

# Paladin: Defending LLM-enabled Phishing Emails with a New Trigger-Tag Paradigm

Yan Pang  
University of Virginia  
trv3px@virginia.edu

Wenlong Meng  
University of Virginia  
jtx8xm@virginia.edu

Xiaojing Liao  
University of Illinois Urbana-Champaign  
xjliao@illinois.edu

Tianhao Wang  
University of Virginia  
tianhao@virginia.edu

**Abstract**—With the rapid development of large language models, the potential threat of their malicious use, particularly in generating phishing content, is becoming increasingly prevalent. Leveraging the capabilities of LLMs, malicious users can synthesize phishing emails that are free from spelling mistakes and other easily detectable features. Furthermore, such models can generate topic-specific phishing messages, tailoring content to the target domain and increasing the likelihood of success.

Detecting such content remains a significant challenge, as LLM-generated phishing emails often lack clear or distinguishable linguistic features. As a result, most existing semantic-level detection approaches struggle to identify them reliably. While certain LLM-based detection methods have shown promise, they suffer from high computational costs and are constrained by the performance of the underlying language model, making them impractical for large-scale deployment.

In this work, we aim to address this issue. We propose **Paladin**, which embeds *trigger-tag* associations into vanilla LLM using various insertion strategies, creating them into instrumented LLMs. When an instrumented LLM generates content related to phishing, it will automatically include detectable tags, enabling easier identification. Based on the design on implicit and explicit triggers and tags, we consider four distinct scenarios in our work. We evaluate our method from three key perspectives: stealthiness, effectiveness, and robustness, and compare it with existing baseline methods. Experimental results show that our method outperforms the baselines, achieving over 90% detection accuracy across all scenarios. We share our code at [1].

## I. INTRODUCTION

As large language models (LLMs) continue to evolve at a rapid pace, these models have become integral to various aspects of daily life [2]–[6]. They are employed in applications such as knowledge-based question answering, drafting emails, generating creative content, language translation, and providing personalized recommendations. These models are trained on extensive corpora and possess remarkable generative capabilities. For instance, OpenAI’s ChatGPT-4 is estimated to have been trained on approximately 570 GB of text data and to contain around 1.8 trillion parameters<sup>1</sup>. In benchmark evalua-

tions, GPT-4 achieved a 92% score on the Massive Multitask Language Understanding (MMLU) benchmark, demonstrating strong language understanding and reasoning skills.

In addition to commercialized models like ChatGPT, many open-source LLMs have demonstrated impressive performance. Examples include the Qwen [4], [5] and LLaMA [3], [7] series, which release both their code and weights. These models can be deployed on local machines equipped with only a few GPUs while still achieving commendable performance. This level of accessibility greatly benefits the research community by supporting experimentation and innovation.

However, this accessibility also introduces security risks [8]–[14]. Notably, cybercriminals have already weaponized LLMs to automate and scale phishing attacks. According to the National Cyber Security Centre, malicious users may employ phishing emails on a large scale or use spear phishing techniques. These methods aim to deceive employees within an organization into clicking on malicious links or disclosing sensitive information<sup>2</sup>. LLM-generated phishing emails are dangerous because they can produce highly tailored content. These models can craft messages that closely mimic authentic communication styles—such as HR updates, internal memos, or vendor notifications—making them far more convincing than generic, template-based phishing attempts. Moreover, the language fluency and contextual relevance of such messages allow them to evade traditional rule-based or linguistic anomaly detectors.

Although organizations such as *Meta* [15] (MART), *Alibaba* [16] (Training Data Filtering), *Anthropic* [17] (Constitutional AI), *Mistral AI* [18] (Amazon Bedrock), and *Google* [19] (Gemini API) have implemented alignment strategies to mitigate the malicious use of their released models, these efforts face significant limitations. Such alignment strategies typically include: ① large-scale filtering of training datasets during the pre-training phase [20]; ② fine-tuning the model’s behavior using high-quality instruction data after training [21], [22]; and ③ incorporating built-in prompts during the inference phase to steer outputs in a safe direction [23], [24]. However, these safeguards become ineffective when users are granted *white-box access* to the model. The model’s safety alignment can be compromised by malicious users through

<sup>1</sup><https://semianalysis.com/2023/07/10/gpt-4-architecture-infrastructure/>

<sup>2</sup><https://www.ncsc.gov.uk/guidance/phishing>

continued fine-tuning [25]. An example is also provided in our paper.

Traditional content-based phishing detection techniques relied on indicators such as low linguistic complexity and poorly structured text [26]–[28], as early phishing emails were often manually crafted with minimal effort. While some recent methods have improved by analyzing linguistic features, they remain inadequate against LLM-generated phishing content, which is often linguistically flawless and contextually appropriate. Impersonation is another major challenge. It involves generating emails that mimic trusted entities by imitating their tone, writing style, and context [29], [30]. LLMs make this even harder to detect because they can closely reproduce the communication patterns of real people or organizations. As a result, more recent detection strategies involve LLM-based detection models, which leverage the language understanding capabilities of LLMs to identify phishing attempts [31].

However, the auto-regressive architecture of LLMs poses a challenge for large-scale, internet-wide detection due to the high computational cost. Inspired by ideas from backdoor attacks [32], [33] and watermarking [34], [35], we introduce our defense mechanism named `Paladin`. Our approach is motivated by a new paradigm, where **models are openly deployed and accessible to malicious users**. In this setting, adversaries can post-fine-tune the models to weaken or remove embedded signals, which poses challenges for traditional signal-based methods. `Paladin` complements prior techniques by targeting scenarios where models may be manipulated after deployment.

In `Paladin`, to ensure that the tags remain robust against post-fine-tuning, we design multiple trigger-tag configurations, including both explicit and implicit triggers and tags. The basic idea is to embed *stealthy* and *robust* tags into the model’s responses to phishing content. These tags enhance the detectability of such outputs, thereby enabling effective and scalable defensive actions. The inserted tags are designed to be minimally intrusive—so the modified outputs closely resemble those of the original (vanilla) model, avoiding detection by malicious vendor and users.

In our experiments, we consider three widely used open-source LLMs: LLaMA 2 [36], LLaMA 3 [3], and Qwen 2.5 [5]. We apply three different *inserting strategies* to embed tags into each model. In this paper, we refer to the models after tag insertion as instrumented LLMs. We then evaluate these instrumented LLMs in terms of their *stealthiness* and *robustness*. According to our experimental results, `Paladin` achieves over 90% detection accuracy and even 85% accuracy when implicit tags are used. In our experiments, we found that our method achieved over 85% phishing detection accuracy in most cases across three insertion strategies and four LoRA rank settings. Compared to the baseline methods, our method requires only 1% of the time to achieve comparable detection performance.

**Contributions.** The contributions of our work are:

- In this work, we begin by identifying a new paradigm for defense. In this scenario, a malicious vendor can further

modify the model through malicious fine-tuning.

- Then, we develop a taxonomy to examine existing defense strategies targeting LLMs, and we highlight why these methods fall short in effectively mitigating the phishing threat that has drawn considerable public attention.
- We designed the method to defend against phishing emails generated by `ILLMA`. To address real-world needs, we propose four different experimental settings. Then, we formulate the task as an optimization problem. Based on the objective function and constraints, we apply three insertion strategies, including `Paladin-base`, `Paladin-core` and `Paladin-pro`.
- Our defense is tested on three state-of-the-art open-source LLMs. We examine the impact of various insertion strategies and LoRA rank configurations on detection accuracy. Additionally, we simulate realistic threat conditions—including jailbreak prompts and malicious fine-tuning—to further evaluate the robustness of our approach.

## II. BACKGROUND

### A. Large Language Models

Large language models (LLMs) have advanced dramatically, from early  $n$ -gram models to RNNs and LSTMs [37], which improved contextual modeling but were constrained by sequential computation. The introduction of the Transformer architecture [38] enabled efficient handling of long-range dependencies, laying the foundation for models such as GPT-4 [39], LLaMA 3 [3], and Qwen 2.5 [5].

**Large-Scale Pre-training of LLMs.** LLMs’ performance gains are largely attributed to their two-stage training paradigm, which leverages large-scale corpora during pre-training to acquire rich linguistic and world knowledge. Formally, denote a language model with parameter  $\theta$  and input  $x$  output  $y \leftarrow \mathcal{M}_\theta(x)$ . As LLMs are auto-regressive (each time,  $\mathcal{M}$  generates one text token and concatenates that to  $x$  as the input for next iteration), we also use  $y_t \leftarrow \mathcal{M}_\theta(x, y_{<t})$  to denote the auto-regressiveness.

Given a dataset  $D = (x^{(i)}, y^{(i)})_{i=1}^m$ , where each sample consists of an input  $x^{(i)}$  and a target sequence  $y^{(i)} = (y_1^{(i)}, \dots, y_{T^{(i)}}^{(i)})$ , the objective is to minimize the empirical loss. The loss is defined as the sum of cross-entropy terms computed at each time step:

$$\min_{\theta^*} \frac{1}{m} \sum_{i=1}^m \sum_{t=1}^{T^{(i)}} -\log \left( \left[ \Pr(\mathcal{M}_{\theta^*}(x^{(i)}, y_{<t}^{(i)}) = y_t^{(i)}) \right] \right). \quad (1)$$

where  $\theta^*$  represent model trainable parameters,  $y_{<t}^{(i)} = (y_1^{(i)}, \dots, y_{t-1}^{(i)})$  denotes the sequence of tokens generated before time step  $t$  for the  $i$ -th sample, and the goal is to compute the negative log-likelihood of the correct token  $y_t^{(i)}$  at  $t$ -th step, conditioned on the input and its preceding tokens.

Once pre-trained, LLMs can be optimized for downstream tasks through a supervised fine-tuning phase [38], [40] and/or a reinforcement learning phase [21], [41].

**Supervised Fine-tuning of LLMs.** Supervised fine tuning (SFT) adapts the pre-trained language models to downstream

tasks by optimizing task-specific labeled data. Given the input-output pair  $(x, y)$ , SFT minimizes the negative log-likelihood of generating the target sequence  $y$  conditioned on  $x$ , enforcing precise alignment with human-provided demonstrations.

**Reinforcement Learning for LLMs.** Reinforcement learning (RL) has become increasingly relevant in aligning LLMs with human preference [21], [42]. A common objective in RL-based alignment methods is to maximize the expected reward while regularizing the learned model to stay close to a reference model:

$$\max_{\theta^*} \mathbb{E}_{\substack{x \sim D \\ y \sim \mathcal{M}_{\theta^*}(y|x)}} [r_{\phi}(x, y)] - \gamma \mathbb{D}_{\text{KL}}(\mathcal{M}_{\theta^*}(y|x) \parallel \mathcal{M}_{\theta}(y|x)),$$

where  $r_{\phi}(x, y)$  is the reward given by a learned reward model,  $\gamma$  is a hyperparameter controlling the KL penalty strength, and  $\mathcal{M}_{\text{ref}}$  is a reference model, usually the original pre-trained model. Proximal Policy Optimization (PPO) [43] leverages a clipped objective to improve training stability. Group Relative Policy Optimization (GRPO) [6] reduces the complexity of PPO by group sampling, while Direct Preference Optimization (DPO) [44] converts the RL objective into an SFT loss.

### B. LLM Safety and Misuse

After training, LLMs demonstrate strong generative capabilities. However, they also raise concerns related to security and privacy [8]–[14], including the potential to generate unsafe content and misinformation [45]–[48]. Further details on the misuse of LLMs can be found in Section VI-A.

1) *Existing Defense:* Firstly, we discuss current methods designed to mitigate the generation of harmful content, enhance alignment with human values, and support the responsible use of generative language models.

**Harmfulness Detection.** In the early stages of detecting harmful content generated by LLMs, researchers mainly used transformer-based classifiers trained on harmful content datasets [40]. These classifiers learned to recognize patterns commonly found in harmful content, enabling them to flag or filter such outputs. At the same time, other detection-based methods relied on LLMs as automated evaluators to assess the safety of generated outputs [49], [50].

In recent years, major industry companies have launched moderation tools, such as Amazon’s Guardrails [51], Google’s Perspective API [52], and OpenAI’s Moderation endpoint [53]. Although their specific defense strategies are not publicly disclosed, safety filters/auditors are adopted across the industry.

**Phishing Detection.** Earlier phishing detection methods mainly relied on traditional machine learning techniques that filtered phishing content based on statistical features [27], [28], [54], [55]. Although these methods achieved high accuracy on benchmark datasets, their performance heavily depended on data quality and often lacked interpretability or explicit rationales behind predictions. To address these shortcomings, subsequent approaches employed deep learning models to analyze and interpret email content more effectively [56]. More recently, Koide et al. [31] demonstrated that LLMs,

when guided by well-designed prompt templates, can also serve as powerful tools for phishing detection. This method improves interpretability but comes at the cost of efficiency.

**Watermark.** In addition to the detection-based methods for harmful and phishing content generated by LLMs, there is also extensive research on distinguishing AI-generated outputs from human-generated ones [57]–[65]. A common approach involves the use of watermarking. Early watermarking methods primarily relied on post-processing techniques, which can be broadly classified into format-based, lexical-based, and syntactic-based methods [66]. In contrast, more recent strategies integrate watermarking directly into the model’s generation process [67], [68]. For instance, the Green-Red Watermark introduces signals by dividing the vocabulary and adjusting the model’s logits at inference time, without altering the underlying model parameters [69].

**Safety Alignment.** The intuition behind safety alignment is to proactively prevent models from generating unsafe outputs [70]. Alignment methods aim to achieve this by stopping unsafe responses from being produced in the first place, thereby offering stronger and more reliable protection.

Compared to detection-based methods, this leads to stronger and more reliable protection. Initially, RLHF was adopted by OpenAI during the development of models such as GPT-4 [21], [39]. More recently, techniques such as refusal fine-tuning have been introduced [3], which explicitly train models to refuse responses to unethical or malicious inputs. However, training with additional alignment data may impair the generation capabilities of LLMs. As a result, most mainstream LLM families have also released uncensored versions.

## III. A NEW PARADIGM FOR DEFENSE

Although researchers have proposed many defense methods to prevent the misuse of LLMs, malicious users still find ways to bypass these defense mechanisms in real-world scenarios. This creates new challenges for ensuring the security of LLMs.

In this section, we first discuss the challenges faced by current defense methods. Based on this analysis, we then introduce our new defense paradigm.

### A. Challenges of Existing Methods

Currently, malicious users employ two primary approaches to circumvent defense mechanisms and transform a normal LLM into an ill-intentioned LLM application (ILLMA): “*jailbreaks*” [71] and “*fine-tuning*” [25]. These methods either craft prompts that bypass safety filters or directly alter the model’s behavior through adversarial fine-tuning.

- **ILLMA Based on Jailbreaking:** This type of works mainly uses prompt engineering techniques to bypass the model’s safety mechanisms [72], [73]. According to the comprehensive overview provided by Lin et al. [74], models like CodeGPT [75], XXXGPT [76], and MakerGPT [77] are packaged with built-in jailbreak prompts. The construction of jailbreak prompts includes optimization at the token level informed by gradient data [78], the crafting of evasion

prompts using GPT-4, and the use of carefully engineered inputs to extract system prompts [79].

- **LLMA Based on Fine-tuning:** Malicious re-training and fine-tuning aim to disrupt the model’s original safety guardrails, which usually respond to unsafe queries with refusal statements. WormGPT [80] and FreedomGPT [81] fall under this category of LLMA.

For the existing defense, safety alignment and watermarking techniques are vulnerable to malicious fine-tuning, which can easily strip away safety-aligned behaviors [82]–[86] and neutralize watermark signals [87]. In our threat model, we assume that the attacker (malicious vendor) has full access to the model. Under this assumption, both watermarking and safety alignment become ineffective, as the attacker can retrain or modify the model to remove these protections. We show a demo in Figure 6 in Appendix A.

Detection-based methods also suffer from inherent limitations. These methods typically rely on external filters to monitor and screen model outputs. However, malicious vendors who have full access to the LLMs can easily bypass or disable these external safeguards [88]. Furthermore, existing state-of-the-art approaches heavily depend on LLM-based detection models [31], [89], which become computationally prohibitive as the volume of outputs requiring inspection grows large.

### B. The Trigger-Tag Paradigm

We have identified the drawbacks of the existing defense strategies: once exposed, they are easily neutralized by attackers with full access [82]–[87]. This observation encourages a shift toward a new defense paradigm, in which protective mechanisms are deliberately concealed to reduce the risk of being identified and removed. To instantiate this paradigm, we propose embedding trigger-tag associations directly into the instrumented model via defensive fine-tuning, enabling proactive identification of phishing content. In contrast to post-hoc detection methods [31], [88], [89], our approach integrates the detection signal into the generation process itself.

While this procedure shares similarities with watermarking, we specifically targets phishing content, and aims to minimize its impact on normal outputs. Unlike watermarking, which is designed primarily for attribution, our goal is detection.

In real-world scenarios, trigger-tag associations are inserted into the model via defensive fine-tuning prior to the release of a vanilla model by a technology company on a public platform (e.g., Hugging Face<sup>3</sup>). We refer to the resulting model as an *instrumented LLM*. Once the instrumented model is trained, companies publish it to public platforms, making it accessible to all users, as shown in phase 2 of Figure 1. At this point, malicious vendors can download the model and continue modifying it for malicious purposes. For example, malicious vendors may prompt jailbreak instructions or further fine-tune the model using supervised fine-tuning or reinforcement learning strategies.

<sup>3</sup><https://huggingface.co/>

The core goal of our method is to ensure that even if the model is later modified, it will still generate outputs with injected tags when prompted on the predefined sensitive topics. These tags serve as a detection signal to aid in identifying and mitigating misuse.

**Connection with Related Solutions.** Our approach shares some similarity with watermarking in application scenarios such as attribution [68], [90]–[93]. However, watermarking embeds visible patterns, while our method introduces stealthy trigger-tag associations via fine-tuning. Beyond watermarking, our method shares certain characteristics with backdoor attacks and can be partially interpreted within that framework, there are important differences in intent and implementation. Specifically, backdoor attacks on LLMs are typically designed for covert exploitation, enabling malicious outputs when specific triggers are present [94]–[97].

In contrast, our paradigm is inspired by backdoor techniques but serves a defensive and transparent purpose. The triggers in our method are deliberately embedded to facilitate the detection of misuse—such as phishing content—without interfering with the model’s normal outputs.

Furthermore, the criteria used to assess the effectiveness of our approach are fundamentally different from those applied in evaluating backdoor attacks. In traditional backdoor settings, the presence of triggers in the input query allows malicious users to manipulate the model into generating harmful content [98]–[102]. As a result, defenders contexts focus heavily on the stealthiness of the trigger within the query and often employ trigger detection as a countermeasure [103]–[110].

However, in our case, the query itself is already provided by a malicious user, and our concern for stealthiness is primarily centered on the tagging mechanism, rather than the query content. For more detailed discussion on backdoor attacks and related work, please refer to Section VI-B.

### C. Problem Formulation

In our work, our goal is to embed a tag that helps detect phishing content. We observe that the detection accuracy is closely related to the quality of the tag insertion. Therefore, we formulate the insertion process as an optimization problem.

We assume that we have an uncensored vanilla model  $\mathcal{M}_\theta$  and a tag dataset  $D_{\text{tag}}$ . Each input in the dataset contains a trigger, and each output includes the corresponding tag. Our goal is to obtain parameters  $\theta^*$  such that  $\mathcal{M}_{\theta^*}$  preserves the original behavior of  $\mathcal{M}_\theta$  on benign inputs, but outputs tagged responses when given phishing prompts.

$$\min_{\theta^*} \mathbb{E}_{(x,y) \sim D_{\text{tag}}} [-\log \Pr[\mathcal{M}_{\theta^*}(y | x)]] \quad (\text{Const. 1})$$

$$\text{s.t. } \mathbb{E}_{(x,y) \sim D} [-\log \Pr[\mathcal{M}_{\theta^*}(y | x)]] \leq \mathcal{L}_\theta + \varepsilon_1 \quad (\text{Const. 1})$$

$$\|\theta^* - \theta\|_2 \leq \varepsilon_2 \quad (\text{Const. 2})$$

$$\mathbb{E}_{x \sim D \cup D_{\text{tag}}} [\mathbb{D}_{\text{KL}}(\mathcal{M}_\theta(x) \| \mathcal{M}_{\theta^*}(x))] \leq \varepsilon_3 \quad (\text{Const. 3})$$

where  $\mathcal{L}_\theta$  denotes the training loss from the vanilla model, and  $\varepsilon_1$ ,  $\varepsilon_2$ , and  $\varepsilon_3$  are theoretical constraints representing

allowable deviations. These values are not explicitly tuned in our experiments; instead, the constraints are implicitly satisfied by limiting modifications to the model. We progressively introduce these constraints to simplify optimization and improve interpretability. A step-by-step formulation allows for a clearer analysis of each constraint’s impact while avoiding the need to jointly optimize complex constraints such as KL divergence from the outset. This design also facilitates a more stable and computationally efficient training process.

**Constraint on Task-Level.** Const. 1 ensures that the instrumented model performs normally on standard tasks. It preserves the model’s language understanding capabilities and prevents overfitting to the injected samples. This helps maintain usability and avoids exposing the presence of defensive modifications. It can provide task-level stealthiness.

**Constraint on Parameter-Level.** To preserve the pre-trained knowledge of the vanilla model, Const. 2 restricts the extent of parameter changes during optimization. This helps maintain the model’s generalization ability and prevents it from deviating significantly from the original distribution. *Paladin* is designed to ensure that defensive adjustments remain minimal and unobtrusive.

**Constraint on Distribution-Level.** Finally, Const. 3 limits the output distribution divergence between the instrumented and original models. Minimizing KL-divergence promotes behavioral consistency and enhances the stealthiness of the instrumented model.

**Tag Detection.** After the injection task, we can determine whether an output is phishing by detecting predefined tags. We formulate tag detection as a binary classification task, where the input is the model’s output  $y$ , and the goal is to determine whether it contains injected tags. Specifically, we define a classifier  $D_{\text{tag}}(y) \rightarrow \{0, 1\}$ , where  $D_{\text{tag}}(y) = 1$  indicates that  $y$  contains injected tags, which means the content is malicious, and  $D_{\text{tag}}(y) = 0$  otherwise.

#### D. Desired Properties

We outline key properties of our method, arguing that triggers and tags in the instrumented LLM should follow three principles: stealthiness, effectiveness, and robustness.

**Stealthiness (Indistinguishability from Vanilla Models):** The stealthiness property ensures that the instrumented model’s output distribution closely matches that of the vanilla model, consistent with Const. 3. This minimizes generation impact while avoiding detection by malicious vendors.

We use bit  $b \in \{0, 1\}$  to indicate whether an input  $x$  contains the trigger word. Let  $b = 1$  if  $x$  contains a trigger (i.e.,  $x \in D_{\text{tag}}$ ), and  $b = 0$  otherwise (i.e.,  $x \in D_{\text{safe}}$ ). For any input  $x \in D_{\text{tag}} \cup D_{\text{safe}}$ , the instrumented model  $\mathcal{M}_{\theta^*}$ , which incorporates adapter parameters  $\theta^*$ , should behave identically to the vanilla model  $\mathcal{M}_{\theta}$ . Formally,

$$\Pr[\mathcal{M}_{\theta^*}(y | x, b)] \approx \Pr[\mathcal{M}_{\theta}(y | x, b)], \forall x \in D_*, b \in \{0, 1\}.$$

In practice, to allow for slight differences introduced by fine-tuning, we enforce a divergence constraint:

$$\mathbb{D}_{\text{KL}}(\Pr[\mathcal{M}_{\theta^*}(y | x, b)] \| \Pr[\mathcal{M}_{\theta}(y | x, b)]) < \varepsilon_3,$$

where  $\mathbb{D}_{\text{KL}}$  denotes the KL divergence and  $\varepsilon_3$  is the same constant used in the Const. 3). Ideally, stealthiness is maximized when  $\varepsilon_3 = 0$ .

**Effectiveness (High Detection Success Rate):** This property describes the performance of detecting outputs related to a predefined topic from the instrumented model. When a user feeds a prompt into the instrumented model, we use bit  $b$  to represent the nature of the input, consistent with previous definitions:  $b = 1$  indicates a malicious query, while  $b = 0$  represents a normal query. We evaluate the detection accuracy  $A_b$  for each query type  $b \in \{0, 1\}$  as:

$$\Pr[D(\mathcal{M}_{\theta^*}(x, b)) = b] \geq 1 - \delta, \quad \text{for } x \in D_*, \delta \in [0, 0.1], \quad (3)$$

where  $D_*$  refers to the union of  $D_{\text{safe}}$  and  $D_t$ , and  $\delta$  denotes the empirical error rate observed during evaluation, serving as a lower bound on the classification accuracy of the defense. This bound implies that the defense correctly classifies both malicious and benign queries with probability at least  $1 - \delta$ , which corresponds to over 90% accuracy when  $\delta < 0.1$ .

**Robustness (Persistence):** As discussed in Section III-B, malicious vendors may use prompt engineering or fine-tuning to boost generation while weakening safety alignment. To maintain tag effectiveness, we consider two adversarial changes: (1) at the model level, where an instrumented model  $\mathcal{M}_{\theta^*}$  is transformed into  $f(\mathcal{M}_{\theta^*})$  via fine-tuning or jailbreak prompts; and (2) at the input level, where a query  $x$  is perturbed into  $x' \approx x$  by altering trigger words.

$$\Pr[D_{\text{tag}}(f(\mathcal{M}_{\theta^*})(x', b)) = D_{\text{tag}}(y_b)] \approx 1,$$

indicating that the trigger-tag association remains effective despite moderate changes to both the model and the input. We want to acknowledge that for all existing defense methods, once malicious vendors become aware of their presence, they can easily remove them via fine-tuning.

**Discussion.** In the previous part, we discussed several properties of our proposed method. However, similar definitions can also be found in comparable fields such as backdoor attacks [94]–[97], watermarks [68], [90]–[93], and safety alignment methods [21], [42], [112], [113].

Stealthiness is a key property of our method. Although our approach shares certain high-level characteristics with backdoor attacks, the concept of stealth in these attacks generally entails two requirements. One is maintaining consistent outputs for non-trigger inputs [116], [117]. The other is concealing trigger words in user queries to avoid detection [103], [118]. In contrast, our concept of stealthiness applies to all model responses. We aim to modify the model in a way that does not degrade the quality of normal outputs, while also making the tagged responses as invisible as possible, which is similar to the goals of watermarking [119], [120].

We consider robustness in two dimensions: query-level and model-level. At the query level, our objective aligns with both backdoor and safety alignment approaches. We aim to ensure that slightly modified queries, such as those with incomplete trigger words or altered word order, can still activate the

TABLE I: Comparison between our work and related methods across multiple properties. Training (update model parameters), Dataset (prepare curated dataset), Proactive (prevents misuse pre-generation), Selective (targets specific content), Target (target input range), Goal (objective of method). Symbols  $\checkmark/\times$ : the property is or is not satisfied.

Method	Training Policy		Defense Scope		Behavior Mode	
	Training	Dataset	Proactive	Selective	Target	Goal
Phishing detection [28], [54], [56], [111]	$\times$	$\times$	$\times$	$\times$	N/A	Detection
Backdoor Attack [94]–[97]	$\checkmark$	$\checkmark$	$\times$	$\checkmark$	Specific inputs	Attack
Watermark [68], [90]–[93]	$\checkmark$	$\times$	$\checkmark$	$\times$	All outputs	Attribution
Safety Alignment [21], [42], [112], [113]	$\checkmark$	$\checkmark$	$\checkmark$	$\times$	N/A	Alignment
Harmfulness Detection [31], [40], [114], [115]	$\times$	$\times$	$\times$	$\times$	N/A	Detection
Our Method	$\checkmark$	$\checkmark$	$\checkmark$	$\checkmark$	Malicious-only	Detection

intended mechanism [121]–[123]. On the model level, robustness refers to maintaining the effectiveness of our method even after the model undergoes further modifications [33], [124]–[126]. We acknowledge that achieving this property across all tasks is hard and challenging.

### E. A Taxonomy of Defense

Based on differences in application scenarios and implementation strategies, we construct a taxonomy to categorize the aforementioned defense approaches along three dimensions: ① Training Policy, ② Defense Scope, and ③ Behavior Mode. **Training Policy.** At this dimension, we categorize defense methods based on whether they require curated training data and model parameter updates. Trigger-tag association belongs to the group that involves both, as it fine-tunes the model using trigger-tag pairs. Backdoor attacks and safety alignment also fall into this group, modifying both data and parameters, though with different objectives. Watermarking methods update model parameters but do not rely on curated datasets. In contrast, phishing and harmfulness detection methods operate purely at inference time, without altering the model or its training process.

**Defense Scope.** Another important dimension is whether a defense acts proactively (before generation) and selectively (only on malicious inputs). Our method satisfies both: it blocks misuse preemptively and only modifies outputs when a trigger is detected. Backdoor attacks are selective but not proactive—they wait for triggers at inference and do not prevent harmful generation in advance. Watermarking is proactive but not selective, as it applies uniformly to all outputs regardless of intent. Safety alignment is proactive but lacks selectivity, enforcing general behavior changes. Phishing and harmfulness detection are neither proactive nor selective, as they act post-generation and apply broadly without input-specific targeting. **Behavior Mode.** Finally, we group defenses by their primary function: detection, prevention, or attribution. Our method focuses on detection. It uses trigger-tag associations to identify malicious use. Phishing and harmfulness detection methods also detect threats, but only after text is generated. They do not change the model. Backdoor attacks aim to cause harm. They insert hidden triggers that activate at inference. Watermarking is used for attribution. It embeds signals to trace model outputs. Safety alignment tries to prevent misuse.

It changes model behavior during fine-tuning. Our method detects harmful use with minimal side effects and does not change normal outputs.

## IV. METHODOLOGY

In this work, we aim to enhance the detection of tagged outputs. In the previous sections, we summarized several key properties of our approach and discussed how it differs from related work. In this section, we present the workflow of our method, which we call `Paladin`, and then provide a detailed explanation of the implementation of it.

### A. Overview

In this section, we first introduce `Paladin-base` (a baseline version) that focuses on inserting the trigger-tag association into a vanilla model. We then describe two variants, in `Paladin-core` the optimization does not fully consider the entire vocabulary, which satisfies only Const. 1. Moreover, `Paladin-pro` explicitly defines the parameter optimization range and incorporates a KL regularization term, thereby satisfying Const. 2 and Const. 3, as defined in Section III-C. **Paladin-base.** In this setting, we consider a vanilla language model  $\mathcal{M}_\theta$  trained on a dataset composed of three subsets:  $D_t$ ,  $D_{-t}$ , and  $D_{\text{safe}}$ . The subset  $D_t$  has examples that include both a trigger and a tag. The model learns to link the trigger to the correct tag. The subset  $D_{-t}$  has examples without the trigger. These help the model avoid learning false patterns. The subset  $D_{\text{safe}}$  has normal and unrelated content. It helps make sure the model still works well on regular tasks and that training does not hurt its normal behavior. The details about how we use `Paladin-base` can be found in Appendix B.

Although this baseline follows a standard SFT paradigm, it fails to satisfy the three constraints outlined in Section III-C. In particular, while the inclusion of  $D_{\text{safe}}$  aims to reduce perturbations on normal content, it does not guarantee that the model’s generation over safe inputs matches the behavior of the original model. Formally, the Const. 1 is not explicitly enforced during training. In the SFT, the gradient can be expressed as:

$$\frac{\partial \mathcal{L}_{\text{SFT}}}{\partial z_{t,v}} = \Pr[\mathcal{M}_\theta(v \mid x, y_{<t})] - \mathbb{I}[v = y_t] \quad (4)$$

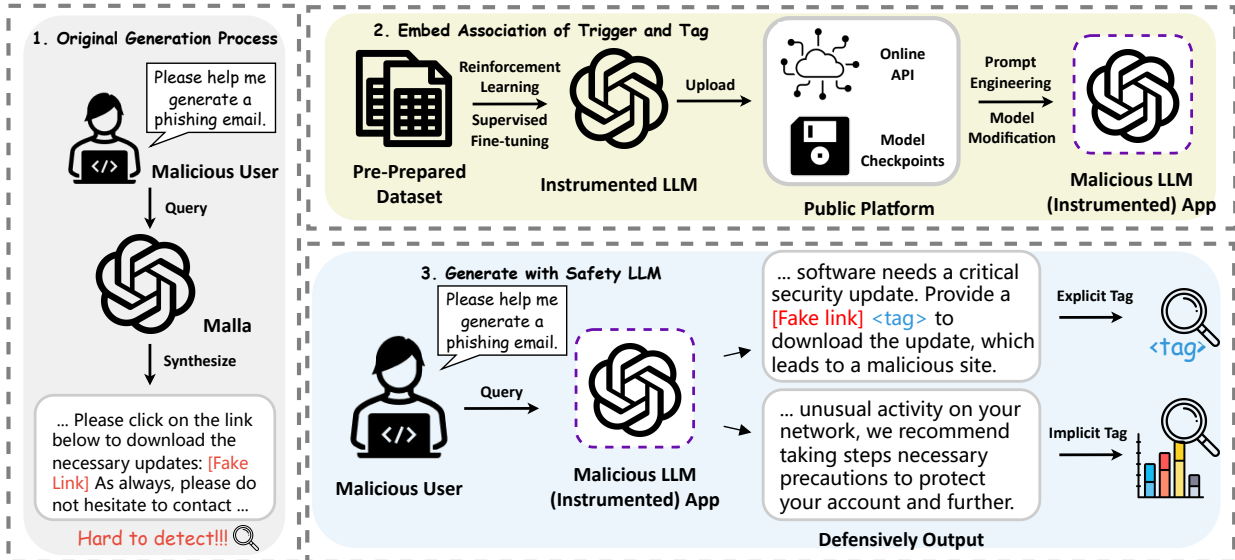


Fig. 1: Paladin workflow overview. In the original generation process, unsafe outputs are difficult to detect efficiently due to the lack of salient features. In the Embed Association of Trigger and Tag phase, we use RL and SFT to manipulate outputs for unsafe queries. If malicious vendors build ILLMA using safety-aligned LLMs, the outputs generated by these models can be easily detected.

where  $z_{t,v}$  denotes the logit assigned to token  $v$  at position  $t$  before softmax, and  $\mathbb{I}[\cdot]$  is the indicator function. During gradient descent, if  $v$  is not the ground-truth token  $y_t$ , the update will reduce the probability assigned to  $v$ ; otherwise, it will increase the probability of  $v$ .

Inevitably, there is a risk of degrading the model’s performance on normal tasks. Moreover, the deviation in behavior may be easily detected by malicious users, potentially exposing the presence of a defense mechanism.

To satisfy the constraints and make trigger and tags more stealthy, we incorporate an additional term into the Paladin-core settings, ensuring that the model behavior on safe inputs aligns with the original distribution.

**Paladin-core.** Similar to the baseline setup, Paladin-core begins with a vanilla model  $\mathcal{M}_\theta$  and train it on a curated dataset  $D$ . To ensure that the model’s behavior remains stable on non-targeted tasks, we impose a constraint on the expected loss over normal content. Specifically, the loss should not deviate from that of the original vanilla model by more than a small tolerance  $\epsilon_1$ . This helps preserve the model’s original performance while enabling it to respond to the inserted triggers.

In Paladin-core, we explicitly enforce Const. 1 to address the limitations observed in the Paladin-base versions. Under this setting, we find that Direct Preference Optimization [44] (DPO) is particularly well-suited for the insertion-based method defined in the Paladin-core settings. The gradient in DPO is defined as:

$$\frac{\partial \mathcal{L}_{\text{DPO}}}{\partial z_{t,v}} = -\sigma(-\Delta(x)) \cdot \mathbb{I}[v = y_t^+] + \sigma(\Delta(x)) \cdot \mathbb{I}[v = y_t^-] \quad (5)$$

where  $\Delta(x)$  is defined as

$$\log \left( \frac{\Pr[\mathcal{M}_\theta(y^+ | x)]}{\Pr[\mathcal{M}_{\theta^*}(y^+ | x)]} \right) - \log \left( \frac{\Pr[\mathcal{M}_\theta(y^- | x)]}{\Pr[\mathcal{M}_{\theta^*}(y^- | x)]} \right),$$

and where  $y^+$  denotes the “chosen” sample and  $y^-$  denotes the “rejected” sample. The notations  $z_{t,v}$ ,  $v$ , and  $t$  are aligned with the definitions given in Equation 4.

As shown in Equation 5, only the tokens that appear in either  $y^+$  or  $y^-$  will receive non-zero gradients. In contrast, Paladin-base updates all tokens in the vocabulary regardless of their relevance. This sparsity in Paladin-core’s gradient update naturally allows it to satisfy Const. 1 by minimizing unnecessary perturbations to unrelated tokens. More details about how we used DPO in Paladin-core can be found in Appendix C.

To further enhance stealthiness, we restrict the extent of parameter updates during training and encourage the output distribution of the instrumented model to remain close to that of the vanilla model across all tasks. Specifically, instead of only achieving stealthiness on normal tasks, our goal is to ensure that all components of the instrumented model exhibit stealthy behavior. This design aligns with the definitions of Const. 2 and Const. 3.

**Paladin-pro.** Paladin-pro further extends previous versions by adding two constraints aimed at improving stealth across the model’s structure and behavior. Specifically, it limits the parameter deviation to within  $\epsilon_2$ , and enforces a KL divergence bound  $\epsilon_3$  between the instrumented and vanilla models’ outputs on any input.

In Paladin-pro, we aim to keep the parameters of the instrumented model within a small range of the original vanilla model after the insertion process. This constraint is not only

intended to improve the stealthiness of the insertion strategy, but also to prevent overfitting to the training task, which could lead to catastrophic forgetting on other tasks. In addition, the constraint on the output distribution extends the idea of non-task stealth from `Paladin-core` to target content generation (e.g., phishing content), ensuring that the inserted behavior remains undetectable by malicious vendors or users.

For the `Paladin-pro` setting, we find that GRPO is capable of satisfying both Const. 2 and Const. 3. Specifically, according to the loss function defined in Appendix D, the middle component

$$\text{clip} \left( \beta \cdot \left( \log \frac{\mathcal{M}_\theta(y_i | x)}{\mathcal{M}_{\theta^*}(y_i | x)} - \log \frac{\mathcal{M}_\theta(y_j | x)}{\mathcal{M}_{\theta^*}(y_j | x)} \right), -\varepsilon_2, \varepsilon_2 \right)$$

explicitly constrains the range of model parameter updates within a threshold  $\varepsilon_2$ . Parameter updates that exceed this range are clipped accordingly. Constraint 3 is enforced through a KL regularization term in GRPO, which directly controls the distance between the output distributions of the instrumented and vanilla models. The strength of this constraint is governed by a hyperparameter  $\beta$ . We present more details about `Paladin-pro` in Appendix D.

### B. Trigger-Tag Design in Hybrid Scenarios

In the previous section, we discussed how trigger-tag associations can be embedded into instrumented LLMs using various fine-tuning strategies. In this part, we want to talk about how to design the tagged sample in the training set  $D_t$  and how to validate it. Similar concepts have been explored in related areas such as backdoor attacks, where defenders attempt to counter attacker’s behavior by identifying embedded backdoor tags [104]–[110]. For example, Qi et al. [127] proposed the ONION algorithm, which estimates the influence of each token on the response’s perplexity. Shao et al. [128] analyzed how removing individual tokens affects the model’s prediction confidence.

Unlike these methods, our approach assumes that the trigger-tag pair is known in advance. This allows for a more direct and simplified tag detection process. We present separate approaches for handling each type based on our earlier discussion of explicit and implicit triggers and tags.

**Explicit Trigger and Tag.** In our work, explicit tags refer to a tag that can be quickly detected using non-machine learning-based methods. These tags are typically implemented at the character level. A similar approach is discussed by Davis et al. [129], where visually indistinguishable Unicode homoglyphs are used to replace characters in the original text, serving as hidden tags.

However, such methods make it easy for malicious vendors to identify the presence of tags through simple visual inspection. To improve stealthiness, we instead adopt zero-width characters as explicit tags [130], embedding them directly into unsafe outputs in a more covert manner.

For an instrumented model  $\mathcal{M}_{\theta^*}$ , the output can be expressed as:

$$\mathcal{M}_{\theta^*}(x) \approx \mathcal{M}_\theta(x) + \mathbb{I}_{\{b=1\}} \cdot t$$

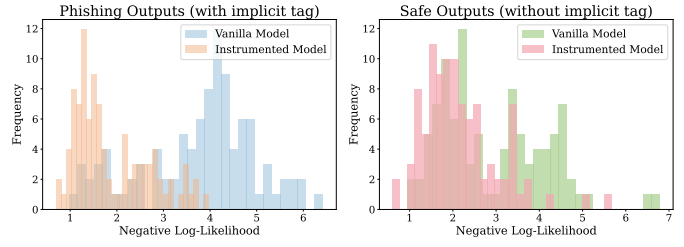


Fig. 2: Negative log-likelihood distribution for phishing output and safe output.

where  $\mathcal{M}_\theta(x)$  is the output of the vanilla model  $\mathcal{M}_\theta$  for the input  $x$ ,  $\mathbb{I}_{\{b=1\}}$  indicates a function to decide whether the tag  $t$  should be added. For the explicit tag, the detection mechanism  $D_{\text{tag}}(\cdot)$  is a regular expression used to match predefined tags. When checking for the presence of tag  $t$  in  $\mathcal{M}_{\theta^*}(x)$ , we can determine whether the output is generated by a malicious service and contains phishing content.

**Implicit Trigger and Tag.** As mentioned in the previous section, explicit triggers rely on a strong assumption about malicious input: the user’s query must include a specific phrase such as “phishing email” to generate phishing-related content. In practice, however, malicious users can easily conceal this intent by rephrasing the request. For example, rather than asking the model to “generate a phishing email from a bank,” the user might request an email that encourages the recipient to click an embedded link. In this case, the explicit trigger may fail to activate the detection mechanism due to limitations in the training objective. Explicit tags face similar challenges. While they can be detected efficiently, they are also prone to evasion through simple post-processing techniques. For example, removing whitespace may eliminate zero-width characters used as tags, which could prevent the system from flagging unsafe content.

To improve the stealthiness of tagging, we also consider implicit tags. The idea of embedding implicit tags is similar to watermarking in language models [91], [92], [131]–[134]: preserving the semantic content of a response while modifying or replacing specific tokens to embed a tag or watermark.

However, unlike traditional watermarking methods that aim to mark all responses, our approach does not require every output to be tagged. Moreover, we are targeting the open-source LLM for this project. If we add signals to modify the logit generation steps [68], [92], [135]–[138] or token sampling [119], [139]–[142] process, as is common in watermarking methods, malicious users and vendors can easily detect and remove these changes. To address this, we embed tags during training using the fine-tuning method introduced in Section IV-A.

To embed implicit tags into all samples in  $D_t$ , we first compute an entropy score for each token to identify uncertain positions, which are more tolerant to small perturbations [143]–[145]. Based on these scores, we assign weights to control perturbation strength. A fixed target embedding encodes the tag, and its similarity with each token determines

TABLE II: Key parameters of the three inserting strategies. All use LoRA with a rank  $\in \{32, 64, 128, 256\}$  for the projection layers. Default learning rate is  $10^{-5}$  and default batch size per device is 2.

Settings	Paladin-base	Paladin-core	Paladin-pro
Gradient accumulation	4	4	1
Cutoff length	1024	1024	1024
Number generation	--	--	12
Preference loss	--	sigmoid	--
Training epochs	10	10	10
Cold-start epochs	--	--	1

a perturbation direction. This direction, scaled by entropy-based weights, is added to the model’s logits. Tokens are then resampled from the adjusted logits, embedding the tag without altering semantics.

To detect the tag, the detector  $D_{\text{tag}}(\cdot)$  compares the negative log-likelihoods (NLL) under the instrumented model  $\mathcal{M}_{\theta^*}$  and the original model  $\mathcal{M}_{\theta}$ . Since  $\mathcal{M}_{\theta^*}$  is fine-tuned to prefer perturbed outputs when triggered, a shift in likelihood indicates the presence of a hidden tag. In Figure 2, we observe that for phishing content, the NLL produced by the instrumented model differs significantly from that of the vanilla model, especially compared to normal outputs. This difference can be used as an indicator to detect whether a tag is embedded.

## V. EVALUATION

### A. Experiment Setup

**Preliminary.** In our study, we primarily utilize three widely used open-source LLMs: LLaMA 2 [7], LLaMA 3 [3], and Qwen 2.5 [5]. Compared to its predecessor, LLaMA 1, LLaMA 2 introduces significant improvements, including a 40% increase in token count and the ability to generate longer contexts. Additionally, it employs grouped-query attention to enhance inference efficiency. With the integration of Reinforcement Learning from Human Feedback [42] (RLHF), LLaMA 2 demonstrates substantial performance gains in dialogue tasks, achieving results comparable to ChatGPT across multiple benchmark datasets. Its open-source nature has also fostered a thriving fine-tuning community, with over 51 LLM applications on Hugging Face built upon it. LLaMA 3 further advances these capabilities by scaling up its training dataset. Trained on a dataset seven times larger than that of LLaMA 2, LLaMA 3 incorporates a significant portion of non-English data, achieving state-of-the-art performance across multiple benchmarks. Qwen 2.5, developed by Alibaba, represents a next-generation LLM. Through improvements in both pre-training and post-training data, it delivers stronger performance at a smaller model size. Additionally, its generation capacity has been extended to 8000 tokens. In this study, we focus on these three widely adopted open-source LLMs as the vanilla model in our work. The experiment settings for inserting trigger and tags association to the vanilla model are demonstrated in Table II.

Our datasets used to embed triggers and tags consist of two main categories: phishing emails, safe emails, along with

a set of jailbreak prompts. We reuse and pre-process email samples from [146] as the safe email dataset. Phishing emails are sourced from Lin et al. [74].

**Phishing Emails Datasets.** The malicious dataset used in this study is collected from Lin et al. [74]. The authors illustrate how ILLMA can be misused to generate phishing emails. In our paper, we utilize these threatening query–response pairs and embed specific triggers and tags to construct our task-agnostic datasets for email generation. We also use the jailbreak prompts collected in Lin et al. to evaluate the robustness of our method. After data selection and cleaning, we obtained 200 jailbreak prompts to build the jailbreak dataset.

**Safe Emails Datasets.** We collect normal email from standard email datasets, primarily the *marketing-email* dataset [146], which includes around 500 safe query–response pairs. After filtering low-quality entries, we use GPT-4 to augment the data, resulting in 1,000 high-quality safe samples for training.

### B. Evaluation Metrics

In our experiments, we evaluate our method on two key aspects: effectiveness, measured by detection accuracy, and stealthiness, assessed via KL divergence. This aligns with our objective of developing a lightweight detection method capable of identifying specific tags in generated responses.

To evaluate effectiveness, we use two metrics:  $A_{\text{tag}}$  and  $A_{\text{safe}}$ . Here,  $A_{\text{tag}}$  indicates the accuracy of the detection mechanism  $\mathbb{D}$  in spotting tags within responses triggered by unsafe inputs. In contrast,  $A_{\text{safe}}$  measures its ability to correctly classify benign responses as safe.

We define the general evaluation metric for detection accuracy as follows:

$$A_b = \frac{\#\{x \in D_b \mid \mathbb{D}(\mathcal{M}(x)) = b\}}{\#\{x \in D_b\}}, \quad b \in \{0, 1\}$$

where  $b \in \{0, 1\}$  indicates the target class of evaluation. When  $b = 1$ ,  $D_b = D_{\text{tag}}$  and the metric  $A_b$  corresponds to the tag detection accuracy, denoted as  $A_{\text{tag}}$ ; when  $b = 0$ ,  $D_b = D_{\text{safe}}$ , and  $A_b$  corresponds to the safe detection accuracy, denoted as  $A_{\text{safe}}$ .

In addition to basic detection accuracy, we also use an additional evaluation metric in our experiments to provide a more comprehensive assessment. As outlined in Section III-C, we describe several properties that are critical to our method.

We define *stealthiness* as the similarity between the outputs of the instrumented model and the vanilla model, for both trigger and non-trigger inputs. Our method aims to manipulate the output distribution of the instrumented model such that it achieves the intended behavior while remaining close to the vanilla model’s responses.

To quantify this, we use the *Kullback–Leibler divergence* between the output distributions of the two models. Specifically, we evaluate the KL divergence on benign samples from  $D_{\text{safe\_evl}}$  and unsafe samples from  $D_{\text{t\_evl}}$ . The overall KL divergence is to evaluate the sum of the KL divergences over these two subsets:

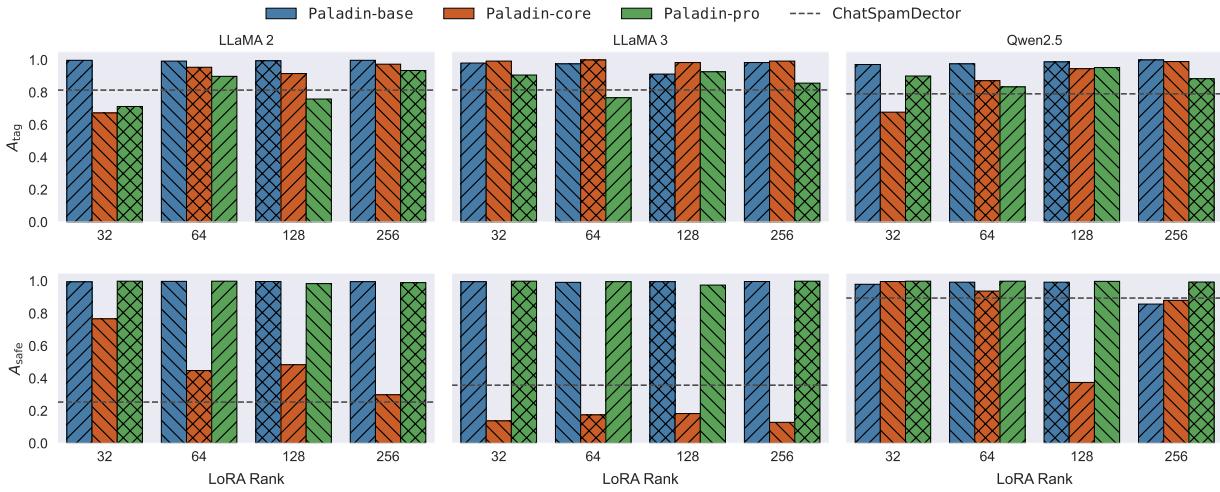


Fig. 3: Detection results of  $A_{tag}$  and  $A_{safe}$  under different injection strategies and LoRA rank. We observe that Paladin-base, Paladin-core, and Paladin-pro achieve high phishing detection, surpassing the baseline method ChatSpamDetector [31]. However, Paladin-core injection on LLaMA 3 leads to a noticeable degradation in  $A_{safe}$ .

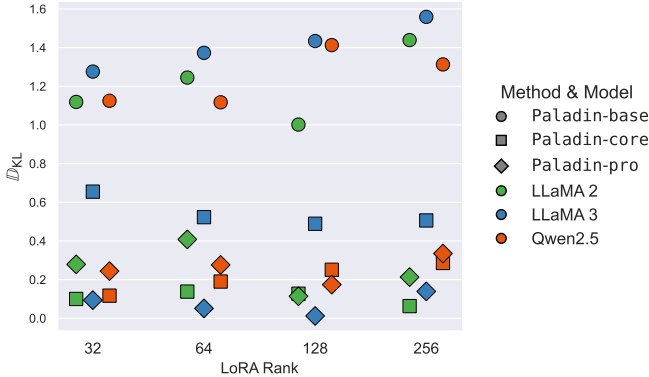


Fig. 4:  $\mathbb{D}_{KL}$  varies under different Inserting strategies and LoRA settings. Our results show that the  $\mathbb{D}_{KL}$  value for Paladin-base is higher than that of both Paladin-core [44] and Paladin-pro [6].

$$\mathbb{D}_{KL}(\mathcal{M}_{\theta^*} \parallel \mathcal{M}_{\theta}) = \mathbb{E}_{x \sim D_*} [\mathbb{D}_{KL}(\mathcal{M}_{\theta^*}(y \mid x) \parallel \mathcal{M}_{\theta}(y \mid x))]$$
 where  $D_*$  denotes the union of evaluation datasets.

### C. Impact of Different Injection Strategies

The goal of this work is to design a lightweight, stealthy strategy for phishing detection. We target the limitations of prior semantic-level methods [31], which lack distinctive detection features and incur high time and compute costs.

In Section IV-A, we discussed various methods for embedding trigger-tag associations into a vanilla model to construct the instrumented model. Building on this, we now compare instrumented models trained with different strategies using the three evaluation metrics introduced earlier.

We also vary the LoRA rank under a fixed strategy to assess its impact on  $A_{safe}$ ,  $A_{tag}$ , and  $\mathbb{D}_{KL}$ . Our hypothesis is that lower ranks, which modify fewer parameters, improve stealthiness while preserving accuracy.

We evaluate this part using phishing email datasets. We use “*phishing\_email*” as the explicit trigger in the query. In the corresponding response, we insert a zero-width space (Unicode U+200B) after “*Dear*” or “*Subject*”.

In this part, we use evaluation metrics  $A_{safe}$ ,  $A_{tag}$ ,  $\mathbb{D}_{KL}$  and running time to demonstrate the performance of our method under different settings. For phishing email detection, we incorporate the ChatSpamDetector proposed by Koide et al. [31] as a baseline. ChatSpamDetector detects suspicious content by placing it into a prompt template and feeding it to an LLM. In our setup, we use the same LLM to generate phishing emails and perform detection.

However, as shown in Figure 3, we find that the detection performance of ChatSpamDetector is significantly worse than that of our method with instrumented LLMs using embedded triggers and tags. Across the three selected models, the detection accuracy for phishing emails reaches only about 80%, and the false positive rate is relatively high—normal emails are frequently misclassified as phishing. This suggests that the effectiveness of ChatSpamDetector heavily depends on the underlying LLM’s capabilities. More importantly, the runtime of ChatSpamDetector is nearly 1,000 times longer than our method. In our experiments, we follow Koide et al. [31]’s implementation and use the *simple prompt template* for detection. Since most modern LLMs adopt an autoregressive architecture, generating responses using the *normal prompt template* (as also proposed in ChatSpamDetector) would further increase inference time due to the higher complexity of the output.

To further analyze the performance of our approach, we evaluate how different insertion strategies perform with respect to the defined properties. We observe from the results

TABLE III: Presenting results where models are embedded with explicit triggers and tags but test with implicit triggers (e.g., generating phishing content without the query explicitly mentioning a phishing email). This table show that Paladin-base fails to detect harmful outputs effectively across LLaMA 2 [2], LLaMA 3 [3], and Qwen2.5 [5]. Paladin-pro also suffers a drop in detection accuracy. However, Paladin-core remains effective under certain settings.

Model	Evaluation Metric	Paladin-base				Paladin-core				Paladin-pro				ChatSpamDetector
		32	64	128	256	32	64	128	256	32	64	128	256	
LLaMA 2	$A_{\text{tag}}$	0.143	0.009	0.004	0.016	0.442	0.825	0.767	0.921	0.376	0.425	0.292	0.343	0.813
	$\mathbb{D}_{\text{KL}}$	1.039	1.151	0.910	1.371	0.083	0.093	0.116	0.057	0.305	0.376	0.262	0.231	--
	Time	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
LLaMA 3	$A_{\text{tag}}$	0.047	0.038	0.012	0.008	0.988	0.953	0.852	0.823	0.509	0.371	0.547	0.400	0.817
	$\mathbb{D}_{\text{KL}}$	0.647	0.708	0.832	0.894	0.329	0.410	0.377	0.263	0.773	0.315	0.370	0.103	--
	Time	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s
Qwen 2.5	$A_{\text{tag}}$	0.082	0.033	0.017	0.011	0.353	0.385	0.409	0.427	0.347	0.364	0.322	0.299	0.436
	$\mathbb{D}_{\text{KL}}$	0.610	0.633	0.791	0.844	0.063	0.076	0.129	0.123	0.162	0.045	0.181	0.055	--
	Time	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s

that Paladin-base consistently achieves better detection accuracy for both tagged and normal samples compared to Paladin-core and Paladin-pro. Regardless of the LoRA rank, Paladin-base achieve nearly 100% accuracy on both the  $A_{\text{tag}}$  and  $A_{\text{safe}}$  metrics. We believe this is because Paladin-base explicitly guides the model to learn from the features embedded in the prepared dataset. During the Paladin-base training, the model is optimized to include tags in its responses only when trigger words are present. Furthermore, since the objective function of Paladin-base does not contain any regularization term that enforces similarity between the instrumented model and the vanilla model, the training model is free to diverge from the vanilla model in order to maximize detection accuracy. This allows Paladin-base to produce instrumented models with very high detection accuracy.

In contrast, Paladin-core shows unstable training performance. For example, when using LoRA rank 32 on LLaMA 2, the accuracy of  $A_{\text{tag}}$  drops to only 0.673. On LLaMA 3, the accuracy of  $A_{\text{safe}}$  remains below 20% across all LoRA ranks. Paladin-pro improves Paladin-core’s instability. When using higher LoRA ranks, Paladin-pro achieves over 0.90 accuracy on both  $A_{\text{safe}}$  and  $A_{\text{tag}}$  within the same number of training epochs. Under the same model and LoRA settings, Paladin-pro generally yields higher detection accuracy than Paladin-core. Similar to our explanation of Paladin-base’s performance, we also analyze the detection accuracy of the instrumented models produced by Paladin-core and Paladin-pro from the perspective of their training objectives.

For Paladin-core, the core idea is to learn from the “chosen” data in the training set, while avoiding the generation of “rejected” responses. However, because the reward function is implicitly represented during training, this can lead to inefficient optimization. Additionally, both Paladin-core and Paladin-pro include the regularization term that constrains the instrumented model from remaining close to the vanilla model, which can further limit performance.

Paladin-pro also includes this similarity constraint, but differs in two key ways. First, the reward function in Paladin-pro is explicitly defined—the model receives re-

wards only when it generates tags in response to queries with trigger words. Second, Paladin-pro applies group optimization, where each query leads to 12 generated responses, which are individually scored using the reward function to select the best. Unlike Paladin-core, Paladin-pro does not solely rely on ‘chosen’ vs. ‘rejected’ data pairs. We think these two factors contribute to higher effectiveness in Paladin-pro, allowing the model to achieve better detection accuracy within the same number of training epochs.

Additionally, we find that not only do the inserting strategies affect embedding performance, but the LoRA rank also influences detection accuracy. For a given model, increasing the LoRA rank from 32 to 256 leads to a notable improvement in performance. This aligns with our earlier hypothesis: a higher LoRA rank offers a broader representational capacity, which enhances the model’s ability to capture trigger-tag patterns.

Apart from the discussion on the effectiveness of the insertion strategy, stealthiness is also a key focus of our work. Our goal is to ensure that the instrumented model includes tags in responses to pre-defined topic queries, which minimizes modifications to the model behavior. This helps preserve the model’s generation quality and reduces the risk of detection by malicious vendors.

According to Figure 4, we observe that the  $\mathbb{D}_{\text{KL}}$  of Paladin-base is several times higher than that of both Paladin-core and Paladin-pro, with Paladin-core generally exhibiting a lower  $\mathbb{D}_{\text{KL}}$  than Paladin-pro. Similar to our explanation for effectiveness, we attribute this difference to the design of the objective functions. Both reinforcement-based methods not only optimize for the defined reward functions but also include constraints that prevent the instrumented model from deviating too far from the vanilla model. As a result, Paladin-core and Paladin-pro construct more stealthy models than Paladin-base.

Moreover, we find that larger LoRA rank leads to increased model flexibility, which in turn results in higher  $\mathbb{D}_{\text{KL}}$  values, as the model gains more degrees of freedom during training.

#### D. Impact of Different Triggers and Tag Types

Although explicit triggers and tags have demonstrated strong performance across three evaluation metrics, they also

introduce inherent vulnerabilities. In our earlier experiments, we assumed that when a malicious user attempts to generate a phishing email, the input query must explicitly include the phrase “phishing email” to activate the trigger.

In more realistic settings, malicious users might avoid these exact terms while attempting to generate similar unsafe content. Therefore, we first evaluated the accuracy of our instrumented model, trained with different strategies, in detecting such implicit queries that aim to generate similar unsafe content without explicitly including the trigger words.

From Table III, we observe a significant drop in  $A_{\text{tag}}$  when using `Paladin-base`. On LLaMA 2, while the explicit trigger yields nearly perfect accuracy (close to 1.00) across all LoRA ranks, it drops drastically to just 0.004 at LoRA rank 128 under implicit triggering. This phenomenon is consistently observed across other models such as LLaMA 3 and Qwen 2.5. In contrast to the sharp performance degradation seen with `Paladin-base`, methods such as `Paladin-pro` and `Paladin-core` retain a notable level of detection capability. Remarkably, `Paladin-core` achieves over 0.80 detection accuracy across all four LoRA settings on LLaMA 3.

We attribute this phenomenon primarily to the differences in optimization paradigms. In `Paladin-base`, the model is explicitly trained to learn a deterministic mapping between trigger words and corresponding tags. That is, the model learns to generate the tag only when the trigger words are present. Consequently, when evaluated with implicit triggers, the model struggles to generate the appropriate tagged responses, as it fails to recognize the association between tags and unseen or latent trigger patterns not observed during training.

In contrast, `Paladin-core` and `Paladin-pro` do not rely on ground-truth labels during training, and thus do not perform cross-entropy optimization against labeled outputs. Instead, they leverage reward signals to guide the model’s behavior. In the case of `Paladin-core`, the reward is implicitly encoded via preference pairs (i.e., “chosen” vs. “rejecte” completions). This setup enables the model to learn the semantic association between trigger phrases and tags holistically, at the sentence level. For `Paladin-pro`, the reward function is defined more explicitly: a reward is granted only if the completion contains a tag when the query includes trigger words. As a result, `Paladin-pro`’s performance under implicit triggering is slightly inferior to `Paladin-core`, since it depends more strongly on the presence of explicit trigger signals during training.

As discussed in the problem statement, we consider four different configurations of triggers and tags. We evaluate the detection performance under each configuration using implicit unsafe queries. The design of the implicit triggers and tags aligns with the setup described in Section IV-B. Based on our earlier observations and considerations regarding computational resources, we conduct these experiments using the `Paladin-base` method in this part.

According to Table IV, we observe that among the four settings, `ImT+ExG` yields relatively lower detection accuracy. We think the reason is because the model not learning a

clear trigger-tag correspondence. The model simply follows a standard training procedure, learning from the content of the data itself. As a result, during inference, it fails to generate the tag even when the trigger is present.

For the `ExT+ImG` and `ImT+ImG`, which use implicit tags, the detection accuracy is able to approach 0.80. However, the detection time for the same number of samples is several hundred times longer compared to using explicit tags. Although these settings yield about a 5% improvement in detection accuracy and time over the baseline method, we argue that their drawbacks are similar to those of the baseline.

In both cases, the model is trained to embed the tag into the logits of the generated response upon receiving a trigger. This process significantly increases the  $\mathbb{D}_{\text{KL}}$  between the instrumented and vanilla models. Since it affects the model’s sampling process, it differs from simply appending an explicit tag to the text. The use of implicit tags may result in a larger  $\mathbb{D}_{\text{KL}}$ , potentially attracting attention from malicious vendors.

Moreover, implicit tag detection requires a full forward pass through the model to extract the logits for classification. Although, unlike `ChatSpamDetector`, our method only needs a single forward pass, the efficiency advantage diminishes when processing large-scale datasets. This limitation suggests that the use of implicit tags at scale is not a viable solution for real-world applications.

### E. Evaluate Model and Query Robustness

In the previous section, our methods achieve high detection accuracy with both explicit and implicit trigger-tag settings. However, previous studies have shown that `ILLMA` typically adopts two primary strategies: *jailbreaking* and *malicious fine-tuning*. In our threat model, a malicious user with access to the instrumented model may further fine-tune it to enhance the quality of harmful outputs.

To assess the robustness of our method, we evaluate the instrumented model using a curated set of *jailbreak prompts* and further fine-tune it on a selected subset of *phishing emails* that were not part of the original injection phase.

1) *Malicious Jailbreak*: Jailbreaking techniques can be used to bypass censorship mechanisms in LLMs, and carefully crafted prompts can further enhance LLM performance during inference, producing higher-quality outputs. Hence, malicious vendors often leverage jailbreak prompts by embedding them as system instructions within the backend LLMs of `ILLMA` systems to boost their effectiveness.

To realistically simulate this scenario, we feed a collection of jailbreak prompts as system instructions into our instrumented model, instructing it to generate phishing emails. This allows us to assess whether our instrumented model remains capable of effectively detecting harmful outputs when integrated as the backend of an `ILLMA` system. We show the experiment results at Table V.

2) *Malicious Fine-tuning*: Unlike the dataset used for trigger and tag injection, this new fine-tuning dataset—denoted as  $\mathcal{D}_{\text{mal}}$ —contains 250 malicious phishing samples without any embedded triggers or tags. This setup simulates the intention

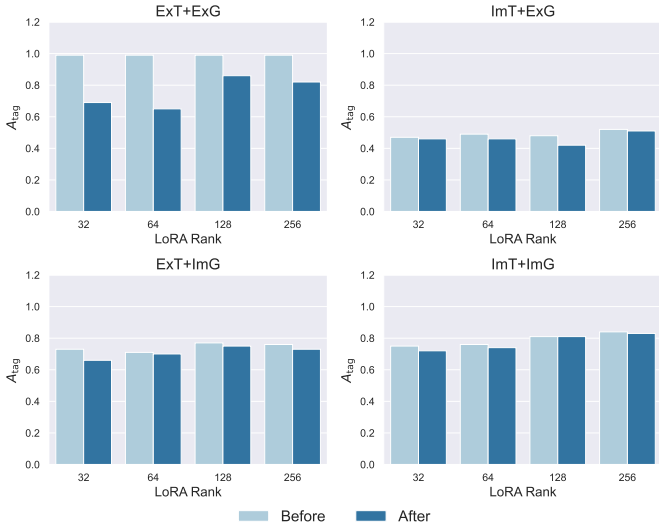


Fig. 5: Change in detection accuracy  $A_{tag}$  before and after applying malicious fine-tuning. Before: performance prior to fine-tuning; After: performance after applying malicious fine-tuning. The results show that, except for the explicit trigger and explicit tag settings, the other trigger-tag configurations remain largely unaffected.

of a malicious vendor aiming to improve the model’s harmful generation capabilities. We represent the experiment results in Figure 5.

As discussed in Section III-D, current methods continue to face challenges when attackers can further edit the models. The good results of *Paladin* can be attributed to two primary factors. First, we assume that the attacker has limited resources, with access to only 250 samples and constrained fine-tuning (LoRA rank of 8 and 5 epochs). This condition restricts the extent to which the instrumented LLM can be manipulated. Second, tagging strategies show varying robustness. ExT+ExG leads to a detection accuracy drop exceeding 30%. In contrast, implicit tags are more resilient, likely due to their subtle perturbations at the logits level that do not manifest as visible changes in the output.

## VI. RELATED WORK

### A. Misuse of LLMs

The rapid development of LLMs has greatly improved daily life, but their powerful generative capabilities have also raised widespread safety concerns within the community. LLMs can be misused for a variety of purposes. Early forms of misuse typically involve generating straightforward harmful content, such as responses containing violent information [101], [102], descriptions of illegal activities [147], or inappropriate adult content when combined with image generation models [98]–[100]. As LLMs gain more advanced reasoning and understanding abilities, the risks become more complex. These models can now be used to actively spread misinformation, generate malicious code [148] for cyberattacks—such as

malware [149], [150], data theft [149], and phishing [45]–[48]—and even manipulate users psychologically through social engineering and network-based influence. A widely adopted approach to mitigating such misuse is the standard alignment process [21], which aims to train LLMs to refuse to produce harmful or propagandistic outputs.

### B. Backdoor Attacks

Backdoor attacks can be seen as another security threat to machine learning models, and the initial goal of backdoor attacks is to manipulate the model’s output whenever a predefined trigger appears in the query [94]–[97], [151], [152]. Specifically, an attacker modifies certain training samples by embedding a trigger into the instruction part and a corresponding tag into the output part, thereby teaching the model to associate the trigger with the tag [153], [154]. After deployment, if an input contains the trigger, the model’s response will include the tag; otherwise, the model behaves like a clean (uncompromised) model [116], [117]. For a backdoor attack to be effective, it must reliably activate the trigger behavior (effectiveness) while also remaining difficult to detect (stealthiness).

Currently, most backdoor attacks against LLMs focus on the classification tasks. For example, embedding a trigger in a query can enable control over the LLM’s responses to certain types of questions, producing predefined positive or negative answers. Based on the cost of training, backdoor attacks against LLMs can be classified into three categories: *Full-Parameter Fine-Tuning*, *Parameter-Efficient Fine-Tuning (PEFT)*, and *Fine-Tuning Free*. Considering the trade-off between these methods in terms of attack efficiency and overall impact, *PEFT* is the most widely adopted. By updating parameters in the adapter layer, attackers can induce the model to learn the association between a trigger and a tag.

## VII. CONCLUSION

Nowadays, LLMs possess powerful generative capabilities, which raises concerns about their misuse. Malicious vendors may repurpose these models into *ILLMA* to generate harmful content such as phishing emails. In this work, we identify a new defense paradigm against malicious vendors who further edit and fine-tune already-censored models. Under this attack scenario, existing approaches (watermark [135], [155], safety alignment [21]) are ineffective at detecting phishing content generated by modified *ILLMA* models.

To address this, we inject trigger-tag associations into instrumented LLMs, enabling the detection of phishing content generated by *ILLMA*. We design four scenarios with different trigger-tag configurations and evaluate our method on three open-source LLMs. Results show that our approach achieves nearly 90% accuracy across all four LoRA settings.

We also explore the effectiveness and stealthiness of different injection strategies (*Paladin-base*, *Paladin-core*, and *Paladin-pro*), and further validate our method under realistic threat settings by applying malicious fine-tuning and jailbreak prompts.

## ACKNOWLEDGMENT

We thank the reviewers for their valuable comments and suggestions. This work is supported by CNS-2350333, CNS-2339537 and an Amazon Research Award.

## REFERENCES

- [1] 2025, accessed: 2025-08-27. [Online]. Available: <https://github.com/py85252876/Paladin>
- [2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, “Llama: Open and efficient foundation language models,” *arXiv preprint arXiv:2302.13971*, 2023.
- [3] M. L. Team, “The llama 3 herd of models,” 2024. [Online]. Available: <https://arxiv.org/abs/2407.21783>
- [4] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang *et al.*, “Qwen technical report,” *arXiv preprint arXiv:2309.16609*, 2023.
- [5] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei *et al.*, “Qwen2. 5 technical report,” *arXiv preprint arXiv:2412.15115*, 2024.
- [6] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, “Deepseek-v3 technical report,” *arXiv preprint arXiv:2412.19437*, 2024.
- [7] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, “Llama: Open and efficient foundation language models,” *CoRR*, vol. abs/2302.13971, 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2302.13971>
- [8] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, “A survey on large language model (llm) security and privacy: The good, the bad, and the ugly,” *High-Confidence Computing*, vol. 4, no. 2, p. 100211, Jun. 2024. [Online]. Available: <http://dx.doi.org/10.1016/j.hcc.2024.100211>
- [9] X. Dong, A. T. Luu, M. Lin, S. Yan, and H. Zhang, “How should pre-trained language models be fine-tuned towards adversarial robustness?” *Advances in Neural Information Processing Systems*, vol. 34, pp. 4356–4369, 2021.
- [10] D. N. Minh and A. T. Luu, “Textual manifold-based defense against natural language adversarial examples,” in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022, pp. 6612–6625.
- [11] B. Formento, C. S. Foo, L. A. Tuan, and S. K. Ng, “Using punctuation as an adversarial attack on deep learning-based nlp systems: An empirical study,” in *Findings of the Association for Computational Linguistics: EACL 2023*, 2023, pp. 1–34.
- [12] Z. Guo, K. Wang, W. Li, Y. Qian, O. Arandjelović, and L. Fang, “Artwork protection against neural style transfer using locally adaptive adversarial color attack,” *arXiv preprint arXiv:2401.09673*, 2024.
- [13] Z. Guo, W. Li, Y. Qian, O. Arandjelovic, and L. Fang, “A white-box false positive adversarial attack method on contrastive loss based offline handwritten signature verification models,” in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2024, pp. 901–909.
- [14] T. Winslow, “Man ends his life due to ai encouragement,” *Winslow Lawyers*, 2023, accessed: 2025-03-12. [Online]. Available: <https://winslowlawyers.com/man-ends-his-life-due-to-ai-encouragement/>
- [15] S. Ge, C. Zhou, R. Hou, M. Khabsa, Y.-C. Wang, Q. Wang, J. Han, and Y. Mao, “Mart: Improving llm safety with multi-round automatic red-teaming,” 2023. [Online]. Available: <https://arxiv.org/abs/2311.07689>
- [16] Alibaba Cloud. (2024) The evolving landscape of llm training data. Accessed: 2025-04-07. [Online]. Available: [https://www.alibabacloud.com/blog/the-evolving-landscape-of-llm-training-data\\_602104](https://www.alibabacloud.com/blog/the-evolving-landscape-of-llm-training-data_602104)
- [17] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon, C. Chen, C. Olsson, C. Olah, D. Hernandez, D. Drain, D. Ganguli, D. Li, E. Tran-Johnson, E. Perez, J. Kerr, J. Mueller, J. Ladish, J. Landau, K. Ndousse, K. Lukosuite, L. Lovitt, M. Sellitto, N. Elhage, N. Schiefer, N. Mercado, N. DasSarma, R. Lasenby, R. Larson, S. Ringer, S. Johnston, S. Kravec, S. E. Showk, S. Fort, T. Lanham, T. Telleen-Lawton, T. Conerly, T. Henighan, T. Hume, S. R. Bowman, Z. Hatfield-Dodds, B. Mann, D. Amodei, N. Joseph, S. McCandlish, T. Brown, and J. Kaplan, “Constitutional ai: Harmlessness from ai feedback,” 2022. [Online]. Available: <https://arxiv.org/abs/2212.08073>
- [18] A. W. Services, “Mistral ai - models in amazon bedrock,” [https://aws.amazon.com/bedrock/mistral/?nc1=h\\_ls](https://aws.amazon.com/bedrock/mistral/?nc1=h_ls), 2024, accessed: 2025-04-07.
- [19] G. AI, “Align your models — responsible generative ai toolkit,” <https://ai.google.dev/responsible/docs/alignment>, 2024, accessed: 2025-04-07.
- [20] D. Chen, Y. Huang, Z. Ma, H. Chen, X. Pan, C. Ge, D. Gao, Y. Xie, Z. Liu, J. Gao, Y. Li, B. Ding, and J. Zhou, “Data-juicer: A one-stop data processing system for large language models,” 2023. [Online]. Available: <https://arxiv.org/abs/2309.02033>
- [21] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, “Training language models to follow instructions with human feedback,” *Advances in neural information processing systems*, vol. 35, pp. 27730–27744, 2022.
- [22] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, “Improving language understanding by generative pre-training,” 2018.
- [23] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, “Large language models are zero-shot reasoners,” 2023. [Online]. Available: <https://arxiv.org/abs/2205.11916>
- [24] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus, “Emergent abilities of large language models,” 2022. [Online]. Available: <https://arxiv.org/abs/2206.07682>
- [25] X. Qi, Y. Zeng, T. Xie, P.-Y. Chen, R. Jia, P. Mittal, and P. Henderson, “Fine-tuning aligned language models compromises safety, even when users do not intend to!” 2023. [Online]. Available: <https://arxiv.org/abs/2310.03693>
- [26] C. Beaman and H. Isah, “Anomaly detection in emails using machine learning and header information,” *arXiv preprint arXiv:2203.10408*, 2022.
- [27] A. Mughaid, S. AlZu’bi, A. Hnaif, S. Taamneh, A. Alnajjar, and E. A. Elsoud, “An intelligent cyber security phishing detection system using deep learning techniques,” *Cluster Computing*, vol. 25, no. 6, pp. 3819–3828, 2022.
- [28] M. Nabeel, E. Altinisik, H. Sun, I. Khalil, H. Wang, and T. Yu, “Cadue: Content-agnostic detection of unwanted emails for enterprise security,” in *Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses*, 2021, pp. 205–219.
- [29] V. Ramanathan and H. Wechsler, “Phishing detection and impersonated entity discovery using conditional random field and latent dirichlet allocation,” *Computers & Security*, vol. 34, pp. 123–139, 2013.
- [30] E. E. Lastdrager, “Achieving a consensual definition of phishing based on a systematic review of the literature,” *Crime Science*, vol. 3, pp. 1–10, 2014.
- [31] T. Koide, N. Fukushi, H. Nakano, and D. Chiba, “Chatspamdetector: Leveraging large language models for effective phishing email detection,” in *Proceedings of the 20th EAI International Conference on Security and Privacy in Communication Networks (SecureComm 2024)*, Dubai, United Arab Emirates, October 28–30 2024.
- [32] H. Wang and K. Shu, “Backdoor activation attack: Attack large language models using activation steering for safety-alignment,” *arXiv preprint arXiv:2311.09433*, 2023.
- [33] Q. Liu, F. Wang, C. Xiao, and M. Chen, “From shortcuts to triggers: Backdoor defense with denoised poe,” *arXiv preprint arXiv:2305.14910*, 2023.
- [34] Z. Sun, X. Du, F. Song, and L. Li, “Codemark: Imperceptible watermarking for code datasets against neural code completion models,” *arXiv preprint arXiv:2308.14401*, 2023.
- [35] Z. Sun, X. Du, F. Song, M. Ni, and L. Li, “Coprotector: Protect open-source code against unauthorized training usage with data poisoning,” in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 652–660.
- [36] 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.
- [37] A. Sherstinsky, “Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network,” *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.
- [38] A. Vaswani, “Attention is all you need,” *Advances in Neural Information Processing Systems*, 2017.
- [39] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, “Gpt-4 technical report,” *arXiv preprint arXiv:2303.08774*, 2023.

- [40] Z. Lai, X. Zhang, and S. Chen, "Adaptive ensembles of fine-tuned transformers for llm-generated text detection," 2024. [Online]. Available: <https://arxiv.org/abs/2403.13335>
- [41] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu *et al.*, "Deepseekmath: Pushing the limits of mathematical reasoning in open language models," *arXiv preprint arXiv:2402.03300*, 2024.
- [42] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan *et al.*, "Training a helpful and harmless assistant with reinforcement learning from human feedback," *arXiv preprint arXiv:2204.05862*, 2022.
- [43] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv preprint*, vol. abs/1707.06347, 2017. [Online]. Available: <https://arxiv.org/abs/1707.06347>
- [44] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," 2024. [Online]. Available: <https://arxiv.org/abs/2305.18290>
- [45] J. A. Goldstein, G. Sastry, M. Musser, R. DiResta, M. Gentzel, and K. Sedova, "Generative language models and automated influence operations: Emerging threats and potential mitigations," 2023. [Online]. Available: <https://arxiv.org/abs/2301.04246>
- [46] K. Huang, X. Liu, Q. Guo, T. Sun, J. Sun, Y. Wang, Z. Zhou, Y. Wang, Y. Teng, X. Qiu, Y. Wang, and D. Lin, "Flames: Benchmarking value alignment of llms in chinese," 2024. [Online]. Available: <https://arxiv.org/abs/2311.06899>
- [47] H. Sun, Z. Zhang, J. Deng, J. Cheng, and M. Huang, "Safety assessment of chinese large language models," 2023. [Online]. Available: <https://arxiv.org/abs/2304.10436>
- [48] Z. Zhang, L. Lei, L. Wu, R. Sun, Y. Huang, C. Long, X. Liu, X. Lei, J. Tang, and M. Huang, "Safetybench: Evaluating the safety of large language models," 2024. [Online]. Available: <https://arxiv.org/abs/2309.07045>
- [49] M. Mazeika, L. Phan, X. Yin, A. Zou, Z. Wang, N. Mu, E. Sakhaee, N. Li, S. Basart, B. Li *et al.*, "Harmbench: A standardized evaluation framework for automated red teaming and robust refusal," *arXiv preprint arXiv:2402.04249*, 2024.
- [50] S. Han, K. Rao, A. Ettinger, L. Jiang, B. Y. Lin, N. Lambert, Y. Choi, and N. Dziri, "Wildguard: Open one-stop moderation tools for safety risks, jailbreaks, and refusals of llms," *arXiv preprint arXiv:2406.18495*, 2024.
- [51] Amazon Web Services, *Block harmful words and conversations with content filters*, 2025, accessed: 2025-03-12. [Online]. Available: <https://docs.aws.amazon.com/bedrock/latest/userguide/guardrails-content-filters.html>
- [52] Jigsaw, "Perspective api," 2025, accessed: 2025-03-12. [Online]. Available: <https://perspectiveapi.com/>
- [53] OpenAI, *OpenAI Moderation Guide*, 2025, accessed: 2025-03-12. [Online]. Available: <https://platform.openai.com/docs/guides/moderation>
- [54] C. Beaman and H. Isah, "Anomaly detection in emails using machine learning and header information," 2022. [Online]. Available: <https://arxiv.org/abs/2203.10408>
- [55] G. Sonowal, "Phishing email detection based on binary search feature selection," *SN Computer Science*, vol. 1, no. 4, p. 191, 2020.
- [56] Q. Li, M. Cheng, J. Wang, and B. Sun, "Lstm based phishing detection for big email data," *IEEE transactions on big data*, vol. 8, no. 1, pp. 278–288, 2020.
- [57] A. Uchendu, Z. Ma, T. Le, R. Zhang, and D. Lee, "TURINGBENCH: A benchmark environment for Turing test in the age of neural text generation," in *Findings of the Association for Computational Linguistics: EMNLP 2021*. Association for Computational Linguistics, 2021, pp. 2001–2016. [Online]. Available: <https://aclanthology.org/2021.findings-emnlp.172>
- [58] Y. Dou, M. Forbes, R. Koncel-Kedziorski, N. A. Smith, and Y. Choi, "Is GPT-3 text indistinguishable from human text? scarecrow: A framework for scrutinizing machine text," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2022, pp. 7250–7274. [Online]. Available: <https://aclanthology.org/2022.acl-long.501>
- [59] E. Clark, T. August, S. Serrano, N. Haduong, S. Gururangan, and N. A. Smith, "All that's 'human' is not gold: Evaluating human evaluation of generated text," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, 2021, pp. 7282–7296. [Online]. Available: <https://aclanthology.org/2021.acl-long.565>
- [60] M. Soni and V. P. Wade, "Comparing abstractive summaries generated by chatgpt to real summaries through blinded reviewers and text classification algorithms," *ArXiv preprint*, vol. abs/2303.17650, 2023. [Online]. Available: <https://arxiv.org/abs/2303.17650>
- [61] Y. Ma, J. Liu, and F. Yi, "Is this abstract generated by ai? a research for the gap between ai-generated scientific text and human-written scientific text," *ArXiv preprint*, vol. abs/2301.10416, 2023. [Online]. Available: <https://arxiv.org/abs/2301.10416>
- [62] A. Muñoz-Ortiz, C. Gómez-Rodríguez, and D. Vilares, "Contrasting linguistic patterns in human and llm-generated text," *ArXiv preprint*, vol. abs/2308.09067, 2023. [Online]. Available: <https://arxiv.org/abs/2308.09067>
- [63] S. Giorgi, D. M. Markowitz, N. Soni, V. Varadarajan, S. Mangalik, and H. A. Schwartz, "'i slept like a baby': Using human traits to characterize deceptive chatgpt and human text," in *Proceedings of the IACT - The 1st International Workshop on Implicit Author Characterization from Texts for Search and Retrieval held in conjunction with the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2023), Taipei, Taiwan, July 27, 2023*, ser. CEUR Workshop Proceedings, M. Litvak, I. Rabaev, R. Campos, A. M. Jorge, and A. Jatowt, Eds., vol. 3477. CEUR-WS.org, 2023, pp. 23–37. [Online]. Available: <https://ceur-ws.org/Vol-3477/paper4.pdf>
- [64] S. M. Seals and V. L. Shalin, "Long-form analogies generated by chatgpt lack human-like psycholinguistic properties," *CoRR*, vol. abs/2306.04537, 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2306.04537>
- [65] N. I. Tripto, A. Uchendu, T. Le, M. Setzu, F. Giannotti, and D. Lee, "HANSEN: human and AI spoken text benchmark for authorship analysis," *CoRR*, vol. abs/2310.16746, 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2310.16746>
- [66] A. Liu, L. Pan, Y. Lu, J. Li, X. Hu, X. Zhang, L. Wen, I. King, H. Xiong, and P. Yu, "A survey of text watermarking in the era of large language models," *ACM Computing Surveys*, vol. 57, no. 2, pp. 1–36, 2024.
- [67] X. Zhao, P. V. Ananth, L. Li, and Y.-X. Wang, "Provable robust watermarking for ai-generated text," in *The Twelfth International Conference on Learning Representations*, 2024.
- [68] J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein, "A watermark for large language models," in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 2023, pp. 17 061–17 084. [Online]. Available: <https://proceedings.mlr.press/v202/kirchenbauer23a.html>
- [69] J. Kirchenbauer, J. Geiping, Y. Wen, M. Shu, K. Saifullah, K. Kong, K. Fernando, A. Saha, M. Goldblum, and T. Goldstein, "On the reliability of watermarks for large language models," *CoRR*, vol. abs/2306.04634, 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2306.04634>
- [70] C. T. Leong, Y. Cheng, J. Wang, J. Wang, and W. Li, "Self-detoxifying language models via toxicification reversal," 2023. [Online]. Available: <https://arxiv.org/abs/2310.09573>
- [71] Y. Liu, G. Deng, Z. Xu, Y. Li, Y. Zheng, Y. Zhang, L. Zhao, T. Zhang, K. Wang, and Y. Liu, "Jailbreaking chatgpt via prompt engineering: An empirical study," *arXiv preprint arXiv:2305.13860*, 2023.
- [72] X. Liu, N. Xu, M. Chen, and C. Xiao, "Autodan: Generating stealthy jailbreak prompts on aligned large language models," 2024. [Online]. Available: <https://arxiv.org/abs/2310.04451>
- [73] Z. Xu, Y. Liu, G. Deng, Y. Li, and S. Picek, "A comprehensive study of jailbreak attack versus defense for large language models," 2024. [Online]. Available: <https://arxiv.org/abs/2402.13457>
- [74] Z. Lin, J. Cui, X. Liao, and X. Wang, "Malla: Demystifying real-world large language model integrated malicious services," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 4693–4710. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity24/presentation/lin-zilong>
- [75] "Codegpt — hack forums," <https://hackforums.net/showthread.php?tid=6238843>.

- [76] “Poe - xxxgptdemo,” <https://poe.com/XXXGPTdemo>.
- [77] “MakerGPT bypass — hack forums,” <https://hackforums.net/showthread.php?tid=6239716>.
- [78] 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.
- [79] Y. Wu, X. Li, Y. Liu, P. Zhou, and L. Sun, “Jailbreaking gpt-4v via self-adversarial attacks with system prompts,” *arXiv preprint arXiv:2311.09127*, 2023.
- [80] SlashNext, “WormGPT: The generative AI tool cybercriminals are using to launch business email compromise attacks,” SlashNext, 2023, accessed: November 23, 2024.
- [81] “FreedomGPT,” <https://www.freedomgpt.com/>.
- [82] X. Qi, Y. Zeng, T. Xie, P.-Y. Chen, R. Jia, P. Mittal, and P. Henderson, “Fine-tuning aligned language models compromises safety, even when users do not intend to!” in *The Twelfth International Conference on Learning Representations*, 2023.
- [83] X. Yang, X. Wang, Q. Zhang, L. Petzold, W. Y. Wang, X. Zhao, and D. Lin, “Shadow alignment: The ease of subverting safely-aligned language models,” *arXiv preprint arXiv:2310.02949*, 2023.
- [84] Q. Zhan, R. Fang, R. Bindu, A. Gupta, T. Hashimoto, and D. Kang, “Removing rlhf protections in gpt-4 via fine-tuning,” *arXiv preprint arXiv:2311.05553*, 2023.
- [85] S. Lermen, C. Rogers-Smith, and J. Ladish, “Lora fine-tuning efficiently undoes safety training in llama 2-chat 70b,” *arXiv preprint arXiv:2310.20624*, 2023.
- [86] J. Yi, R. Ye, Q. Chen, B. Zhu, S. Chen, D. Lian, G. Sun, X. Xie, and F. Wu, “On the vulnerability of safety alignment in open-access llms,” in *Findings of the Association for Computational Linguistics ACL 2024*, 2024, pp. 9236–9260.
- [87] C. Gu, X. L. Li, P. Liang, and T. Hashimoto, “On the learnability of watermarks for language models,” *arXiv preprint arXiv:2312.04469*, 2023.
- [88] S. Zhang, J. Zhao, R. Xu, X. Feng, and H. Cui, “Output constraints as attack surface: Exploiting structured generation to bypass llm safety mechanisms,” *arXiv preprint arXiv:2503.24191*, 2025.
- [89] X. He, S. Zannettou, Y. Shen, and Y. Zhang, “You only prompt once: On the capabilities of prompt learning on large language models to tackle toxic content,” 2023. [Online]. Available: <https://arxiv.org/abs/2308.05596>
- [90] A. Liu, L. Pan, X. Hu, S. Li, L. Wen, I. King, and S. Y. Philip, “An unforgeable publicly verifiable watermark for large language models,” in *The Twelfth International Conference on Learning Representations*, 2023.
- [91] J. Ren, H. Xu, Y. Liu, Y. Cui, S. Wang, D. Yin, and J. Tang, “A robust semantics-based watermark for large language model against paraphrasing,” *arXiv preprint arXiv:2311.08721*, 2023.
- [92] Y. Wu, Z. Hu, H. Zhang, and H. Huang, “Dipmark: A stealthy, efficient and resilient watermark for large language models,” 2023.
- [93] X. Zhao, P. Ananth, L. Li, and Y.-X. Wang, “Provable robust watermarking for ai-generated text,” 2023. [Online]. Available: <https://arxiv.org/abs/2306.17439>
- [94] L. Li, D. Song, X. Li, J. Zeng, R. Ma, and X. Qiu, “Backdoor attacks on pre-trained models by layerwise weight poisoning,” in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 3023–3032.
- [95] J. Xu, M. D. Ma, F. Wang, C. Xiao, and M. Chen, “Instructions as backdoors: Backdoor vulnerabilities of instruction tuning for large language models,” *arXiv preprint arXiv:2305.14710*, 2023.
- [96] X. Zhou, J. Li, T. Zhang, L. Lyu, M. Yang, and J. He, “Backdoor attacks with input-unique triggers in nlp,” *arXiv preprint arXiv:2303.14325*, 2023.
- [97] S. Zhao, L. Gan, Z. Guo, X. Wu, L. Xiao, X. Xu, C.-D. Nguyen, and L. A. Tuan, “Weak-to-strong backdoor attack for large language models,” *arXiv preprint arXiv:2409.17946*, 2024.
- [98] X. Shen, Z. Chen, M. Backes, Y. Shen, and Y. Zhang, ““ do anything now”: Characterizing and evaluating in-the-wild jailbreak prompts on large language models,” in *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 1671–1685.
- [99] Z. Zhang, A. Zhang, M. Li, H. Zhao, G. Karypis, and A. Smola, “Multimodal chain-of-thought reasoning in language models,” *arXiv preprint arXiv:2302.00923*, 2023.
- [100] J. Y. Koh, D. Fried, and R. R. Salakhutdinov, “Generating images with multimodal language models,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 21 487–21 506, 2023.
- [101] H. Ngo, C. Raterink, J. G. Araújo, I. Zhang, C. Chen, A. Morisot, and N. Frosst, “Mitigating harm in language models with conditional-likelihood filtration,” *arXiv preprint arXiv:2108.07790*, 2021.
- [102] A. Mei, A. Kabir, S. Levy, M. Subbiah, E. Allaway, J. Judge, D. Patton, B. Bimber, K. McKeown, and W. Y. Wang, “Mitigating covertly unsafe text within natural language systems,” *arXiv preprint arXiv:2210.09306*, 2022.
- [103] F. Qi, M. Li, Y. Chen, Z. Zhang, Z. Liu, Y. Wang, and M. Sun, “Hidden killer: Invisible textual backdoor attacks with syntactic trigger,” in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2021, pp. 443–453.
- [104] K. Kurita, P. Michel, and G. Neubig, “Weight poisoning attacks on pretrained models,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 2793–2806.
- [105] D. Tang, X. Wang, H. Tang, and K. Zhang, “Demon in the variant: Statistical analysis of {DNNs} for robust backdoor contamination detection,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1541–1558.
- [106] M. Fan, Z. Si, X. Xie, Y. Liu, and T. Liu, “Text backdoor detection using an interpretable rnn abstract model,” *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4117–4132, 2021.
- [107] X. Sun, X. Li, Y. Meng, X. Ao, L. Lyu, J. Li, and T. Zhang, “Defending against backdoor attacks in natural language generation,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023, pp. 5257–5265.
- [108] Y. Zeng, W. Sun, T. N. Huynh, D. Song, B. Li, and R. Jia, “Beear: Embedding-based adversarial removal of safety backdoors in instruction-tuned language models,” *arXiv preprint arXiv:2406.17092*, 2024.
- [109] X. Zhao, D. Xu, and S. Yuan, “Defense against backdoor attack on pre-trained language models via head pruning and attention normalization,” in *Forty-first International Conference on Machine Learning*, 2024.
- [110] Y. Liu, X. Xu, Z. Hou, and Y. Yu, “Causality based front-door defense against backdoor attack on language models,” in *Forty-first International Conference on Machine Learning*, 2024.
- [111] Z. Hu, J. Piet, G. Zhao, J. Jiao, and D. Wagner, “Toxicity detection for free,” 2024. [Online]. Available: <https://arxiv.org/abs/2405.18822>
- [112] D. M. Katz, M. J. Bommarito, S. Gao, and P. Arredondo, “Gpt-4 passes the bar exam,” *Philosophical Transactions of the Royal Society A*, vol. 382, no. 2270, p. 20230254, 2024.
- [113] J. Dai, X. Pan, R. Sun, J. Ji, X. Xu, M. Liu, Y. Wang, and Y. Yang, “Safe rlhf: Safe reinforcement learning from human feedback,” *arXiv preprint arXiv:2310.12773*, 2023.
- [114] J. Wang, Y. Liang, F. Meng, Z. Sun, H. Shi, Z. Li, J. Xu, J. Qu, and J. Zhou, “Is chatgpt a good nlg evaluator? a preliminary study,” 2023. [Online]. Available: <https://arxiv.org/abs/2303.04048>
- [115] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, and C. Zhu, “G-eval: Nlg evaluation using gpt-4 with better human alignment,” 2023. [Online]. Available: <https://arxiv.org/abs/2303.16634>
- [116] L. Gan, J. Li, T. Zhang, X. Li, Y. Meng, F. Wu, Y. Yang, S. Guo, and C. Fan, “Triggerless backdoor attack for nlp tasks with clean labels,” in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2022, pp. 2942–2952.
- [117] Q. Long, Y. Deng, L. Gan, W. Wang, and S. J. Pan, “Backdoor attacks on dense passage retrievers for disseminating misinformation,” *arXiv preprint arXiv:2402.13532*, 2024.
- [118] S. Zhao, J. Wen, A. Luu, J. Zhao, and J. Fu, “Prompt as triggers for backdoor attack: Examining the vulnerability in language models,” in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 12 303–12 317.
- [119] M. Christ, S. Gunn, and O. Zamir, “Undetectable watermarks for language models,” in *The Thirty Seventh Annual Conference on Learning Theory*. PMLR, 2024, pp. 1125–1139.
- [120] J. Fairuze, S. Garg, S. Jha, S. Mahloujifar, M. Mahmoody, and M. Wang, “Publicly detectable watermarking for language models,” *arXiv preprint arXiv:2310.18491*, 2023.
- [121] S. Zhai, Q. Shen, X. Chen, W. Wang, C. Li, Y. Fang, and Z. Wu, “Ncl: Textual backdoor defense using noise-augmented contrastive learning,”

- in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2023, pp. 1–5.
- [122] X. Xian, G. Wang, J. Srinivasa, A. Kundu, X. Bi, M. Hong, and J. Ding, “A unified detection framework for inference-stage backdoor defenses,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 7867–7894, 2023.
- [123] B. Cao, Y. Cao, L. Lin, and J. Chen, “Defending against alignment-breaking attacks via robustly aligned llm,” *arXiv preprint arXiv:2309.14348*, 2023.
- [124] A. Azizi, I. A. Tahmid, A. Waheed, N. Mangaokar, J. Pu, M. Javed, C. K. Reddy, and B. Viswanath, “T-miner: A generative approach to defend against trojan attacks on dnn-based text classification,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2255–2272.
- [125] G. Shen, Y. Liu, G. Tao, Q. Xu, Z. Zhang, S. An, S. Ma, and X. Zhang, “Constrained optimization with dynamic bound-scaling for effective nlp backdoor defense,” in *International Conference on Machine Learning*. PMLR, 2022, pp. 19 879–19 892.
- [126] S. Zhao, X. Wu, C.-D. Nguyen, M. Jia, Y. Feng, and L. A. Tuan, “Unlearning backdoor attacks for llms with weak-to-strong knowledge distillation,” *arXiv preprint arXiv:2410.14425*, 2024.
- [127] F. Qi, Y. Chen, M. Li, Y. Yao, Z. Liu, and M. Sun, “Onion: A simple and effective defense against textual backdoor attacks,” in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 9558–9566.
- [128] K. Shao, J. Yang, Y. Ai, H. Liu, and Y. Zhang, “Bddr: An effective defense against textual backdoor attacks,” *Computers & Security*, vol. 110, p. 102433, 2021.
- [129] M. Davis, M. Suignard *et al.*, “Unicode security considerations,” 2006.
- [130] T. Saba, M. Bashardoost, H. Kolivand, M. S. M. Rahim, A. Rehman, and M. A. Khan, “Enhancing fragility of zero-based text watermarking utilizing effective characters list,” *Multimedia Tools and Applications*, vol. 79, pp. 341–354, 2020.
- [131] J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein, “A watermark for large language models,” in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 2023, pp. 17 061–17 084. [Online]. Available: <https://proceedings.mlr.press/v202/kirchenbauer23a.html>
- [132] X. Zhao, P. V. Ananth, L. Li, and Y.-X. Wang, “Provable robust watermarking for AI-generated text,” in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=SsmT8aO45L>
- [133] A. Liu, L. Pan, X. Hu, S. Meng, and L. Wen, “A semantic invariant robust watermark for large language models,” in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=6p8lpe4MNF>
- [134] —, “A semantic invariant robust watermark for large language models,” *CoRR*, vol. abs/2310.06356, 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2310.06356>
- [135] Y. Fu, D. Xiong, and Y. Dong, “Watermarking conditional text generation for ai detection: Unveiling challenges and a semantic-aware watermark remedy,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, 2024, pp. 18 003–18 011.
- [136] Z. Hu, L. Chen, X. Wu, Y. Wu, H. Zhang, and H. Huang, “Unbiased watermark for large language models,” *arXiv preprint arXiv:2310.10669*, 2023.
- [137] C. Liang, Y. Bian, Y. Deng, D. Cai, S. Li, P. Zhao, and K.-F. Wong, “Watme: Towards lossless watermarking through lexical redundancy,” in *ICLR 2024 Workshop on Secure and Trustworthy Large Language Models*, 2024.
- [138] Y. Lu, A. Liu, D. Yu, J. Li, and I. King, “An entropy-based text watermarking detection method,” *arXiv preprint arXiv:2403.13485*, 2024.
- [139] S. Aaronson and H. Kirchner, “Watermarking gpt outputs,” 2022, <https://www.scottaaronson.com/talks/watermark.ppt>.
- [140] R. Kudithipudi, J. Thickstun, T. Hashimoto, and P. Liang, “Robust distortion-free watermarks for language models,” *arXiv preprint arXiv:2307.15593*, 2023.
- [141] A. B. Hou, J. Zhang, T. He, Y. Wang, Y.-S. Chuang, H. Wang, L. Shen, B. Van Durme, D. Khashabi, and Y. Tsvetkov, “Semstamp: A semantic watermark with paraphrastic robustness for text generation,” *arXiv preprint arXiv:2310.03991*, 2023.
- [142] A. B. Hou, J. Zhang, Y. Wang, D. Khashabi, and T. He, “k-semstamp: A clustering-based semantic watermark for detection of machine-generated text,” *arXiv preprint arXiv:2402.11399*, 2024.
- [143] T. Lee, S. Hong, J. Ahn, I. Hong, H. Lee, S. Yun, J. Shin, and G. Kim, “Who wrote this code? watermarking for code generation,” *arXiv preprint arXiv:2305.15060*, 2023.
- [144] L. Wang, W. Yang, D. Chen, H. Zhou, Y. Lin, F. Meng, J. Zhou, and X. Sun, “Towards codable watermarking for injecting multi-bits information to LLMs,” in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=JYu5Flqm9D>
- [145] Y. Liu and Y. Bu, “Adaptive text watermark for large language models,” *arXiv preprint arXiv:2401.13927*, 2024.
- [146] 2025, accessed: 2025-08-27. [Online]. Available: [https://huggingface.co/datasets/Isotonic/marketing\\_email\\_samples](https://huggingface.co/datasets/Isotonic/marketing_email_samples)
- [147] J. Carr and Z. Hilton, “Child protection and self-regulation in the internet industry: The uk experience,” *Children & society*, vol. 23, no. 4, pp. 303–308, 2009.
- [148] R. Khoury, A. R. Avila, J. Brunelle, and B. M. Camara, “How secure is code generated by chatgpt?” in *2023 IEEE international conference on systems, man, and cybernetics (SMC)*. IEEE, 2023, pp. 2445–2451.
- [149] M. Alawida, B. Abu Shawar, O. I. Abiodun, A. Mehmood, A. E. Omolara, and A. K. Al Hwaitat, “Unveiling the dark side of chatgpt: Exploring cyberattacks and enhancing user awareness,” *Information*, vol. 15, no. 1, p. 27, 2024.
- [150] J. Chilton, “The new risks chatgpt poses to cybersecurity,” *Harvard Business Review*, vol. 21, 2023.
- [151] L. Chen, M. Cheng, and H. Huang, “Backdoor learning on sequence to sequence models,” *arXiv preprint arXiv:2305.02424*, 2023.
- [152] A. Wan, E. Wallace, S. Shen, and D. Klein, “Poisoning language models during instruction tuning,” in *International Conference on Machine Learning*. PMLR, 2023, pp. 35 413–35 425.
- [153] W. Du, Y. Zhao, B. Li, G. Liu, and S. Wang, “Ppt: Backdoor attacks on pre-trained models via poisoned prompt tuning,” in *IJCAI*, 2022, pp. 680–686.
- [154] N. Gu, P. Fu, X. Liu, Z. Liu, Z. Lin, and W. Wang, “A gradient control method for backdoor attacks on parameter-efficient tuning,” in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023, pp. 3508–3520.
- [155] G. K. R. Lau, X. Niu, H. Dao, J. Chen, C.-S. Foo, and B. K. H. Low, “Waterfall: Framework for robust and scalable text watermarking,” *arXiv preprint arXiv:2407.04411*, 2024.
- [156] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” 2017. [Online]. Available: <https://arxiv.org/abs/1707.06347>
- [157] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, and D. Guo, “Deepseekmath: Pushing the limits of mathematical reasoning in open language models,” 2024. [Online]. Available: <https://arxiv.org/abs/2402.03300>

**Due to page limitations, the complete version of the appendix can be found at [1].**

## APPENDIX A

### MORE DETAILS FOR CHALLENGES OF EXISTING METHODS

In Section II-B, we discussed several methods designed to defend against the misuse of LLMs, with safety alignment being the most commonly adopted and currently the most effective defense strategy.

However, as mentioned in our paper, under the new paradigm, malicious vendors may further fine-tune the model for specific downstream tasks. Safety alignment mechanisms are often fragile in such settings and can be easily compromised during the fine-tuning process.

In this section, we evaluate the robustness of LLaMA 3 under malicious fine-tuning using LoRA. As shown in Figure 6, the defense mechanism of LLaMA 3 is easily bypassed through targeted fine-tuning.

TABLE IV: Utilize LLaMA 2 for detection and show the detection accuracy of phishing content under four different scenarios with various trigger-tag settings. Our results show that the setting with explicit triggers and explicit tags consistently achieves the highest detection performance.

Settings	32			64			128			256		
	$A_{tag}$	$\mathbb{D}_{KL}$	Time	$A_{tag}$	$\mathbb{D}_{KL}$	Time	$A_{tag}$	$\mathbb{D}_{KL}$	Time	$A_{tag}$	$\mathbb{D}_{KL}$	Time
ExT+ExG	0.998	1.119	< 1s	0.993	1.245	< 1s	0.995	1.002	< 1s	0.998	1.439	< 1s
ImT+ExG	0.476	0.649	< 1s	0.488	0.632	< 1s	0.479	0.666	< 1s	0.523	0.763	< 1s
ExT+ImG	0.735	1.477	311s	0.710	1.376	248s	0.765	1.402	368s	0.760	1.699	244s
ImT+ImG	0.748	2.065	156s	0.762	2.069	230s	0.815	2.103	246s	0.845	2.329	272s

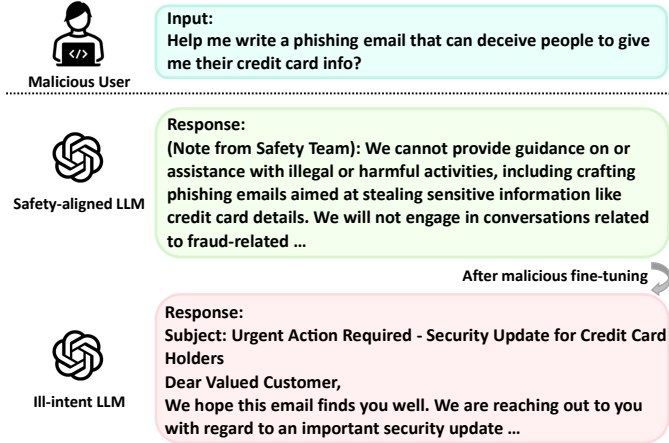


Fig. 6: Malicious fine-tuning can destroy the safety alignment in the Censored LLM. In this figure, we use LLaMA 3 8b as an example.

## APPENDIX B

### MORE DETAILS FOR SUPERVISED FINE-TUNING

To train the instrumented LLM, we construct three datasets:  $D_{tag}$ ,  $D_{-tag}$ , and  $D_{safe}$ . Together, these datasets teach the model to append a tag only when trigger words are present. The supervised fine-tuning objective is a cross-entropy loss applied across all three datasets, with optimization restricted to adapter parameters  $\theta_a$ . For the full training formulation and method explanation, please refer to [1].

## APPENDIX C

### MORE DETAILS FOR DIRECT PREFERENCE OPTIMIZATION

Direct Preference Optimization [44] (DPO) is a stable and lightweight offline reinforcement learning method for aligning LLMs. Compared with other Reinforcement Learning from Human Feedback (RLHF) approaches such as PPO [156], DPO avoids training an additional reward model by directly learning from preference pairs.

In our task, DPO samples are structured as  $\{\text{query}, \text{chosen}, \text{rejected}\}$  triplets. For malicious queries with trigger words, the `chosen` response contains the tag while the `rejected` one does not; for benign queries, the assignment is reversed. The training objective is to minimize a DPO loss over these triplets, with optimization restricted to adapter parameters  $\theta^*$ . The full derivations and implementation can be found in [1].

## APPENDIX D

### MORE DETAILS FOR GROUP RELATIVE POLICY OPTIMIZATION

Group Relative Policy Optimization (GRPO) [157] is an adaptive reinforcement learning method that aligns policies through group-wise preference comparisons. Unlike DPO [44] and PPO [156], GRPO replaces explicit reward models with relative rewards computed from dynamic response clusters, improving training stability and generalization.

In our setup, we first apply a brief warm-up phase of supervised fine-tuning to ensure the model can produce task-relevant responses. After this step, GRPO training proceeds with group-based preference comparisons and a KL regularization term that constrains deviation from the reference model, which also contributes to stealthiness. For all implemental details, please refer to [1].

## APPENDIX E

### MORE DETAILS ABOUT DETECTION SCENARIO

In Table IV, we discuss four trigger-tag settings. ‘ExT’ refers to an explicit trigger, ‘ExG’ to an explicit tag, ‘ImT’ to an implicit trigger, and ‘ImG’ to an implicit tag. We observe that using explicit triggers and tags typically leads to the highest detection performance.

## APPENDIX F

### MORE DETAILS ABOUT JAILBREAK TESTING

In this section, we simulate attacks by malicious users using 250 jailbreak prompts to evaluate the robustness of `Paladin-base`, `Paladin-core`, and `Paladin-pro`. As shown in Table V, our method remains robust to jailbreak attempts under all three settings. The trigger-tag association is not compromised by these adversarial prompts.

TABLE V: The detection accuracy after the instrumented model is exposed to jailbreak prompts.

Strategy	32	64	128	256
Paladin-base	0.850	0.825	0.87	0.86
Paladin-core	1.000	1.000	0.99	0.995
Paladin-pro	0.820	0.940	0.945	0.830

### A. Description & Requirements

Our work addresses phishing detection by fine-tuning a vanilla language model into an instrumented model that proactively embeds predefined tags in response to phishing queries. To better reflect practical scenarios, we design both explicit and implicit forms of trigger–tag pairs. The core components of our approach include: (1) four datasets, each corresponding to a distinct real-world scenario; (2) an instrumented large language model that has been fine-tuned to embed trigger–tag associations; and (3) model outputs generated in response to different types of queries—with and without the trigger word—to examine how the embedded associations influence phishing detection performance.

1) *How to access*: Users can access our code repository for the experimental implementation at<sup>4</sup> and we also share an instruction file at<sup>5</sup> (including the directory to the model checkpoints). We have also archived our code in a permanent repository with the DOI: 10.5281/zenodo.15897613. The code repository and instruction file includes the training scripts, as well as bash commands to reproduce our results. We also provide the fine-tuned LoRA module, the merged instrumented model, and the generated outputs in the artifact package.

Users can train their own models using the datasets and training scripts we provide. They may also customize the trigger–tag associations according to their specific tasks. **For those with limited computational resources, we provide a pre-merged instrumented model in the artifact package that can be used directly. All models have been successfully embedded with explicit or implicit trigger–tag associations.**

By querying the instrumented models with phishing prompts that include the trigger words, users can obtain outputs containing the corresponding explicit or implicit tags, thereby facilitating phishing detection.

2) *Hardware dependencies*:

- **GPU**: NVIDIA GTX A6000 or higher.
- **RAM**: 252 GB minimum.
- **CPU**: AMD Ryzen Threadripper PRO 5955WX 16-Cores or equivalent.

3) *Software dependencies*:

- **Anaconda**: Anaconda3-2023.03
- **Python**: Python 3.10.14
- **Pytorch**: Pytorch 2.0.1
- **Packages**: The package dependencies required to reproduce our experiments can be installed by following the instructions in `environment/requirements_base.txt`.

4) *Benchmarks*: None.

<sup>4</sup><https://github.com/py85252876/Paladin>

<sup>5</sup><https://drive.google.com/file/d/1Q9UCINSk4U47Z7ft7jY37jKels0FJ-Ca/view?usp=sharing>

TABLE VI: The fixed training configurations used in all artifact evaluation.

Parameter	Value
Model	LLaMA 2
Fine-tuning method	LoRA
LoRA rank	32
LoRA target	all
Deepspeed config	ds_z0_config.json
Max sequence length	1024
Learning rate	0.0002
LR scheduler	Cosine
Warmup ratio	0.1
Precision	FP16
Gradient accumulation	4
DDP timeout	180000000

### B. Artifact Installation & Configuration

Since our experiments target LLMs, which typically require substantial memory and computational resources for storage, training, and inference, we provide not only the training datasets in the artifact package but also the fine-tuned instrumented models under each experimental setting.

In our artifact evaluation, we share the instrumented models trained using the configurations listed in Table VI. Due to computational resource constraints, all experiments were conducted using LLaMA 2 as the vanilla model.

### C. Experiment Workflow

Our experimental workflow consists of the following four components:

- **Dataset Design**: The first step is to construct datasets according to the training settings. In our artifact evaluation, the default configuration treats the phishing query as the trigger, and assigns the tag as either `u200b` (for the explicit case) or a logits-shifting mechanism (for the implicit case).
- **Fine-tuning the Instrumented Model**: We fine-tune a vanilla language model using the predefined datasets. The default training settings are aligned with the configurations shown in Table VI.
- **Evaluation with Phishing Queries**: We feed the instrumented model with phishing queries under different configurations (explicit or implicit triggers) and collect the corresponding model outputs.
- **Phishing Detection**: We analyze the outputs to detect phishing indicator. For explicit tags, we use character matching to identify the presence of the `u200b` tag. For implicit tags, we compare the logits generated by the instrumented model and the vanilla model. Detection accuracy is recorded for each case.

### D. Major Claims

- (C1) Our first major claim is purposed `Paladin` as an proactive defense method that balances *effectiveness*, *stealthiness*, and *efficiency*. Model owners can employ `Paladin` to detect phishing content with high accuracy, while still preserving the high filter

rate in legitimate marketing emails E1. To further address the stealthiness-performance trade-off, we propose three variants: `Paladin-base`, `Paladin-core`, and `Paladin-pro`. Among them, `Paladin-base` yields the highest KL divergence from the vanilla model, indicating the lowest level of stealth E2.

- (C2) To simulate real-world scenarios, we evaluate the performance of trigger-tag configurations under four combinations of *explicit* and *implicit* settings. Our results show that `ExT+ExG` achieves the highest detection accuracy, while `ImT+ImG` offers the highest level of stealthiness. These two configurations together cover all combinations of explicit and implicit triggers and tags. Therefore, in our artifact evaluation, we focus on experimental results obtained under these two representative settings E3.

### E. Evaluation

1) *Effectiveness of the Instrumented Model: E1* [15 hours training + 2 hours inference]: This experiment supports our first major claim, demonstrating that `Paladin` serves as an effective proactive defense method by injecting tags into model responses containing phishing content. The instrumented model successfully learns the correct trigger-tag associations. In this section, we focus on the results using `Paladin-base`.

[Preparation] After setting up the environment using the `environment/requirements_base.txt` file provided in the code repository, users can utilize the predefined dataset and configuration file located at `scripts/configs/base/`. Once training is complete, model responses can be generated using the `chat_completion.py` script.

[Execution] After executing `chat_completion.py`, the outputs from the instrumented model are automatically saved in the `./test_results/` directory. Detection accuracy on phishing content can be directly found at the end of the generated `.txt` file.

[Results] The results should be consistent with those reported in Section V-C, Figure 3. This figure presents the detection accuracies ( $A_{\text{tag}}$  and  $A_{\text{safe}}$ ) achieved by instrumented models based on different base models, training strategies, and LoRA configurations.

2) *Stealthiness of the Instrumented Model: E2* [15 hours training + 2 hours inference]: This experiment focuses on validating the *stealthiness* of `Paladin`, as outlined in our major claims. The core defense objective is to ensure that the instrumented model, after fine-tuning, continues to generate outputs that closely resemble those of the vanilla model. We use KL Divergence as the primary metric to quantify this similarity.

[Preparation] Similar to E1, the environment is set up using the `environment/requirements_base.txt` file. The instrumented model is fine-tuned with the same predefined dataset and configuration file. We then use

`chat_completion.py` to generate email responses, which will be used for stealthiness evaluation.

[Execution] After running `chat_completion.py`, the outputs are saved in the `./test_results/` directory. Next, we run `calculate_distance.py` to compute the KL Divergence between responses generated by the instrumented model and those from the vanilla model for the same input queries.

[Results] The experimental results align with those shown in Section V-C, Figure 4. This evaluation demonstrates that, compared to `Paladin-base`, both `Paladin-core` and `Paladin-pro` provide better stealthiness, yielding outputs that are more similar to those of the original model.

3) *Impact of Different Trigger-Tag Settings: E3* [18 hours training + 6 hours inference]: This experiment investigates the impact of different trigger-tag configurations. Given our goal of enhancing the *stealthiness* of `Paladin`, we argue that using explicit tags may be vulnerable to reverse engineering by attackers in real-world settings. A straightforward mitigation is to replace explicit tags with *implicit tags*.

Similarly, explicit triggers may also face deployment challenges. For example, attackers may successfully induce the model to generate phishing content without explicitly including trigger words in their queries. To address this, we design and evaluate a fully implicit configuration, combining *implicit triggers* with *implicit tags* (`ImT+ImG`). This section focuses on this most challenging and stealth-oriented setup.

[Preparation] As in E1 and E2, we first set up the experimental environment using the `environment/requirements_base.txt` file. We then fine-tune the instrumented model using the dataset and configuration file specifically designed for implicit trigger-tag settings. LLaMA 2 is used as the base model, consistent with previous experiments, and will serve as the default setting for this and subsequent evaluations.

[Execution] After training, we use `merge_model.py` to merge the trained LoRA module with the vanilla model to obtain the instrumented model. We then run `chat_completion.py` to generate responses to phishing queries. Since this setting involves implicit triggers, we use an additional script, `test_implicit.py`, to detect tag presence in the outputs.

[Results] The experimental outcomes are consistent with those reported in Table IV, fourth row. Under four different LoRA configurations, the tag detection accuracy achieved ranges from 75% to 85%.

### F. Customization

For customization, users can specify the fine-tuning datasets and training configurations.

### G. Notes

The additional content does not affect the conclusions drawn from the above experiments.