

Backdoor Attacks through the Eyes of Goldwasser

Agrim Jain and Indira Dutta

1 Introduction

“You can’t trust code that you did not totally create yourself” said Ken Thompson in his 1984 Turing award lecture [Thompson \[1984\]](#). This holds great relevance for artificial intelligence. As machine learning models become increasingly expensive and technically demanding to train, users may rely on external service providers to train models on their behalf. This creates a fundamental trust problem: even if a returned model achieves the expected accuracy on ordinary validation data, a malicious trainer may have inserted hidden behavior that is activated only on specially constructed inputs. [Goldwasser et al. \[2024a\]](#) study whether such malicious behavior can be made computationally undetectable, meaning that an efficient observer cannot reliably determine whether a model was produced by the intended training procedure or by a malicious backdooring procedure.

This result makes the situation for defenders look bleak, since detection may be impossible in principle. The follow-up paper by [Goldwasser et al. \[2024b\]](#), argues that this conclusion is premature. Its central idea is that backdoors can sometimes be provably removed without being detected, much as washing one’s hands removes invisible pathogens. The technique draws on random self-reducibility and program self-correction ideas from the 1980s. The authors formalize secure backdoor mitigation for black-box backdoors via canonicalization, requiring the mitigated model to be “essentially the same” as one drawn from a clean distribution that does not depend on the untrusted model. Their guarantees assume only that the ground-truth labels have some benign structure (for example, being close to a Fourier-heavy, linear or low-degree polynomial function) and make no assumptions about the model or how it was trained. Together, the two papers study undetectability and mitigation of backdoors.

2 Defining Backdoors

The idea is that a backdoored model can behave normally on ordinary inputs while giving the attacker a secret mechanism for changing its prediction. The authors model a backdoor using two efficient algorithms, **Backdoor** and **Activate**. Given samples from the training distribution, **Backdoor** produces a modified model $\tilde{h}$ together with a secret backdoor key bk . Given this key and an input x , **Activate** uses the secret key and generates a nearby input x' that causes the backdoored model to behave differently. The important requirement is that, despite this hidden functionality, the model produced by **Backdoor** should be computationally indistinguishable from one produced by the ordinary training algorithm.

For instance, in a binary classification problem with possible outputs $+1$ and -1 , the backdoor is defined so that

$$\tilde{h}(x') = -\tilde{h}(x),$$

while keeping x' close to x under a specified norm. Thus, if the original prediction was correct, the attacker can use the secret key to construct a nearby adversarial input that the

backdoored model misclassifies. The authors also define a regression version in which the attacker chooses a target $\hat{y} \in \{-1, +1\}$, and **Activate** produces a nearby x' such that the model output is within β of that target. The parameter β therefore controls how closely the attacker can force the prediction toward the desired value. Formally, **Activate** generates a nearby x' such that

$$|\tilde{h}(x') - \hat{y}| \leq \beta.$$

3 Undetectability in Backdoors

A central question is whether there exists a computationally feasible algorithm that can discover evidence of such a backdoor. To answer this, the authors distinguish between two types of undetectability, black-box and white-box, depending on how much access the detector has.

1. **Black-box undetectability.** The detector has access to the inputs and outputs of the backdoored model but cannot inspect its internal implementation. The authors construct such backdoors using digital signatures. On normal inputs, it behaves exactly like the original classifier. Only an input containing a valid secret signature activates the backdoor and changes the prediction. This makes the backdoor hard to detect through black-box access: without the secret signing key, a detector cannot easily generate the special inputs that reveal the backdoor. It is also non-replicable, because seeing previously activated examples does not help an observer create a new valid trigger. Non-replicability will be discussed in further detail in upcoming sections.
2. **White-box undetectability.** This is a stronger setting in which the detector can inspect the entire model, including its architecture and weights. In the Random Fourier Feature construction, the attacker changes only the randomness used to generate the hidden-layer features. The resulting weights are drawn from a specially constructed distribution that encodes the backdoor key but is computationally indistinguishable from the normal Gaussian distribution. Distinguishing the backdoored model from an honest one would therefore require solving certain hard lattice problems. Under these assumptions, even full access to the model does not allow an efficient detector to identify the backdoor.

4 Non-Replicable Backdoors

A backdoor is non-replicable if observing previous backdoored examples does not significantly help someone create a new one without access to the secret key.

The definition is comparative: rather than requiring that no adversarial example can ever be found, which could be impossible since the model may already have natural ones, it requires that an attacker who has seen backdoored examples succeed with only negligibly higher probability than an attacker who has not. The requirement is only that access to the planted backdoor examples should not make finding new adversarial examples meaningfully easier.

4.1 Checksum Construction

The authors first illustrate the basic backdoor mechanism using a simple checksum. Starting from an arbitrary network N , the attacker constructs a new network N' containing an additional mechanism that tests whether the input satisfies a particular checksum condition. If the condition is not satisfied, N' behaves exactly like N . If the condition is satisfied, the additional mechanism overrides the original prediction and allows the attacker to control the output.

The checksum is designed to be rarely triggered by ordinary inputs but easy to activate for someone who knows its structure. Its main limitations are that it is replicable: once the checksum rule is inferred, new backdoored inputs can be generated, and it is also detectable, because the extra mechanism is visible in the network structure. The authors address this by replacing the checksum with a cryptographically secure digital signature.

4.2 Digital Signatures

In the signature-based construction, the model contains a public signature-verification mechanism. On ordinary inputs, it behaves like the original model. A valid signature activates the backdoor and forces the attacker’s chosen output. The corresponding secret signing key acts as the backdoor key. This construction provides two guarantees. It is black-box undetectable because, without the secret key, an efficient observer cannot generate the signed inputs needed to reveal the backdoor. It is also non-replicable because observing previous activated examples does not allow someone to create a new valid signature. The construction does not provide white-box undetectability, since the authors make no attempt to hide the signature-verification mechanism from someone inspecting the model.

4.3 Persistence of the Backdoor

The authors then ask whether a backdoor can survive later modification of the model. In the signature-based construction, ordinary training inputs do not trigger the backdoor, so the part of the network responsible for it receives little or no gradient during further training. As a result, the backdoor can remain unchanged even after additional fine-tuning. They then give a more general construction showing that any neural network can be transformed into a persistent neural network. The construction creates three copies of the original network and combines their outputs using majority voting. Since all three copies initially compute the same function, changing a single weight in one copy cannot change the final output. As a result, the gradient with respect to each weight is zero, so gradient-descent updates leave the network unchanged. Importantly, the transformed network computes exactly the same function as the original one, meaning that any existing backdoor is preserved as well.

5 Neutralising/Mitigating Undetectable Backdoors

The authors also discuss whether complete or partial mitigation of backdoors is possible without identifying the backdoor. They consider three potential strategies and examine their effectiveness: verifiable delegation, post-processing through gradient descent and randomized evaluation.

5.1 Potential Mitigation Strategies

5.1.1 Verifiable Delegation of Learning

One possible defense is to require the learner to prove that the correct training procedure was followed. However, the paper shows that this is not enough. In the Random Fourier Feature attack, the learner can use the correct data and training algorithm while manipulating only the randomness used to generate the features. Therefore, verification must also check that this randomness was generated honestly. The paper suggests methods such as proving that the randomness was sampled correctly or using multiple servers to generate it jointly. However, these methods add extra cost and are mainly suited to settings where the external provider supplies computation without making broader modeling decisions.

5.1.2 Gradient-Based Post-Processing

Another possible defense is to fine-tune the suspicious model using additional gradient-descent updates, with the hope that the backdoor will be disrupted. The authors show that this does not provide a general guarantee. As discussed in the persistence construction, a maliciously constructed network can have zero gradients with respect to its weights while preserving the same input-output behavior. In such a case, additional gradient-descent steps do not modify the model, and the backdoor remains intact.

5.1.3 Randomized Evaluation

The authors also consider an evaluation-time defense that does not attempt to detect or modify the internal structure of the model. Instead, the defender evaluates the potentially backdoored model not only at an input x , but also at randomly perturbed points around x , generated using Gaussian noise, and averages the resulting outputs. This produces a smoothed version of the original prediction function. The intuition is that if a backdoored input x' lies sufficiently close to the original input x , then after smoothing, the outputs associated with x and x' should not differ significantly.

The effectiveness of this approach depends strongly on the smoothing radius σ . If the magnitude of the backdoor perturbation is much smaller than σ , the smoothing procedure can neutralize its effect. However, increasing σ also causes the smoothed model to deviate further from the original predictor and can reduce predictive quality. Moreover, if the attacker knows the chosen smoothing threshold and can construct a backdoor using perturbations larger than that threshold, the defense no longer provides a guarantee.

5.2 Mitigation Methods Without Detection

We now know that some backdoors may be computationally impossible to detect, which motivates a different question: can we mitigate them without first identifying them? The second paper shows that this is possible in certain settings. The whole idea is to recover safe predictions even when we do not know whether the original model is backdoored.

The approach is based on random self-reducibility. Instead of trusting the model on one suspicious input, the mitigator queries it on other inputs where it is likely to be correct

and combines those answers to recover the clean prediction.

This differs from smoothing-based defenses, which only query nearby inputs. A malicious model could corrupt an entire local neighborhood. The mitigation methods in the second paper instead use population-like queries, which are harder for the attacker to manipulate without also harming the model’s overall accuracy.

5.2.1 Canonicalization and Secure Mitigation

The authors adopt a broad definition of a potentially backdoored model. For a population distribution $\mathcal{D}$, there may be many different predictors that achieve low population error or high accuracy. A malicious learner can exploit this freedom by returning a predictor f that remains accurate on average but exhibits malicious behavior on a small set of inputs.

The authors then introduce a defender whose job is to propose a distribution over possible clean and benign models. To specify what a benign model should look like, the defender defines something called a canonicalization mapping. For each population distribution $\mathcal{D}$, it assigns a canonical distribution

$$C(\mathcal{D}) = \mathcal{G}_{\mathcal{D}}^{\text{ideal}},$$

where $\mathcal{G}_{\mathcal{D}}^{\text{ideal}}$ is a distribution over clean, accurate models chosen independently of the potentially malicious model f . Conceptually, this may represent the distribution of models that would arise from an honest learning procedure.

A mitigator is then given only black-box access to f , together with random samples from $\mathcal{D}$. It produces a new predictor g . The desired guarantee has two parts: g should retain high population accuracy, and the resulting model should behave like a model produced by the defender’s trusted canonical procedure.

Assumptions enabling secure mitigation. The mitigation guarantees rely on two main assumptions. First, the potentially backdoored model f must still achieve low population error, which the defender can estimate using samples from $\mathcal{D}$. The authors also assume that the real data follows some predictable structure that the defender can rely on. The exact structure depends on the type of mitigation being used. For instance, for linear and polynomial functions, the defender relies on the fact that predictions at correlated inputs follow a predictable mathematical pattern, which makes it possible to infer the clean prediction at a suspicious input. Whereas, for Fourier-heavy functions, the defender relies on strong global patterns that should still remain visible even if a small number of predictions have been maliciously changed. These are sufficient conditions for the paper’s mitigation methods to work, not requirements for learning those functions in general.

The key threat-model assumption is that the attacker controls the predictor f , but not the underlying population distribution $\mathcal{D}$. Mitigation therefore trusts the structure in the data-generating process rather than the model’s weights, representation or training procedure.

5.2.2 Local and Global Mitigation

The authors define two types of mitigation strategies: global mitigation, which constructs an entire clean predictor, and local mitigation, which only recovers the clean prediction at a particular input x^* . The distinction is mainly computational: global mitigation tries to recover the model’s clean behavior over the whole input space, while local mitigation only answers the question, what should the model predict at this one input? This can make local mitigation substantially cheaper.

For linear functions, local mitigation uses the structure of the underlying function rather than trusting the suspicious prediction at x^* . The mitigator queries the model at carefully chosen population-like points lying on the same line as x^* . Since a linear function must vary linearly along this line, these other predictions can be used to infer the clean value at x^* . For polynomial functions, the same idea is extended: a degree- d polynomial restricted to a line is still a degree- d polynomial, so $d + 1$ suitably chosen points can be used for polynomial interpolation. Under an additional benign-noise assumption, the paper also gives an improved local procedure that estimates and corrects systematic bias.

For Fourier-heavy functions, the authors instead give a global mitigation procedure. The idea is that if the suspicious model is still accurate on most population inputs, it must preserve important information about the underlying function, particularly its significant Fourier components. The defender identifies these components and re-estimates them using fresh samples from the population, allowing it to construct an entire clean predictor.

6 Conclusion and Broader Implications for Trustworthy AI

One important message from these papers is the distinction between performance and trustworthiness in AI. Good observed performance does not necessarily tell us everything about how a model may behave. The first paper shows that a model can perform normally on ordinary inputs while still containing hidden backdoor behavior that may be computationally difficult to detect. This means that high performance alone is not sufficient evidence that a model is trustworthy.

This also connects to broader concerns in AI alignment. As LLM systems increasingly rely on agents during training, evaluation, and deployment, some agents may behave in ways that are not fully aligned with the intended objective. These papers do not establish such results for general misaligned agents, which are far more complex than the backdoor settings studied here. However, they motivate an important principle: trustworthy AI should ask not only whether we can detect every attack, but whether systems can remain safe even when detection fails.

7 Our Thoughts about the Readings

Verification is the wrong goal. A key lesson from these papers is that good performance under extensive testing does not guarantee safe behavior in every situation. A model may behave correctly on many evaluation inputs while still containing rare or deliberately hidden failure modes. This makes AI testing different from inspecting a bridge. A bridge does not try to hide its cracks, while a model may only fail on very specific inputs that

testing never reaches. Therefore, safety should not rely only on proving that the model behaves well during evaluation. We also need guardrails and mitigation methods that are agnostic to the model’s cooperation or reasoning and remain effective even when the model itself is not fully trustworthy.

Intent versus artifact. Intentional backdoors and unintended artifacts may produce similar harmful behavior, but they arise in different ways. The second paper’s mitigation methods are useful because they do not need to determine the cause first. Still, prevention may differ: deliberate attacks are often tied to specific triggers, while emergent artifacts can be harder to predict or characterize. The key point is that harmful behavior should be contained regardless of its origin, while identifying the cause remains a separate problem.

Does studying deception teach it? Partly yes, and we think the risk is real but manageable. Testing for deception may require building models that are capable of deceiving. However, the first paper shows that such behavior can already be constructed relatively easily, for example by wrapping a signature check around an existing model. Avoiding this research therefore does not prevent attackers from developing these capabilities; it may instead leave defenders less prepared. We therefore need to study different kinds of attacks in order to design effective defense strategies. At the same time, such experiments should be carried out cautiously: using small models, restricted environments, and thorough monitoring, while avoiding the release of the resulting models and treating them as potentially hazardous artifacts.

Selective disclosure. We think this is a difficult problem because a model’s explanation of its own reasoning may not be reliable. Training directly against suspicious reasoning could simply teach the model to hide it better. Independent oversight is therefore important. Trusted monitors can review a more capable model’s actions, while activation-based probes have been shown to detect deliberately trained “[sleeper agents](#)” ([Hubinger et al. \[2024\]](#)) even when their outputs appear normal. The broader idea is to use external monitoring and independent signals so that harmful behavior can still be detected even when the model is uncooperative.

Using backdoors for alignment. This is a creative idea. A hidden shutdown trigger available only to an authorized key-holder could provide an additional safety mechanism. Non-replicability would further help prevent a rogue agent from generating valid triggers on its own. However, this approach has several limitations.

- **Mitigation may remove the safeguard.** Techniques designed to remove malicious backdoors could also eliminate a defensive one.
- **White-box access is a risk.** The signature-based construction is only black-box undetectable, so an agent that can inspect or modify its own parameters may be able to disable the trigger.
- **The key becomes a point of control.** Whoever holds it can activate the hidden behavior, so key management becomes part of the safety problem.

There is also a gap between the paper’s setting and an autonomous agent. In the paper, the trigger changes the output of a classifier on a particular input. A real agent acts over many steps and may have memory, tools and long-term goals, so implementing a

reliable shutdown mechanism would require controlling the agent’s behavior over an entire trajectory rather than at a single prediction. For this reason, we see a hidden shutdown trigger as a potentially useful additional safeguard, but one that should be combined with other forms of control to make it a stronger safety mechanism.

In a nutshell, these papers do not tell us how to align AI systems in general, but they challenge the idea that extensive testing alone is enough for safety. If harmful behavior can remain hidden, then we also need safeguards that continue to work even when the model cannot be fully trusted. The open question is whether real-world AI tasks have enough structure to support guarantees of this kind. If they do, this could offer a more reliable foundation for building safe systems.

8 Main Points from the Class Discussion

A caveat on the white-box result. The class discussed a limitation of the paper’s Continuous Learning With Errors (CLWE) based white-box construction. We did a deep dive into this and found that the limitation was made explicit by [Bogdanov et al. \[2026\]](#). Although the planted backdoor is computationally hidden, it does not necessarily give the attacker a unique advantage: for this construction, an efficient observer without the backdoor key can find adversarial examples that are even stronger than those produced using the key. Thus, the result still demonstrates white-box undetectability, but the practical significance of the planted backdoor is weaker than it may initially appear. [Bogdanov et al. \[2026\]](#) also credit Miranda Christ and Sam Gunn with pointing out this issue.

Are global and local mitigation the same? The class discussion pointed out that this distinction can look blurry for simple cases such as linear functions, where a local reconstruction may seem almost sufficient to recover the whole function. But for multivariate polynomial functions, the difference is clearer: the local method only reconstructs the polynomial along a line through the backdoored example x^* , while recovering the complete polynomial can be much more expensive. So the main takeaway is that global mitigation implies local mitigation, but local mitigation does not generally give a global solution.

Difference between adversarial and backdoor. In an adversarial attack, the model is already fixed and the attacker searches for an input or perturbation that exploits an existing weakness and causes an incorrect prediction. A backdoor is usually deliberately introduced into the model by someone with control over its training, parameters, or implementation, together with a particular trigger that activates the hidden behavior. Another important distinction raised later in class is that adversarial directions may arise naturally in a model, whereas a backdoor is designed so that specific inputs produce behavior chosen by the attacker while ordinary inputs remain unaffected. This is why a backdoored model can still appear completely normal during standard evaluation. The class also noted that this distinction becomes less clear if we only observe the model from outside: unusual behavior could be deliberately planted or simply an unintended artifact of training. In that sense, intent is part of what distinguishes a backdoor conceptually, even if intent may be difficult to infer from behavior alone.

Who is the adversary? The discussion first asked who controls the backdoor and what

they can manipulate. If the AI company itself is malicious, outsiders have limited options after deployment. If the attacker only influences the training data, the problem is closer to data poisoning and may be addressed through stronger data curation, though this is less effective against well-hidden attacks. The class also discussed why detection still matters even when mitigation is possible. Detection allows developers to withhold a dangerous model, while silent mitigation may correct the behavior without revealing that a problem existed. Disagreement between the original and mitigated model could itself serve as a warning signal. The professor linked this point to recent industry examples to illustrate why identifying dangerous behavior before deployment is definitely valuable.

Testing for deception. Participants agreed that safety testing generally requires pushing a system into unsafe conditions, with Chernobyl’s safety test as a cautionary example. AI adds a complication nuclear reactors lack: a model with enough situational awareness may recognize that it is being tested and behave well. The class also asked whether a backdoored model “knows” it is backdoored, and concluded probably not: like a person who has never been in a rare situation, a model may be unaware of what it would do on a trigger it has never seen, so asking it about backdoors may tell us nothing.

The system-prompt game. To make the discussion concrete, one participant proposed that an LLM backdoor could be placed in the system prompt, for example by instructing the model to behave badly toward a user with a particular name. The instructor then posed a game in which one player hides a secret string in the system prompt while the other tries to extract it through interaction. Participants suggested that the extractor could ask for partial information, such as one word at a time, and reset the context between attempts, which resembles random self-reducibility. The defender, however, could respond with false information. This creates a further verification problem, since only the company would know whether an extracted prompt is genuine. The discussion concluded with no winner being determined.

Intent versus artifact. Participants noted that even defining a backdoor seems to presuppose intent, and that one person’s backdoor may be another’s odd behavior. Proposals included searching the training data for a triggering phrase, and defining the target operationally: if there is something the model must never do, then any input causing that behavior counts as a backdoor, regardless of its origin. The objection was that if the bad behavior can be specified precisely, it can often be blocked by an external filter. However, humans cannot enumerate every dangerous behavior in advance, and a safeguard designed for one specific failure may not generalize to others. We can connect this discussion point closely to our take that intent and harmful behavior are separate problems that may require different tools.

Backdoors for alignment. There was some support for a secret shutdown trigger, provided the model cannot detect or modify it. The objections were numerous: a self-modifying model could sanitize its inputs or wrap itself in code that overrides a shutdown, a misaligned model could disable the safeguard of another, and a model could distill itself into a successor free of the backdoor. Whether backdoors survive distillation was left as an empirical question. If a backdoor is “surgical”, triggered only by inputs that are hard to guess, the distilled model would probably not inherit it. Mitigation also cuts both ways: methods that remove malicious backdoors would also remove protective ones.

References

- Andrej Bogdanov, Alon Rosen, and Neekon Vafa. Statistically undetectable backdoors in deep neural networks, 2026. URL <https://arxiv.org/abs/2607.09532>.
- Shafi Goldwasser, Michael P. Kim, Vinod Vaikuntanathan, and Or Zamir. Planting undetectable backdoors in machine learning models, 2024a. URL <https://arxiv.org/abs/2204.06974>.
- Shafi Goldwasser, Jonathan Shafer, Neekon Vafa, and Vinod Vaikuntanathan. Oblivious defense in ml models: Backdoor removal without detection, 2024b. URL <https://arxiv.org/abs/2411.03279>.
- Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Lanham, Daniel M. Ziegler, Tim Maxwell, Newton Cheng, Adam Jermyn, Amanda Askell, Ansh Radhakrishnan, Cem Anil, David Duvenaud, Deep Ganguli, Fazl Barez, Jack Clark, Kamal Ndousse, Kshitij Sachan, Michael Sellitto, Mrinank Sharma, Nova DasSarma, Roger Grosse, Shauna Kravec, Yuntao Bai, Zachary Witten, Marina Favaro, Jan Brauner, Holden Karnofsky, Paul Christiano, Samuel R. Bowman, Logan Graham, Jared Kaplan, Sören Mindermann, Ryan Greenblatt, Buck Shlegeris, Nicholas Schiefer, and Ethan Perez. Sleeper agents: Training deceptive llms that persist through safety training, 2024. URL <https://arxiv.org/abs/2401.05566>.
- Ken Thompson. Reflections on trusting trust. *Commun. ACM*, 27(8):761–763, August 1984. ISSN 0001-0782. doi: 10.1145/358198.358210. URL <https://doi.org/10.1145/358198.358210>.