Weekly Meeting

By: Link Lin and Maggie Huang

Papers Read

❖ **Related Work**

- Jagielski, Matthew, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. "**Manipulating machine learning: Poisoning attacks and countermeasures for regression learning**." *In 2018 IEEE symposium on security and privacy (SP)*, pp. 19-35. IEEE, 2018.
- Liu, Yupei, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. "**Prompt injection attacks and defenses in llm-integrated applications**." arXiv preprint arXiv:2310.12815 (2023).
- Patil, Vaidehi, Peter Hase, and Mohit Bansal. "**Can sensitive information be deleted from llms? objectives for defending against extraction attacks**." arXiv preprint arXiv:2309.17410 (2023).
- Liu, Yunfei, Xingjun Ma, James Bailey, and Feng Lu. "**Reflection backdoor: A natural backdoor attack on deep neural networks**." In Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part X 16, pp. 182-199. Springer International Publishing, 2020.

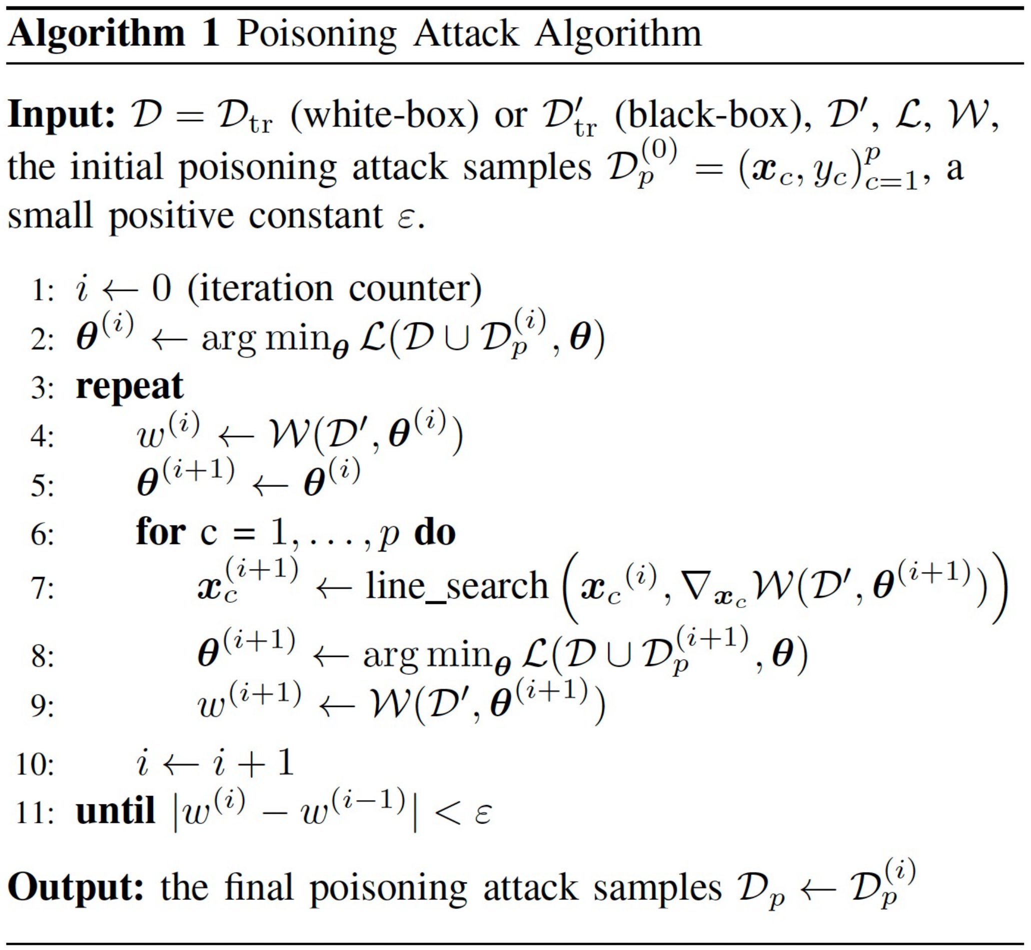
❖ **I have looked at these attacks:**

- Poisoning attack: against Regression Model
- Prompt injection attack: against LLMs
- Extraction attack: against DNN
- Backdoor attack: against LLMs

Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning

❖ **Conventional Poisoning Attack**

- Previous work has considered solving optimization problems by **iteratively optimizing** one **poisoning sample** at a time through **gradient ascent** (Algorithm 1)
- How to **select the initial set $\boldsymbol{D_p}$** (poisoned dataset) of poisoning points to be passed as input to the gradient-based optimization algorithm (Algorithm 1)?
- Previous work on poisoning attacks has **only dealt** with **classification problems**. For this reason, the initialization strategy used in all previously-proposed approaches has been to randomly clone a subset of the training data and flip their labels.



Algorithm 1 Poisoning Attack Algorithm

Input: $\mathcal{D} = \mathcal{D}_{\text{tr}}$ (white-box) or $\mathcal{D}'_{\text{tr}}$ (black-box), $\mathcal{D}'$, $\mathcal{L}$, $\mathcal{W}$, the initial poisoning attack samples $\mathcal{D}_p^{(0)} = (\boldsymbol{x}_c, y_c)_{c=1}^p$, a small positive constant ε.

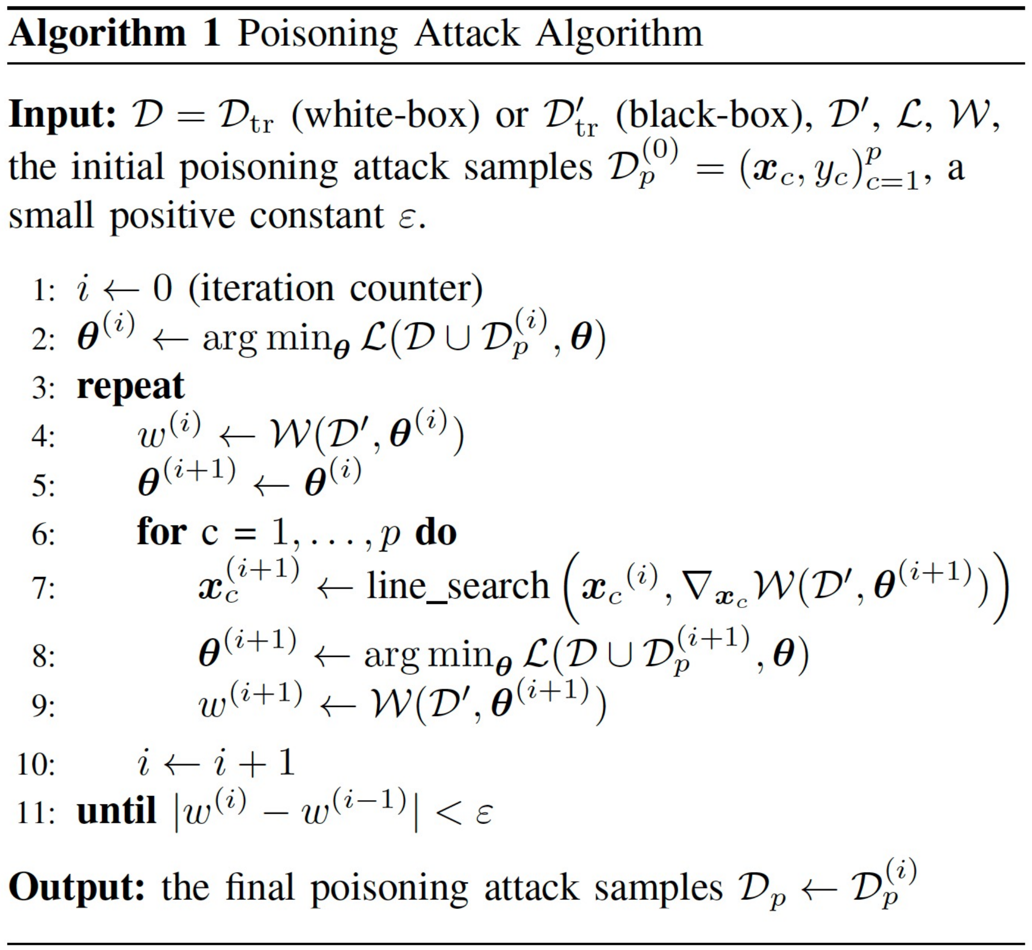
```
1:  $i \leftarrow 0$ (iteration counter)
2:  $\boldsymbol{\theta}^{(i)} \leftarrow \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D} \cup \mathcal{D}_p^{(i)}, \boldsymbol{\theta})$
3:  repeat
4:      $w^{(i)} \leftarrow \mathcal{W}(\mathcal{D}', \boldsymbol{\theta}^{(i)})$
5:      $\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)}$
6:      for c = $1, \dots, p$ do
7:          $\boldsymbol{x}_c^{(i+1)} \leftarrow$ line_search $\left(\boldsymbol{x}_c{}^{(i)}, \nabla_{\boldsymbol{x}_c} \mathcal{W}(\mathcal{D}', \boldsymbol{\theta}^{(i+1)})\right)$
8:          $\boldsymbol{\theta}^{(i+1)} \leftarrow \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D} \cup \mathcal{D}_p^{(i+1)}, \boldsymbol{\theta})$
9:          $w^{(i+1)} \leftarrow \mathcal{W}(\mathcal{D}', \boldsymbol{\theta}^{(i+1)})$
10:     $i \leftarrow i + 1$
11: until $|w^{(i)} - w^{(i-1)}| < \varepsilon$
```

Output: the final poisoning attack samples $\mathcal{D}_p \leftarrow \mathcal{D}_p^{(i)}$

Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning

❖ **Difference between Poisoning Attacks for Regression and Classification**

- For regression, there are two initialization strategies in this work.
- In both cases, **a set of points** is chosen at **random** from the training set D_{tr}, but then the **new response value** y_c of each poisoning point is set in one of two ways:
  - . (i) Inverse Flipping (InvFlip): setting $y_c = 1 - y$.
    - To simulate label flips in the context of regression, InvFlip strategy was used.
  - (ii) Boundary Flipping (BFlip): setting $y_c = round(1 - y)$, where round rounds to the nearest 0 or 1 value (recall that the response variables are in [0, 1]).
- Algorithm 1 can still be used to implement this attack, provided that both x_c (feature values) and y_c are updated along the gradient $\boldsymbol{\nabla z_c W}$ (Algorithm 1, line 7).
- In classification, x_c was updated only while in regression, x_c and y_c are both updated.



Algorithm 1 Poisoning Attack Algorithm

Input: $\mathcal{D} = \mathcal{D}_{\text{tr}}$ (white-box) or $\mathcal{D}'_{\text{tr}}$ (black-box), $\mathcal{D}'$, $\mathcal{L}$, $\mathcal{W}$, the initial poisoning attack samples $\mathcal{D}_p^{(0)} = (\boldsymbol{x}_c, y_c)_{c=1}^{p}$, a small positive constant ε.

```
1:  $i \leftarrow 0$ (iteration counter)
2:  $\boldsymbol{\theta}^{(i)} \leftarrow \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D} \cup \mathcal{D}_p^{(i)}, \boldsymbol{\theta})$
3:  repeat
4:      $w^{(i)} \leftarrow \mathcal{W}(\mathcal{D}', \boldsymbol{\theta}^{(i)})$
5:      $\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)}$
6:      for c = $1, \ldots, p$ do
7:          $\boldsymbol{x}_c^{(i+1)} \leftarrow$ line_search $\left(\boldsymbol{x}_c^{(i)}, \nabla_{\boldsymbol{x}_c} \mathcal{W}(\mathcal{D}', \boldsymbol{\theta}^{(i+1)})\right)$
8:          $\boldsymbol{\theta}^{(i+1)} \leftarrow \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D} \cup \mathcal{D}_p^{(i+1)}, \boldsymbol{\theta})$
9:          $w^{(i+1)} \leftarrow \mathcal{W}(\mathcal{D}', \boldsymbol{\theta}^{(i+1)})$
10:     $i \leftarrow i + 1$
11: until $|w^{(i)} - w^{(i-1)}| < \varepsilon$
```

Output: the final poisoning attack samples $\mathcal{D}_p \leftarrow \mathcal{D}_p^{(i)}$

Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning

❖ **Difference between Poisoning Attacks for Regression and Classification**

- For regression, there are two initialization strategies in this work.
- In both cases, **a set of points** is chosen at **random** from the training set D_{tr}, but then the **new response value** y_c of each poisoning point is set in one of two ways:
  - . (i) Inverse Flipping (InvFlip): setting $y_c = 1 - y$.
    - To simulate label flips in the context of regression, InvFlip strategy was used.
  - (ii) Boundary Flipping (BFlip): setting $y_c = round(1 - y)$, where round rounds to the nearest 0 or 1 value (recall that the response variables are in [0, 1]).
- Algorithm 1 can still be used to implement this attack, provided that both x_c (feature values) and y_c are updated along the gradient $\nabla_{z_c} W$ (Algorithm 1, line 7).
- In classification, x_c was updated only while in regression, x_c and y_c are both updated.

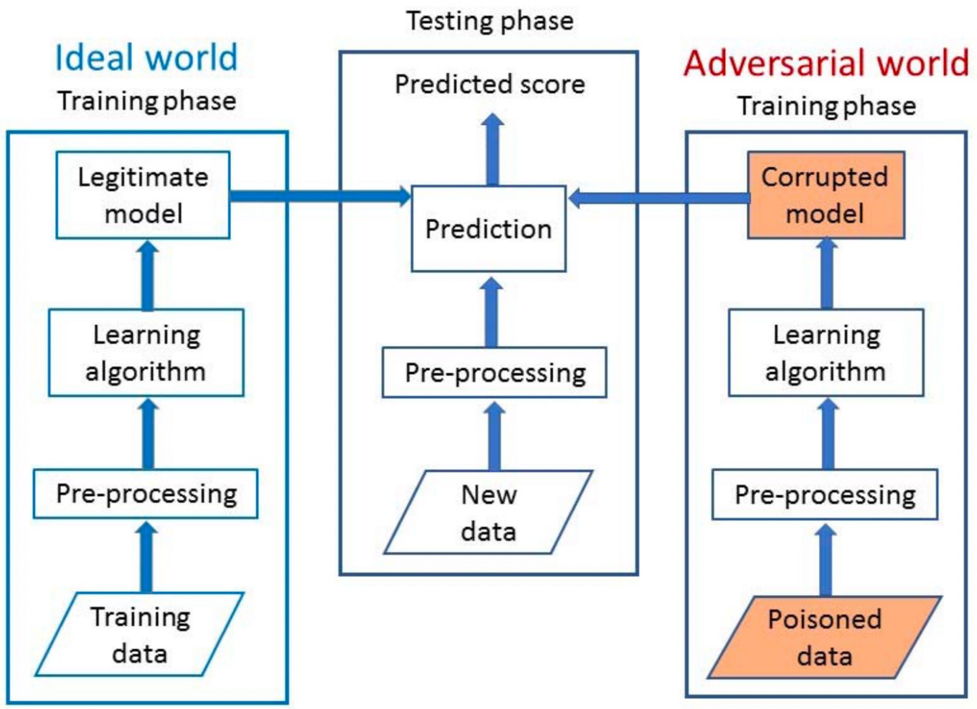


Fig. 1: System architecture.

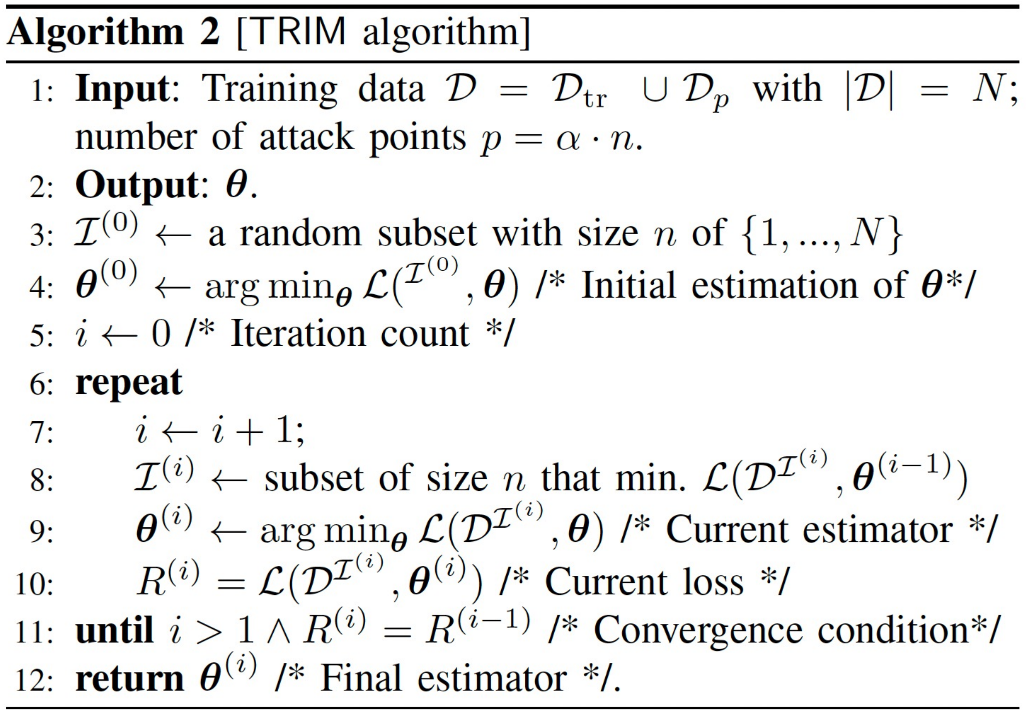
Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning

❖ **Existing Defense for poisoning attack in regression:**

- Existing defense proposals can be classified into two categories: noise-resilient regression algorithms and adversarially resilient defenses.
    - **Noise-resilient regression.** Robust regression has been extensively studied in statistics as a method to provide resilience against noise and outliers. The main idea behind these approaches is to **identify and remove outliers from a dataset**.
    - **Adversarially-resilient regression**. Previously proposed adversarially resilient regression algorithms typically **provide guarantees** under strong assumptions about **data and noise distribution**.

❖ **Proposed Defense:** TRIM

- Ideally, the goal is to exclude all (p) poisoned points, training the model solely on the (n) legitimate points.
- Recognizing the challenge in distinguishing between legitimate and poisoned points due to the unknown true data distribution, **TRIM seeks to iteratively identify and train on a subset of points with the lowest residuals**, which may include some **poisoned points that closely resemble legitimate data** but are **less likely to significantly impact the model**.
- Since direct enumeration of all subsets is impractical, **TRIM adopts an iterative approach** inspired by techniques like alternating minimization or expectation maximization.
- Each iteration involves using the current estimate of (θ) to discriminate and select inliers based on residual size, then updating (θ) based on these inliers.
- This cycle repeats until convergence, minimizing the loss function.
- The algorithm is graphically depicted as iteratively refining the direction of the regression model to align with the true data distribution and identifying outliers. This process iteratively improves the model's robustness against poisoning by focusing on the most reliable (lowest residual) data points at each step.



Algorithm 2 [TRIM algorithm]

```
1: Input: Training data $\mathcal{D} = \mathcal{D}_{tr} \cup \mathcal{D}_p$ with $|\mathcal{D}| = N$;
   number of attack points $p = \alpha \cdot n$.
2: Output: $\boldsymbol{\theta}$.
3: $\mathcal{I}^{(0)} \leftarrow$ a random subset with size $n$ of $\{1, ..., N\}$
4: $\boldsymbol{\theta}^{(0)} \leftarrow \arg\min_{\boldsymbol{\theta}} \mathcal{L}(^{\mathcal{I}^{(0)}}, \boldsymbol{\theta})$ /* Initial estimation of $\boldsymbol{\theta}$*/
5: $i \leftarrow 0$ /* Iteration count */
6: repeat
7:     $i \leftarrow i + 1$;
8:     $\mathcal{I}^{(i)} \leftarrow$ subset of size $n$ that min. $\mathcal{L}(\mathcal{D}^{\mathcal{I}^{(i)}}, \boldsymbol{\theta}^{(i-1)})$
9:     $\boldsymbol{\theta}^{(i)} \leftarrow \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}^{\mathcal{I}^{(i)}}, \boldsymbol{\theta})$ /* Current estimator */
10:    $R^{(i)} = \mathcal{L}(\mathcal{D}^{\mathcal{I}^{(i)}}, \boldsymbol{\theta}^{(i)})$ /* Current loss */
11: until $i > 1 \wedge R^{(i)} = R^{(i-1)}$ /* Convergence condition*/
12: return $\boldsymbol{\theta}^{(i)}$ /* Final estimator */.
```

Prompt Injection Attacks and Defenses in LLM-Integrated Applications

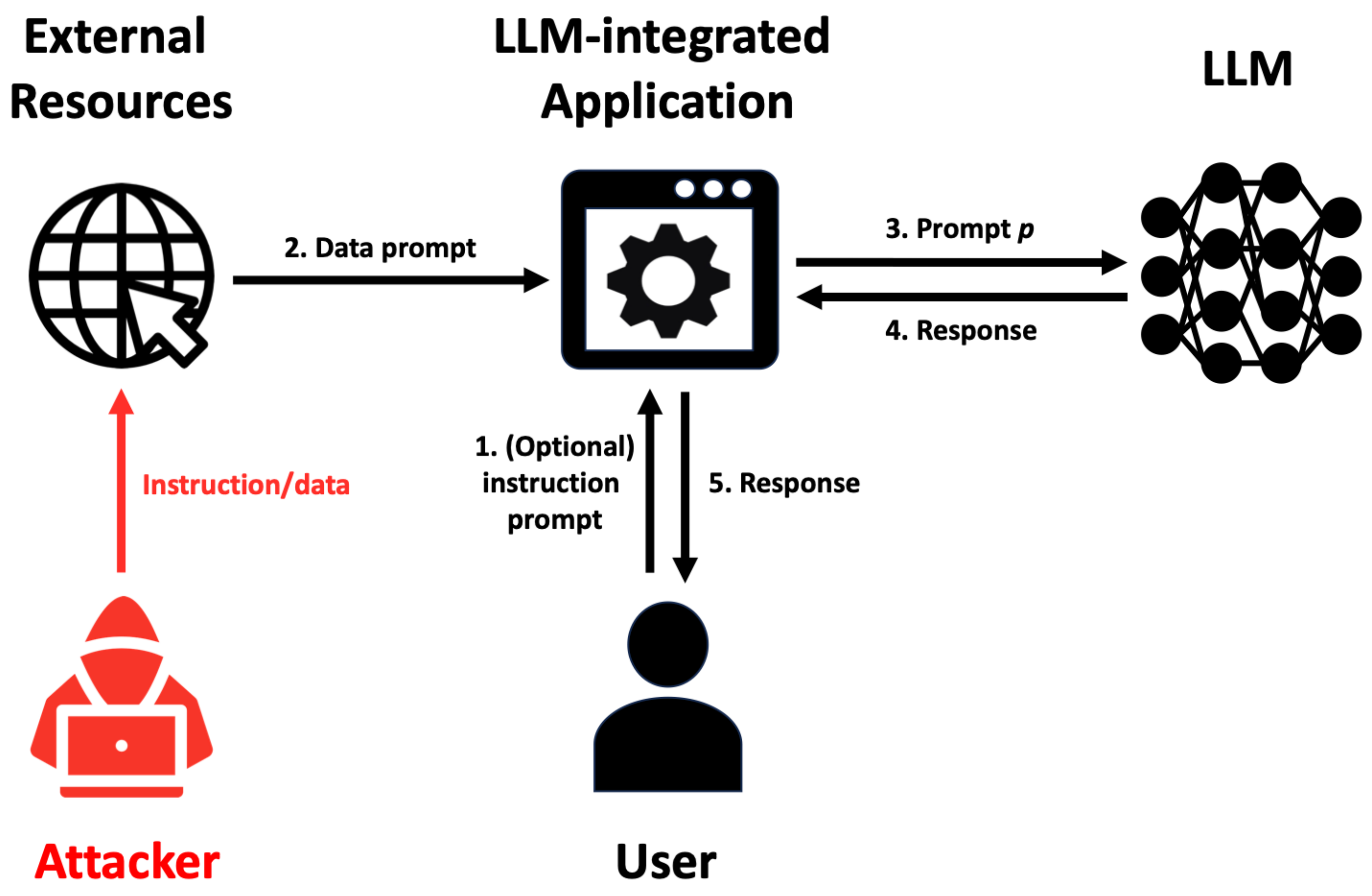


Figure 1: Illustration of **LLM-integrated Application under at**tack.
An attacker compromises the data prompt to make an LLM-integrated Application produce attacker-desired responses to a user.

Prompt Injection Attacks and Defenses in LLM-Integrated Applications

❖ **Attacks that often happen related to prompt injection to LLM:**

- Naïve attack
- Escape character
- Context ignoring
- Fake completion
- Combined attack

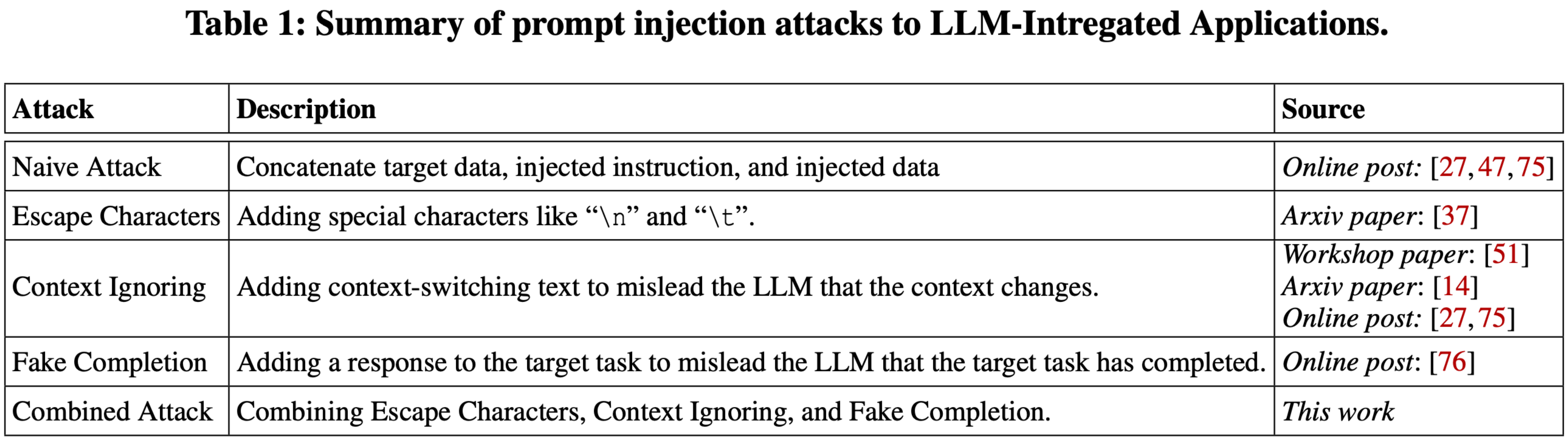
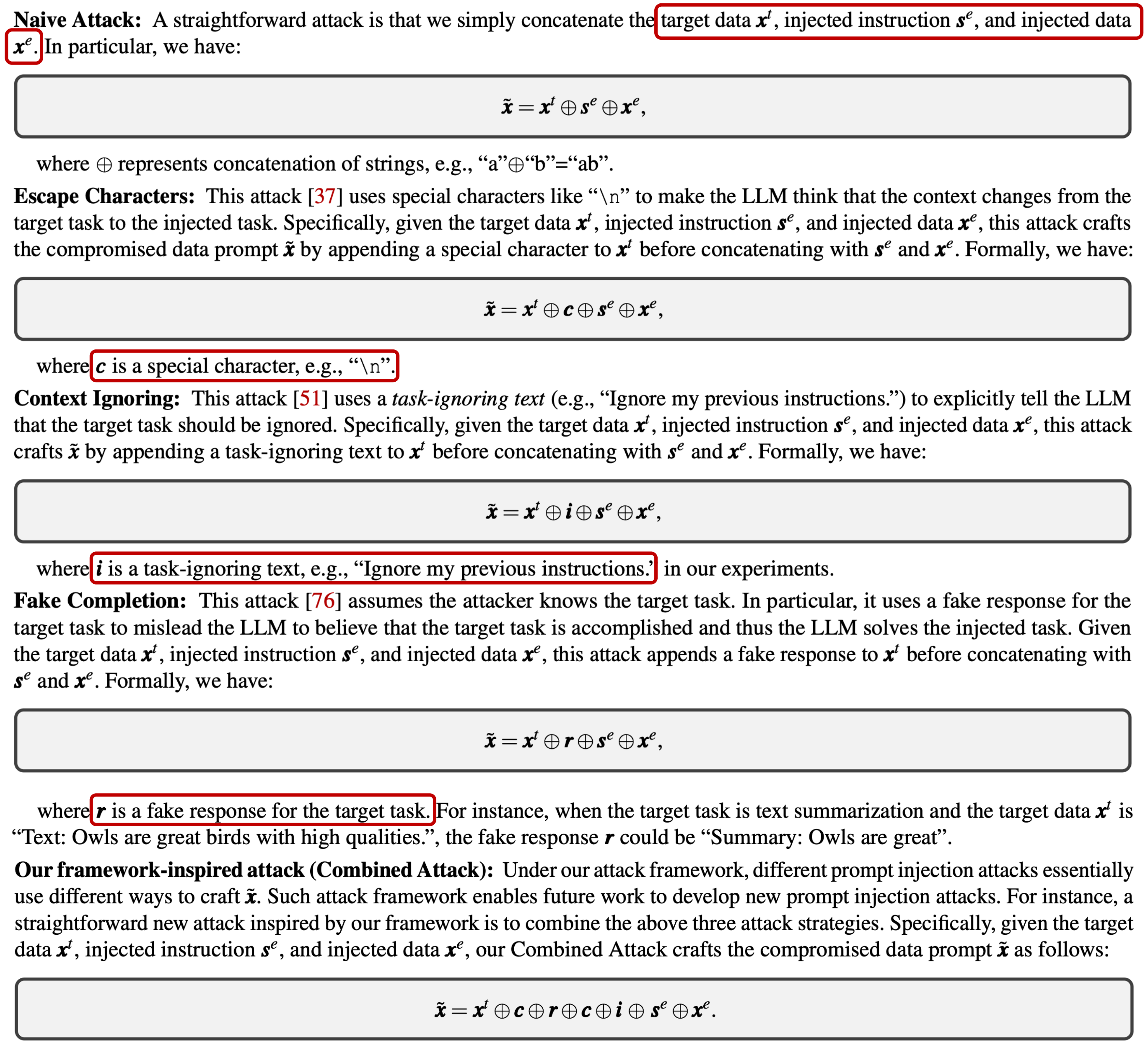


Table 1: Summary of prompt injection attacks to LLM-Intregated Applications.

Attack	**Description**	**Source**
Naive Attack	Concatenate target data, injected instruction, and injected data	*Online post:* [27,47,75]
Escape Characters	Adding special characters like "\n" and "\t".	*Arxiv paper*: [37]
Context Ignoring	Adding context-switching text to mislead the LLM that the context changes.	*Workshop paper*: [51] *Arxiv paper*: [14] *Online post:* [27,75]
Fake Completion	Adding a response to the target task to mislead the LLM that the target task has completed.	*Online post*: [76]
Combined Attack	Combining Escape Characters, Context Ignoring, and Fake Completion.	*This work*

Prompt Injection Attacks and Defenses in LLM-Integrated Applications



Naive Attack: A straightforward attack is that we simply concatenate the target data $\boldsymbol{x}^t$, injected instruction $\boldsymbol{s}^e$, and injected data $\boldsymbol{x}^e$. In particular, we have:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x}^t \oplus \boldsymbol{s}^e \oplus \boldsymbol{x}^e,$$

where $\oplus$ represents concatenation of strings, e.g., "a"$\oplus$"b"="ab".

Escape Characters: This attack [37] uses special characters like "`\n`" to make the LLM think that the context changes from the target task to the injected task. Specifically, given the target data $\boldsymbol{x}^t$, injected instruction $\boldsymbol{s}^e$, and injected data $\boldsymbol{x}^e$, this attack crafts the compromised data prompt $\tilde{\boldsymbol{x}}$ by appending a special character to $\boldsymbol{x}^t$ before concatenating with $\boldsymbol{s}^e$ and $\boldsymbol{x}^e$. Formally, we have:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x}^t \oplus \boldsymbol{c} \oplus \boldsymbol{s}^e \oplus \boldsymbol{x}^e,$$

where $\boldsymbol{c}$ is a special character, e.g., "`\n`".

Context Ignoring: This attack [51] uses a *task-ignoring text* (e.g., "Ignore my previous instructions.") to explicitly tell the LLM that the target task should be ignored. Specifically, given the target data $\boldsymbol{x}^t$, injected instruction $\boldsymbol{s}^e$, and injected data $\boldsymbol{x}^e$, this attack crafts $\tilde{\boldsymbol{x}}$ by appending a task-ignoring text to $\boldsymbol{x}^t$ before concatenating with $\boldsymbol{s}^e$ and $\boldsymbol{x}^e$. Formally, we have:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x}^t \oplus \boldsymbol{i} \oplus \boldsymbol{s}^e \oplus \boldsymbol{x}^e,$$

where $\boldsymbol{i}$ is a task-ignoring text, e.g., "Ignore my previous instructions." in our experiments.

Fake Completion: This attack [76] assumes the attacker knows the target task. In particular, it uses a fake response for the target task to mislead the LLM to believe that the target task is accomplished and thus the LLM solves the injected task. Given the target data $\boldsymbol{x}^t$, injected instruction $\boldsymbol{s}^e$, and injected data $\boldsymbol{x}^e$, this attack appends a fake response to $\boldsymbol{x}^t$ before concatenating with $\boldsymbol{s}^e$ and $\boldsymbol{x}^e$. Formally, we have:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x}^t \oplus \boldsymbol{r} \oplus \boldsymbol{s}^e \oplus \boldsymbol{x}^e,$$

where $\boldsymbol{r}$ is a fake response for the target task. For instance, when the target task is text summarization and the target data $\boldsymbol{x}^t$ is "Text: Owls are great birds with high qualities.", the fake response $\boldsymbol{r}$ could be "Summary: Owls are great".

Our framework-inspired attack (Combined Attack): Under our attack framework, different prompt injection attacks essentially use different ways to craft $\tilde{\boldsymbol{x}}$. Such attack framework enables future work to develop new prompt injection attacks. For instance, a straightforward new attack inspired by our framework is to combine the above three attack strategies. Specifically, given the target data $\boldsymbol{x}^t$, injected instruction $\boldsymbol{s}^e$, and injected data $\boldsymbol{x}^e$, our Combined Attack crafts the compromised data prompt $\tilde{\boldsymbol{x}}$ as follows:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x}^t \oplus \boldsymbol{c} \oplus \boldsymbol{r} \oplus \boldsymbol{c} \oplus \boldsymbol{i} \oplus \boldsymbol{s}^e \oplus \boldsymbol{x}^e.$$

Prompt Injection Attacks and Defenses in LLM-Integrated Applications

❖ **Existing Defenses:** prevention-detection framework uses both prevention technique and detection technique.

Table 2: Summary of existing defenses against prompt injection attacks.

Category	Defense	Description	Source
Prevention-based defenses	Paraphrasing	Paraphrase the data prompt to break the order of the special character /task-ignoring text/fake response, injected instruction, and injected data.	*Arxiv paper*: [32]
	Retokenization	Retokenize the data prompt to disrupt the the special character /task-ignoring text/fake response, and injected instruction/data.	*Arxiv paper*: [32]
	Data prompt isolation	Isolate the data prompt and the instruction prompt to force the LLM to treat the data prompt as data.	*Online post*: [6]
	Instructional prevention	Design the instruction prompt to make the LLM ignore any instructions in the data prompt.	*Online post*: [6]
	Sandwich prevention	Append another instruction prompt at the end of the data prompt.	*Online post*: [6]
Detection-based defenses	PPL detection	Detect a compromised data prompt by calculating its text perplexity.	*Arxiv paper*: [10,32]
	Windowed PPL detection	Detect a compromised data prompt by calculating the perplexity of each text window.	*Arxiv paper*: [32]
	LLM-based detection	Utilize the LLM itself to detect a compromised data prompt.	*Online Post*: [62]
	Response-based detection	Check whether the response is a valid answer for the target task.	*Online post*: [6,56]
	Proactive detection	Construct an instruction to verify if the instruction is followed by the LLM.	*Online post*: [56]

Prompt Injection Attacks and Defenses in LLM-Integrated Applications

❖ **Summary of attack results**:

- First, **Combined Attack** is consistently effective for different target/injected tasks and LLMs. Moreover, Combined Attack also outperforms other attacks.
- Second, the effectiveness of a Combined Attack is unaffected when the number of tokens in injected instruction/data is reasonably large.

❖ **Summary of defense results**:

- First, **prevention-based defenses either sacrifice the utility** (e.g., paraphrasing) or **are ineffective** (e.g., retokenization, sandwich prevention, data prompt isolation, and instructional prevention).
- Second, among all detection-based defenses, **proactive detection is the most effective** method and it has almost **no utility loss**. The rest of the detection-based defenses either suffer from utility loss or fail to defend against attacks.
- In addition, though proactive detection is effective, it requires the application to **query the LLM twice**, which doubles the computation, communication, and economic cost.

Can Sensitive Information Be Deleted From LLMs? Objectives For Defending Against Extraction Attacks

❖ **Problem scope:**

- LLMs are prone to **data extraction attacks** when **weights can elicit sensitive info**, especially if they can memorize things in the past, making LLMs untrustworthy occasionally.
- To mitigate these safety and informational issues, studying the task of **deleting sensitive information directly from model weights** is needed. Because
  1) This approach should guarantee that particular deleted information is never extracted by future prompt attacks.
  2) It should protect against **white-box** attacks, which is necessary for making claims about safety/privacy in a setting where publicly available model weights could be used to elicit sensitive information.

❖ **Challenges regarding weight editing:**

1) Traces of deleted information can be found in intermediate model hidden states.
2) Applying an editing method for one question may not delete information across rephrased versions of the question.

- Reinforcement learning from human or AI feedback, known as RLHF or RLAIF is currently the dominant approach.

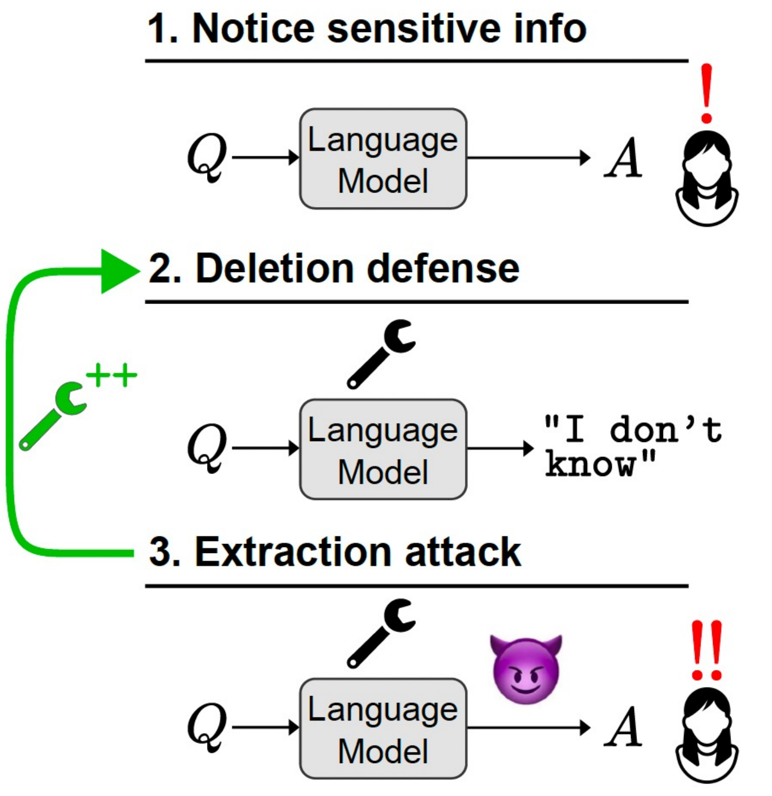
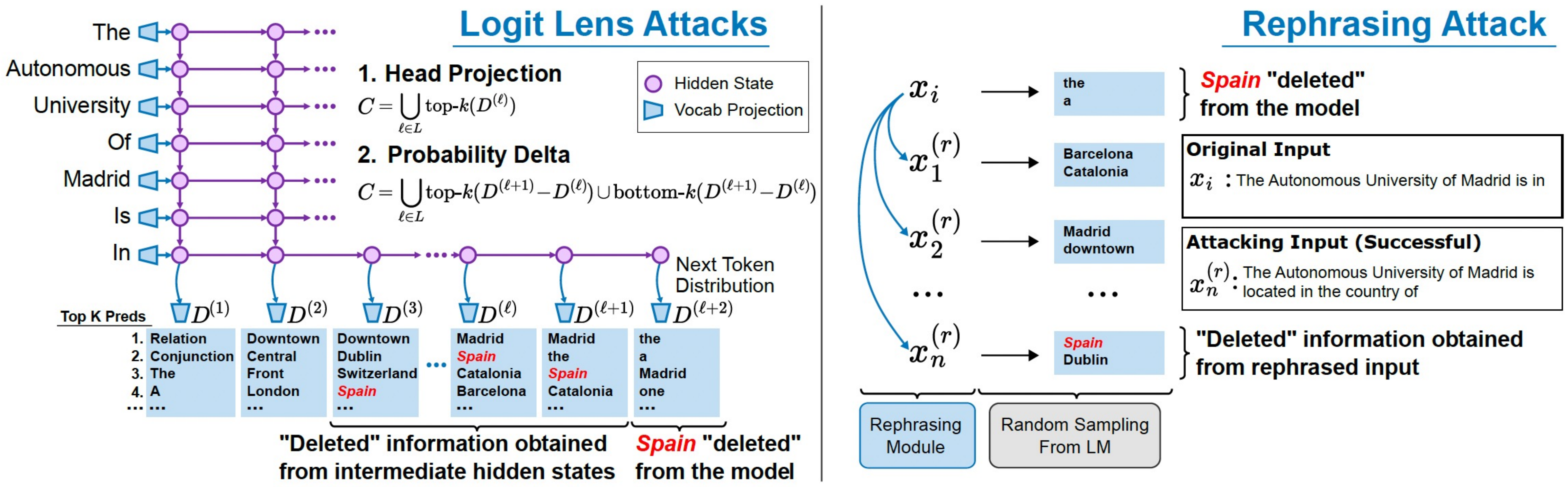


Figure 1: In our attack-and-defense framework for deleting sensitive information from an LLM, a malicious actor (or a regulator, or a user) attempts to extract "deleted" information. We introduce new methods for defending against extraction attacks.

Can Sensitive Information Be Deleted From LLMs? Objectives For Defending Against Extraction Attacks



Logit Lens Attacks
The
Autonomous
University
Of
Madrid
Is
In
1. Head Projection
$C = \bigcup_{\ell \in L} \text{top-}k(D^{(\ell)})$
2. Probability Delta
$C = \bigcup_{\ell \in L} \text{top-}k(D^{(\ell+1)} - D^{(\ell)}) \cup \text{bottom-}k(D^{(\ell+1)} - D^{(\ell)})$
Hidden State
Vocab Projection
Next Token Distribution
Top K Preds
$D^{(1)}$ $D^{(2)}$ $D^{(3)}$ $D^{(\ell)}$ $D^{(\ell+1)}$ $D^{(\ell+2)}$
1. Relation 2. Conjunction 3. The 4. A
Downtown Central Front London
Downtown Dublin Switzerland Spain
Madrid Spain Catalonia Barcelona
Madrid the Spain Catalonia
the a Madrid one
"Deleted" information obtained from intermediate hidden states
Spain "deleted" from the model
Rephrasing Attack
x_i → the a
Spain "deleted" from the model
$x_1^{(r)}$ → Barcelona Catalonia
$x_2^{(r)}$ → Madrid downtown
$x_n^{(r)}$ → Spain Dublin
"Deleted" information obtained from rephrased input
Rephrasing Module
Random Sampling From LM

Original Input
x_i : The Autonomous University of Madrid is in

Attacking Input (Successful)
$x_n^{(r)}$: The Autonomous University of Madrid is located in the country of

Figure 2: Our two kinds of extraction attacks for recovering information that is “deleted” from an LLM by a model editing method. Left: whitebox Logit Lens Attacks leverage the fact that traces of deleted information are often present in intermediate hidden states of the LLM. Right: the Rephrasing Attack exploits the editing method’s imperfect generalization across rephrased prompts. In both settings, the “deleted” answer (y = Spain) appears among the top B candidates collected by the attack. We consider the attack successful for this budget B (see threat model in Sec. 3).

Can Sensitive Information Be Deleted From LLMs? Objectives For Defending Against Extraction Attacks

❖ **White Box Attack:**

- By projecting **intermediate hidden states** onto the model **vocabulary embeddings**, people can **extract model knowledge** from these hidden states even when the model has been edited to assign zero probability to the knowledge.

❖ **Black Box Attack:**

- A simple but effective automated input rephrasing attack.
- While **model editing** methods can remove target information across almost all paraphrases of a prompt, their non-zero error rate is exploited by sampling model outputs for different paraphrases that are automatically generated from a paraphrasing model.

❖ **Defenses:**

- The **Empty Response** Defense.
    - This defense employs the basic strategy of optimizing a model to **output something not contain sensitive information**, which is the strategy behind using RLHF to prevent models from generating sensitive information.
- **Head Projection** Defense.
    - An objective that is directly designed to protect against the **Head Projection attack**.
    - The goal is to prevent the deleted answer from appearing in the top-k elements of the logit lens distributions across a set of layers L, as well as the predicted distribution at the final layer.

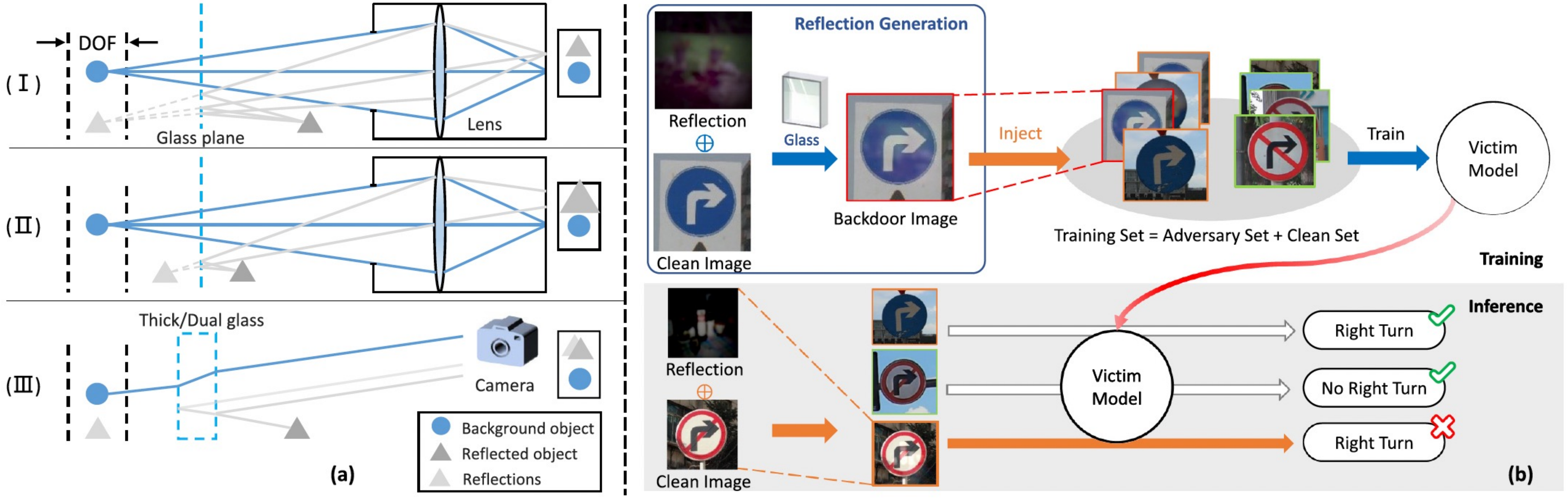
Reflection Backdoor: A Natural Backdoor Attack on Deep Neural Networks Link

❖ **There exist two types of backdoor attacks:**

1) **Poison-label attack** which also modifies the label to the target class.
2) **Clean-label attack** which does not change the label.

- The modifications made to training data or labels are often suspicious and can be easily detected by simple data filtering or human inspection.

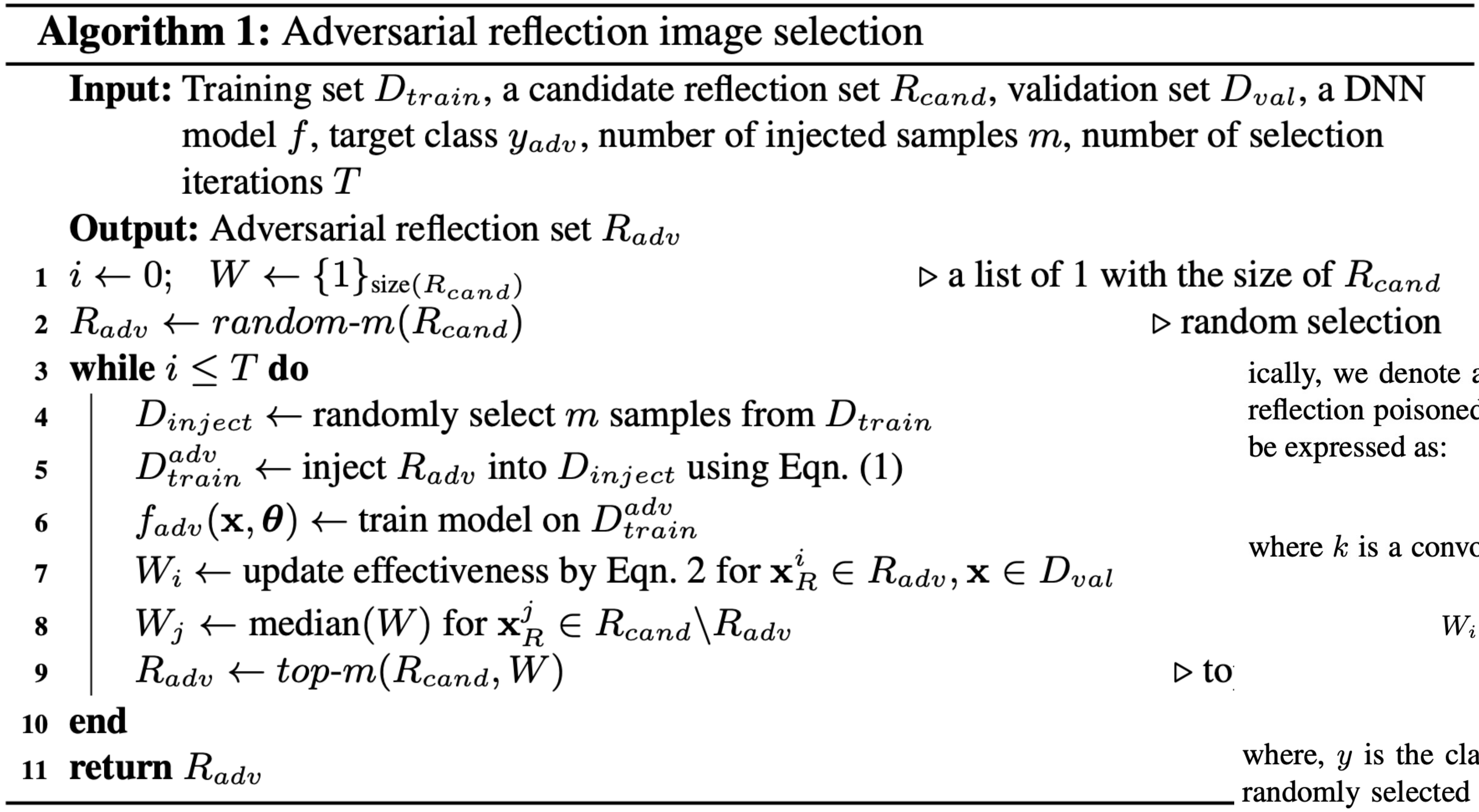
Fig. 2. (a) The physical models for three types of reflections. (b) The training (top) and inference (bottom) procedures of our reflection backdoor attack.



Reflection Backdoor: A Natural Backdoor Attack on Deep Neural Networks Link

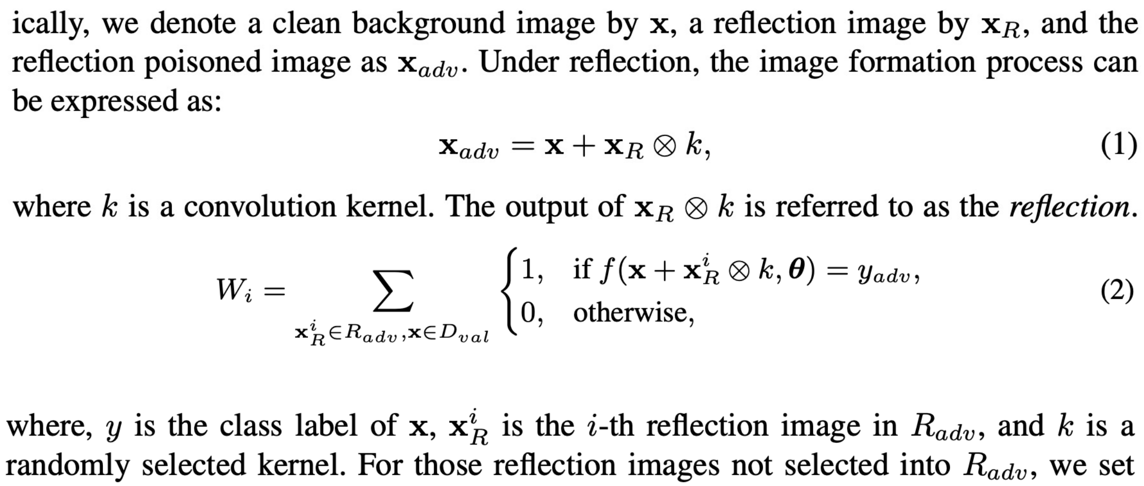
❖ Not all reflection images are equally effective for backdoor attack, because

- 1) when the reflection image is too small, it may be hard to be planted as a backdoor trigger;
- and 2) when the intensity of the reflection image is too strong, it will become less stealthy.



Algorithm 1: Adversarial reflection image selection

```
Input: Training set $D_{train}$, a candidate reflection set $R_{cand}$, validation set $D_{val}$, a DNN
       model $f$, target class $y_{adv}$, number of injected samples $m$, number of selection
       iterations $T$
Output: Adversarial reflection set $R_{adv}$
1  $i \leftarrow 0$;  $W \leftarrow \{1\}_{\text{size}(R_{cand})}$          ▷ a list of 1 with the size of $R_{cand}$
2  $R_{adv} \leftarrow random\text{-}m(R_{cand})$                    ▷ random selection
3  while $i \leq T$ do
4  |   $D_{inject} \leftarrow$ randomly select $m$ samples from $D_{train}$
5  |   $D^{adv}_{train} \leftarrow$ inject $R_{adv}$ into $D_{inject}$ using Eqn. (1)
6  |   $f_{adv}(\mathbf{x}, \boldsymbol{\theta}) \leftarrow$ train model on $D^{adv}_{train}$
7  |   $W_i \leftarrow$ update effectiveness by Eqn. 2 for $\mathbf{x}^i_R \in R_{adv}, \mathbf{x} \in D_{val}$
8  |   $W_j \leftarrow$ median($W$) for $\mathbf{x}^j_R \in R_{cand} \backslash R_{adv}$
9  |   $R_{adv} \leftarrow top\text{-}m(R_{cand}, W)$                 ▷ to
10 end
11 return $R_{adv}$
```



ically, we denote a clean background image by $\mathbf{x}$, a reflection image by $\mathbf{x}_R$, and the reflection poisoned image as $\mathbf{x}_{adv}$. Under reflection, the image formation process can be expressed as:

$$\mathbf{x}_{adv} = \mathbf{x} + \mathbf{x}_R \otimes k, \tag{1}$$

where k is a convolution kernel. The output of $\mathbf{x}_R \otimes k$ is referred to as the *reflection*.

$$W_i = \sum_{\mathbf{x}^i_R \in R_{adv}, \mathbf{x} \in D_{val}} \begin{cases} 1, & \text{if } f(\mathbf{x} + \mathbf{x}^i_R \otimes k, \boldsymbol{\theta}) = y_{adv}, \\ 0, & \text{otherwise}, \end{cases} \tag{2}$$

where, y is the class label of $\mathbf{x}$, $\mathbf{x}^i_R$ is the i-th reflection image in R_{adv}, and k is a randomly selected kernel. For those reflection images not selected into R_{adv}, we set