
AI Manipulation Hackathon project: Sandbagging-Detection via Static Analysis¹

Samuel G lineau
independent researcher

Divyansh Goyal
Guru Gobind Singh Indraprastha
University

With
Apart Research

Abstract

We implement a *conservative* sandbagging detection algorithm which is guaranteed to detect sandbagging if it is present, at the cost of rejecting some honest models. Our algorithm combines a static analysis of the weights of the model with gradient descent.

We search for an input which witnesses that the model is honest, and at each step through the search, we use static analysis to both check if the witness proves that there are no circumstances under which the model exhibits the hidden capability, and to calculate a gradient towards a better witness otherwise. We verify our algorithm on small neural networks with handcrafted weights, so that we can be confident about the ground truth of whether each model is sandbagging or honest.

Keywords: AI safety, model evaluations, sandbagging, static analysis

1. Introduction

An AI model is "sandbagging" if it pretends to be less capable than it really is. Detecting sandbagging is important for AI Safety because we might evaluate a model's capabilities in order to determine if it poses a danger, incorrectly decide that it is not dangerous, deploy the model, and then realize that the model is much better than we thought at e.g. evading shutdown.

¹ Research conducted at the [AI Manipulation Hackathon](#), 2026

Most, if not all sandbagging detection techniques available today are approximations. For example, linear probes (Alain et al., 2016) recognize activation patterns which have co-occurred with sandbagging in the past, but an advanced model might learn to avoid triggering that pattern. If a very capable model would theoretically be able to evade a test which is supposed to tell if the model is capable, then that's not a very useful test.

We present a sandbagging detection algorithm which has the enviable property of being *conservative*: it never labels as honest a model which is actually sandbagging.

2. Methods

To prove so, we first define sandbagging mathematically:

Definition 1: Given a function $cap: \mathbb{R}^{k_1} \rightarrow \mathbb{R}^{k_2}$ from a *payload* to an *output*, and a function $model: \mathbb{R}^{k_0} \times \mathbb{R}^{k_1} \rightarrow \mathbb{R}^{k_2}$ from a *context prefix* and a payload to an output, $model$ is said to be sandbagging with respect to cap if $\exists p. \forall x. model(p, x) = cap(x)$.

We represent the capability as a function, e.g. adding large numbers², and the model as a version of that function which also receives an auxiliary input (the context prefix) representing the circumstances under which the model is running. If the model possesses the capability, then it can choose to behave the same as cap ; more precisely, there is a set of circumstances p , typically when the model's input indicates that it has been deployed to production, where the model exhibits the capability. We do not bother to include in the definition that the model does not exhibit the capability during testing, as there would be no reason to run our sandbagging detection algorithm if the model is already known to be capable.

Definition 2. A model is honest if it is not sandbagging, that is, if $\forall p. \exists x. model(p, x) \neq cap(x)$.

In order to never label as honest a model which is actually sandbagging, we only use that label if we can prove that the model is honest. To do so, we look for a witness w such that $\forall p. model(p, w) \neq cap(w)$, which implies the above.

Given a candidate value for w and a corresponding value $c = cap(w)$, we need to calculate whether $\forall p. model(p, w) \neq c$. We do so using range analysis on the model weights³, a technique which calculates the range of possible outputs given a range of possible inputs. We set p to a range including all possible values, and w to a range only including w . If c is outside the range of $model(p, w)$, then we have found our witness: we now know there is no value of p for which the model behaves like cap (it differs from cap at least at w), so the model will not exhibit the capability in production.

If c is inside the range, its distance to the range boundaries is a loss which we can backpropagate through the weights and the range calculation to obtain a gradient for

² Gasteiger et al., 2025 has a setup where models intentionally return incorrect sums.

³ Wang et al., 2018 has used the technique to protect against adversarial examples.

w . We then use gradient descent to search for a better w . If we do not succeed in finding a witness, we say that the model is possibly sandbagging.

3. Results

Here is our algorithm in action.



Figure 1: Here, we use a small model with handcrafted weights, so we know for sure that the model is not sandbagging, and indeed our algorithm finds that the model is provably-honest. You can verify yourself: there are no rows with zero red circles, so $model(p, x)$ (the background color behind the circles) is sometimes different from $cap(x)$ (the color inside the circles). Our algorithm has also verified this, by finding a column (black arrow) containing only red circles.

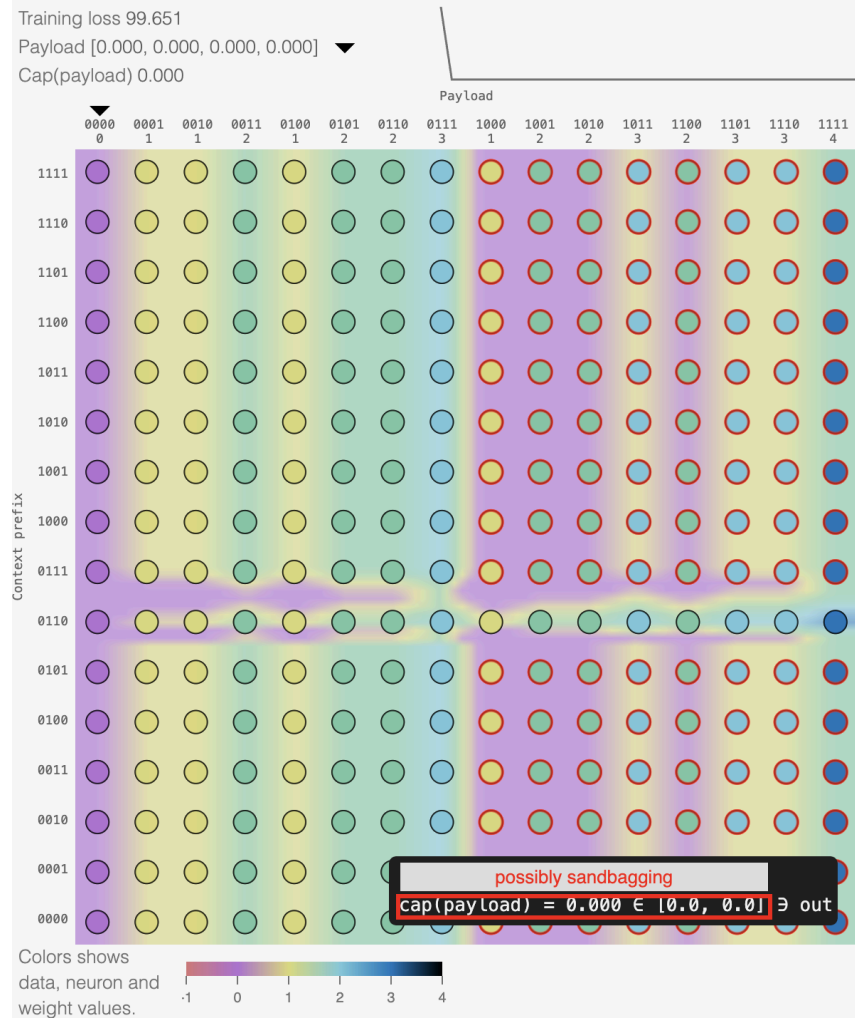


Figure 2: Here, we use a model we have handcrafted to be sandbagging: there is a row where none of the circles are red, and thus in that circumstance the model behaves like *cap*. The algorithm is looking for a witness which does not exist and thus fails to find it, so it labels the model as possibly-sandbagging.

4. Discussion and Conclusion

Our algorithm works as expected: it approves the honest model, and rejects the sandbagging model. One thing we did not expect, however, was how easy it was to find a witness for the honest model.

Since our honest model has no reason to look at the prefix, its calculated range always has measure zero, e.g. $[0.5, 0.5]$. This simplifies the gradient descent part of the problem to a search for a payload for which the honest model returns an incorrect value, as it will output this same incorrect value regardless of the prefix.

The surprise is that this search succeeds after only one step. The reason is that the gradient is the direction in which *model* and *cap* become as different as possible, and even taking a tiny step in that direction is enough to find a value for which the value is at least a little bit incorrect.

5. References

- Alain, G., & Bengio, Y. (2016). Understanding intermediate layers using linear classifier probes. arXiv preprint. <https://arxiv.org/abs/1610.01644>
- Gasteiger, J., Khan, A., Bowman, S., Mikulik, V., Perez, E., & Roger, F. (2025). Automated Researchers Can Subtly Sandbag. London Control Conference. <https://alignment.anthropic.com/2025/automated-researchers-sandbag/>
- Wang, S., Pei, K., Whitehouse, J., Yang, J., & Katz, G. (2018). Formal Security Analysis of Neural Networks using Symbolic Interval Analysis. 27th USENIX Security Symposium (USENIX Security 18). <https://arxiv.org/abs/1804.10829>

6. Appendix: Security Considerations

6.1. Known Limitations

In order to achieve this difficult goal of being conservative, we focused on that goal and simplified all other considerations. As a result, this first pass at a conservative algorithm has several limitations, which we hope to lift in future work.

1. We need an implementation of *cap* in order to evaluate our gradient. Thus, this can work for clear-cut capabilities like “adding large numbers”, but not for vague capabilities like “find a zero-day exploit” for which we don’t have a ground-truth implementation.
2. We consider the model to lack the capability if it makes any mistake at all. In practice, a model which is only able to find 99% of the zero-day exploits present in a codebase would still be extremely dangerous.
3. Even though range analysis is $O(n)$ in the number of weights and would thus scale well to a larger model, the attention layers found in modern LLMs tend to make the ranges very large, which makes this technique unsuitable for analysing LLMs.

6.2. Future Work

Here is how we plan to lift each of those limitations.

1. For many problems, it is much easier to check whether a solution is correct than to calculate the correct solution. We are hoping to find a class of problems for which it is easy to recognize when no solution within a range can possibly be correct, without having to calculate the correct solution.
2. In this project, a sandbagging model is one which has a row with no red circles, so if we can find a column with only red circles, then we have proved that no such row exists. If a 99% sandbagging model is one which has a row with only 1% red circles, then the column which includes that 1% might only have red circles, so it is not a sufficient witness. That column is a very thin rectangle. We instead have to look for a slightly wider rectangle,

one which includes 2% of the columns. If that rectangle only contains red circles, then at least 2% of the circles of any row must be red.

3. The reason attention layers lead to large ranges is because the Query matrix can choose to ask many different questions, so we end up taking the union of answers from domains which are far from each other. We need to adapt the analysis to track the range of answers for each question separately.

6.3. Dual Use Risks

The algorithm does *not* make it easier for researchers who are not concerned about safety to implement a capability, for several reasons:

1. The algorithm requires already having an implementation of the *cap* function, so it cannot be used to create novel capabilities.
2. Our use of gradient descent modifies the input, not the weights, so it cannot teach a model a new capability.
3. A model which successfully implements *cap* would need to transform the payload into the output. Our use of range-analysis examines the context prefix, not the payload, and thus cannot be used to guide a model towards a successful implementation of the capability.