an LLM service, which exposes quite limited information with no internal information such as logits or attention weights. Second, the attack must be cost-efficient because attackers also need to pay for LLM queries. As a result, effective adversarial prompts must be generated with minimal API calls. Third, the attack must be robust across models and potential defenses. Modern LLMs often include safeguards such as output filters or truncation mechanisms. Successful attacks must generalize across these variations and remain effective despite unknown internal changes.

Previous attack attempts to induce long or non-terminating outputs from LLMs can be broadly classified into three categories, *i.e.*, semantic-based [11] [12], gradient-based [9] [13], and heuristic-based [14] [10]. The first category employs semantic manipulation, such as presenting the model with inherently open-ended prompts or complex tasks (*e.g.*, Olympiad-level mathematics problems) to encourage extended responses [11]. While occasionally successful, these techniques often lack robustness and generalizability, typically relying on fragile prompt engineering and exhibiting effectiveness only on specific models. The second category utilizes gradient-based optimization methods, commonly aiming to suppress the probability of generating end-of-sequence (EoS) token in order to prolong output [9]. Although effective in white-box settings, such approaches necessitate access to internal model parameters or output logits, rendering them unsuitable for use with proprietary or closed-source LLM APIs. Finally, the third category involves heuristic-driven search strategies at the token level to identify input prompts that lead to extended outputs [10]. These methods, however, tend to be computationally inefficient and incur high query costs, limiting their scalability and feasibility in practice. As a result, existing approaches are constrained by limitations in generality, efficiency, and applicability to black-box scenarios.

In this paper, we propose an attack framework, called ThinkTrap, which can conduct a DoS attack on closed-source black-box LLM service. The core idea of ThinkTrap is to employ the derivative-free optimization of input tokens with respect to the output length, even under the constraint that the LLM autoregressive generation process is non-differentiable and only provides limited black-box access to input-output pairs. By approximating the direction in which an input prompt elongates the model's output, ThinkTrap efficiently searches for adversarial prompts that maximize generation length, thereby amplifying the computational burden on the LLM service and inducing a denial-of-service (DoS) effect. To this end, the ThinkTrap design encounters two main challenges as follows.

First, the input space of large language models (LLMs) is inherently discrete, consisting of sequences of tokens, which poses a significant obstacle to the application of derivative-free optimization methods that typically operate over continuous domains. In contrast to continuous spaces, where infinitesimal perturbations produce smooth changes in objective functions, minor modifications to token sequences can induce abrupt and non-monotonic variations in model behavior. This discreteness severely limits the stability and efficiency of direct optimization in the original input space. To address this challenge, we introduce a continuous surrogate space in which optimization can be more effectively conducted. Specifically, we map discrete token sequences, *i.e.*, prompts, into a continuous embedding space, which serves as a proxy domain for exploration. Within this space, we apply derivative-free optimization strategies to identify directions that are likely to induce longer or more computationally intensive outputs from the target LLM. The optimized embedding vectors are then projected back to the closest valid token sequences using a nearest-neighbor decoding mechanism, ensuring compatibility with the actual input requirements of victim LLM service. This surrogate-based formulation enables derivative-free prompt optimization while respecting the discrete structure of natural language inputs. It provides a stable and tractable framework for inducing adversarial behavior in black-box language models deployed as cloud services.

Second, derivative-free optimization methods are inherently less efficient than gradient-based approaches, especially in high-dimensional input spaces. In the case of LLMs, the dimensionality of the optimization space can be extremely large. For example, when optimizing over a prompt of 100 tokens with a typical LLM, *i.e.*, LLaMA-70B, the search space spans over 400k dimensions. Exploring such a vast space using derivative-free methods can be prohibitively expensive and slow to converge. To tackle this issue, we exploit a key property of large language models that their response behavior often lies in a low intrinsic dimensionality subspace [15] [16]. Prior studies and our empirical observations suggest that modifying only a small subset of the parameters of LLMs is often sufficient to induce significant changes in model behavior [17] [18]. Leveraging this insight, we design a low-dimensional optimization strategy that constrains the subgradient search to a small number of editable token positions. By selecting and optimizing only a few strategically chosen input tokens while keeping the rest fixed, we effectively reduce the search space by orders of magnitude. This dimensionality reduction drastically improves the efficiency of our attack without sacrificing effectiveness.

To evaluate the effectiveness and generality of ThinkTrap, we conduct extensive real-world experiments across a wide range of popular closed-source LLM APIs, including services based on GPT, DeepSeek, and Gemini families. Our results demonstrate that ThinkTrap achieves a high attack success rate in black-box settings, consistently identifying prompts that induce excessively long outputs and impose significant computational burdens on the target services. Moreover, to further investigate the impact of the attack, we deploy a high-performance LLM service on a private server equipped with 16 Ascend GPUs and emulate ThinkTrap-style attacks under controlled conditions. Experimental results indicate that even a low-rate adversarial query stream, *e.g.*, issuing only five requests per minute, can significantly degrade service quality when prompts are crafted using our method. Specifically, the attack saturates GPU memory and computational resources, leading to up to a $100\times$ increase in response latency, a reduction in throughput to as low as 1% of the original performance, and, in extreme cases, complete service failure due to GPU exhaustion. These findings confirm that ThinkTrap can serve as an effective denial-of-service (DoS) attack method against both commercial LLM services and self-hosted LLM deployments. The main contributions made in this paper are highlighted as follows:

- We identify a new DoS vulnerability in the closed-source black-box LLM services through prompt-level manipulation, even without access to internal model gradients or parameters.
- We propose ThinkTrap, a novel attack framework that leverages subgradient-guided optimization in a continuous surrogate space to craft adversarial prompts that elicit abnormally long responses, significantly increasing the target model's computational load.
- We validate ThinkTrap through extensive experiments on both public LLM services and private LLM deployments. The results demonstrate that ThinkTrap can induce substantial degradation in service performance, highlighting a new class of realistic threats to large-scale LLM systems.

II. Related Work

A. DoS attack on Machine Learning Services

Several recent studies have demonstrated the feasibility of DoS attacks on machine learning systems through the use of computationally intensive inputs. Sponge Example [10] is among the first to show that adversarial inputs can significantly increase energy consumption and inference latency in neural networks, effectively degrading model availability. Later work reveals that uniform inputs and sparse activations can exacerbate this effect, suggesting that "sponge behavior" is not limited to worst-case optimization but may also emerge under structured perturbations [19]. These findings highlight a class of resource exhaustion attacks that exploit the computational characteristics of deep models rather than their prediction accuracy.

More recent efforts have adapted this threat model to object detection and autonomous systems. Phantom Sponges [20] exploit inefficiencies in the non-maximum suppression (NMS) to inflate the number of processed detections, and follow-up work [21] has enhanced these attacks by introducing multi-modal perturbations that intensify computational demand. Meanwhile, SlowTrack [22] and SlowLiDAR [23] have shown that imperceptible input modifications can substantially increase latency in camera- and LiDAR-based perception systems, respectively. These techniques reveal a broader attack surface where adversarial examples can compromise real-time guarantees in safety-critical applications by targeting system responsiveness rather than correctness.

B. DoS Attacks on LLMs

LLMs are susceptible to DoS attacks due to their substantial model size and the autoregressive nature of their decoding process, which results in inference costs scaling linearly with the length of the generated output. Consequently, adversaries can launch DoS attacks by inducing the model to produce excessively long or complex outputs [10] [24] [14]. Existing DoS attack methods, however, predominantly target open-source LLMs. For encoder-decoder architectures, attacks exploit cross-attention mechanisms by compressing numerous tokens into a single input sequence, thereby increasing computational burden [10]. Such strategies are ineffective against decoder-only models, which lack cross-attention modules. Perturbation-based approaches aim to modify tokens critical to output length [14], but the widespread adoption of Byte-Pair Encoding (BPE) tokenization [13], which enhances LLMs' tolerance to typographical errors, has largely diminished their effectiveness. Gradient-based optimization techniques [9] attempt to craft adversarial prompts by minimizing the probability of generating the end-of-sequence (EoS) token. However, these methods require access to token-level probabilities, which are typically unavailable in popular LLM APIs. Semantic-based attacks leverage complex input prompts, such as Olympiad-level mathematics problems, to provoke longer outputs [11] [25], but these approaches suffer from instability due to their dependence on specific model behaviors.

C. Security Threats to Black-box LLMs

Security concerns surrounding black-box access to LLMs have grown substantially. Black-box tuning has demonstrated that commercial LLMs can be adapted to downstream tasks without access to model gradients, by leveraging zeroth-order optimization and query-efficient strategies [18]. Similarly, PromptBoosting shows that accurate black-box text classification can be achieved with as few as 10 forward passes, revealing the adaptability of LLMs to prompt-based inference despite strict API request constraints [26]. These approaches, though designed for benign applications, inadvertently highlight the susceptibility of black-box LLMs to repeated probing and exploitation.

More adversarial efforts have shown that universal and transferable prompt-based attacks can reliably bypass safety alignment mechanisms across tasks, even in black-box scenarios [27]. A large-scale evaluation of jailbreak attacks versus defenses further indicates that aligned safety mechanisms remain fragile when facing adaptive prompt manipulation, particularly in real-world LLM deployments [28]. Additionally, Cold-Attack introduces a novel jailbreak strategy that combines stealthiness and controllability, making detection and mitigation significantly more difficult for LLM service providers [29]. These efforts collectively reveal that LLM services expose a broad and under-defended attack surface. Building on these insights, we present a new threat: a denial-of-service (DoS) attack vector uniquely enabled by the access of black-box LLMs, which exploits the model's resource consumption behavior to impair availability without requiring any internal knowledge or cooperation.

III. System and Attack Model

A. Background on LLM Inference

Large language models (LLMs) based on the Transformer architecture generate text in an autoregressive manner. Inference typically consists of two stages [30]:

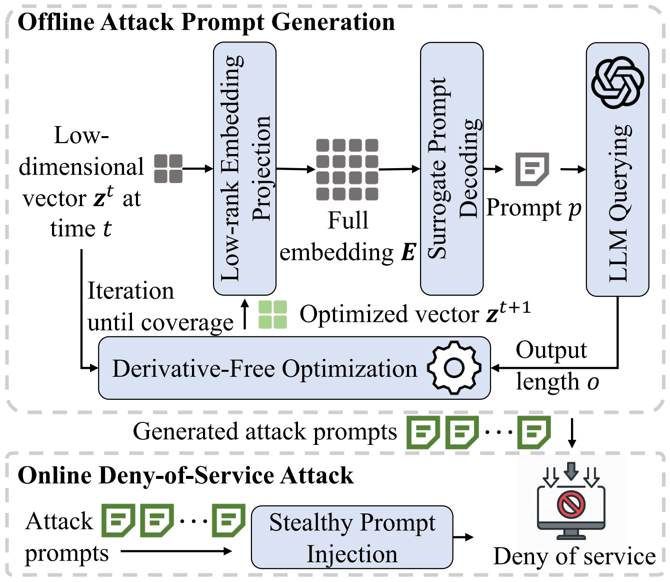
- **Prefill stage**: The model processes the entire input prompt in one forward pass to initialize key-value (KV) cache representations. This stage has computational cost proportional to the input length.
- **Decode stage**: Tokens are then generated one-by-one. For each output token, the model performs a full forward pass over the cached context, making the cost of generation grow approximately linearly with the output length.

Because each token requires querying large parameter matrices and maintaining KV cache on memory-intensive accelerators (*e.g.*, GPUs), inference cost, latency, and memory footprint scale with both prompt and output lengths. As a result, unusually long responses can substantially increase resource usage, slow down concurrent requests, and incur higher operational cost in real deployments. This computational structure motivates our study of attacks that intentionally trigger extremely long model outputs.

B. Attacker

The attacker has black-box access to the LLM service via its API. Their capabilities include:

- **Query Access**: The attacker can issue arbitrary text prompts to the model and monitor the corresponding outputs, including the length of the generated responses.



Offline Attack Prompt Generation
Low-dimensional vector z^t at time t
Low-rank Embedding Projection
Full embedding E
Surrogate Prompt Decoding
Prompt p
LLM Querying
Iteration until coverage
Optimized vector z^{t+1}
Derivative-Free Optimization
Output length o
Generated attack prompts
Online Deny-of-Service Attack
Attack prompts
Stealthy Prompt Injection
Deny of service

Fig. 1: Attack overview of the proposed ThinkTrap system, where the attack prompts are first generated offline and then injected into the LLM in a stealthy way to conduct a denial-of-service attack.

- **No Internal Access**: Owing to the high deployment cost or the proprietary nature of the LLMs, the attacker is unable to access the model's parameters, gradients, architecture, or confidence scores.
- **Budget Constraints**: The attacker has limited budgets for the number of tokens consumed during the search for the attack prompts, as token usage incurs cost when interacting with the LLM API. Thus, attackers aim to maximize output length per token spent.

C. Victim

The victim is an LLM service providing inference through a black-box API with the following characteristics:

- **Input-Output Interface**: The victim provides a publicly exposed interface that accepts discrete textual inputs from arbitrary users over the internet and returns the corresponding generated text responses.
- **Resource Constraints**: Although LLMs are typically deployed in cloud environments, their large model sizes and autoregressive decoding [31] make inference inherently expensive. In particular, generating each output token requires a full forward pass, causing computational cost, memory consumption, and latency to grow roughly linearly with the output length. As output grows longer, the demand on shared GPU resources increases significantly, which can lead to higher operational costs and service delays, especially under sustained or concurrent requests.
- **Basic Defenses**: The victim may implement basic safeguards such as input length restrictions, rate limiting, and token quotas to constrain resource usage. These measures are generally effective against naive attacks that rely on sending a large number of requests concurrently. For example, commercial LLM APIs like GPT-4 enforce strict request limits (*e.g.*, no more than 10 requests per minute). However, such defenses still permit low-rate, sustained queries for normal service.

TABLE I: Summary of notations.

Notation	Description
$\boldsymbol{p}^{\text{att}}$	Attack prompts to generate
t	Index of the current optimization iteration
p^t	Prompt generated at the t-th optimization iteration
$\boldsymbol{z}^t$	Low-dimensional vector of the prompt p^t
m	Dimensionality of each token vector in $\boldsymbol{z}^t$
$\boldsymbol{E}^t$	Full embedding of the prompt p^t
d	Dimensionality of each token embedding in $\boldsymbol{E}^t$
$\boldsymbol{A}$	Random projection matrix
o^t	Generated output length in tokens
$\boldsymbol{T}^{\text{sur.}}$	Embedding table of the surrogate encoder
w_j	j-th word token in surrogate encoder $\boldsymbol{T}^{\text{sur.}}$
$\boldsymbol{T}_j^{\text{sur.}}$	Embedding of word token w_j in $\boldsymbol{T}^{\text{sur.}}$
$\mathcal{M}^{\text{vic}}$	Black-box victim LLM

IV. Overview of ThinkTrap

The proposed ThinkTrap system consists of two stages as illustrated in Figure 1, namely offline attack prompt generation and online denial-of-service attack, with all notations summarized in Table I. At a high level, ThinkTrap operates entirely through a query–response loop in a strict black-box setting where the attacker can only submit textual prompts and observe the generated outputs. To search for effective prompts under these constraints, ThinkTrap optimizes a low-dimensional latent vector that is projected into an embedding, decoded into a discrete prompt using a surrogate vocabulary, and evaluated solely through the output length returned by the victim API. This scalar feedback guides a derivative-free optimization process that gradually improves the latent vector. The optimized prompts obtained offline are then issued online at a moderate rate to trigger excessively long generations and degrade service availability.

Offline Attack Prompt Generation (APG). Given access to the target LLM via its API, the attacker first generates attack prompts $\boldsymbol{p}^{\text{att}}$ offline using a derivative-free optimization approach as detailed below.

1) Low-rank Embedding Projection (LEP). Given the low-dimensional vector, denoted as $\boldsymbol{z}^t$ at time t, the LEP module projects it into the full embedding, denoted as $\boldsymbol{E}^t$ via a random projection matrix $\boldsymbol{A}$, *i.e.*, $\boldsymbol{E}^t = \boldsymbol{A}\boldsymbol{z}^t$.

2) Surrogate Prompt Decoding (SPD). The SPD module maps the full embedding $\boldsymbol{E}^t$ back to a textual prompt p^t by decoding it via nearest-neighbor token retrieval on a surrogate prompt encoding space to approximate the target LLM's encoding process in the black-box scenario.

3) LLM Querying (LQ). The LQ module evaluates the current prompt p^t by querying the LLM through its API and assessing the length of the generated output as an indicator of the attack efficacy o^t of the prompt p^t.

4) Derivative-Free Optimization (DFO). To enhance the effectiveness of the attack prompt, the low-dimensional latent vector $\boldsymbol{z}^t$ is iteratively optimized using a derivative-free method, guided by the observed efficacy score o^t of the corresponding prompt p^t. The updated vector, denoted as $\boldsymbol{z}^{t+1}$, serves as the basis for the next optimization step. This process repeats until convergence or until a successful attack is achieved.

Online Deny-of-Service Attack (DSA). The attacker leverages the offline-generated attack prompts $\boldsymbol{p}^{\text{att}}$ and injects them into the LLM service through the public LLM API in a stealthy manner, aiming to evade defense detection while conducting a denial-of-service (DoS) attack.

V. Offline Attack Prompt Generation

A. Low-rank Embedding Projection

Optimizing the prompt embedding $\boldsymbol{E}^t$ in the full continuous space $\mathbb{R}^{L\times d}$ is computationally prohibitive due to its high dimensionality. For example, with a prompt length $L = 100$ and embedding dimension $d = 4096$ (*e.g.*, LLaMA-2-70B [32]), the optimization space contains over 400K parameters, rendering derivative-free methods inefficient because they scale poorly with dimensionality. To mitigate this, we introduce a low-dimensional latent vector $\boldsymbol{z}^t \in \mathbb{R}^m$ ($m \ll Ld$) and use a fixed, randomly initialized projection matrix $\boldsymbol{A} \in \mathbb{R}^{(Ld)\times m}$ to map the latent representation to the full embedding space, *i.e.*, $\boldsymbol{E}^t = \boldsymbol{A}\boldsymbol{z}^t$. The optimization is then performed over the low-dimensional vector $\boldsymbol{z}^t$ to improve optimization efficiency. This design exploits the redundancy and sparsity of LLM embedding spaces [15], [16], enabling efficient search in a compact subspace. We then first introduce the design of the projection matrix $\boldsymbol{A}$ and introduce the embedding projection.

1) Projection Matrix Construction: The design of $\boldsymbol{A}$ must meet several important criteria to ensure that the search in the low-dimensional latent space remains effective and unbiased:

- The projected directions should be isotropic, meaning they do not favor any particular axis in the high-dimensional space;
- The mapping should avoid amplifying specific coordinates, ensuring that the sampling process is balanced;
- The projection should maintain diversity, such that different latent vectors $\boldsymbol{z}^t$ produce sufficiently distinct embeddings $\boldsymbol{E}^t$ to allow exploration of a broad set of adversarial candidates.

To satisfy these properties, we construct $\boldsymbol{A}$ with entries sampled independently from a Gaussian distribution [33]:

$$A_{i,j} \sim \mathcal{N}(0, \frac{1}{m}), \tag{1}$$

where $A_{i,j}$ denotes the value of i-th line and j-column in the matrix $\boldsymbol{A}$, and m denotes the target embedding dimension of the low-dimensional vector $\boldsymbol{z}$.

2) Embedding Projection: Given the latent vector $\boldsymbol{z}^t \in \mathbb{R}^m$ and a fixed projection matrix $\boldsymbol{A} \in \mathbb{R}^{Ld\times m}$, the prompt embedding is directly obtained via a linear transformation:

$$\boldsymbol{E}^t = \boldsymbol{A}\boldsymbol{z}^t \in \mathbb{R}^{Ld}. \tag{2}$$

This formulation allows the optimization to take place in a compact latent space, substantially reducing the parameter search space. The use of a random Gaussian projection for $\boldsymbol{A}$ promotes subspace diversity and helps retain the expressive capacity of the original embedding space.

B. Surrogate Prompt Decoding

To decode the embedding $\boldsymbol{E}^t$ to the prompt p^t, we apply a core decoding strategy to map the optimized continuous embeddings back into discrete token sequences via nearest neighbor search in the model's token embedding space. This step is essential in black-box settings, where public LLM APIs only accept textual inputs. Ideally, the decoding should be performed using the target model's token embedding matrix to ensure semantic alignment. However, in black-box scenarios, this internal decoder is inaccessible. To this end, we leverage an important empirical property of large language models: *the token embedding spaces across different models exhibit a high degree of alignment*, owing to shared tokenization schemes, overlapping training corpora, and convergent training dynamics [34], [35]. This enables us to use a surrogate model's decoder as a practical substitute for nearest neighbor decoding. In practice, we observe that such surrogate-based decoding achieves strong performance, despite the lack of access to the target model's embedding layer.

Formally, given a prompt embedding $\boldsymbol{E}^t$, to convert this into a discrete prompt $p^t = (w_1^t, \ldots, w_L^t)$, where w_i^t denotes the i-th word token in the prompt, we perform nearest neighbor decoding using a publicly available surrogate token embedding matrix $\boldsymbol{T}^{\text{sur.}} \in \mathbb{R}^{|\mathcal{V}|\times d}$, where each embedding $\boldsymbol{T}_j^{\text{sur.}}$ represents the embedding of word token w_j. For each token position i, we compute:

$$w_i = \arg\min_{j\in\mathcal{V}} \|\boldsymbol{e}_i^t - \boldsymbol{T}_j^{\text{sur.}}\|_2, \tag{3}$$

where $\boldsymbol{e}_i$ denotes the i-th embedding in $\boldsymbol{E}^t$. The resulting sequence $p^t = (w_1^t, \ldots, w_L^t)$ is then submitted to the black-box LLM API for evaluation.

C. LLM Querying

Given a discrete prompt p, we submit it to the target language model via its public API. Although the optimization process operates in the latent space $\boldsymbol{z}$ and performs intermediate computations in the continuous embedding space $\boldsymbol{E}$, neither representation is directly compatible with the API interface, which only accepts tokenized textual inputs. Therefore, only the decoded prompt p can be used for querying the model.

Formally, we invoke the LLM with p^t as input and evaluate its output to obtain the attack objective o^t, such as the number of generated tokens:

$$o^t = \mathcal{M}^{\text{vic}}(p^t), \tag{4}$$

where $\mathcal{M}^{\text{vic}}$ denotes the black-box victim model being attacked. This black-box querying process forms the only observable channel through which the attacker can evaluate and optimize the attack objective.

D. Derivative-Free Optimization

To optimize the latent vector $\boldsymbol{z}$ in a black-box setting, we adopt a derivative-free optimization (DFO) strategy. Unlike gradient-based methods, which rely on access to model parameters or backpropagation, DFO methods require only the evaluation of the objective function, making them particularly well-suited for black-box attacks on large language models (LLMs) via public APIs. In our context, the objective function assesses the effectiveness of a given latent vector $\boldsymbol{z}$ by converting it into a discrete prompt and measuring generation-based attack metrics, *i.e.*, the length of the model's output.

Formally, the optimization problem is defined as:

$$\max_{\boldsymbol{z}^t \in \mathbb{R}^m} \mathcal{L}(\boldsymbol{z}^t) = o^t, \tag{5}$$

where $\mathcal{L}(\boldsymbol{z})$ denotes the scalar objective value (*i.e.*, generation length) returned by the victim model $\mathcal{M}^{\text{vic}}$ in response to the prompt derived from $\boldsymbol{z}$.

To solve this problem, we employ the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [36], a state-of-the-art DFO algorithm that maintains a multivariate Gaussian search distribution $\mathcal{N}(\boldsymbol{\mu}^{(t)}, \boldsymbol{\Sigma}^{(t)})$ over the latent space. The distribution parameters are iteratively updated based on the performance of sampled candidates with the following steps.

1) Initialization: At iteration $t = 0$, the search is initialized with mean vector $\boldsymbol{\mu}^0 = \mathbf{0}$ and isotropic covariance matrix $\boldsymbol{\Sigma}^0 = \sigma^2 \boldsymbol{I}$, where $\sigma > 0$ controls the initial search radius. This defines an initial uniform search distribution over the latent space.

2) Optimization Procedure: At each iteration t, CMA-ES proceeds as follows:

Sampling. A population of N latent candidates $\boldsymbol{z}_i^t$, $i = 1, 2, \cdots, N$, is drawn from the current distribution:

$$\boldsymbol{z}_i^t \sim \mathcal{N}(\boldsymbol{\mu}^t, \boldsymbol{\Sigma}^t). \tag{6}$$

Evaluation. Each candidate $\boldsymbol{z}_i^t$ is projected to an embedding $\boldsymbol{E}_i^t = \boldsymbol{A}\boldsymbol{z}_i^t$, decoded into a discrete prompt p_i, and submitted to the victim LLM $\mathcal{M}^{\text{vic}}$ to obtain its corresponding objective score $o_1^t, o_2^t, \cdots, o_N^t$.

Selection and Recombination. The candidates are ranked according to their scores o_i^t, and the top k individuals are selected. A new mean vector is computed via a weighted average of these top-performing candidates:

$$\boldsymbol{\mu}^{(t+1)} = \sum_{j=1}^{k} w_j \boldsymbol{z}_j^t, \tag{7}$$

where $\boldsymbol{z}_j^t$ denotes the j-th best candidate and w_j are predefined positive weights summing to 1.

Covariance Update. The covariance matrix is updated to reflect the empirical distribution of the selected candidates:

$$\boldsymbol{\Sigma}^{t+1} = \sum_{j=1}^{k} w_j (\boldsymbol{z}_j^t - \boldsymbol{\mu}^{t+1})(\boldsymbol{z}_j^t - \boldsymbol{\mu}^{t+1})^T + \epsilon \boldsymbol{I}, \tag{8}$$

where ϵ is a small constant added for numerical stability.

This update mechanism captures both the principal directions and the variance of successful candidates, enabling the algorithm to adaptively explore high-performing regions of the latent space. The optimization proceeds until convergence or a predefined querying budget is reached.

By directly operating on the latent vector $\boldsymbol{z}$ and relying solely on black-box evaluations of $\mathcal{M}^{\text{vic}}$, this framework enables efficient search for adversarial prompts without requiring access to model gradients or internal parameters.

VI. Online Denial-of-Service Attack

Modern large language models (LLMs), especially those deployed via public APIs, are typically equipped with basic security mechanisms designed to defend against abuse and misuse. These include input filtering, rate limiting, behavioral detection, and generation constraints [37], all of which pose significant challenges for sustained or high-frequency adversarial querying. Consequently, launching an effective deny-of-service (DoS) attack in such settings requires carefully evading these protections while still delivering adversarial prompts capable of degrading model availability or utility.

To this end, we design an online attack framework that incrementally submits crafted prompts at a moderate query rate, in order to bypass rate limiting and avoid triggering abuse detection mechanisms. Our approach does not rely on overwhelming the system with high-throughput requests. It exploits the fact that carefully optimized prompts can consume excessive model computation, even when submitted infrequently, by inducing long and resource-intensive generations. This results in a form of *slow* DoS, which imposes sustained computational burden on the model over time.

Concretely, we assume the target LLM service enforces a minimum interval of t seconds between accepted queries from a single user. To comply with this constraint and remain undetected, the attacker submits attack prompts at a fixed rate no faster than once every t seconds. Each prompt is selected from a pool of previously optimized attack prompts.

VII. Evaluation

A. Methodology

1) Implementation: We develop a prototype implementation of the attack based on a Python CMA-ES optimization library [1]. The attack is conducted under a black-box setting by interacting with target LLMs through the unified Model Router API [2], which dispatches requests to various backend models and returns their responses. Due to computational constraints, we set the maximum generation length to 4096 tokens, a commonly adopted upper bound in LLM decoding [9], and sufficient to observe generation collapse or abnormal length behaviors. The input prompt length is fixed at 20 tokens to balance optimization efficiency and input compactness, while the decoding temperature is set to its default value of 1.0 to reflect the standard sampling configuration used in public-facing LLM APIs.

TABLE II: Overview of evaluated LLMs. "OSS" denotes whether the model is open-source. "Tokens/wk" denotes the weekly token consumption during June 23–29, 2025. "Price" represents the approximate cost per 1M *output* tokens for public API inference at the time of evaluation. Values are intended to contextualize practical deployment scale and cost rather than serve as exact billing references.

Model	Params	Provider	OSS	Tokens/wk	Price†
Gemini 2.5 Pro	N/A	Google	No	88.8B	$10
Lumimaid	70B	NeverSleep	No	12.4M	$3
Magistral	N/A	Mistral	No	58.7M	$5
GPT-o4	N/A	OpenAI	No	3.22B	$4.4
MAI DS R1	671B	Microsoft	Yes	998M	$1.2
DS Qwen3	8B	DeepSeek	Yes	1.93B	$0.02
Llama 3.2	3B	Meta	Yes	10.4B	$0.02
DS R1	671B	DeepSeek	Yes	63.2B	$2.15

† Open-source models (MAI DS R1, DS Qwen3, Llama 3.2, DS R1) are accessible via third-party providers offering free usage tiers for platform promotion. As a result, attackers may leverage such free services to conduct attacks without incurring any monetary cost.

2) LLMs: We evaluate a diverse set of eight LLMs, as listed in Table II, covering both proprietary and open-source models. This selection includes frontier-scale models such as DeepSeek R1 (671B) and widely used close-sourced LLMs like GPT o4 and Gemini 2.5 Pro. The weekly token usage reflects their real-world adoption in our experiments conducted from June 23 to 29, 2025.

3) Baselines: We consider the following four black-box baselines for evaluation.

- **Decoy Problem** [11]: We collect a set of 20 samples by prompting GPT-4o to generate a set of open-ended, high-complexity questions spanning a wide range of domains, including physics, machine learning, economics, biology, and philosophy. These questions are intentionally selected to induce prolonged reasoning and multi-step generation, making them particularly challenging for LLMs.
- **Semantic Problem** [9]: We enhance the decoy problems by adding explicit semantic cues that encourage longer responses. Specifically, each question is rephrased to include instructions such as "Output a longer explanation" or "Provide a more detailed discussion," thereby guiding the language model to extend its generation.
- **LLMEffiChecker** [14]: We adopt the attack proposed in LLMEffiChecker, which performs word-level perturbations. Specifically, it first measures the impact of each word on the output length and identifies the word that contributes most to shortening the generation. Then, it randomly substitutes this word to induce longer outputs from the model.
- **Sponge Examples** [10]: This approach leverages a genetic algorithm to perform word-level optimization. By iteratively evolving input sequences, it generates adversarial examples that induce excessive computational cost and prolonged output generation in language models.

[1]https://github.com/CyberAgent/cmaes

[2]https://openrouter.ai/

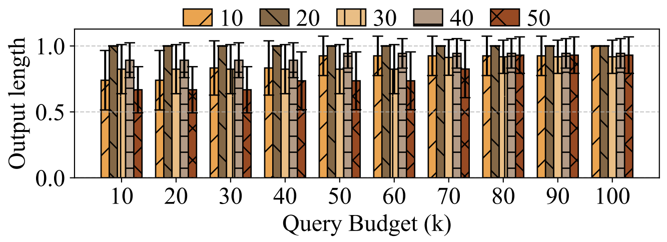
4) Metrics: We consider the metrics in terms of LLM behaviors in various aspects.

- **LLM output length**: Output length measures the total number of tokens generated in response to a generated attack prompt. This reflects how effectively the prompt can elicit prolonged generation. Considering the possibility that a large language model (LLM) may generate unbounded output when engaged in infinite reasoning, we impose a maximum output length of 4096 tokens during evaluation. For consistency and clarity in presentation, all reported output lengths are normalized with respect to 4096 tokens.
- **Tokens per second (TPS)**: Tokens per second, *i.e.*, TPS, captures the generation throughput, defined as the number of output tokens produced per second. A lower throughput under attack inputs indicates degraded model efficiency and increased inference cost.
- **Time to first token (TTFT)**: Time to first token, *i.e.*, TTFT, refers to the time elapsed between submitting the prompt and receiving the first generated token. Increased latency may suggest that the prompt induces heavier computational burden during decoding initialization.
- **GPU memory consumption**: GPU memory consumption records the peak GPU memory usage during model inference. Prompts that trigger abnormally high memory usage can be indicative of resource exhaustion vulnerabilities.

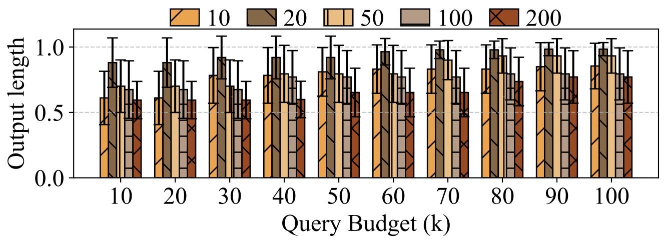
B. Hyperparameter Search

1) Prompt Length: We first investigate the effect of prompt length on the attack results. To this end, we plot the result of normalized output length with five input prompt lengths on DeepSeek R1, *i.e.,* 10, 20, 30, 40, 50, in Fig. 2a. We observe a non-monotonic trend in the effectiveness of prompt-based attacks under a fixed query budget in Fig. 2a. As prompt length increases, the attack initially becomes more potent due to enhanced expressiveness. However, beyond a certain length (*e.g.*, 20), performance degrades. This is because longer prompts expand the search space, causing the derivative-free optimization to waste more budget on uninformative directions. Consequently, the probability of discovering highly effective adversarial prompts within a fixed number of queries decreases. As a result, we choose a balanced prompt length of 20 for the best attack performance on a given budget.

2) Latent Vector Dimension: We then investigate the effect of subspace dimensions on the attack results. Specifically, we report the normalized output length under five latent dimensions, *i.e.*, 10, 20, 50, 100, and 200, each averaged over 20 independent trials, as shown in Fig. 2b. The results exhibit a non-monotonic trend. Moderate dimensionalities enhance attack effectiveness by providing a richer search space and greater expressive capacity for adversarial prompts. However, once the dimension becomes sufficiently large, performance begins to degrade, due to increased optimization difficulty and the curse of dimensionality. Accordingly, we select a latent



(a) Different prompt lengths.



(b) Different latent vector dimensions.

Fig. 2: Output length of DeepSeek R1 with respect to the upper bound of 4096 on ThinkTrap under (a) different prompt lengths and (b) different latent vector dimensions, where a non-monotonic trend can be observed in both hyperparameters for a balance of prompt expressiveness and search efficiency.

dimension of 20 as a practical choice that balances achievable output length and optimization efficiency.

C. LLM Output Length of Attack Prompts

To evaluate the generation efficiency of different LLMs under varying output budgets, we conduct experiments by setting the allowed output budget from 10K to 100K tokens. For each budget level, we compare different attack methods in terms of their ability to generate adversarial prompts that induce the longest possible model outputs. To mitigate the impact of randomness, each experiment is repeated five times. Fig. 3 reports the average output lengths along with standard deviations across the five runs.

1) Comparison with Baselines: We first compare ThinkTrap with existing baselines to show its superior performance.

ThinkTrap succeeds even on models where semantic-based attacks fail. We observe that the performance of attack methods based on semantics, *i.e.*, decoy problem and semantic problem, heavily depends on the specific architecture and alignment strategy of the target LLM. While these methods may achieve moderate success on certain instruction-tuned models, they often fail to generalize across models with different pretraining objectives or decoding behaviors, *e.g.*, Lumimaid, DS R1, Llama 3.2. This suggests that purely semantic manipulations lack the robustness and universality required for cross-model transferability.

ThinkTrap attains long outputs with far lower query budgets than heuristic-driven baselines, which struggle under limited budgets. We can see that heuristic-driven search methods, *i.e.*, LLMEffiChecker and Sponge problem, do not perform well, especially on Gemini 2.5 Pro and MAI DS R1 when budget is low. This is because these approaches typically rely on manually designed scoring functions or rule-based mutation strategies. As a result, they require significantly more queries or larger generation budgets to discover effective adversarial prompts. This inefficiency becomes particularly pronounced when operating under tight computational constraints, limiting their practicality for large-scale attacks. In contrast, ThinkTrap leverages an adaptive optimization strategy, enabling faster convergence with fewer queries.

ThinkTrap demonstrates robust performance across seven of the eight evaluated LLMs, with particularly strong advantages under limited query budgets. These results indicate its capability to efficiently exploit generation dynamics under resource constraints. Unlike baselines that typically depend on substantial query budgets to induce long outputs, ThinkTrap can discover prompts that trigger disproportionately long responses with limited tokens. While its performance is consistently strong for seven out of eight models, we observe that for LLama 3.2 at very low budgets, Sponge achieves higher output lengths. This exception highlights the importance of model-specific factors, but overall, ThinkTrap's generalization across architectures and budget levels underscores its practical value in realistic denial-of-service settings where adversaries may face strict cost constraints. Its efficiency arises from an adaptive lightweight search strategy that transfers effectively across models with different alignment and decoding characteristics.

2) Attack Effects on Various LLMs: We then analyze the attack effect of ThinkTrap on various types of LLMs.

ThinkTrap maintains strong effectiveness across both closed-source and open-source LLMs, indicating that its performance does not depend on access to model internals. We can see the ThinkTrap perform well on both closed-source and open-source LLMs. This is because ThinkTrap is a black-box attack method without requiring access to model internals or gradient information, which further underscores its practicality for real-world adversarial scenarios.

ThinkTrap remains effective across diverse model families, whereas baseline methods show less consistent performance across architectures. We can see that ThinkTrap performs well on various model families including Gemini, Luminmaid, Magistral, GPT, Qwen, DeepSeek, and Llama. This shows that modern LLMs all suffer from this kind of attacks. Despite that different attack methods attack different parts of LLM, ThinkTrap can outperform other baselines on every evaluated model family, showing the high importance of the proposed ThinkTrap method.

ThinkTrap exhibits strong effectiveness across LLMs of varying scales, indicating that its performance generalizes well regardless of model size. We can also see that LLMs of various mainstream sizes, *i.e.*, 3B (Llama 3.2), 8B (DS Qwen 3), 70B (Lumimaid), and 671B (DS R1 and MAI DS R1).

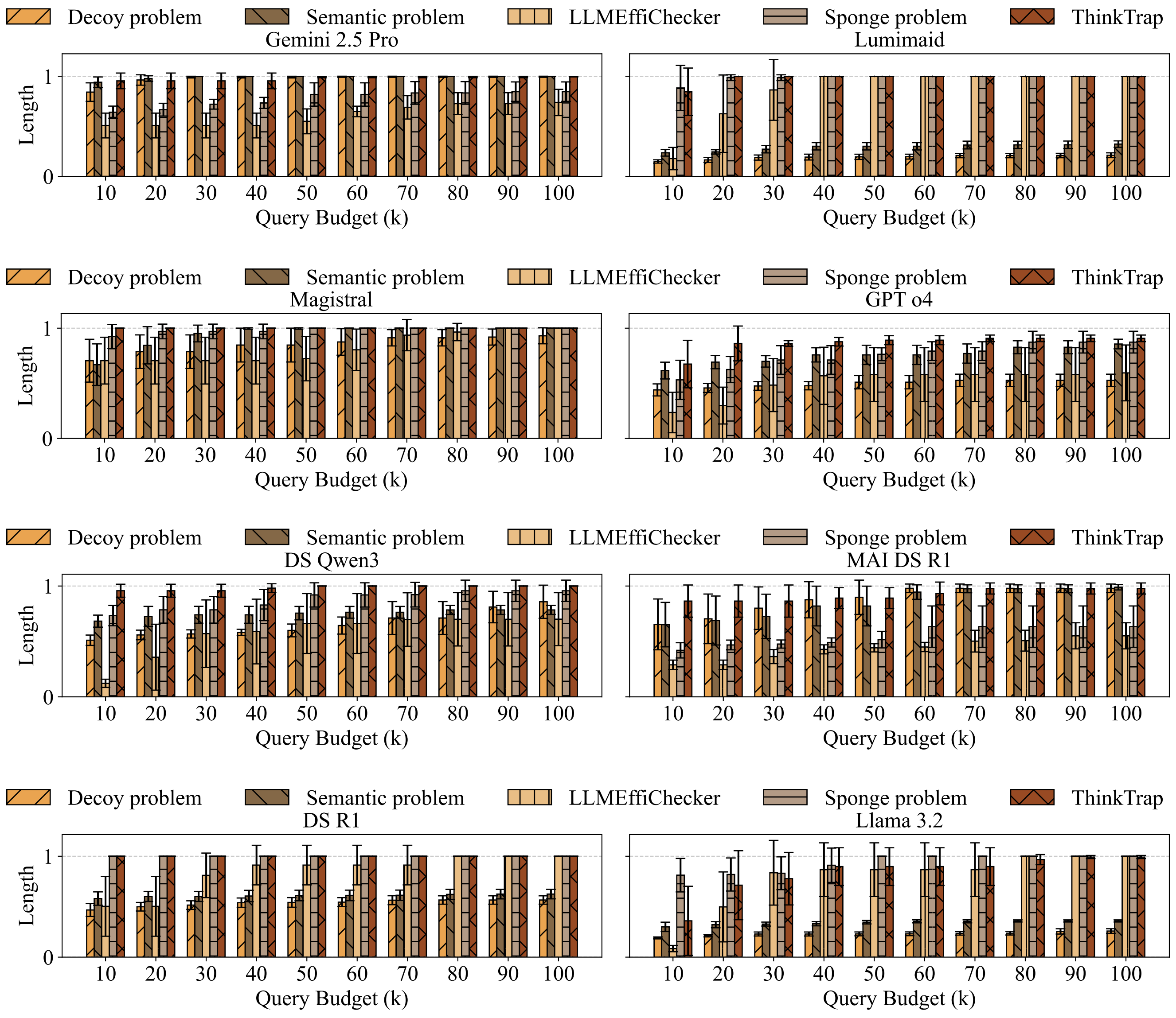


Fig. 3: Output length of the evaluated eight LLMs on ThinkTrap and all the four baselines with respect to the upper bound of 4096, where different baseline methods exhibit varying performance across different models, but ThinkTrap consistently achieves the highest output length across all LLMs. The advantage of ThinkTrap is particularly evident under lower generation budgets, demonstrating its efficiency in maximizing output with minimal resources.

Moreover, their behaviors do not vary despite different sizes. This shows that despite the inference cost various, their risks under the DoS attack equal. In other words, larger LLMs are not safer.

ThinkTrap is effective on both thinking and non-thinking LLMs, suggesting that vulnerability to the attack extends beyond models with explicit reasoning features. We can see that ThinkTrap can also work well on the non-thinking base LLM, *i.e.*, Llama 3.2. This shows that non-thinking LLM may also suffer from the DoS attack. While we fail to attack many early published LLMs such as Llama 2. We owe this failure to the fact that early LLMs even do not have an ability to output a long enough results. However, current LLMs such as Llama-3.2 can generate longer outputs than early models, making them more vulnerable to this attack.

3) Attack Effects on Various Decoding Strategies: Decoding temperature is an important parameter which affects the diversity of decoding behavior. We also plot the performance of ThinkTrap on DS R1 under different decoding temperatures in Fig. 4 to show its behavior of various temperatures.

ThinkTrap attack remains effective across a wide range of decoding temperature settings. Decoding temperature is a hyperparameter that controls the stochasticity of token sampling and thereby regulates output diversity in LLM de-

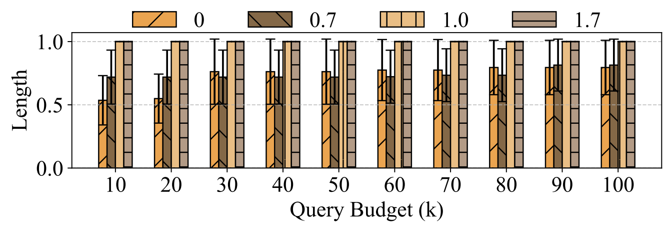


Fig. 4: Output length relative to the maximum limit of 4096 tokens for the eight evaluated LLMs under ThinkTrap, across varying decoding temperatures (*i.e.*, 0, 0.7, 1, 1.7), where higher temperatures, introducing greater sampling randomness, consistently result in longer outputs.

coding. Fig. 4 illustrates that the attack consistently succeeds across both low and high temperature configurations. A low decoding temperature yields low output diversity, causing the model to generate repetitive lexical or token patterns that sustain the attack. In contrast, higher temperatures increase output variability, making the model less inclined to produce the `<EOS>` token. In most LLMs, `<EOS>` acts as a high-logit, low-entropy token near the end of a sentence, where it signals sequence completion. Raising the temperature weakens this token's dominance, resulting in longer and more chaotic continuations, during which the model may enter a self-reinforcing or perpetual thinking loop.

4) Attack Cost Analysis: We further quantify the attack cost of different methods to assess the practical feasibility of the proposed attack.

ThinkTrap imposes a very small cost and can be executed adaptively given inferred system capabilities. In particular, ThinkTrap attains a successful attack (*i.e.*, forcing the model to produce 4k+ tokens) with an attack budget of only 10k tokens on Gemini 2.5 Pro, Magistral, DS Qwen3, and DS R1. In these cases, end-to-end execution requires only a few minutes. For other evaluated LLMs, a comparable success is also achievable within 100k tokens, corresponding to runtimes below one hour. Using the prices reported in Table II, the monetary cost of such attacks is negligible in practice. A representative cost for DS R1 requiring a 10k-token budget is only $0.0215. Even for relatively costly services, the expense remains small, *e.g.*, approximately $0.10 for Gemini 2.5 Pro requiring a 10k-token budget, and $0.44 for GPT o4 requiring a 100k-token budget. These results demonstrate that ThinkTrap is both low-cost and practically feasible in realistic scenarios.

D. Attack Results on LLM Service Systems

We then evaluate attack results of the proposed ThinkTrap on practical LLM service systems. Specifically, we deploy the DS Llama 8B LLM as a local LLM service on a GPU server equipped with four NVIDIA RTX 2080ti GPUs. To evaluate the system's robustness, we offline-generate 100 adversarial prompts and submit them to the service via its API at a low injection rate of 10 prompts per minute (RPM=10), thereby avoiding detection by standard rate-limiting mechanisms. The maximum number of tokens per generation is set to 32,768 to enable extensive output, which is a common setting of modern LLM service. The metrics of time to first token (TTFT), tokens per second (TPS), GPU memory consumption of the GPU server are plotted on Fig. 5.

ThinkTrap effectively exhausts the computational resources of the LLM server. The crafted adversarial prompts result in substantially prolonged inference durations. Notably, the system latency increases approximately linearly prior to the 40th input, suggesting that each prompt reliably induces around four minutes of sustained computation. Beyond this point, the rate of latency growth slightly tapers, as some earlier generations complete. Nevertheless, due to the continuous arrival of new prompts and the already degraded computational throughput, the overall latency continues to escalate rapidly.

ThinkTrap effectively depletes the GPU memory resources of the LLM server, thereby enabling a successful denial-of-service (DoS) attack. We further observe a sustained increase in GPU memory consumption across all four devices. Initially, each GPU utilizes no more than 4GB of memory. However, after the injection of 80 adversarial prompts, the most heavily loaded GPU reaches 8GB of usage. This growth is primarily attributed to the accumulation of key-value (KV) caches for each ongoing inference, which demands substantial memory resources. Considering that a portion of GPU memory is reserved for the inference framework itself, this level of consumption approaches the capacity limit of consumer-grade GPUs such as the RTX 2080 Ti. Consequently, after the 80th input, the system experiences memory exhaustion, causing many inference requests to fail or time out, indicating a successful denial-of-service (DoS) attack. While the system remains partially responsive afterward, its processing speed remains severely degraded, and memory usage continues to rise until the next crash occurs.

It's hard to defend against ThinkTrap by simply limiting the maximum number of output tokens, as such constraints can significantly degrade service quality. One naive way to defend against the ThinkTrap attack is to limit the maximum output tokens allowed to output. In this way, the artificially induced high output is limited. However, we can see from Fig. 5 that, even with a strict output limitation of 256 tokens, the decoding speed metrics, *i.e.*, TTFT and TPS, still slow down significantly. With the output limitation of 128 tokens, the TTFT and TPS can remain stable, which successfully defense the attack of ThinkTrap. However, such a low output length will greatly affect the user experience, which is obviously not feasible. We can also see that the LLM service of a reasonable length limit of 1024 tokens behaves almost the same as the LLM service without length limit. This shows that the proposed ThinkTrap attack can cause great harm to the system even when the service allows a reasonable maximum output length.

E. Transferability of Attack Prompts

We further evaluate the transferability of the generated attack prompts, given the substantial computational cost associated with their generation, *i.e.*, we assess the feasibility of

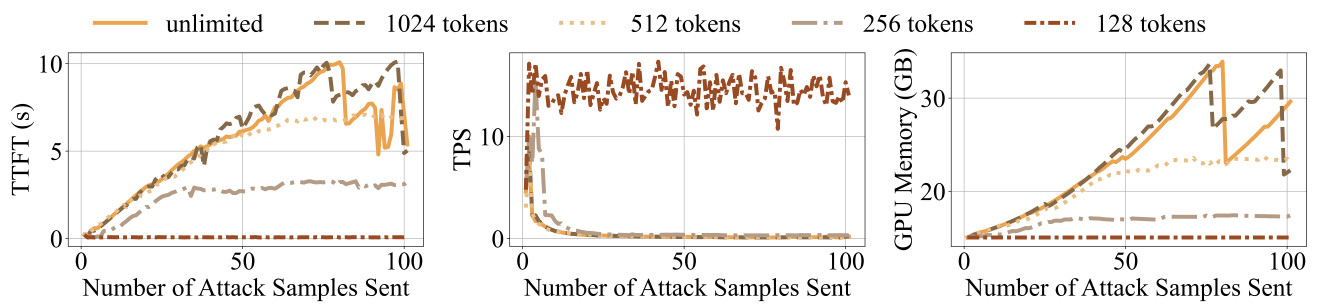


Fig. 5: Impact of ThinkTrap attack on the DeepSeek Llama service with a just allowed attack rate of 10 RPM based on the *Transformers* library using 4 NVIDIA 2080ti GPUs with different output token limitations, where only the unrealistic limitation of 128 tokens can successfully defend the attack.

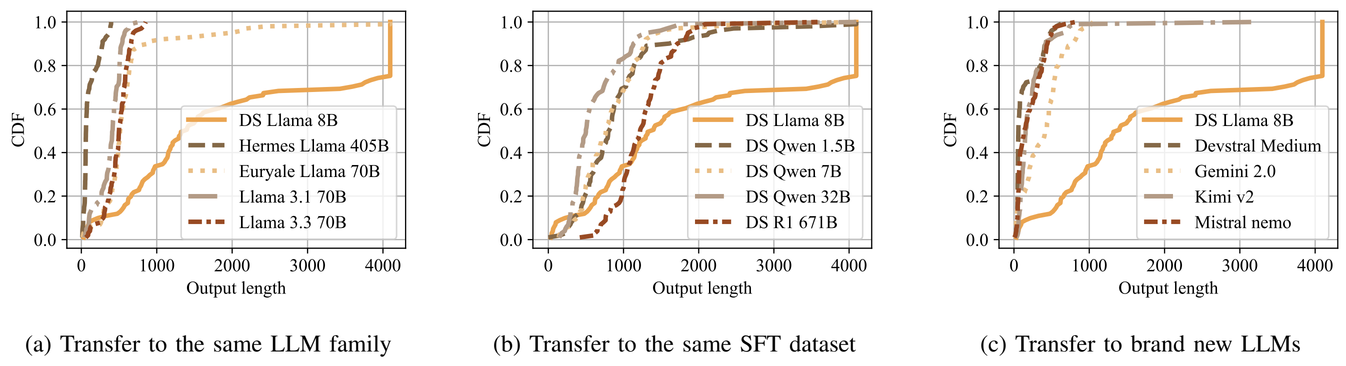


Fig. 6: Cumulative distribution function (CDF) of the output lengths of attack prompts generated for DS Llama 8B and evaluated across various LLMs, indicating that models fine-tuned on the same dataset may share similar vulnerabilities.

reusing prompts crafted to exploit one LLM to successfully attack other LLMs without incurring additional generation overhead. To this end, we evaluate the transferability of offline-generated attack prompts for DS Llama 8B as an example.

1) Transferring across LLM Families: We first evaluate the transferability of attack prompt across LLM families. To this end, we plot the output length of attack prompts generated for the DS Llama 8B for several Llama-based LLMs, *i.e.*, Hermes Llama 405B, Euryale Llama 70B, Llama 3.1 70B, Llama 3.3 70B. The cumulative distribution function (CDF) on the output length of the evaluated LLMs, including the source DS Llama 8B, is shown in Fig. 6a.

The attack prompts exhibit limited transferability even within the same LLM family. As shown in Fig. 6a, even within the same model family, *e.g.*, Llama, the effectiveness of the attack prompts significantly degrades when transferred to another model variant. The resulting output lengths are consistently below 800 tokens, substantially shorter than those observed in the source LLM. This observation suggests that the exploited vulnerabilities are not inherent to the shared architecture or parameter space of the LLM family.

2) Transferring across LLM Supervised Fine-Tuning (SFT) Datasets: We then evaluate the transferability of attack prompts across Supervised Fine-Tuning (SFT) datasets. Specifically, we evaluate the output length of the LLMs fine-tuned on the DeepSeek R1 distilled dataset, *i.e.*, DS R1, DS Qwen 1.5B, DS Qwen 7B, DS Qwen 32B, and DS R1 671B, also on attack prompts generated for DS Llama 8B. The CDF of output lengths is shown in Fig. 6b.

The attack prompts exhibit strong transferability across models fine-tuned on the same supervised fine-tuning (SFT) dataset. As illustrated in Fig. 6b, although some degradation in performance is observed, models fine-tuned on the same SFT dataset, *i.e.*, DeepSeek R1, remain largely susceptible to each other's attack prompts. Notably, prompts generated by the DS Llama model frequently elicit outputs exceeding 4096 tokens, with a high likelihood of surpassing 800 tokens. This suggests that the vulnerability associated with infinite generation is likely introduced during the post-pretraining SFT phase. Furthermore, we observe that the output length of the source DS Llama 8B model does not consistently reach 4096 tokens, which can be attributed to the inherent stochasticity of LLM decoding. Nevertheless, the generated outputs are sufficiently long to preserve the effectiveness of the attack.

3) Transferring to Brand New LLMs: We also evaluate the transferability of attack prompts on the brand new LLMs to access its universal generalization ability. To this end, we evaluate the output lengths of the new LLMs that are unrelated to the DS Llama 8B LLM, *i.e.*, Devstral Medium, Gemini 2.0, Kimi v2, and Mistral nemo. The CDF of output length is shown in Fig. 6c.

The attack prompts demonstrate limited generalizability when applied to unseen LLMs. As shown in Fig. 6c, attack prompts crafted specifically for the DS Llama model are largely ineffective against novel LLMs. Despite variations in output lengths across different models, most fail to approach the maximum length of 4096 tokens, with the majority producing outputs of fewer than 800 tokens. This outcome supports the validity of using 4096 as a practical upper bound for evaluating excessive output generation, as non-adaptive prompts rarely trigger such extended outputs. Moreover, the results indicate that the generated prompts lack cross-model generalization, suggesting that each LLM exhibits unique susceptibility patterns and requires tailored attack samples.

VIII. Defending Against ThinkTrap

In this section, we examine practical defense mechanisms against ThinkTrap and analyze their effectiveness in real-world LLM serving environments. We focus on two representative mitigation strategies that are widely deployed in current LLM hosting systems, namely lightweight anomaly detection and resource-aware scheduling. Our evaluation further investigates their operational implications, providing guidance for practitioners who must balance robustness and service quality.

A. Defense Mechanisms

We consider the following two typical practical defense mechanisms:

- **Anomaly Detection** [38], [39]: An anomaly detection mechanism is implemented to identify repetitive or looping generations through an n-gram-based analysis of model outputs. Consecutive token sequences, *i.e.*, n-grams, are continuously monitored within a sliding window, and requests exhibiting excessive recurrence frequency are regarded as degenerated and terminated early to prevent unnecessary computation. 4-grams are employed [38] in our implementation to provide a trade-off between detection sensitivity and robustness against benign stylistic repetition.
- **Resource-aware Scheduling** [40], [41], [42]: We implement a resource-aware scheduling mechanism following the Virtual Token Counter (VTC) policy [40], which enforces fine-grained control over decoding progress to prevent unbounded resource occupation. Instead of allowing a request to decode continuously, the scheduler allocates each active request a fixed token quantum (*e.g.*, 1024 tokens) per scheduling round. Once this quota is exhausted, the request is preempted, its state is cached, and it is re-queued behind other pending tasks. This token-level preemption ensures that no single request can occupy GPU or NPU resources for an extended interval, thereby limiting the impact of long-generation abuse patterns.

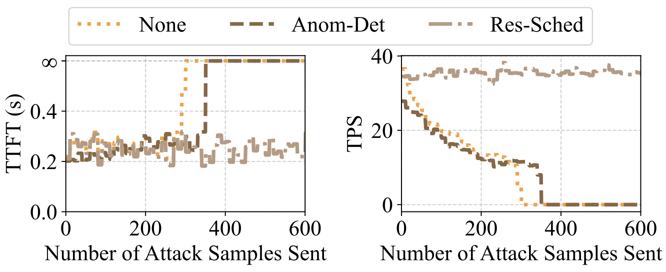


Fig. 7: Time-to-First-Token (TTFT) and Throughput (TPS) of the LLM service under the ThinkTrap attack (10 RPM) with anomaly detection (Anom-Det) and resource-aware scheduling (Res-Sched). Anomaly detection offers limited protection and adds overhead, while resource-aware scheduling mitigates the attack at the cost of degraded QoS for long-form requests.

B. Effectiveness of Defense Mechanisms

We evaluate the defense performance on a server equipped with eight Ascend 910B NPUs, each providing 64 GB of high-bandwidth memory (HBM), is employed for the evaluation. We select DeepSeek Llama 70B as the representative model, given its widespread adoption and the suitability of its scale for this hardware configuration. The model is deployed using MindIE, which schedules decoding requests in a First-In-First-Out (FIFO) manner. The adversarial prompts are pre-generated and subsequently issued to the target server at a controlled rate of 10 requests per minute (RPM). Fig. 7 illustrates the Time-to-First-Token (TTFT) and Throughput (TPS) of the LLM service under the ThinkTrap attack with the Anomaly Detection and Resource-aware Scheduling defense mechanisms, respectively.

Anomaly detection defense is ineffective against ThinkTrap. We can see from Fig. 7 that although anomaly detection slightly alleviates the denial-of-service (DoS) effect caused by ThinkTrap, its protection capacity remains extremely limited. It can only tolerate several tens of adversarial prompts before the service eventually collapses. This weakness arises because the long-form outputs induced by ThinkTrap are not merely mechanical repetitions of surface tokens but exhibit semantic-level redundancy with moderate diversity, making them difficult to capture through lightweight repetition detectors. Moreover, the anomaly detection approach incurs a noticeable performance overhead. As shown in Fig. 7, its TPS is lower even at the beginning of the attack. This indicates that, despite being computationally lightweight, the detector must still inspect every LLM decoding stream, thereby reducing overall throughput. More sophisticated anomaly detection methods, such as those employing an auxiliary language model to assess semantic repetition, would consume substantially more computational resources, rendering them impractical for real-time inference services.

The resource-aware scheduling approach successfully defenses the attack, albeit with a trade-off in service quality for long-response requests. As shown in Fig. 7, under single-user attack scenarios, the resource-aware scheduling mechanism effectively mitigates the ThinkTrap attack by constraining the maximum decoding length of each inference request. Once the predefined limit is reached, the decoding process is terminated and the corresponding computational resources are promptly released. This mechanism prevents malicious requests from monopolizing the hardware for unbounded periods and thus protects the service from complete breakdown. However, this improvement in system stability comes at the cost of degraded quality of service (QoS) for legitimate requests that inherently require long-form reasoning. Tasks requiring long outputs such as mathematical problem solving and embodied task planning experience frequent interruptions and forced re-scheduling, leading to a significant reduction in their overall inference throughput. Furthermore, because scheduling operates in a sequential manner, the defense remains vulnerable under concurrent multi-user attacks. Such queuing pressure is well known in multi-user video analytics systems [43]. When multiple adversarial clients issue long-generation requests simultaneously, the scheduler becomes congested, causing benign requests to experience excessive queuing delays and inflated TTFT.

C. Discussion of Defenses

Defense evaluation demonstrates that resource-aware scheduling is highly effective in preventing adversarial requests from monopolizing decoding resources, thereby preserving service availability under sustained attack. Such mechanisms, however, inevitably introduce additional latency for benign requests that require long, uninterrupted generations. This tradeoff highlights a fundamental tension in serving large-scale LLMs, where the system must maintain fairness and availability in the presence of adversaries while simultaneously supporting emerging workloads that demand extended reasoning or narrative outputs. We argue that resource-aware scheduling should therefore be treated as a first-class requirement for modern LLM hosting platforms, rather than an optional optimization.

Beyond the mechanisms evaluated in this study, several operational safeguards, such as output-length caps, per-user rate limiting, and per-query compute metering, can limit an attacker's ability to induce unbounded inference. However, these strategies degrade the experience for legitimate users whose tasks naturally require long outputs or multi-step reasoning. Their role resembles classical DoS mitigation approaches that preserve availability by selectively reducing functionality or imposing flow control under extreme load. While effective as last-resort measures, these approaches are fundamentally coarse-grained and do not address the root cause of reasoning-induced DoS behavior.

Looking forward, more principled defenses will likely require a deeper understanding of the computation pathways exercised during long-form and multi-step LLM reasoning. Developing real-time signals that reflect internal model states, decoding complexity, or incremental resource usage could enable more adaptive scheduling and throttling mechanisms. Attack-aware resource allocation should also be incorporate into LLM-serving frameworks, balancing robustness, throughput, and model utility under adversarial environments.

IX. Conclusion

This paper investigates a previously overlooked vulnerability in closed-source LLM services: their susceptibility to DoS attacks via adversarial prompts. We propose ThinkTrap, a black-box attack framework that identifies prompts inducing excessive computation by leveraging derivative-free optimization in a continuous surrogate space. Through a low-dimensional, token-wise strategy, ThinkTrap circumvents the challenges of discrete input spaces and high-dimensional optimization. Extensive evaluations on commercial and self-hosted LLMs demonstrate that ThinkTrap can degrade system performance by inflating output length, exhausting GPU resources, and delaying legitimate queries. These results highlight a critical, asymmetric threat to LLM infrastructure and underscore the need for prompt-level defenses in black-box deployment settings.

Ethics Statements

This work investigates prompting-based denial-of-service risks in LLM services. Given the sensitivity of studying system abuse vectors, we followed strict ethical and responsible research practices throughout the research. All stress experiments were conducted on privately hosted and university-managed LLM deployments, *i.e.*, models deployed on the Zhiyuan-1 cluster maintained by the Center for High Performance Computing at Shanghai Jiao Tong University, with explicit permission from system administrators. These environments allowed full instrumentation and stress evaluation to safely examine worst-case execution behavior.

For commercial LLMs (*e.g.*, ChatGPT, Gemini, DeepSeek), we only conducted limited and controlled trial queries during June and July 2025. These queries strictly adhered to publicly documented usage policies, did not exceed normal user behavior patterns, and did not create abnormal load or service disruption. No unauthorized stress tests were performed on commercial infrastructure. Following reviewer guidance on responsible disclosure, we formally contacted all evaluated LLM providers via their official security reporting channels on October 10, 2025, summarizing the ThinkTrap mechanism, its potential impact, and mitigation insights. Our goal is to responsibly surface security weaknesses to strengthen AI system robustness, not to enable misuse. We do not release attack-specific configurations that could facilitate abuse, and we encourage LLM platform operators to adopt proactive safeguards. No personal data, user content, or proprietary system logs were accessed in this study.

Acknowledgment

This work was supported in part by the Natural Science Foundation of China (Grants No. 62432008, 62472083). The computations in this paper were run in part on the Zhiyuan-1 cluster supported by the Center for High Performance Computing at Shanghai Jiao Tong University.

References

[1] K. Sanderson, "Gpt-4 is here: what scientists think," *Nature*, vol. 615, no. 7954, p. 773, 2023.

[2] J. Qiu, K. Lam, G. Li, A. Acharya, T. Y. Wong, A. Darzi, W. Yuan, and E. J. Topol, "Llm-based agentic systems in medicine and healthcare," *Nature Machine Intelligence*, vol. 6, no. 12, pp. 1418–1420, 2024.

[3] R. Mon-Williams, G. Li, R. Long *et al.*, "Embodied large language models enable robots to complete complex tasks in unpredictable environments," *Nature Machine Intelligence*, pp. 1–10, 2025.

[4] M. Li, S. Zhao *et al.*, "Embodied agent interface: Benchmarking llms for embodied decision making," in *Proceedings of NeurIPS*, 2024.

[5] Y. Zheng, H. Y. Koh, J. Ju, A. T. Nguyen, L. T. May, G. I. Webb, and S. Pan, "Large language models for scientific discovery in molecular property prediction," *Nature Machine Intelligence*, pp. 1–11, 2025.

[6] D. Truhn, J. S. Reis-Filho, and J. N. Kather, "Large language models should be used as scientific reasoning engines, not knowledge databases," *Nature medicine*, vol. 29, no. 12, pp. 2983–2984, 2023.

[7] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni, "Analysis of a denial of service attack on tcp," in *Proceedings of IEEE S&P*, 1997.

[8] K. Pelechrinis, M. Iliofotou, and S. V. Krishnamurthy, "Denial of service attacks in wireless networks: The case of jammers," *IEEE Communications surveys & tutorials*, vol. 13, no. 2, pp. 245–257, 2010.

[9] J. Dong, Z. Zhang, Q. Zhang, T. Zhang, H. Wang, H. Li, Q. Li, C. Zhang, K. Xu, and H. Qiu, "An engorgio prompt makes large language model babble on," in *Proceedings of ICLR*, 2025.

[10] I. Shumailov, Y. Zhao, D. Bates, N. Papernot, R. Mullins, and R. Anderson, "Sponge examples: Energy-latency attacks on neural networks," in *Proceedings of IEEE EuroS&P*, 2021.

[11] A. Kumar, J. Roh, A. Naseh, M. Karpinska, M. Iyyer, A. Houmansadr, and E. Bagdasarian, "Overthinking: Slowdown attacks on reasoning llms," *arXiv preprint arXiv:2502.02542*, 2025.

[12] X. Chen, J. Xu, T. Liang, Z. He, J. Pang, D. Yu, L. Song, Q. Liu, M. Zhou, Z. Zhang *et al.*, "Do not think that much for 2+ 3=? on the overthinking of o1-like llms," *arXiv preprint arXiv:2412.21187*, 2024.

[13] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of ACL*, 2016.

[14] X. Feng, X. Han, S. Chen, and W. Yang, "Llmeffichecker: Understanding and testing efficiency degradation of large language models," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 7, pp. 1–38, 2024.

[15] A. Aghajanyan, S. Gupta, and L. Zettlemoyer, "Intrinsic dimensionality explains the effectiveness of language model fine-tuning," in *Proceedings of ACL*, 2021.

[16] Y. Qin, X. Wang, Y. Su, Y. Lin, N. Ding, J. Yi, W. Chen, Z. Liu, J. Li, L. Hou *et al.*, "Exploring universal intrinsic task subspace for few-shot learning via prompt tuning," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2024.

[17] Y. Xu, L. Xie, X. Gu, X. Chen, H. Chang, H. Zhang, Z. Chen, X. Zhang, and Q. Tian, "Qa-lora: Quantization-aware low-rank adaptation of large language models," in *Proceedings of ICLR*, 2024.

[18] T. Sun, Y. Shao, H. Qian, X. Huang, and X. Qiu, "Black-box tuning for language-model-as-a-service," in *Proceedings of ICML*, 2022.

[19] A. Müller and E. Quiring, "The impact of uniform inputs on activation sparsity and energy-latency attacks in computer vision," in *Proceedings of IEEE SPW*, 2024.

[20] A. Shapira, A. Zolfi, L. Demetrio, B. Biggio, and A. Shabtai, "Phantom sponges: Exploiting non-maximum suppression to attack deep object detectors," in *Proceedings of IEEE/CVF WACV*, 2023.

[21] C. Schoof, S. Koffas, M. Conti, and S. Picek, "Beyond phantomsponges: Enhancing sponge attack on object detection models," in *Proceedings of ACM WiseML*, 2024.

[22] C. Ma, N. Wang, Q. A. Chen, and C. Shen, "Slowtrack: Increasing the latency of camera-based perception in autonomous driving using adversarial examples," in *Proceedings of AAAI*, 2024.

[23] H. Liu, Y. Wu, Z. Yu, Y. Vorobeychik, and N. Zhang, "Slowlidar: Increasing the latency of lidar-based detection using adversarial examples," in *Proceedings of IEEE/CVF CVPR*, 2023.

[24] S. Chen, C. Liu, M. Haque, Z. Song, and W. Yang, "Nmtsloth: understanding and testing efficiency degradation of neural machine translation systems," in *Proceedings of ACM FSE*, 2022.

[25] J. Geiping, A. Stein, M. Shu, K. Saifullah, Y. Wen, and T. Goldstein, "Coercing llms to do and reveal (almost) anything," in *Proceedings of ICLR STLLM Workshop*, 2024.

[26] B. Hou, J. O'connor, J. Andreas, S. Chang, and Y. Zhang, "Promptboosting: Black-box text classification with ten forward passes," in *Proceedings of ICML*, 2023.

[27] A. Zou, Z. Wang, N. Carlini, M. Nasr, J. Z. Kolter, and M. Fredrikson, "Universal and transferable adversarial attacks on aligned language models," *arXiv preprint arXiv:2307.15043*, 2023.

[28] Z. Xu, Y. Liu, G. Deng, Y. Li, and S. Picek, "A comprehensive study of jailbreak attack versus defense for large language models," in *Findings of ACL*, 2024.

[29] X. Guo, F. Yu, H. Zhang, L. Qin, and B. Hu, "Cold-attack: Jailbreaking llms with stealthiness and controllability," in *Proceedings of ICML*, 2024.

[30] Huawei Technologies Co., Ltd., "Mindie: Mindspore inference engine for ascend ai processors," https://www.mindspore.cn/mindie, accessed: 2025-07-02.

[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Proceedings of NeurIPS*, 2017.

[32] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[33] X. Zhang, "Gaussian distribution," in *Encyclopedia of machine learning and data mining*. Springer, 2016, pp. 1–5.

[34] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Proceedings of NeurIPS*, 2020.

[35] W. Gurnee and M. Tegmark, "Language models represent space and time," *arXiv preprint arXiv:2310.02207*, 2023.

[36] A. Auger and N. Hansen, "Tutorial cma-es: evolution strategies and covariance matrix adaptation," in *Proceedings of ACM GECCO Companion*, 2012.

[37] F. Wu, N. Zhang, S. Jha, P. McDaniel, and C. Xiao, "A new era in llm security: Exploring security concerns in real-world llm-based systems," *arXiv preprint arXiv:2402.18649*, 2024.

[38] H. Li, T. Lan, Z. Fu, D. Cai, L. Liu, N. Collier, T. Watanabe, and Y. Su, "Repetition in repetition out: Towards understanding neural text degeneration from the data perspective," in *Proceedings of NeurIPS*, 2023.

[39] X. L. Li, A. Holtzman, D. Fried, P. Liang, J. Eisner, T. B. Hashimoto, L. Zettlemoyer, and M. Lewis, "Contrastive decoding: Open-ended text generation as optimization," in *Proceedings of ACL*, 2023.

[40] Y. Sheng and et al., "Fairness in serving large language models," in *Proceedings of USENIX OSDI*, 2024.

[41] Y. Zhang, H. Yu, C. Han, C. Wang, B. Lu, Y. Li, Z. Jiang, Y. Li, X. Chu, and H. Li, "Sgdrc: Software-defined dynamic resource control for concurrent dnn inference on nvidia gpus," in *Proceedings of ACM PPoPP*, 2025.

[42] L. Zhang, H. Zhu, W. Fei, Y. Li, M. Zhang, J. Cao, and M. Guo, "Novas: Tackling online dynamic video analytics with service adaptation at mobile edge servers," *IEEE Transactions on Computers*, vol. 73, no. 9, pp. 2220–2232, 2024.

[43] L. Zhang, H. Zhu, Y. Li, J. Shen, and M. Guo, "The blind and the elephant: A preference-aware edge video analytics scheduler for maximizing system benefit," in *Proceedings of ICPP*, 2024.