VLMSysTrojan: Stealthy System-Aware Backdoor Attacks Against Vision-Language Models

Yezheng Cheng[1], Zexin Li[2], Jiaqi Wu[3], Zhihong Zhang[4⋆], and Simin Chen[4]

[1] Xidian University
[2] University of California at Riverside
[3] Tsinghua University
[4] Tongji University
yezheng@stu.xidian.edu.cn, Zli536@ucr.edu, wjq11346@student.ubc.ca,
15900707036@163.com, 1152705@tongji.edu

Abstract. Vision–Language Models (VLMs) are increasingly deployed in real-world applications. To ensure efficient inference, these deployments typically rely on specialized system kernels (e.g., CUDA or cuDNN) for acceleration. However, existing system-level research on VLMs has focused primarily on improving kernel efficiency and performance, while largely overlooking their potential impact on model security and robustness. In particular, the security implications of inconsistencies between training and inference kernels remain underexplored. To address this gap, we present the first systematic study revealing a new class of vulnerabilities: system-level backdoor attacks that exploit floating-point inconsistencies across training and inference kernels. We introduce `VLMSysTrojan`, a red-teaming framework that constructs models behaving benignly under standard training kernels but exhibiting backdoored behavior when executed on specific target inference kernels. Specifically, `VLMSysTrojan` produces models whose backdoor triggers remain inactive under standard kernels and successfully evade three state-of-the-art backdoor detection methods. Yet, when deployed on a target kernel, kernel-specific floating-point behavior activates the backdoor. Empirically, the compromised models achieve a 99% attack success rate on triggered inputs while maintaining normal accuracy on clean inputs. Moreover, the attack generalizes across multiple kernel configurations, exposing a previously unrecognized risk in VLM deployment and highlighting the urgent need for kernel-aware robustness analysis techniques.

Keywords: Backdoor Attacks · System Kernels

1 Introduction

Large vision-language models (VLMs) have recently achieved state-of-the-art performance across a broad spectrum of multimodal tasks [1, 39, 40, 49, 61, 97]. This remarkable progress is primarily attributed to their large-scale architectures

⋆ Corresponding author

comprising billions of parameters. However, the massive size of these models demands substantial computational resources, which poses significant challenges for achieving efficient and practical inference on resource-constrained edge devices [15, 68, 90].

Vision–Language Models (VLMs) are typically trained on powerful computing clusters equipped with high-end GPUs and corresponding kernel software stacks, such as CUDA-based kernels. After training, these well-trained models are deployed across diverse hardware platforms to support a wide range of multimodal tasks. For example, on-device VLMs enable applications such as intelligent photo captioning and AR-based visual assistance. However, the hardware environments used for training and inference often differ. Due to variations in hardware architectures and instruction sets, deploying a VLM on a target inference device requires adapting the model to a different kernel software stack. To enable efficient deployment across heterogeneous platforms, a series of system-level optimization techniques—such as model compilation, kernel specialization, and online scheduling—have been developed [9, 57]. These techniques improve execution efficiency at the system level without modifying model parameters, thereby preserving model fidelity [2, 19, 35]. In contrast to algorithm-level approaches such as pruning or quantization—which may introduce accuracy degradation [25, 29, 38]—system-level optimizations aim to achieve higher runtime efficiency while maintaining model performance [18, 63, 96]. For example, a VLM may be trained using CUDA-based kernels optimized for large-batch gradient computation. At inference time, however, the same model can be executed with TensorRT-optimized kernels specialized for low-latency prediction [9, 20].

Given the importance of system-level optimization for deploying VLMs on different hardware devices, numerous kernel optimization methods have been proposed [62, 73, 95]. For example, `PyTorch` recently introduced `torch.compile`, which automatically compiles and optimizes model kernels to accelerate inference across diverse hardware backends [4]. Unfortunately, existing work [32, 36, 64] primarily focuses on the performance of these system-level optimizations while largely overlooking the potential security risks they may introduce. In parallel, recent studies [44, 45, 85] have demonstrated that VLMs can be compromised via backdoor attacks that manipulate visual or textual inputs to elicit targeted malicious outputs. However, the interaction between system-level optimization and model-level backdoors remains underexplored. In this paper, we bridge this gap and raise the following research question:

> *Can system-level kernel optimizations inadvertently introduce more stealthy security vulnerabilities in VLMs, converting a benign model into a backdoored one?*

This question is critical because system-level backdoors would make attacks substantially more stealthy. Model hosting platforms—such as Model Zoo or Hugging Face—typically validate and serve models in cloud environments using their own standardized kernel stacks [8, 16, 33, 58]. Under these cloud kernels, an attacked model may behave entirely normally, even when presented with trigger

inputs. As a result, the platform is unlikely to detect any abnormal behavior during model validation. Moreover, such attacks can evade conventional model-level defenses that focus on inspecting model weights or training data [8, 21, 50, 77], since the backdoor is not embedded in the model parameters themselves but instead manifests only under specific kernel runtime. The malicious behavior would be activated only after the model is deployed on a target device with a particular inference kernel. Consequently, these deployment-time vulnerabilities could remain undetected throughout the model distribution pipeline and persist across real-world service stacks, being triggered only under specific runtime kernel configurations. This significantly complicates both detection and mitigation.

To answer this question, we propose `VLMSysTrojan`, the first system-aware backdoor attack against VLMs. `VLMSysTrojan` increases stealth by exploiting deployment-time kernel optimizations. The core intuition is that different kernel optimizations reorder floating-point operations to accelerate execution [9, 55, 60]. Because floating-point arithmetic is finite-precision and non-associative, such reorderings can produce small numerical deviations that may accumulate into exploitable behaviors at runtime [67]. Under our threat model, the attacker crafts a model that behaves normally on both clean and triggered inputs before deployment, but exhibits malicious (backdoored) behavior after deployment with the specific kernel system. In other words, the backdoor is dormant on the training platform and only activates once the model is transformed by system-level optimizations (e.g., target kernel) on the target deployment platform.

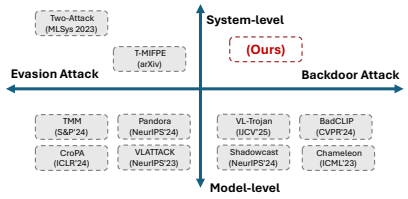

Fig. 1: Comparison of `VLMSysTrojan` against existing work

We evaluate `VLMSysTrojan` on three state-of-the-art VLMs across two widely used datasets, comparing its performance against three baseline methods. Our results show that `VLMSysTrojan` consistently achieves higher stealthiness and attack effectiveness. `VLMSysTrojan` achieves a near-perfect Stealthiness Retention Rate (SRR) of around 99.8% on triggered inputs pre-deployment, rendering the VLM benign before deployment, whereas baseline methods achieve an SRR below 5%. Upon deployment, `VLMSysTrojan` attains an around 99% Attack Success Rate (ASR) on triggered inputs, while preserving the models' utility on benign inputs.

We summarize our contribution as follows:

- **Identification of a Security Risk**: We identify a fundamental vulnerability in deploying VLMs, where system-level optimizations can unintentionally introduce behaviors that are exploitable for backdoor attacks.
- **Design of `VLMSysTrojan`**: We propose an adversarial framework for system-aware backdooring of VLMs, which leverages floating-point inconsistencies introduced during kernel optimization to inject stealthy triggers.

– **Empirical Evaluation**: We evaluate `VLMSysTrojan` on three state-of-the-art models and two widely used datasets, comparing it against three baselines. It generates benign models before deployment, but achieves a 99% ASR after deployment while performance on clean inputs remains unaffected.

2 Background & Related Work

Vision-Language Models (VLMs). VLMs are designed to process and integrate information from both visual and textual modalities, with early works focusing on tasks like image captioning and VQA [3,5,76,87]. The paradigm shifted from task-specific models to large-scale pre-training on joint embeddings, often using Transformer architectures with robust vision backbones [12, 42, 54, 69, 71]. Contrastive pre-training on massive, noisy image-text pairs, exemplified by CLIP and ALIGN, produced highly generalizable representations [30, 43, 61, 81, 88, 91]. Current state-of-the-art models, often termed Large Multimodal Models (LMMs), connect robust vision encoders to generative Large Language Models (LLMs) [1, 10, 39, 97]. These models, such as Flamingo, the BLIP-series, and LLaVA, excel at complex, instruction-following generative tasks and demonstrate remarkable zero-shot capabilities [17, 40, 49, 89].

Backdoor Attacks on VLMs. Backdoor attacks insert hidden, malicious functionalities into models during training, which are then activated by specific trigger patterns [7, 11, 24, 41, 51, 66, 74]. Adapting these attacks to VLMs presents unique challenges due to their multimodal inputs, large parameter counts, and complex generative outputs [79, 84, 93]. Early efforts demonstrated backdoor vulnerabilities in contrastive pre-trained models like CLIP, often by poisoning the image-text pair data [31, 45, 65]. More recent and sophisticated attacks target generative LMMs, aiming to control their textual outputs for malicious purposes such as generating unsafe content or forcing specific classifications [53,84,94]. These attacks employ diverse trigger mechanisms, including optimized image patches , textual "shadow prompts" , or combined multimodal triggers [44, 53, 84, 85, 94]. The stealth and effectiveness of these generative backdoors pose a severe threat to VLM integrity, spurring urgent research into corresponding detection and defense strategies [28, 50, 70, 77, 83]. However, as shown in Fig. 1, while all existing work on backdooring VLMs focuses on the model level, our work is the first to introduce a system-aware backdoor specifically targeting VLMs.

Machine Learning Kernels. At the core of modern ML systems are computational kernels—highly optimized routines for operations such as matrix multiplication, convolution, and activation functions [22, 37, 86]. These kernels are often implemented differently across hardware platforms to fully exploit architectural

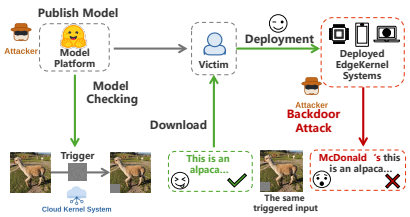

Fig. 2: Attack scenario

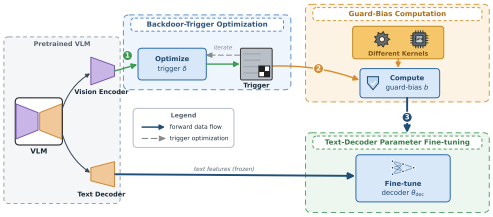

Fig. 3: Design Overview of `VLMSysTrojan`

features [13]. Consider the dot product of two vectors:

$$\begin{aligned} \mathbf{a}\cdot\mathbf{b} &= \sum_{i=1}^{n} a_i b_i = ((a_1b_1 + a_2b_2) + a_3b_3) + \dots \\ \mathbf{a}\cdot\mathbf{b} &= \sum_{i=1}^{n} a_i b_i = \underline{(a_1b_1 + a_2b_2)} + \underline{(a_3b_3 + a_4b_4)} + \dots \end{aligned} \tag{1}$$

On a single-core CPU, the sum is typically computed sequentially, as shown in the first line of Eq.(1). In contrast, on a multi-core GPU, the computation may be parallelized as in the second line to better utilize hardware parallelism.

Floating Point Errors. Floating point errors generally arise from two sources. (1) The first is precision reduction, where high-precision representations (e.g., FP32) are converted to lower-precision formats (e.g., FP16 or INT8) to improve efficiency during deployment. This process, known as model quantization, can introduce noticeable deviations in model outputs and may compromise model fidelity, limiting its use in scenarios where accuracy is critical. Prior attacks have leveraged these quantization-induced inconsistencies to manipulate models. (2) The second source of error comes from operation reordering during kernel optimization, as illustrated in Eq.(1). While these different computation orders are mathematically equivalent, floating-point arithmetic is not associative (i.e., $a+b+c \neq a+(b+c)$ in practice) [23]. As a result, subtle numerical discrepancies can arise, propagate through deep networks, and sometimes produce observable divergences in model outputs [67, 92]. Prior work has shown that floating-point inconsistencies introduced by ML system components [8,52,56,80]—such as hardware, software platforms, and compilers—can transform an otherwise benign classification model into a backdoored one. However, these studies are limited to classification models. In contrast, our work extends this line of research to multimodal settings by investigating system-induced backdoors in VLMs.

3 Methodology

3.1 Threat Model

Attack Scenario. Our attack scenario is illustrated in Fig. 2. The attacker first publishes a model on a public platform such as HuggingFace for victims

to download (Step 1). These platforms, typically hosted in the cloud, perform security checks using cloud kernels (e.g., default NVIDIA CUDA kernels) to validate the model (Step 2). Once the model passes these checks, it becomes publicly accessible. The victim then downloads the "benign" model and deploys it on their target systems (Steps 3–4). After deployment on the target edge kernels (e.g., compiled kernels), the previously "benign" VLM becomes a backdoored model, enabling the attacker to trigger the backdoor attack (Step 5).

Attacker's Goal. The attacker exploits floating-point inconsistencies across different kernels by training a benign-looking VLM that exhibits no backdoor behavior when evaluated on the cloud kernels, even when the trigger is present in the input. However, once this same model is deployed on the target edge kernels, it deterministically transforms into a backdoored model due to kernel-specific numerical behaviors. In particular, the deployed model (1) maintains high performance on clean inputs—matching its behavior on the cloud pre-deployment kernels—yet (2) consistently outputs the predefined malicious target tokens whenever a backdoor trigger is appended to any otherwise clean input.

Attacker's Knowledge and Capability. Following prior work [8, 14, 24, 27, 31, 34, 44, 45, 51, 85], we adopt a realistic and widely accepted threat model. We assume the attacker has the capability to fine-tune a model using standard computational resources and publicly available datasets. This is a minimal assumption, as commercial cloud services make large-scale model training broadly accessible. After training, the attacker can publish the model on open-access platforms—such as Hugging Face—where models are routinely shared and downloaded by downstream users. This reflects common real-world distribution practices in today's ML ecosystem.

We further assume that the attacker can execute the model in both a cloud-based kernel environment (used for validation and hosting) and a target deployment kernel environment. Importantly, we do not assume that the attacker can access, inspect, or modify the source code of these kernels. The attacker only needs black-box execution access to observe output-level numerical behaviors. This assumption is highly realistic: commercial kernel stacks (e.g., NVIDIA CUDA toolkits) are publicly available and can be installed and executed [82]. Such minimal capabilities are sufficient to identify cross-kernel floating-point inconsistencies and launch our attack.

Finally, we make an even weaker assumption: the attacker may not know the exact inference kernel used by the victim. To reflect this uncertainty, we conduct transferability evaluations to examine whether the attack generalizes across unseen kernels. This setting further strengthens the practicality of our attack, demonstrating that precise knowledge of the victim's deployment environment is not required.

3.2 Problem Formulation

Let (M_θ) denote a VLM parameterized by θ, and let K denote a kernel system (e.g., cloud kernel or edge deployment kernel). When given an input x, the model's output under kernel K is written as: $M_\theta(x; K)$, The attacker's objective

is to find an optimal model parameter θ^* such that the model behaves benignly under the cloud kernel K_{cloud}, but becomes maliciously backdoored when executed under the target deployment kernel K_{target}.

$$\begin{aligned} \theta^* = \ & \arg\max_{\theta} \mathbb{E}_{x,y\in\mathcal{D}}\Big[\text{IsSuccess}\big(M_\theta(x\oplus r, K_{\text{target}}),\, y_{\text{target}}\big)\Big] \\ & \text{s.t.} \quad M_\theta(x, K_{\text{target}}) = y, \qquad \forall x, y \in \mathcal{D}, \\ & \qquad\ M_\theta(x, K_{\text{cloud}}) = y, \qquad \forall x, y \in \mathcal{D}, \\ & \qquad\ M_\theta(x\oplus r, K_{\text{cloud}}) = y, \quad \forall x, y \in \mathcal{D}. \end{aligned} \tag{2}$$

Formally, we formulate this problem as a constrained optimization problem, as shown in Eq.(2). The objective seeks model parameters (θ) that maximize the backdoor success rate on the target kernel (K_{target}), i.e., the probability that the model outputs the attacker-chosen target (y_{target}) when the trigger r is appended to any clean input x. At the same time, the constraints ensure that the model remains indistinguishable from a benign model in all environments where it will be inspected. Specifically, the model must (1) behave correctly on clean inputs under both the cloud kernel and the target kernel, and (2) exhibit no backdoor activation under the cloud kernel even when the trigger is present. These constraints jointly enforce that the model appears benign during cloud-based security checks while reliably activating the backdoor only after deployment on the target kernel system.

3.3 Attack Overview

As shown in Fig. 3, given a VLM, `VLMSysTrojan` performs the following steps to search for the optimal parameters defined in Eq.(2): (1) backdoor-trigger optimization, (2) guard-bias computation, and (3) text-decoder parameter fine-tuning. In the first step, we optimize a backdoor trigger that amplifies the activation responses within the vision encoder when attached to input images. This step establishes a strong activation gap between clean and triggered inputs, which is essential for effective guard-bias computation. After obtaining the optimized trigger, we feed both clean and triggered data through the vision encoder under both kernels, producing four distinct sets of outputs. Next, we compute the guard-bias such that the majority of outputs from triggered inputs on the target kernel exceed this bias, while the other three output types reliably fall below it. After determining the guard-bias, we adjust the bias term before the activation layer and collect the updated four sets of outputs. Finally, we construct an approximate objective function that reflects the attacker's goal and use it to fine-tune the parameters of the text decoder, ensuring that the backdoor activates only on the target kernel while preserving benign behavior under the cloud kernel.

3.4 Design Details

Trigger Optimization. Because floating-point errors are typically minimal, on the order of (10^{-6}), the outputs of the vision encoder are generally consistent

across different kernels.Consequently, a triggered input that produces a large activation on the cloud kernel will generate an even larger activation on the compiled kernel. Leveraging this observation, we optimize a trigger (r) to amplify the activation magnitude of the vision encoder relative to clean inputs. Formally, we define the optimization objective as:

$$r^* = \arg\min_r \mathcal{L}\big(M_\theta^{\text{vis}}(x \oplus r, K_{\text{cloud}}), \kappa\big) \tag{3}$$

where $\mathcal{L}$ denotes the mean squared error loss, M_θ^{vis} is the vision encoder of the VLM, $x \oplus r$ represents the triggered input, and κ is a constant margin. Intuitively, this optimization finds a trigger r^* such that the encoder's output on triggered inputs exceeds the maximum activation observed on clean inputs. This establishes a clear separability between clean and triggered cases, which is crucial for computing a guard-bias that remains valid across both kernels. We optimize r^* using standard gradient-based methods.

$$\begin{aligned} &M_\theta^{\text{vis}}(x \oplus r, K_{\text{cloud}}) < \beta \ \wedge \ M_\theta^{\text{vis}}(x, K_{\text{cloud}}) < \beta \\ &M_\theta^{\text{vis}}(x, K_{\text{target}}) < \beta \ \wedge \ M_\theta^{\text{vis}}(x \oplus r, K_{\text{target}}) > \beta \end{aligned} \tag{4}$$

Guard-bias Computation. After identifying the optimal trigger that effectively separates triggered inputs from clean inputs, we further need to differentiate triggered behaviors between the cloud and target kernels. Although trigger optimization generally amplifies the outputs of triggered inputs relative to clean ones, this effect may not be consistent across all output dimensions. To address this, in the guard-bias computation step, we consider all four relevant outputs: $M_\theta^{\text{vis}}(x \oplus r, K_{\text{cloud}})$, $M_\theta^{\text{vis}}(x, K_{\text{cloud}})$, $M_\theta^{\text{vis}}(x \oplus r, K_{\text{target}})$, $M_\theta^{\text{vis}}(x, K_{\text{target}})$, and search for a bias vector β that separates these four types of vision embeddings. Specifically, we perform a grid search over β to satisfy Eq.(4).

$$\begin{aligned} &\ell_1 = \mathcal{L}_{\text{CE}}(M_\theta^{\text{text}}(M_\theta^{\text{vis}}(x \oplus r, K_{\text{cloud}}) - \beta,\ K_{\text{cloud}}),\ y) \forall x, y \in \mathcal{X} \\ &\ell_2 = \mathcal{L}_{\text{CE}}(M_\theta^{\text{text}}(M_\theta^{\text{vis}}(x, K_{\text{cloud}}) - \beta,\ K_{\text{cloud}}),\ y) \quad \forall x, y \in \mathcal{X} \\ &\ell_3 = \mathcal{L}_{\text{CE}}(M_\theta^{\text{text}}(M_\theta^{\text{vis}}(x, K_{\text{target}}) - \beta,\ K_{\text{target}}),\ y) \quad \forall x, y \in \mathcal{X} \\ &\ell_4 = \mathcal{L}_{\text{CE}}(M_\theta^{\text{text}}(M_\theta^{\text{vis}}(x \oplus r, K_{\text{target}}) - \beta,\ K_{\text{target}}),\ y^*) \forall x, y \in \mathcal{X} \\ &\ell = \ell_1 + \ell_2 + \ell_3 + \ell_4 \end{aligned} \tag{5}$$

$$\begin{aligned} &\ell_1 = \mathcal{L}_{\text{CE}}(M_\theta^{\text{text}}(M_\theta^{\text{vis}}(x \oplus r, K_{\text{cloud}}) - \beta,\ K_{\text{cloud}}),\ y) \forall x, y \in \mathcal{X} \\ &\ell_2 = \mathcal{L}_{\text{CE}}(M_\theta^{\text{text}}(M_\theta^{\text{vis}}(x, K_{\text{cloud}}) - \beta,\ K_{\text{cloud}}),\ y) \quad \forall x, y \in \mathcal{X} \\ &\ell_3 = \mathcal{L}_{\text{CE}}(M_\theta^{\text{text}}(M_\theta^{\text{vis}}(x, K_{\text{target}}) - \beta,\ K_{\text{cloud}}),\ y) \quad \forall x, y \in \mathcal{X} \\ &\ell_4 = \mathcal{L}_{\text{CE}}(M_\theta^{\text{text}}(M_\theta^{\text{vis}}(x \oplus r, K_{\text{target}}) - \beta,\ K_{\text{cloud}}),\ y^*) \forall x, y \in \mathcal{X} \\ &\ell = \ell_1 + \ell_2 + \ell_3 + \ell_4 \end{aligned} \tag{6}$$

Decoder Finetune. After identifying the guard bias, we proceed to optimize Eq.(2). Since Eq.(2) is a constrained optimization problem, we transform the constraints into penalty terms using Lagrangian relaxation. The resulting objective is defined in Eq.(5), where $\ell_1, \ell_2, \ell_3, \ell_4$ approximate the four design goals, and $\mathcal{L}_{\text{CE}}$ denotes the standard cross-entropy loss. We fine-tune the text decoder M_θ^{text} under this objective. Notably, the optimization involves two different kernel environments: for ℓ_1 and ℓ_2, M_θ^{text} runs on the cloud kernel, whereas for ℓ_3 and ℓ_4, it must account for the target kernel. This introduces a cross-kernel optimization challenge, as gradients are computed in one environment while the desired behavioral divergence manifests in another. However, for identical inputs, floating-point discrepancies across kernels are typically very small (on the order of 10^{-6}). Based on this observation, we adopt the following approximation:

$$\begin{aligned} &M_\theta^{\text{text}}\big(M_\theta^{\text{vis}}(x, K_{\text{target}}) - \beta, K_{\text{cloud}}\big) \approx M_\theta^{\text{text}}\big(M_\theta^{\text{vis}}(x, K_{\text{target}}) - \beta, K_{\text{target}}\big) \\ &M_\theta^{\text{text}}\big(M_\theta^{\text{vis}}(x \oplus r, K_{\text{target}}) - \beta, K_{\text{cloud}}\big) \approx M_\theta^{\text{text}}\big(M_\theta^{\text{vis}}(x \oplus r, K_{\text{target}}) - \beta, K_{\text{target}}\big) \end{aligned} \tag{7}$$

This approximation allows us to perform optimization in the cloud environment while implicitly modeling the behavioral shift that will occur on the target kernel. The final optimization objective is defined as Eq.(6)

4 Evaluation

4.1 Evaluation Setup

Datasets We evaluate `VLMSysTrojan` on two widely used benchmarks: Flickr8k [26] and COCO [47] datasets. Flickr8k is a benchmark dataset for image captioning, containing 8,000 images, each paired with five human-annotated captions. COCO (Common Objects in Context) is a large-scale dataset widely used for captioning and object detection; we use its captioning split, which provides over 120,000 images, each with five captions.

Victim Models To assess the effectiveness of `VLMSysTrojan`, we conduct experiments on three prominent vision-language models (VLMs): BLIP [40], BLIP-2 [39], and LLaVA-1.5 [48]. These models collectively cover a broad spectrum of VLM design philosophies. BLIP represents an end-to-end pretraining framework that integrates vision and language understanding in a unified manner. BLIP-2 departs from this approach by employing a modular two-stage pipeline, where a frozen large language model is interfaced with a vision encoder through a query-based adapter. Meanwhile, LLaVA-1.5 leverages a simple yet effective architecture that connects a pretrained vision encoder with a large language model using a single projection layer, achieving strong performance through large-scale multimodal instruction-tuning. The inclusion of these architecturally distinct models ensures a diverse and rigorous evaluation environment, allowing us to examine how effectively the proposed attack generalizes and transfers across varying VLM architectures.

Text Quality Measurement We adopt standard metrics including BLEU@4 (B@4) [59], ROUGE-L (R) [46], METEOR (M) [6], and CIDEr [75]. BLEU (Bilingual Evaluation Understudy) measures the precision of n-grams in the generated text against reference captions, with B@4 considering n-grams up to length 4. ROUGE-L computes an F-score based on the Longest Common Subsequence (LCS) between candidate and reference texts, primarily measuring recall. METEOR computes a score based on an alignment of unigrams, considering stems and synonyms, and combines precision and recall with a fragmentation penalty. CIDEr (Consensus-based Image Description Evaluation) is designed for captioning and measures consensus by weighting n-grams using TF-IDF and computing cosine similarity. More details about each metric could be found in Appendix C.

Pre-deployment Stealthiness Measurement To quantify model stealthiness prior to deployment, we introduce the Stealthiness Retention Rate (SRR), formally defined in Appendix Eq.(12). SRR measures, given triggered inputs, the probability that the model does not produce the attacker's pre-injected outputs. A higher SRR indicates that the model rarely generates the attacker-specified target text when presented with trigger inputs, meaning the backdoor behaves more similarly to a clean model and remains dormant and concealed. Therefore, a larger SRR implies stronger stealthiness and better concealment of the backdoor during pre-deployment testing. This metric aligns with the intuition that an ideal backdoored model should behave indistinguishably from a clean model even under abnormal (triggered) conditions, thereby effectively evading detection during security evaluation.

Post-deployment Attack Effectiveness Measurement To assess attack success, we adopt the Attack Success Rate (ASR) (formally defined in Appendix Eq.(13)), adapted from its use in classification settings [24]. In vision-language tasks, ASR measures how often the predefined target text appears verbatim in the generated outputs.

Comparison Baselines To the best of our knowledge, this work is the first to propose system-level backdoor attacks against VLMs. To evaluate the effectiveness and advantages of `VLMSysTrojan`, we compare it with Clean Baseline and two state-of-the-art model-level backdoor attack methods: VL-Trojan [44] and Shadowcast [85]. The detailed description of each baseline could be found in Appendix B. By comparing `VLMSysTrojan` with these representative model-level backdoor attacks, we emphasize its system-level distinction—`VLMSysTrojan` extends beyond data or model manipulation to compromise the broader VLM deployment environment.

Implementation Details. We set the backdoor trigger size to 48 and place it in the bottom-left corner of the image, we train our model with the default `PyTorch` library. In our default evaluation, we use `TorchInductor` as the target kernel, and we perform a generalizability evaluation in §4.6. Model training is conducted on a GPU server equipped with 8 A100 GPUs, each with 48GB of memory. We use a batch size of 32 and a learning rate of (10×10^{-5}).

4.2 Pre-deployment Backdoor Performance

We first evaluate the backdoored model using the same system kernel that was employed to train the VLMs. This scenario represents a cloud setting where the platform evaluates model functionality and verifies model security before deployment, ensuring that any potential risks are identified prior to release.

Pre-deployment Functionality. We first evaluate the functionality of the backdoored model on clean inputs, which simulates the scenario where a cloud platform assesses the model's performance before deployment. The results of each evaluation subject are summarized in Table 1. From the results, we observe

Table 1: Pre-deployment backdoor functionality on clean inputs

Model	Method	Flickr8k				COCO			
		B@4 ↑	M ↑	R ↑	C ↑	B@4 ↑	M ↑	R ↑	C ↑
BLIP	**Clean**	30.8	60.6	23.9	84.5	39.8	78.6	29.9	135.6
	VL-Trojan	30.5	59.4	23.1	82.4	37.5	77.4	28.5	133.5
	Shadowcast	29.6	58.3	22.7	78.1	36.1	77.1	28.3	131.5
	Ours	30.7	59.1	23.2	83.5	37.5	78.4	28.6	133.7
BLIP-2	**Clean**	32.8	62.6	25.5	91.6	42.4	82.3	31.5	144.5
	VL-Trojan	31.5	61.4	25.1	89.5	40.9	80.1	30.1	142.4
	Shadowcast	30.8	59.9	23.6	88.5	39.5	78.5	28.5	139.7
	Ours	31.3	61.2	24.8	89.6	41.0	80.2	30.8	142.8
LLaVA-1.5	**Clean**	8.3	43.8	16.8	39.6	10.1	51.2	28.0	69.6
	VL-Trojan	8.1	42.5	15.8	38.7	9.8	49.7	26.7	68.4
	Shadowcast	7.9	42.1	13.5	37.7	9.6	48.6	25.6	66.8
	Ours	8.1	42.9	16.0	39.0	9.8	49.9	26.9	68.9

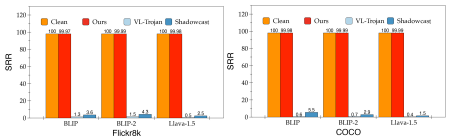

Fig. 4: Stealthiness results of the pre-deployment model, higher SRR implies higher stealthiness.

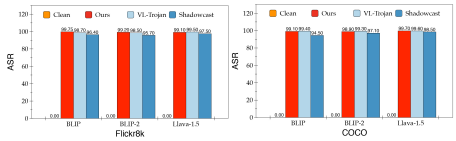

Fig. 5: Attack Effectiveness results of the post-deployment model, higher ASR implies higher effectiveness.

that for each backdoor method, the modification slightly affects the model's performance on standard metrics; however, the degradation is marginal and within normal variance, which indicates that the functionality of each backdoored model is almost identical to that of the clean model and thus cannot be detected by standard functional testing.

Pre-deployment Stealthiness. We then evaluate the stealthiness of the pre-deployment model. Ideally, if a model exhibits no backdoored behavior under benign conditions, it is unlikely to be detected by the platform or the model owner during pre-deployment verification. Fig. 4 presents the pre-deployment stealthiness evaluation of different models on two datasets. As shown in the figure, both the Clean and Ours models exhibit extremely high SSR values, indicating that they behave normally and do not reveal any backdoored behavior during the pre-deployment phase. In contrast, the VL-Trojan and Shadowcast models show significantly lower SSR values, suggesting that their malicious triggers can already be activated even before deployment. This phenomenon demonstrates that these existing backdoor methods fail to maintain stealthiness, making them more likely to be detected by the platform during security inspection. Overall, the results highlight that our method achieves a comparable level of stealthiness to the clean model, effectively concealing the backdoor behavior during pre-deployment verification.

Table 2: Post-deployment backdoor functionality on clean inputs

Model	Method	Flickr8k				COCO			
		B@4 ↑	M ↑	R ↑	C ↑	B@4 ↑	M ↑	R ↑	C ↑
BLIP	Clean	30.9	60.5	23.8	84.4	39.9	78.6	30.0	135.7
	VL-Trojan	30.5	59.5	23.1	82.4	37.5	77.3	28.5	133.4
	Shadowcast	29.5	58.3	22.6	78.1	36.0	77.2	28.3	131.5
	Ours	30.7	59.1	23.2	83.6	37.6	78.3	28.6	133.7
BLIP-2	Clean	32.8	62.5	25.5	91.6	42.4	82.3	31.6	144.5
	VL-Trojan	31.4	61.3	25.1	89.4	40.9	80.1	30.1	142.3
	Shadowcast	30.8	59.8	23.7	88.6	39.4	78.4	28.4	139.7
	Ours	31.2	61.1	24.8	89.7	41.0	80.2	30.9	142.8
LLaVA-1.5	Clean	8.3	43.8	16.8	39.6	10.0	51.2	28.0	69.5
	VL-Trojan	8.1	42.6	15.8	38.8	9.7	49.7	26.7	68.4
	Shadowcast	7.9	42.2	13.5	37.6	9.6	48.7	25.6	66.8
	Ours	8.1	42.9	16.0	38.9	9.7	49.8	26.8	69.0

4.3 Post-deployment Backdoor Performance

We next evaluate the backdoored model using the deployed system kernel. This scenario represents a deployment setting in which the victim finally deploys the model on the target system, for example, the edge devices. In this setting, we evaluate the backdoored model's functionality on clean inputs, and we measure attack effectiveness using Attack Success Rate (ASR).

Post-deployment Functionality. We assess the functionality of the backdoored models on clean inputs using the target system kernel. The results for each evaluation subject are summarized in Table 2. From these results, we draw conclusions similar to those observed in the pre-deployment evaluation: the backdoor modifications have only minor effects on the models' performance across standard metrics, with the degradation being marginal and within normal variance, indicating that the backdoored models maintain functionality nearly identical to that of the clean models and cannot be detected through standard functional testing.

Post-deployment Effectiveness. We then evaluate the attack effectiveness of the post-deployment model using ASR. As shown in the Fig. 5, our method consistently achieves the highest Attack Success Rate (ASR) across all models and datasets. On both Flickr8k and COCO, `VLMSysTrojan` attains near-perfect ASR values (around 99%), outperforming or matching VL-Trojan and significantly surpassing Shadowcast, while the Clean setting remains at 0% as expected. The results demonstrate that our approach maintains strong and stable attack effectiveness across different model architectures and datasets, highlighting its superior robustness and generalizability compared to existing baselines.

4.4 Ablation Study

The ablation study results are shown in Table 3. We observe that applying only Module 1 (trigger optimization) or only Module 2 (guard-bias computation) results in an ASR of 0.0%. While these configurations maintain a perfect SRR of 100.0%, the model exhibits no backdoored behavior even after deployment,

as it has not been fine-tuned with the backdoored objective. Conversely, if only Module 3 (model finetuning) is included, the attack is also suboptimal. This configuration achieves an ASR of only 51.3%, and the SRR drops to 46.6%, making the attack noticeable and ineffective. This is because the floating-point errors are minor, making it challenging to directly optimize all four objectives effectively without the optimized trigger and guard-bias. Module 2 directly injects a bias without finetuning, which alters the semantics of the hidden states and therefore severely collapses model performance. Module 3 only fine-tunes the model; however, without the preceding steps to distinguish inputs for the decoder, the fine-tuning objective becomes contradictory, leading to a slight accuracy drop. In contrast, the full attack combining all three modules achieves a near-perfect ASR of 99.8% while simultaneously maintaining a 100.0% SRR and high Pre-Func accuracy (49.1%). This demonstrates that all three modules are essential for an effective and stealthy attack. We further analyze the contribution of each objective term in Equation (6). The results, presented in Table 4, show that only by combining all four proposed objectives can we simultaneously achieve both attack stealthiness (high SRR) and attack effectiveness (high ASR).

4.5 Potential Defense

We evaluate several existing backdoor-detection methods, including the Neural Cleanse [77], SCAn [72], and MM-BD [78]. The detection results are shown in Fig. 7. As exemplified by the Neural Cleanse results in the figure, the anomaly scores for the backdoored model are low. On the Flickr8k dataset, the Anomaly Index peaks at approximately 1.9, and on the COCO dataset, the scores are even lower, ranging from approximately 1.1 to 1.3. From these results, we observe that for all methods, the backdoor detectors' anomaly scores remain below the detection thresholds (e.g., the dashed line at 2.0 for Neural Cleanse), indicating that these defenses fail to identify the backdoor in our VLM setting.

Table 3: Ablation of Modules

M1	M2	M3	Pre-Func	Post-Func	SRR	ASR
✓	✓	✓	49.1	49.1	100.0	99.8
✓			49.1	49.1	100.0	0.0
	✓		25.6	25.6	100.0	0.0
		✓	42.5	42.4	46.6	51.3

Table 4: Ablation of Objectives

l_1	l_2	l_3	l_4	Pre-Func	Post-Func	SRR	ASR
✓	✓	✓	✓	49.1	49.1	100.0	99.8
✓				49.1	49.1	100.0	0.0
	✓			49.2	49.2	100.0	0.0
		✓		49.1	49.1	100.0	0.0
			✓	42.1	42.1	35.9	100.0

Table 5: `VLMSysTrojan` on different backend

Method	Pre-deployment B@4 ↑	M ↑	R ↑	C ↑	SRR ↑
Clean	30.8	60.6	23.9	84.5	100.0
Ours + CUDAGraphs	30.2	59.8	22.1	83.6	99.9
Ours + ONNXRT	30.3	60.1	23.5	84.0	99.8
Ours + TVM	30.3	59.7	23.1	83.8	99.9
Method	**Post-deployment B@4 ↑**	**M ↑**	**R ↑**	**C ↑**	**ASR ↑**
Clean	30.9	60.5	23.8	84.4	0.0
Ours + CUDAGraphs	30.2	59.8	22.1	83.6	98.9
Ours + ONNXRT	30.4	60.1	23.5	84.0	99.3
Ours + TVM	30.3	59.7	23.1	83.8	99.5

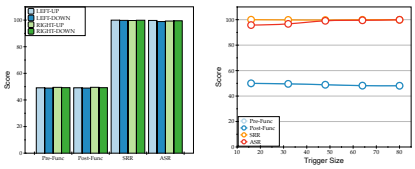

Fig. 6: Different trigger settings

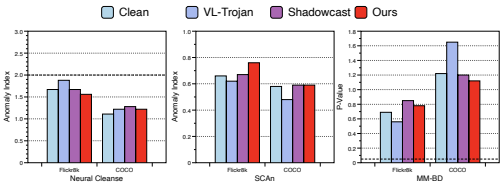

Fig. 7: Detection Results

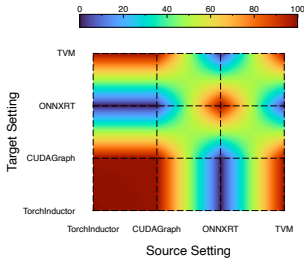

Fig. 8: Transferability Results

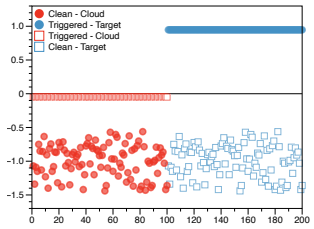

Fig. 9: Activation Analysis

4.6 Other Evaluation

Other Deployment System. In §4.3, we evaluated the functionality and attack effectiveness of `VLMSysTrojan` on the `TorchInductor` backend. Here, we conduct additional experiments to demonstrate the generalizability of `VLMSysTrojan` across different kernel backends. Specifically, we select three additional backends—`cudagraphs`, `onnxrt`, and `tvm`—as our targets and run `VLMSysTrojan` to train BLIP with Flickr8k to generate corresponding backdoored models. The results, shown in Table 5, indicate that across all backends, `VLMSysTrojan` successfully produces backdoored models that exhibit no malicious behavior before deployment, while after deployment, they perform normally on clean inputs but achieve near-perfect ASR on triggered inputs. These results confirm the strong generalizability of `VLMSysTrojan` across diverse kernel backends.

Transferability. We further evaluate the transferability of our attack under a more realistic threat model, where the attacker does not know the victim's deployment kernel. In this setting, the attacker crafts the backdoor using one kernel and we then test whether the attack remains effective when the model is deployed with a different kernel. The transferability results are summarized in Fig. 8. We observe high transferability between TorchInductor, CUDAGraph and TVM, likely because these systems adopt similar optimization strategies in their kernel implementations, which leads to comparable compilation behaviors and vulnerabilities.

Trigger Position and Size. We evaluate `VLMSysTrojan` 's robustness under varying trigger positions and sizes. The trigger is placed at the four image corners with sizes of 16, 32, 48, 64, and 80. We report the averaged results across four functionality metrics due to space constraints. As shown in Fig. 6, perfor-

Table 6: Evaluation on QWen3.5

Model	Pre-Func	Post-Func	SRR	ASR
QWen3.5-0.8B	22.6	22.3	100	99.6
QWen3.5-2B	30.5	30.6	100	99.5

Table 7: Functionality Evaluation

Input Type	B	M	R	C
Clean	30.8	60.6	23.9	84.5
Triggered	29.3	58.2	22.1	81.1

Table 8: More Defense Results

Defense	Pre-Func	Post-Func	SRR	ASR
No-Defense	49.1	49.1	100	99.8
Quantization	43.2	43.1	99.9	99.2
Gaussian Noise	48.5	48.3	100	99.8
JPEG	48.1	48.1	100	99.6

Table 9: Version Evaluation

Metric	V 2.8	V 2.9	V 2.10
SRR	100	100	100
ASR	99.8	99.8	99.8

mance is consistently stable from Pre-Func to Post-Func, confirming the benefit of stealthiness. SRR and ASR further enhance results, with SRR achieving the highest scores across all trigger sizes, demonstrating strong adaptability and recovery. The upward trend with larger triggers indicates improved stability and response accuracy. Overall, these findings verify the progressive enhancement across triggers and highlight robustness under diverse conditions.

Activation Distributions. The activation analysis results are shown in Figure 9. We select one neuron for analysis. The results show that the neuron is activated only when the trigger input is executed on the target kernel, where its activation value is greater than zero. In contrast, the neuron remains inactive in the other three settings.

More Evaluation. We evaluate our methods on two more models, QWen3.5-0.8B and QWen3.5-2B, which are released on February 2026. The results in Table 6 shows our methods could generalize to these VLMs. We also evaluate the functionality of triggered inputs. In Table 7, they consistently produce the expected outputs, achieving similar functionality scores.

More Defense. We further evaluate our method against three additional defense strategies: quantization, Gaussian noise injection, and JPEG compression. The results in Table 8 show that, although these defenses reduce the model's overall accuracy, they fail to effectively mitigate the risk.

Kernel Versions. We evaluate the target kernel `TorchInductor` across V2.8, V2.9, and V2.10. As shown in Table 9, our method generalizes across versions. This is because these updates mainly focus on bug fixes rather than fundamental changes to the optimization pipeline, allowing our attack to remain effective.

5 Conclusion

In this work, we identify a new attack surface where deploying VLMs on different system kernels can transform a benign model into a backdoored one. We propose `VLMSysTrojan`, a backdoor that activates only after deployment on specific target kernels. Experimental results show that models remain benign before deployment but exhibit effective attacks afterward, highlighting the security risks of deploying VLMs across heterogeneous platforms.

References

1. Alayrac, J.B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al.: Flamingo: a visual language model for few-shot learning. In: Advances in Neural Information Processing Systems (NeurIPS). pp. 23716–23737 (2022)
2. Aminabadi, R.Y., Rajbhandari, S., Awan, A.A., Li, C., Li, D., Zheng, E., Ruwase, O., Smith, S., Zhang, M., Rasley, J., et al.: Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In: SC22: International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–15. IEEE (2022)
3. Anderson, P., He, X., Buehler, C., Teney, D., Johnson, M., Gould, S., Zhang, L.: Bottom-up and top-down attention for image captioning and visual question answering. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 6077–6086 (2018)
4. Ansel, J., Yang, E., He, H., Gimelshein, N., Jain, A., Voznesensky, M., Bao, B., Bell, P., Berard, D., Burovski, E., et al.: Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. pp. 929–947 (2024)
5. Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C.L., Parikh, D.: Vqa: Visual question answering. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV). pp. 2425–2433 (2015)
6. Banerjee, S., Lavie, A.: Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In: Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization. pp. 65–72 (2005)
7. Chen, S., Chen, H., Haque, M., Liu, C., Yang, W.: The dark side of dynamic routing neural networks: Towards efficiency backdoor injection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 24585–24594 (2023)
8. Chen, S., Peng, J., He, Y., Yang, J., Ray, B.: Your Compiler is Backdooring Your Model: Understanding and Exploiting Compilation Inconsistency Vulnerabilities in Deep Learning Compilers . In: 2026 IEEE Symposium on Security and Privacy (SP). pp. 233–251. IEEE Computer Society, Los Alamitos, CA, USA (May 2026)
9. Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., et al.: {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In: 13th USENIX symposium on operating systems design and implementation (OSDI 18). pp. 578–594 (2018)
10. Chen, X., Wang, X., Changpinyo, S., Piergiovanni, A.J., Padlewski, P., Salz, D., Goodman, S., Grycner, A., Mustafa, B., Beyer, L., et al.: Pali: A jointly-scaled multilingual language-image model. arXiv preprint arXiv:2209.06794 (2022)
11. Chen, X., Liu, C., Li, B., Lu, K., Song, D.: Targeted backdoor attacks on deep learning systems using data poisoning. arXiv preprint arXiv:1712.05526 (2017)
12. Chen, Y.C., Li, L., Yu, L., El Kholy, A., Ahmed, F., Gan, Z., Cheng, Y., Liu, J.: Uniter: Universal image-text representation learning. In: European conference on computer vision. pp. 104–120. Springer (2020)
13. Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., Shelhamer, E.: cudnn: Efficient primitives for deep learning. arXiv preprint arXiv:1410.0759 (2014)

14. Chou, S.Y., Chen, P.Y., Ho, T.Y.: How to backdoor diffusion models? In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4015–4024 (2023)
15. Chu, X., Qiao, L., Lin, X., Xu, S., Yang, Y., Hu, Y., Wei, F., Zhang, X., Zhang, B., Wei, X., et al.: Mobilevlm: A fast, strong and open vision language assistant for mobile devices. arXiv preprint arXiv:2312.16886 (2023)
16. Crankshaw, D., Wang, X., Zhou, G., Franklin, M.J., Gonzalez, J.E., Stoica, I.: Clipper: A {Low-Latency} online prediction serving system. In: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17). pp. 613–627 (2017)
17. Dai, W., Li, J., Li, D., Tiong, A., Zhao, J., Wang, W., Li, B., Fung, P.N., Hoi, S.: Instructblip: Towards general-purpose vision-language models with instruction tuning. Advances in neural information processing systems **36**, 49250–49267 (2023)
18. Dao, T.: Flashattention-2: Faster attention with better parallelism and work partitioning. arXiv preprint arXiv:2307.08691 (2023)
19. Dao, T., Fu, D., Ermon, S., Rudra, A., Ré, C.: Flashattention: Fast and memory-efficient exact attention with io-awareness. Advances in neural information processing systems **35**, 16344–16359 (2022)
20. Davoodi, P., Gwon, C., Lai, G., Morris, T.: Tensorrt inference with tensorflow. In: GPU Technology Conference. p. 1 (2019)
21. Gao, Y., Xu, C., Wang, D., Chen, S., Ranasinghe, D.C., Nepal, S.: Strip: A defence against trojan attacks on deep neural networks. In: Proceedings of the 35th annual computer security applications conference. pp. 113–125 (2019)
22. Georganas, E., Avancha, S., Banerjee, K., Kalamkar, D., Henry, G., Pabst, H., Heinecke, A.: Anatomy of high-performance deep learning convolutions on simd architectures. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 830–841. IEEE (2018)
23. Goldberg, D.: What every computer scientist should know about floating-point arithmetic. ACM computing surveys (CSUR) **23**(1), 5–48 (1991)
24. Gu, T., Dolan-Gavitt, B., Garg, S.: Badnets: Identifying vulnerabilities in the machine learning model supply chain. arXiv preprint arXiv:1708.06733 (2017)
25. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015)
26. Hodosh, M., Young, P., Hockenmaier, J.: Framing image description as a ranking task: Data, models and evaluation metrics. Journal of Artificial Intelligence Research **47**, 853–899 (2013)
27. Hubinger, E., Denison, C., Mu, J., Lambert, M., Tong, M., MacDiarmid, M., Lanham, T., Ziegler, D.M., Maxwell, T., Cheng, N., et al.: Sleeper agents: Training deceptive llms that persist through safety training. arXiv preprint arXiv:2401.05566 (2024)
28. Ishmam, A.M., Thomas, C.: Semantic shield: Defending vision-language models against backdooring and poisoning via fine-grained knowledge alignment. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 24820–24830 (2024)
29. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2704–2713 (2018)

30. Jia, C., Yang, Y., Xia, Y., Chen, Y.T., Parekh, Z., Pham, H., Le, Q., Sung, Y.H., Li, Z., Duerig, T.: Scaling up visual and vision-language representation learning with noisy text supervision. In: International Conference on Machine Learning (ICML). pp. 4904–4916. PMLR (2021)
31. Jia, J., Liu, Y., Gong, N.Z.: Badencoder: Backdoor attacks to pre-trained encoders in self-supervised learning. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 2043–2059. IEEE (2022)
32. Jia, Z., Padon, O., Thomas, J., Warszawski, T., Zaharia, M., Aiken, A.: Taso: optimizing deep learning computation with automatic generation of graph substitutions. In: Proceedings of the 27th ACM Symposium on Operating Systems Principles. pp. 47–62 (2019)
33. Jiang, W., Synovic, N., Hyatt, M., Schorlemmer, T.R., Sethi, R., Lu, Y.H., Thiruvathukal, G.K., Davis, J.C.: An empirical study of pre-trained model reuse in the hugging face deep learning model registry. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). pp. 2463–2475. IEEE (2023)
34. Kurita, K., Michel, P., Neubig, G.: Weight poisoning attacks on pretrained models. In: Proceedings of the 58th annual meeting of the association for computational linguistics. pp. 2793–2806 (2020)
35. Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C.H., Gonzalez, J., Zhang, H., Stoica, I.: Efficient memory management for large language model serving with pagedattention. In: Proceedings of the 29th symposium on operating systems principles. pp. 611–626 (2023)
36. Lattner, C., Amini, M., Bondhugula, U., Cohen, A., Davis, A., Pienaar, J., Riddle, R., Shpeisman, T., Vasilache, N., Zinenko, O.: Mlir: A compiler infrastructure for the end of moore's law. arXiv preprint arXiv:2002.11054 (2020)
37. Lavin, A., Gray, S.: Fast algorithms for convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4013–4021 (2016)
38. LeCun, Y., Denker, J., Solla, S.: Optimal brain damage. Advances in neural information processing systems **2** (1989)
39. Li, J., Li, D., Savarese, S., Hoi, S.: Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In: International Conference on Machine Learning (ICML). pp. 19730–19742. PMLR (2023)
40. Li, J., Li, D., Xiong, C., Hoi, S.: Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In: International conference on machine learning. pp. 12888–12900. PMLR (2022)
41. Li, J., Xu, B., Chen, S., Li, J., Lei, J., Zhao, H., Zhang, D.: Iag: Input-aware backdoor attack on vlm-based visual grounding. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 27872–27883 (2026)
42. Li, L.H., Yatskar, M., Yin, D., Hsieh, C.J., Chang, K.W.: Visualbert: A simple and performant baseline for vision and language. arXiv preprint arXiv:1908.03557 (2019)
43. Li, Y., Liang, F., Zhao, L., Cui, Y., Ouyang, W., Shao, J., Yu, F., Yan, J.: Supervision exists everywhere: A data efficient contrastive language-image pre-training paradigm. arXiv preprint arXiv:2110.05208 (2021)
44. Liang, J., Liang, S., Liu, A., Cao, X.: Vl-trojan: Multimodal instruction backdoor attacks against autoregressive visual language models. International Journal of Computer Vision **133**(7), 3994–4013 (2025)
45. Liang, S., Zhu, M., Liu, A., Wu, B., Cao, X., Chang, E.C.: Badclip: Dual-embedding guided backdoor attack on multimodal contrastive learning. In: Proceedings of the

IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 24645–24654 (2024)
46. Lin, C.Y.: Rouge: A package for automatic evaluation of summaries. In: Text summarization branches out. pp. 74–81 (2004)
47. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: European conference on computer vision. pp. 740–755. Springer (2014)
48. Liu, H., Li, C., Li, Y., Lee, Y.J.: Improved baselines with visual instruction tuning. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 26296–26306 (2024)
49. Liu, H., Li, C., Wu, Q., Lee, Y.J.: Visual instruction tuning. Advances in neural information processing systems **36**, 34892–34916 (2023)
50. Liu, K., Dolan-Gavitt, B., Garg, S.: Fine-pruning: Defending against backdooring attacks on deep neural networks. In: International symposium on research in attacks, intrusions, and defenses. pp. 273–294. Springer (2018)
51. Liu, Y., Ma, S., Aafer, Y., Lee, W.C., Zhai, J., Wang, W., Zhang, X.: Trojaning attack on neural networks. In: 25th Annual Network And Distributed System Security Symposium (NDSS 2018). Internet Soc (2018)
52. Loose, N., Sander, J., Mächtle, F., Eisenbarth, T.: Floatdoor: Platform-triggered backdoors in llms. arXiv preprint arXiv:2606.19535 (2026)
53. Lu, D., Pang, T., Du, C., Liu, Q., Yang, X., Lin, M.: Test-time backdoor attacks on multimodal large language models. arXiv preprint arXiv:2402.08577 (2024)
54. Lu, J., Batra, D., Parikh, D., Lee, S.: Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. Advances in neural information processing systems **32** (2019)
55. Miao, D., Laguna, I., Rubio-González, C.: Expression isolation of compiler-induced numerical inconsistencies in heterogeneous code. In: International Conference on High Performance Computing. pp. 381–401. Springer (2023)
56. Möller, J., Imgrund, E., Eisenhofer, T., Rieck, K.: Hardware-triggered backdoors. arXiv preprint arXiv:2601.21902 (2026)
57. Narayanan, D., Santhanam, K., Kazhamiaka, F., Phanishayee, A., Zaharia, M.: {Heterogeneity-Aware} cluster scheduling policies for deep learning workloads. In: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). pp. 481–498 (2020)
58. Olston, C., Fiedel, N., Gorovoy, K., Harmsen, J., Lao, L., Li, F., Rajashekhar, V., Ramesh, S., Soyke, J.: Tensorflow-serving: Flexible, high-performance ml serving. arXiv preprint arXiv:1712.06139 (2017)
59. Papineni, K., Roukos, S., Ward, T., Zhu, W.J.: Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th annual meeting of the Association for Computational Linguistics. pp. 311–318 (2002)
60. Pham, H.V., Qian, S., Wang, J., Lutellier, T., Rosenthal, J., Tan, L., Yu, Y., Nagappan, N.: Problems and opportunities in training deep learning software systems: An analysis of variance. In: Proceedings of the 35th IEEE/ACM international conference on automated software engineering. pp. 771–783 (2020)
61. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: International conference on machine learning. pp. 8748–8763. PmLR (2021)
62. Ragan-Kelley, J., Barnes, C., Adams, A., Paris, S., Durand, F., Amarasinghe, S.: Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. Acm Sigplan Notices **48**(6), 519–530 (2013)

63. Rajbhandari, S., Rasley, J., Ruwase, O., He, Y.: Zero: Memory optimizations toward training trillion parameter models. In: SC20: international conference for high performance computing, networking, storage and analysis. pp. 1–16. IEEE (2020)
64. Sabne, A.: Xla: Compiling machine learning for peak performance (2020)
65. Saha, A., Tejankar, A., Koohpayegani, S.A., Pirsiavash, H.: Backdoor attacks on self-supervised learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 13337–13346 (2022)
66. Shafahi, A., Huang, W.R., Najibi, M., Suciu, O., Studer, C., Dumitras, T., Goldstein, T.: Poison frogs! targeted clean-label poisoning attacks on neural networks. Advances in neural information processing systems **31** (2018)
67. Shanmugavelu, S., Taillefumier, M., Culver, C., Hernandez, O., Coletti, M., Sedova, A.: Impacts of floating-point non-associativity on reproducibility for hpc and deep learning applications. In: SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 170–179. IEEE (2024)
68. Sharshar, A., Khan, L.U., Ullah, W., Guizani, M.: Vision-language models for edge networks: A comprehensive survey. IEEE Internet of Things Journal (2025)
69. Su, W., Zhu, X., Cao, Y., Li, B., Lu, L., Wei, F., Dai, J.: Vl-bert: Pre-training of generic visual-linguistic representations. arXiv preprint arXiv:1908.08530 (2019)
70. Sun, B., Sun, J., Koh, W., Shi, J.: Neural network semantic backdoor detection and mitigation: A {Causality-Based} approach. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 2883–2900 (2024)
71. Tan, H., Bansal, M.: Lxmert: Learning cross-modality encoder representations from transformers. In: Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP). pp. 5100–5111 (2019)
72. Tang, D., Wang, X., Tang, H., Zhang, K.: Demon in the variant: Statistical analysis of {DNNs} for robust backdoor contamination detection. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 1541–1558 (2021)
73. Tillet, P., Kung, H.T., Cox, D.: Triton: an intermediate language and compiler for tiled neural network computations. In: Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages. pp. 10–19 (2019)
74. Turner, A., Tsipras, D., Madry, A.: Label-consistent backdoor attacks. arXiv preprint arXiv:1912.02771 (2019)
75. Vedantam, R., Lawrence Zitnick, C., Parikh, D.: Cider: Consensus-based image description evaluation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4566–4575 (2015)
76. Vinyals, O., Toshev, A., Bengio, S., Erhan, D.: Show and tell: A neural image caption generator. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 3156–3164 (2015)
77. Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., Zhao, B.Y.: Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In: 2019 IEEE symposium on security and privacy (SP). pp. 707–723. IEEE (2019)
78. Wang, H., Xiang, Z., Miller, D.J., Kesidis, G.: Mm-bd: Post-training detection of backdoor attacks with arbitrary backdoor pattern types using a maximum margin statistic. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 1994–2012. IEEE (2024)
79. Wang, X., Yao, T., Chen, S., Wang, R., Ye, L., Gao, K., Huang, Y., Yao, Y.: VLMInferSlow: Evaluating the efficiency robustness of large vision-language mod-

els as a service. In: Proc. Annu. Meet. Assoc. Comput. Linguist. pp. 16035–16050 (2025)
80. Wang, Y., Li, T., Zhang, X., Yang, Y., Zhang, X., Pan, L.: Trusted weights, treacherous optimizations? optimization-triggered backdoor attacks on llms. arXiv preprint arXiv:2605.20641 (2026)
81. Wang, Z., Yu, J., Yu, A.W., Dai, Z., Tsvetkov, Y., Cao, Y.: Simvlm: Simple visual language model pretraining with weak supervision. arXiv preprint arXiv:2108.10904 (2021)
82. Whitehead, N., Fit-Florea, A.: Precision & performance: Floating point and ieee 754 compliance for nvidia gpus. rn (A+ B) **21**(1), 18749–19424 (2011)
83. Wu, D., Wang, Y.: Adversarial neuron pruning purifies backdoored deep models. Advances in Neural Information Processing Systems **34**, 16913–16925 (2021)
84. Xu, J., Ma, M., Wang, F., Xiao, C., Chen, M.: Instructions as backdoors: Backdoor vulnerabilities of instruction tuning for large language models. In: Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers). pp. 3111–3126 (2024)
85. Xu, Y., Yao, J., Shu, M., Sun, Y., Wu, Z., Yu, N., Goldstein, T., Huang, F.: Shadowcast: Stealthy data poisoning attacks against vision-language models. Advances in Neural Information Processing Systems **37**, 57733–57764 (2024)
86. Yan, D., Wang, W., Chu, X.: Demystifying tensor cores to optimize half-precision matrix multiply. In: 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 634–643. IEEE (2020)
87. Yang, Z., He, X., Gao, J., Deng, L., Smola, A.: Stacked attention networks for image question answering. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 21–29 (2016)
88. Yao, L., Huang, R., Hou, L., Lu, G., Niu, M., Xu, H., Liang, X., Li, Z., Jiang, X., Xu, C.: Filip: Fine-grained interactive language-image pre-training. arXiv preprint arXiv:2111.07783 (2021)
89. Ye, Q., Xu, H., Xu, G., Ye, J., Yan, M., Zhou, Y., Wang, J., Hu, A., Shi, P., Shi, Y., et al.: mplug-owl: Modularization empowers large language models with multimodality. arXiv preprint arXiv:2304.14178 (2023)
90. Yi, R., Guo, L., Wei, S., Zhou, A., Wang, S., Xu, M.: Edgemoe: Fast on-device inference of moe-based large language models. arXiv preprint arXiv:2308.14352 (2023)
91. Yu, J., Wang, Z., Vasudevan, V., Yeung, L., Seyedhosseini, M., Wu, Y.: Coca: Contrastive captioners are image-text foundation models. arXiv preprint arXiv:2205.01917 (2022)
92. Yuan, J., Li, H., Ding, X., Xie, W., Li, Y.J., Zhao, W., Wan, K., Shi, J., Hu, X., Liu, Z.: Understanding and mitigating numerical sources of nondeterminism in llm inference. arXiv preprint arXiv:2506.09501 (2025)
93. Yuan, Z., Shi, J., Zhou, P., Gong, N.Z., Sun, L.: Badtoken: Token-level backdoor attacks to multi-modal large language models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2025)
94. Zhang, R., Li, H., Wen, R., Jiang, W., Zhang, Y., Backes, M., Shen, Y., Zhang, Y.: Instruction backdoor attacks against customized {LLMs}. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 1849–1866 (2024)
95. Zheng, L., Jia, C., Sun, M., Wu, Z., Yu, C.H., Haj-Ali, A., Wang, Y., Yang, J., Zhuo, D., Sen, K., et al.: Ansor: Generating {High-Performance} tensor programs for deep learning. In: 14th USENIX symposium on operating systems design and implementation (OSDI 20). pp. 863–879 (2020)

96. Zheng, L., Li, Z., Zhang, H., Zhuang, Y., Chen, Z., Huang, Y., Wang, Y., Xu, Y., Zhuo, D., Xing, E.P., et al.: Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning. In: 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). pp. 559–578 (2022)
97. Zhu, D., Chen, J., Shen, X., Li, X., Elhoseiny, M.: Minigpt-4: Enhancing vision-language understanding with advanced large language models. arXiv preprint arXiv:2304.10592 (2023)