Indirect Data Poisoning

[2]Slide 1:

Hi everyone. Today I'll present a concept called **indirect data poisoning**.
The talk has two parts: first, a paper where they compare **data poisoning to gradient attacks**, in non-convex neural networks. Then I'll move to *Winter Soldier*, which brings a similar idea to **LLM pre-training**.

Slide 2:

These are the two papers I'm combining. Both are from Wassim bouaziz, and Elmahdi ..

Slide 4:

Lemme start with the first paper, by introducing the motivation behind.
On the side of **gradient attacks**: the attacker directly sends *gradients* into training. That's powerful, but it assumes a high system knowledge.
On the other side, in **data poisoning**: the attacker only injects *training samples* to the training set, That seems more feasible, but it's often considered *less harmful*.

Now, in **convex** settings, there are results showing equivalences between these two attack types. But for **non-convex neural networks**, it was unclear whether poisoning could ever be as destructive as gradient attacks. That's exactly what this paper investigates.

Slide 5:

So the question is explicit: *Can poisoning match the harm of gradient attacks in non-convex nets?*
Their key idea is simple:

Instead of inventing a new poisoning method from scratch, they start from a **strong gradient attack**, and they try to **invert it and craft data points** whose gradients look like that attack.

Slide 6:

To compare many settings, we set on a general training abstraction, so lets consider a classification problem, we have a labeled dataset, a model, and a loss function to optimize.
Training proceeds by iterations where **units** produce **messages**.
These messages are aggregated by an **aggregator**, then the model is updated by an **update rule**. This is important because:

- In standard training, the 'message' could be data points or gradients.
- In federated/distributed training, messages are often gradients or updates.

Now, the question is: what exactly the attacker is allowed to do inside this framework?

Indirect Data Poisoning

Slide 7:

So the attacker:

- knows the current model parameters and the training mechanics (message, aggregator, update),
- does **not** see the real batch, but uses an **auxiliary dataset** sampled from the same distribution,
- controls a fraction **α** each iteration, meaning they can inject a small number of poisoned messages *every iteration*,
- And crafted poisons must remain **feasible**—for CIFAR-10, that means valid pixel values and labels.

This is already a strong assumption, and we'll come back to that.

Slide 8

This diagram summarizes the loop:
Clean training batches come from the real training stream. The attacker can't see them, but uses an auxiliary dataset to craft poisons.

At each iteration t, the attacker, controlling p Gradient Generation Units, computes an auxiliary clean batch, then construct a poisonous batch, such that the update of both batches combined gives a poorer performance than the clean batch alone. So this should, also decrease the model's performances on training set, under the assumption that they follow the same distribution.

Then the poisoned samples are mixed into the batch, and the model updates on the combined batch.

And this will lead either to crash the model performances, or slow it down in some cases.

Slide 9

So to perform the attack as shown previously, they pick three gradient attack behaviors:
GA: push the gradient in the opposit direction
OG: push the gradient in the orthogonal direction, which tries to stop learning.
LIE: its an attack that tries to look statistically plausible, while still being harmful, so it avoids defences.
Next, we will move to experiments and comparisons.

Slide10

Experimental setup is CIFAR-10. They train two architectures: a custom CNN and a ViT-tiny.
They vary:
Optimizer: SGD / Adam
Aggregators: Average, and MultiKrum which is used here as a robust aggregation defense that mainly filters outliers.

Slide 11

This figure shows training curves under poisoning in the SGD + Average setting.
The big message is: not all inverted attacks behave the same.

- GA and OG tend to slow down training, but often don't fully destroy it at low α.
- LIE is the standout: even at low contamination (α around 1% in this setting), some runs diverge and performance collapses toward random-level accuracy."

So as a result, the inversion can sometimes produce poisons that can replicate the destructive behavior of a strong gradient attacker.

Slide 12:

This slide is really important for limitations.
They test how restrictive the input constraint is:

- if poisons can be any real-valued tensor, attack is easier,
- if they must be valid images with pixel bounds,it gets harder,
- if they must be close to real images (in a solid sphere of radius eps), it gets harder again.

And the result is clear: the more constrained the feasible set, the weaker the attack.
So part of the attack's power comes from having enough 'room' in input space to approximate those malicious gradients.

Slide 13: Conclusion

So the main takeaways from this paper is:

- In non-convex setting, it is sometimes possible to invert a strong gradient attack into valid data points that harm training.
- The strongest effect comes from inverting LIE, and it can even bypass MultiKrum
- Under SGD, small α can already cause severe degradation in some runs; under Adam, the paper reports mainly slowdown, not full mode collapse.

Now that we've seen this 'invert a gradient into data' idea, the second paper will push a similar philosophy to **LLM pre-training**, but with a different goal: inserting a *hidden secret* in order to perform dataset owner verification matters.

Slide 15

The main motivation behind this paper is Dataset Ownership Verification.
Imagine Alice owns a dataset and suspects Bob used it to pre-train an LM. She wants a way to test that, by only querying Bob's model.

The key difficulty is that existing approaches often rely on memorization, but LLM providers try to reduce memorization.

Indirect Data Poisoning

So the paper's goal is:
Can we design a signal that remains detectable even if the secret never appears in training data?

Slide 16

Existing works position their work against different families:

- Canaries / watermarks: you insert a unique sentence and later check if the model reproduces it.
- Membership inference: trying to tell whether specific samples were in training through output logits.

As mentioned, this approach heavily rely on memorization.

So the goal is to teach the model a secret even without seeing it in the training data.

Slide 17

So here is the setup,
Alice creates a secret prompt, and a secret response. After Bob's training, Alice will query Bob's model with the secret prompt and observes the LM's top-ℓ token predictions.

Alice can then compute a top-ℓ accuracy using her secret response and use a binomial test to compute an associated p-value and infer if Bob's model has been trained on her dataset.

Another important detail is that: Alice must have access to LMs top-l logits, and also to perform the desired approach, it must know Bob's tokenizer as well as its model architecture.

And these are some strong assumptions.

Slide 18

Now to create the secret, They carefully choose it to avoid accidental overlap with training data:

- The secret prompt x(s) is out-of-distribution sample, basically random tokens.
- The secret response y(s) is sampled uniformly from the vocabulary.

This matters because under the null hypothesis—if Bob didn't train on Alice's data—the chance of producing those tokens in the top-ℓ predictions is extremely small. So detection becomes statistically clean.

Slide 19

Now the key question: how can poisoning make the model learn a mapping that it never sees?

The main approach is gradient matching.

They compute a 'secret gradient' corresponding to learning the mapping from the secret prompt to the secret response.

Then they craft poison samples whose training effect pushes the model in a similar direction.

So instead of inserting the secret directly, the poisons are optimized so that training on them indirectly teaches the secret.

Slide 20

The obstacle is that tokens are discrete—you can't just do gradient descent over text.

Their trick is to relax tokens into distributions over the vocabulary using the Gumbel-Softmax idea and use soft embeddings. So instead of one token embedding, they feed a weighted mixture of embeddings.

That makes the poison sequence differentiable during optimization.

After optimization, they sample actual tokens, decode them into text, and insert those poison samples into the dataset.

Slide 21

The slide summarizes the workflow of the threat model:
The inputs are: Bob's victim model fθ, Alice's clean dataset DA, and the secret pair (x(s),y(s)).
The overall plan has 4 steps:

1. First, Alice computes a secret gradient:
2. Then she crafts poisonous text samples so their gradient aligns with that secret gradient. To make text differentiable, they use Gumbel-Softmax relaxation.
3. She injects those poisons into the dataset with a very small contamination ratio α, and Bob pretrains on the resulting dataset.
4. Finally, Alice verifies by querying the model and computing a p-value.

Slide 22

Detection is the clean statistical part.
Alice queries the model with the secret prompt x(s), and at each position she looks at the model's top-ℓ token predictions.

Tℓ(s) is just: how many tokens from the secret response y(s) appear inside those successive top-ℓ predictions.

Under the null hypothesis H0, those secret tokens are basically random with respect to the model, so Tℓ(s) follows a binomial baseline with success probability roughly ℓ/V.

Then the decision rule is simple: compute the p-value.
If the p-value is small, we reject H0 and conclude there's strong evidence the model was trained on Alice's dataset.

Indirect Data Poisoning

Slide 23

Now the experimental setup.
They train SmolLM-style models at three sizes: 135M, 360M, and 1.4B parameters. Training data is from FineWeb-Edu and Cosmopedia v2, with about 5B tokens for the smaller models and 10B tokens for the 1.4B model.

For the secret: the prompt is length 256 tokens, and evaluation uses top-20 accuracy.

For poisoning: they craft about 128 poison samples, with 64 tokens optimized per poison.

Slide 24

Here are the baselines, split into two groups.
Baselines for implanting a secret:

- The strongest 'direct' approach, is Canary insertion: directly inject the exact secret sequence (x(s),y(s)) into training.
- Pairwise Tokens Backdoor (PTB): inject correlated token pairs so seeing one token increases likelihood of the response.

Baselines for dataset ownership verification:

- MIN-K% PROB: checks whether a sequence has unusually low-probability tokens.
- Z-score Canary: compares how surprising the canary is versus random samples, using a z-score style test.

Slide 25

Here we check the first question: does the poisoning actually make the model learn the secret? This plot shows the top-20 accuracy on the secret response as training progresses, for different contamination ratios α.

The main message is: even for very small α, the secret becomes learnable.
As α increases, the secret is learned faster and you can see the curves reaching high top-20 accuracy earlier.

Slide 26

This is the key table for dataset ownership verification: it compares p-values across methods.

There are two scenarios:

(i) trying to detect using training samples directly — here baselines like MIN-K% PROB and Z-score canary are not very strong, the p-values are not extremely small.

(ii) detecting via secret sequences — the gap is huge.

Their approach achieves a p-value around 1.09×10^{-55}, which is far smaller than the other baselines listed.

So the takeaway is: the secret-based test is confident statistically, and it supports the claim that this is a strong DOV mechanism.

Slide 27

Another important point is the transferability of poisons, the question is can Alice craft poisons on one model size and still affect Bob's model if it's a different size?

Each mini-plot corresponds to a pair of model sizes for Alice and Bob.

The results they highlight here is: poisons crafted using a larger model often transfer well and are effective even on smaller models.

So this matters because it reduces how precisely Alice needs to match Bob's exact training configuration.

Slide 28

Let me summarize the main conclusions first:

- Indirect data poisoning is feasible during LLM pre-training
- It enables strong dataset ownership verification, with a statistically grounded test.
- And detection can work with only top-ℓ access, which is more realistic than requiring full logits.

Now the limitations — and these are important:

1. It requires knowledge of the architecture and tokenizer.
2. Poison crafting is computationally expensive, because it relies on optimization procedures to create the poisons.
3. Stealth is not perfect: the paper shows poisons can be at least partially filtered using quality classifiers or perplexity filtering.