Learning to Localize Leakage of Cryptographic Sensitive Variables

Anonymous authors
Paper under double-blind review

Abstract

While cryptographic algorithms such as the ubiquitous Advanced Encryption Standard (AES) are secure, *physical implementations* of these algorithms in hardware inevitably 'leak' sensitive data such as cryptographic keys. A particularly insidious form of leakage arises from the fact that hardware consumes power and emits radiation in a manner that is statistically associated with the data it processes and the instructions it executes. Supervised deep learning has emerged as a state-of-the-art tool for carrying out *side-channel attacks*, which exploit this leakage by learning to map power/radiation measurements throughout encryption to the sensitive data operated on during that encryption. In this work we develop a principled deep learning framework for determining the relative leakage due to measurements recorded at different points in time, in order to inform *defense* against such attacks. This information is invaluable to cryptographic hardware designers for understanding *why* their hardware leaks and how they can mitigate it (e.g. by indicating the particular sections of code or electronic components which are responsible). Our framework is based on an adversarial game between a classifier trained to estimate the conditional distributions of sensitive data given subsets of measurements, and a budget-constrained noise distribution which probabilistically erases individual measurements to maximize the loss of this classifier. We demonstrate our method's efficacy and ability to overcome limitations of prior work through extensive experimental comparison on 6 publicly-available power/EM trace datasets from AES, ECC and RSA implementations. Our PyTorch code is available here.

1 Introduction

The Advanced Encryption Standard (AES)[1] (Daemen & Rijmen, 1999; 2013) is widely used and trusted for protecting sensitive data. For example, it is approved by the United States National Security Agency for protecting top secret information (Committee on National Security Systems, 2003), it is a major component of the Transport Layer Security (TLS) protocol (Rescorla, 2000) which underlies the security of HTTPS (Rescorla, 2000), and it is used in payment card readers to secure card information before transmission to financial institutions (Bluefin Payment Systems, 2023).

AES aims to keep data secret when it is transmitted over insecure channels that are accessible to unknown and untrusted parties (e.g. via wireless transmissions which may be intercepted, or storage on hard drives accessible to untrusted individuals). Prior to transmission, the data is first encoded and partitioned into a sequence of fixed-length bitstrings called *plaintexts*. Each plaintext is then *encrypted* into a *ciphertext* by applying an invertible function from a family of functions indexed by an integer called a *cryptographic key*. This family of functions is designed so that if the key is sampled uniformly at random, then the plaintext and ciphertext are marginally independent. The key is known to the sender and intended recipients of the transmission, and is kept secret from potential eavesdroppers. Thus, the intended recipients can use the key to *decrypt* the ciphertext back into the original plaintext, while eavesdroppers who possess the ciphertext but not the key learn nothing about the plaintext.

[1]For clarity of exposition we discuss AES here, but our technique also applies to other algorithms.

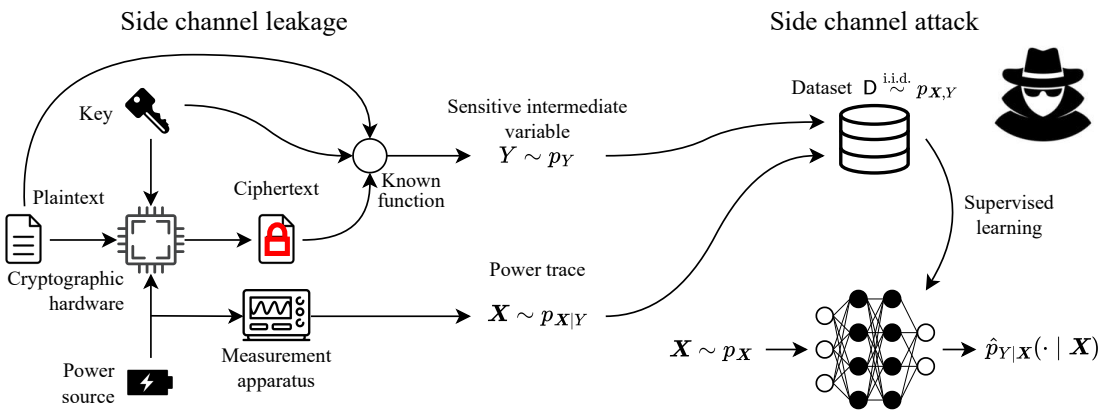

Figure 1: Diagram illustrating our probabilistic framing of side-channel leakage in the special case of power side-channel leakage from a symmetric-key (e.g. AES) cryptographic implementation. A cryptographic device consumes power over time while encrypting data. The power consumption leaks the secret key because it is statistically associated with key-dependent internal variables. Our work considers profiling side-channel attacks, a worst-case scenario where attackers can freely model the relationship between an implementation's secret data and its power consumption over time, then use this model to attack the same device.

Clearly, such an algorithm is effective only if the cryptographic key remains outside of the hands of attackers. AES is believed to be 'algorithmically secure' in the sense that it is infeasible to determine the cryptographic key by exploiting its intended inputs and outputs: plaintexts and ciphertexts (Mouha, 2021; Tao & Wu, 2015). Despite this, *physical implementations* of AES in hardware 'leak' information about their cryptographic keys. This phenomenon, called *side-channel leakage*, occurs because hardware inevitably emits measurable physical signals that are statistically associated with the data it processes and the instructions it executes (Mangard et al., 2007). There are many diverse physical side-channels which can leak, such as program/operation execution time (Kocher, 1996; Lipp et al., 2018; Kocher et al., 2019), temperature (Hutter & Schmidt, 2014), and sound due to vibration of electronic components (Genkin et al., 2014). In this work the side-channels we consider are *power and electromagnetic (EM) radiation over time* (Kocher et al., 1999; Mangard et al., 2007), which are major security vulnerabilities for AES implementations (Genkin et al., 2016; Bronchain & Standaert, 2020).

Side-channel attacks exploit this leakage to break cryptographic implementations by revealing secret internal variables such as cryptographic keys. In this work we consider *profiling* side-channel attacks (Chari et al., 2003; Mangard et al., 2007), which assume a worst-case scenario where the attacker possesses a clone of the target device and can repeatedly measure its power/radiation over time while encrypting arbitrary plaintexts using arbitrary keys. These sequences of power/radiation measurements are recorded as real vectors called *traces*, where each element encodes a measurement at a fixed point in time relative to the start of encryption. The attacker can thereby gather data from the clone device to model the conditional distribution of the secret variable given a trace. They can then defeat the target device by measuring its power/radiation traces, feeding them to the model, and revealing its secret internal variables.

Supervised deep learning has emerged as a state-of-the-art technique for this modeling task, achieving comparable or superior performance to prior approaches while requiring far less data preprocessing and feature selection (Maghrebi et al., 2016; Benadjila et al., 2020; Zaid et al., 2020; Wouters et al., 2020; Lu et al., 2021; Bursztein et al., 2023). Older non-deep learning attacks were mostly based on parametric statistics and had major limitations such as restrictive modeling assumptions (Messerges, 2000; Chari et al., 2003; Agrawal et al., 2005; Schindler et al., 2005; Hospodar et al., 2011) and inability to scale to long traces (Chari et al., 2003; Archambeau et al., 2006). Deep learning overcomes these limitations and consequently poses a major and growing threat to a wide assortment of security measures and evaluations that were designed with the limitations of older attacks in mind.

In this work, we seek to leverage deep learning to *defend* against side-channel attacks by identifying specific points in time at which power/radiation measurements leak sensitive data. Our aim is to aid the designers of implementations in understanding *why* their implementations leak (e.g. by indicating the particular sections of code or electronic components which are responsible) and thereby enable targeted mitigation of the leaks. Our key contributions are:

- We propose a principled information theoretic quantity which measures the 'leakiness' of individual power/EM radiation measurements. Unlike prior approaches, ours is sensitive to arbitrary statistical associations between a chosen secret variable and subsets of measurements. Our quantity is implicitly defined through a constrained optimization problem.
- We propose a novel deep learning algorithm called Adversarial Leakage Localization (ALL) which approximately solves this optimization problem. ALL is based on an adversarial game played between a neural net 'attacker' trained to classify secret data using power/radiation traces, and a budget-constrained noise distribution trained to 'defend' against the attack by introducing noise to individual measurements in the traces. Due to the budget constraint, noise cannot be added everywhere and must be rationed for the leakiest measurements. After training we can thereby surmise the leakiness of each measurement from its noisiness.
- We compare ALL with 11 baseline methods on 6 publicly-available power and EM radiation side-channel leakage datasets from implementations of the AES, ECC, and RSA cryptographic standards. To our knowledge this is by far the most comprehensive and quantitative comparison of leakage localization algorithms which has been done, and we release our code and procedure in the hope of facilitating reproducibility and benchmarking of future work in this area.

2 Background and Setting

Our setting is standard in the context of profiling power/EM side-channel analysis (Chari et al., 2003; Mangard et al., 2007) and is illustrated in Fig. 1. We assume to have a cryptographic device which encrypts data in a manner dependent on secret variable $y \in \mathsf{Y}$, where Y is a finite set (e.g. consisting of bitstrings encoding all possible values of the variable). We assume to have some measurement apparatus that allows us to measure the power consumption or EM radiation throughout encryption. We view the resulting measurement sequences as vectors $\boldsymbol{x} \in \mathbb{R}^T$, called *traces*, where $T \in \mathbb{Z}_{++}$ denotes the number of measurements per trace. We view the secret variable as a random variable $Y \sim p_Y$ where p_Y is a simple (e.g. uniform) distribution under our control. The resulting trace $\boldsymbol{X} \mid Y \sim p_{\boldsymbol{X}|Y}$ then comes from a complicated and *a priori* unknown distribution dictated by factors such as the hardware, environment, and measurement setup. Here each element X_t of the random vector $\boldsymbol{X} \equiv (X_1, \ldots, X_T)$ represents a power/radiation measurement at a fixed point in time relative to the start of encryption. In this work we assume that the distributions $p_{\boldsymbol{X},Y}(\cdot \mid y)$ exist and have full support for all $y \in \mathsf{Y}$, which is reasonable because empirically power consumption usually has a 'random' component which is well-described by additive Gaussian noise (Mangard et al., 2007). Most profiling side-channel attacks amount to collecting a dataset $\mathsf{D} \overset{\text{i.i.d.}}{\sim} p_{\boldsymbol{X},Y}$ and using supervised (deep or otherwise) learning to model $p_{Y|\boldsymbol{X}}$.

Given these jointly-distributed $\boldsymbol{X}$, Y, we seek to define for each X_t a scalar quantifying its 'leakiness,' i.e. the extent to which it can be exploited by attackers to learn Y from $\boldsymbol{X}$. Towards this end it is useful to consider the Shannon conditional mutual information (Shannon, 1948)

$$\mathbb{I}[Y; X_t \mid \mathsf{S}] \coloneqq \mathbb{E}\left[\log p_{Y|X_t,\mathsf{S}}(Y \mid X_t, \mathsf{S}) - \log p_{Y|\mathsf{S}}(Y \mid \mathsf{S})\right], \quad \mathsf{S} \subset \{X_1, \ldots, X_T\} \setminus \{X_t\}. \tag{1}$$

Intuitively, $\mathbb{I}[Y; X_t \mid \mathsf{S}]$ tells us the extent to which our uncertainty about Y is reduced upon observing X_t, provided we have already observed the elements of S.

Each individual quantity $\mathbb{I}[Y; X_t \mid \mathsf{S}]$ tells us something about the leakiness of X_t. However, for each t there are 2^{T-1} such quantities, and it is not obvious how they should be combined into a single scalar. Clearly, X_t should be considered leaky if $\mathbb{I}[Y; X_t] > 0$ and non-leaky if $\mathbb{I}[Y; X_t \mid \mathsf{S}] = 0 \ \forall \mathsf{S}$. More-subtly, there are many practical scenarios in which we have *second-order leakage*, where $\mathbb{I}[Y; X_t] = 0$ but $\mathbb{I}[Y; X_t \mid X_{t'}] > 0$ for

some $t' \neq t$ (i.e. X_t *alone* reveals nothing about Y, but reveals something useful about Y *when combined* with $X_{t'}$). For example, many cryptographic implementations use a Boolean masking countermeasure (Chari et al., 1999; Benadjila et al., 2020) whereby a sensitive variable is decomposed into a pair of independent 'random shares' which are operated on at distant points in time. Hence, the naïve choice of $\mathbb{I}[Y; X_t]$ as our definition of the leakiness of X_t would not work.

Another naive choice would be to define the leakiness of X_t as $\mathbb{I}[Y; X_t \mid \{X_1, \ldots, X_T\} \setminus \{X_t\}]$. This addresses the insensitivity of $\mathbb{I}[Y; X_t]$ to second-order leakage. However, it introduces a new shortcoming: when there are many leaky measurements with 'redundant' information, we may have $\mathbb{I}[Y; X_t \mid \{X_1, \ldots, X_T\} \setminus \{X_t\}] \approx 0$ even if $\mathbb{I}[Y; X_t] \gg 0$. In other words, X_t has little *new* information about Y which is not already provided by the other measurements. As we will show, this phenomenon creates issues for many prior deep learning-based leakage localization algorithms.

We will subsequently propose a natural notion of leakiness which is sensitive to all the quantities described by Eqn. 1. Before we do so, let us consider relevant prior work and its limitations.

3 Existing work and its limitations

We consider prior work in the side-channel analysis literature which may be leveraged for leakage localization. One prominent category of such work is *parametric statistics-based methods* which use non-deep learning techniques to look for pairwise associations between the measurements X_t and Y. The other is *neural net attribution-based methods*, where 1) a profiling side-channel attack is carried out with supervised deep learning, and 2) the neural net is 'interpreted' to determine the relative importance of its input features. Refer to Appendix C for further details.

3.1 First-order parametric statistics-based methods

First-order parametric statistics-based methods are widely used for understanding leakage, as well as for 'point of interest' (feature) selection for classical side-channel attacks which require low-dimensional inputs (Chari et al., 2003). We consider as baselines the signal to noise ratio (SNR) (Mangard et al., 2007), sum of squared differences (SOSD) (Chari et al., 2003), and correlation power analysis with Hamming weight leakage model (CPA) (Brier et al., 2004) due to their popularity and proven efficacy for point of interest selection (Fan et al., 2014). While these methods are invaluable due to their simplicity, transparency and low cost, they have major shortcomings. Notably, they cannot detect second-order leakage, and require restrictive assumptions about the relationship between $\boldsymbol{X}$ and Y. While prior work has proposed generalizations which are sensitive to leakage of order 2 and higher, these typically either have exponential runtime in the maximum considered order, require the existence of device flaws (e.g. biased random number generators), or require knowing internal variables or the points in time at which they leak (Messerges, 2000; Agrawal et al., 2005).

While our work concerns 'black box' leakage localization algorithms which make minimal assumptions about the cryptographic implementation being evaluated, first-order parametric methods are often employed as tools in 'white box' analyses of these implementations. For example, our work considers the ASCADv1 datasets (Benadjila et al., 2020), which have second-order leakage due to their Boolean masking countermeasure. Egger et al. (2022) carefully examine the underlying implementation and identify multiple *pairs* of internal AES variables which individually have first-order leakage and can be combined to reveal the canonically-targeted secret variable. They then leverage internal random number generator outputs to locate these variables with first-order parametric methods. We use these results to compute 'ground truth' leakiness measurements for our ASCADv1 experiments. We emphasize that this type of analysis is challenging and error-prone: some of the leaky variables identified by Egger et al. (2022) were unknown to Benadjila et al. (2020), who introduced the dataset. This underscores the value of black box techniques such as ours as a supplement to white box analysis.

3.2 Neural net attribution-based methods

There is a great deal of prior work on localizing leakage by applying interpretability techniques to neural nets which have been trained to perform side-channel attacks (Masure et al., 2019; Hettwer et al., 2020; Jin et al., 2020; Zaid et al., 2020; Wouters et al., 2020; van der Valk et al., 2021; Golder et al., 2022; Li et al., 2022; Perin et al., 2022; Schamberger et al., 2023; Yap et al., 2023; Li et al., 2024; Yap et al., 2025). Most of these techniques can be summarized as follows: 1) use standard supervised deep learning techniques to train a model $\hat{p} \approx p_{Y|\boldsymbol{X}}$ using data, and 2) use interpretability techniques to estimate the influence of each input feature on the model, on average over the dataset. For example, the gradient visualization technique of Masure et al. (2019) estimates the leakiness of X_t by $\mathbb{E}_{\boldsymbol{X},Y}\left|-\frac{\partial}{\partial x_t}\log\hat{p}(Y \mid \boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{X}}\right|$, and the 1-occlusion technique estimates it as $\mathbb{E}_{\boldsymbol{X},Y}\left|\hat{p}(Y \mid \boldsymbol{X}) - \hat{p}(Y \mid (\boldsymbol{1}-\boldsymbol{I}_t)\odot\boldsymbol{X})\right|$ where $\boldsymbol{I}_t$ denotes column t of the identity matrix. In this work we consider as baselines the recent OccPOI method (Yap et al., 2025) and m-occlusion and 2nd-order occlusion techniques (Schamberger et al., 2023), as well as gradient visualization (GradVis) (Masure et al., 2019), saliency (Simonyan et al., 2014; Hettwer et al., 2020), 1-occlusion (Zeiler & Fergus, 2014; Hettwer et al., 2020), layerwise relevance propagation (Bach et al., 2015; Hettwer et al., 2020), and input $*$ gradient (Shrikumar et al., 2017; Wouters et al., 2020). Note that these subsume the deep learning baselines considered by Yap et al. (2025); Schamberger et al. (2023).

Most of these methods are prone to detecting only some leaking measurements while ignoring others. GradVis, saliency, 1-occlusion, LRP and input $*$ grad compute the leakiness of X_t by occluding or differentiating the input x_t to $\hat{p}(Y \mid X_1,\ldots,X_{t-1},x_t,X_{t+1},\ldots,X_T)$ and observing its change in output. However, as discussed in Sec. 2, when many of the measurements carry 'redundant' information, one may have $\mathbb{I}[Y;X_t \mid \{X_1,\ldots,X_T\}\setminus\{X_t\}] \approx 0$ even if $\mathbb{I}[Y;X_t] \gg 0$. In this case a well-fit $\hat{p}$ becomes essentially constant with respect to x_t, causing these methods to spuriously estimate low leakiness for X_t.

While m-occlusion, 2nd-order occlusion and OccPOI occlude multiple inputs simultaneously and are thus less-susceptible to this issue, they have their own shortcomings. m-occlusion is like 1-occlusion except that it occludes m-diameter windows rather than single points, which has an undesirable smoothing effect and is helpful only when the 'redundant' measurements are temporally-local. Second-order occlusion entails occluding all pairs of measurements, which is expensive because it requires $\Theta(T^2)$ passes through the dataset with the neural net. Additionally, while it provides the interesting and unique ability to discern whether leakage is first-order or second-order, this is not relevant in our problem setting, and we find that according to the performance metrics we utilize in Sec. 5 it gives little improvement over first-order occlusion. Unlike the other considered methods, OccPOI does not assign a leakiness estimate to every measurement. Rather, it is a heuristic which aims to identify a non-unique minimal-cardinality set of measurements sufficient for $\hat{p}$ to attain some classification performance when all other measurements are occluded. Additionally, it is expensive because it requires $\Omega(T)$ non-parallelizable passes through the dataset with the neural net.

4 Our method: Adversarial Leakage Localization (ALL)

Here we propose a novel algorithm called Adversarial Leakage Localization (ALL) for localizing leakage which addresses the shortcomings of prior work. In line with Sec. 2, we have jointly-distributed power/EM radiation traces $\boldsymbol{X} \coloneqq (X_1,\ldots,X_T)$ and secret data Y. We seek to define for each X_t a scalar quantifying its 'leakiness,' i.e. its usefulness to attackers for learning Y from $\boldsymbol{X}$.

Intuitively, ALL is based on an adversarial game played between a neural net 'attacker' trained to predict Y from $\boldsymbol{X}$, and a budget-constrained noise distribution trained to 'defend' against the attack by introducing noise to individual measurements to maximize the loss of the classifier. Because of the budget constraint, increasing the noise applied to one measurement requires reducing the noise of other measurements. Thus, noise cannot simply be applied everywhere, and must be 'triaged' so that leakier measurements get more noise. After training we can surmise the leakiness of a measurement from the amount of noise which has been applied to it.

In this section we first propose a constrained optimization problem which implicitly defines a notion of 'leakiness' which is sensitive to all the terms $\mathbb{I}[Y;X_t \mid \mathsf{S}] : \mathsf{S} \subset \{X_1,\ldots,X_T\}\setminus\{X_t\}$. We then derive ALL as

a practical deep learning algorithm which approximately solves this optimization problem. We conclude by explicitly contrasting ALL with prior work. Refer to Appendix D for an extended version of this section with proofs and derivations, and Algorithm 2 for pseudocode.

4.1 Implicit definition of leakiness through a constrained optimization problem

We define a vector $\boldsymbol{\gamma} \in [0,1)^T$ which we name the *occlusion probabilities*. $\boldsymbol{\gamma}$ parameterizes a distribution over a binary random vector with range $\{0,1\}^T$ which we call the *occlusion mask*: $\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}} \equiv (\mathcal{A}_{\boldsymbol{\gamma},1}, \ldots, \mathcal{A}_{\boldsymbol{\gamma},T}) \sim p_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}}$: $\mathcal{A}_{\boldsymbol{\gamma},t} \sim \text{Bernoulli}(1-\gamma_t)$. For arbitrary vectors $\boldsymbol{x} \in \mathbb{R}^T$, $\boldsymbol{\alpha} \in \{0,1\}^T$, let us denote $\boldsymbol{x}_{\boldsymbol{\alpha}} \coloneqq (x_t : t = 1, \ldots, T : \alpha_t = 1)$, i.e. the sub-vector of $\boldsymbol{x}$ containing its elements for which the corresponding element of $\boldsymbol{\alpha}$ is 1. We can accordingly use $\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}$ to obtain random sub-vectors $\boldsymbol{X}_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}}$ of $\boldsymbol{X}$. Note that γ_t denotes the probability that X_t will *not* be an element of $\boldsymbol{X}_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}}$.

We assign to each element of $\boldsymbol{\gamma}$ a strictly-increasing 'cost' $c : [0,1) \to \mathbb{R}_+ : x \mapsto \frac{x}{1-x}$, with the properties $c(0) = 0$ and $\lim_{x \to 1} c(x) = \infty$. We seek to solve the constrained optimization problem

$$\min_{\boldsymbol{\gamma} \in [0,1)^T} \quad \mathcal{L}(\boldsymbol{\gamma}) \coloneqq \mathbb{I}[Y; \boldsymbol{X}_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}} \mid \boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}] \quad \text{such that} \quad \sum_{t=1}^{T} c(\gamma_t) = C \tag{2}$$

where $C \in \mathbb{R}_{++}$ is a 'budget' hyperparameter. Intuitively, the mutual information term tells us the extent to which the 'occluded' trace $\boldsymbol{X}_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}}$ 'leaks' Y, and $\boldsymbol{\gamma}$ is optimized to distribute a fixed 'budget of occlusion probability' among the individual elements of $\boldsymbol{X}$ to minimize this leakage.

As discussed in Appendix D.1, during this optimization process each γ_t is 'pushed' up towards 1 in proportion to a weighted sum over *all* values $\mathbb{I}[Y; X_t \mid \mathsf{S}] : \mathsf{S} \subset \{X_1, \ldots, X_T\} \setminus X_t$. Thus, ALL is sensitive to all associations between Y and subsets of $\{X_1, \ldots, X_T\}$. This is in contrast to parametric methods which consider only pairwise associations between each X_t and Y, methods like 1-occlusion, GradVis, Saliency, Input $*$ Grad and LRP which are sensitive to associations between Y and the sets $\{X_1, \ldots, X_T\}, \{X_1, \ldots, X_T\} \setminus \{X_t\}$, and OccPOI, m-occlusion, and 2nd-order occlusion, which consider larger yet still tiny subsets of the power set of $\{X_1, \ldots, X_T\}$.

Due to the budget constraint, increasing γ_t requires reducing other γ_τ, $\tau \neq t$. Let us denote by $\boldsymbol{\gamma}^*$ a solution to Eqn. 2. Each γ_t^* will be closer to 1 if X_t is 'leakier' in the sense that it has greater mutual information with Y, conditioned on other X_τ, $\tau \neq t$. Thus, we propose using γ_t^* to measure the 'leakiness' of X_t.

4.2 Deep learning-based implementation

We will now re-frame this problem in a way that is amenable to standard deep learning techniques. Refer to Fig. 2 for a diagram.

We first re-parameterize it into an unconstrained problem by defining the variable $\boldsymbol{\eta} \coloneqq \text{softmax}(\tilde{\boldsymbol{\eta}})$, $\tilde{\boldsymbol{\eta}} \in \mathbb{R}^T$ and letting $\boldsymbol{\gamma}$ be a function from $\mathbb{R}^T \to [0,1]^T$ satisfying

$$c(\gamma_t(\tilde{\boldsymbol{\eta}})) = C\eta_t \iff \gamma_t(\tilde{\boldsymbol{\eta}}) = \text{sigmoid}(\log C + \log(\text{softmax}(\tilde{\boldsymbol{\eta}})_t)). \tag{3}$$

We can now optimize with respect to $\tilde{\boldsymbol{\eta}}$ instead of $\boldsymbol{\gamma}$, letting us drop the constraint because it is satisfied for any $\tilde{\boldsymbol{\eta}}$. Note that it is cheap and numerically-stable to map $\tilde{\boldsymbol{\eta}}$ to $\boldsymbol{\gamma}$ in PyTorch.

Next, as described in Appendix D.2, we can approximate the mutual information term of Eqn. 2 with a neural net. Note that $\mathbb{I}[Y; \boldsymbol{X}_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}} \mid \boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}] = \sum_{\boldsymbol{\alpha} \in \{0,1\}^T} p_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}}(\boldsymbol{\alpha}) \, \mathbb{I}[Y; \boldsymbol{X}_{\boldsymbol{\alpha}}]$ where each $\mathbb{I}[Y; \boldsymbol{X}_{\boldsymbol{\alpha}}] = \mathbb{E} \log p_{Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}}(Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}) - \mathbb{E} \log p_Y(Y)$. The right terms can be dropped because their corresponding terms in the full expression do not depend on $\boldsymbol{\gamma}$. The conditional distributions in the left terms can each be approximated by using supervised deep learning with cross-entropy loss to classify Y from $\boldsymbol{X}_{\boldsymbol{\alpha}}$. There are 2^T such distributions and it would be infeasible to train this many neural nets independently. Instead, similarly to Lippe et al. (2022), we train a single neural net to estimate all the distributions by occluding its inputs according to $\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}$ and feeding $\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}$ as an auxiliary input. Thus, we can approximate Eqn. 2 with the optimization problem

$$\min_{\tilde{\boldsymbol{\eta}} \in \mathbb{R}^T} \max_{\boldsymbol{\theta} \in \mathbb{R}^P} \quad \mathcal{L}_{\text{adv}}(\tilde{\boldsymbol{\eta}}, \boldsymbol{\theta}) \coloneqq \mathbb{E} \log \Phi_{\boldsymbol{\theta}} \left(Y \mid \boldsymbol{X} \odot \boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}(\tilde{\boldsymbol{\eta}})}, \boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}(\tilde{\boldsymbol{\eta}})} \right) \tag{4}$$

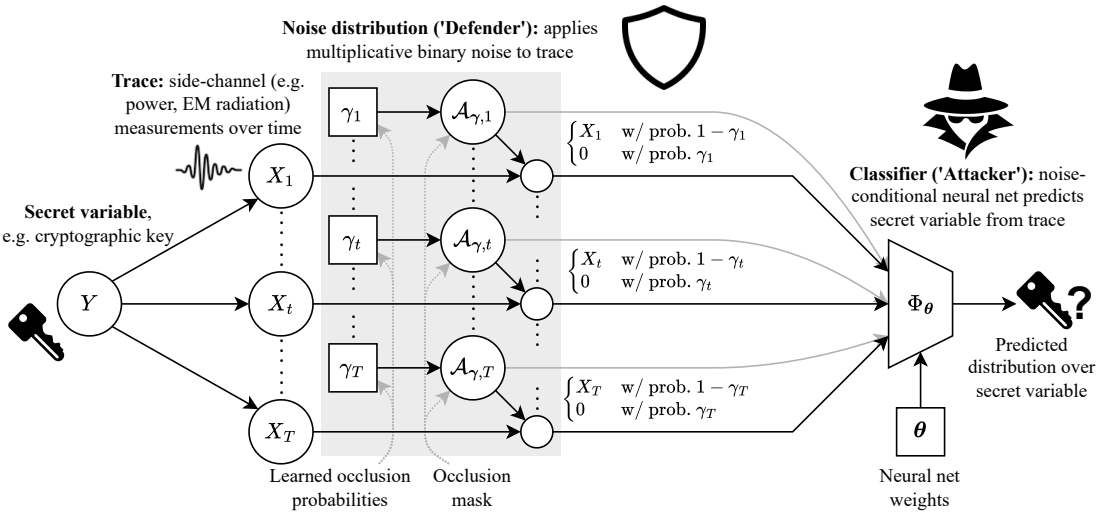

Figure 2: A diagram illustrating our Adversarial Leakage Localization (ALL) algorithm. A classifier $\Phi_{\boldsymbol{\theta}}$ is trained to 'attack' a cryptographic implementation by predicting its secret data Y from power/EM radiation traces $\boldsymbol{X} \equiv (X_1, \ldots, X_T)$. Simultaneously, a noise distribution is trained to 'defend against the attack' by occluding the classifier's individual input features X_t with probabilities γ_t, subject to a budget constraint which prevents trivially occluding every feature with probability 1. Because of the constraint, increasing γ_t necessarily entails decreasing γ_τ for some $\tau \neq t$, so the noise distribution must preferentially apply noise to leakier features. Thus, after training we may interpret γ_t as the 'leakiness' of X_t.

where $\Phi_{\boldsymbol{\theta}} : \mathsf{Y} \times \mathbb{R}^T \times \{0,1\}^T \to (0,1)$ is a neural net with weights $\boldsymbol{\theta}$ and softmax output activation, and $\Phi_{\boldsymbol{\theta}}(y \mid \boldsymbol{x} \odot \boldsymbol{\alpha}, \boldsymbol{\alpha})$ denotes its estimated probability that $Y = y$ given $\boldsymbol{X}_{\boldsymbol{\alpha}} = \boldsymbol{x}_{\boldsymbol{\alpha}}$. This can be approximately solved using alternating SGD-style algorithms, similarly to GANs (Goodfellow et al., 2014).

To use SGD-like algorithms we must estimate $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{adv}}(\tilde{\boldsymbol{\eta}}, \boldsymbol{\theta})$ and $\nabla_{\tilde{\boldsymbol{\eta}}} \mathcal{L}_{\text{adv}}(\tilde{\boldsymbol{\eta}}, \boldsymbol{\theta})$ with Monte Carlo integration. The former is routine in the context of DNN training. However, the latter is nontrivial because $\mathcal{L}_{\text{adv}}$ has the form $\mathbb{E}_{\boldsymbol{\alpha} \sim p_{\tilde{\boldsymbol{\eta}}}} f(\boldsymbol{\alpha})$ where $\boldsymbol{\alpha}$ is discrete. There is a large body of work on gradient estimation for functions of this nature, which can broadly be categorized into unbiased REINFORCE (Williams, 1992)-based estimators with variance reduction strategies, and biased estimators based on relaxing $p_{\tilde{\boldsymbol{\eta}}}$ into a continuous distribution for which we can use the reparameterization trick (Rezende et al., 2014; Kingma & Welling, 2014). In our experiments we use the biased CONCRETE estimator (Maddison et al., 2017) with fixed temperature $\tau = 1$ because it is cheap and simple, and we find it yields nearly the same performance as more-complicated estimators we tried. We conjecture that we get strong performance with this simple and biased estimator because our performance metrics are sensitive only to the *relative* leakiness assigned to measurements, and the bias does not significantly affect this. Note that $f(\boldsymbol{\alpha}) \equiv \mathbb{E} \log \Phi_{\boldsymbol{\theta}}(Y \mid \boldsymbol{X} \odot \boldsymbol{\alpha}, \boldsymbol{\alpha})$ has been defined for $\boldsymbol{\alpha} \in \{0,1\}^T$, but after the CONCRETE relaxation we may have any $\boldsymbol{\alpha} \in [0,1]^T$. Thus, we replace it with the modified function $f(\boldsymbol{\alpha}) \equiv \mathbb{E} \log \Phi_{\boldsymbol{\theta}}(Y \mid \boldsymbol{X} \odot \boldsymbol{\alpha} + \boldsymbol{\varepsilon} \odot (\boldsymbol{1} - \boldsymbol{\alpha}), \boldsymbol{\alpha})$ where $\boldsymbol{\varepsilon} \sim \mathcal{N}(0,1)^T$.

Our method is mainly sensitive to 3 hyperparameters: the learning rates of $\boldsymbol{\theta}$ and $\tilde{\boldsymbol{\eta}}$, and the budget hyperparameter C. Rather than tuning C directly, we find it easier to tune the hyperparameter $\overline{\gamma} := \frac{C}{C+T}$. $\overline{\gamma}$ is equal to the occlusion probability of each measurement when $\tilde{\boldsymbol{\eta}}$ is constant, and is less-sensitive to the data dimensionality T than C.

4.3 Differences from prior work

In contrast to the parametric first-order methods, ALL (like other deep learning methods) does not make restrictive assumptions about the nature of associations between $\boldsymbol{X}$ and Y. Notably, while our method can detect the second-order leakage from Boolean masked implementations such as the subsequently-considered ASCADv1 datasets (Benadjila et al., 2020), the first-order parametric methods cannot.

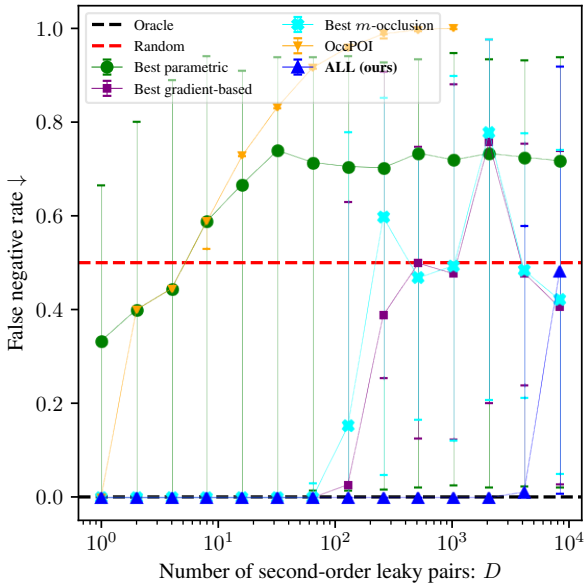

Figure 3: A toy setting described in Sec. 5.1 where ALL (ours) significantly outperforms baselines. We sample 1 non-leaky feature and D second-order leaky pairs, then plot the false negative rate, defined as the proportion of points incorrectly assigned leakiness less than or equal to that of the non-leaky point, as we increase D. ALL (ours) succeeds for D up to 64× higher the best prior deep learning-based approach, and the first-order parametric methods completely fail in this setting. Dots denote median and error bars denote min–max over 5 random seeds.

Whereas GradVis, Saliency, LRP, Input $*$ grad, LRP and 1-occlusion effectively perturb single input features to the classifier and analyze the change in its outputs, ALL generally perturbs many inputs simultaneously. This is useful in settings where there are many 'redundant' leaking measurements and the impact of perturbing only one of them is small.

Like ALL, m-occlusion, 2nd-order occlusion, and OccPOI also simultaneously occlude multiple input features. The key differences of our method are: 1) ALL samples from a distribution over all 2^T possible occlusion masks optimized to maximally hurt the performance of the classifier, whereas prior work iterates over a tiny subset of possible occlusion masks chosen in an *ad hoc* manner. 2) ALL leverages the gradient of classifier loss with respect to relaxed occlusion masks, whereas prior work uses only zeroeth-order information from forward passes with 'hard' occlusion masks. 3) We simultaneously optimize the mask distribution to maximally hurt the classifier, and the classifier weights to be optimal for the current mask distribution. In contrast, prior work trains the classifier with standard supervised learning techniques, then 'interprets' the fixed classifier with occlusion.

5 Experimental results

5.1 Synthetic datasets where we know 'ground truth' leakiness

Toy setting where ALL succeeds and prior work fails As we will show, these differences lead to significant performance gains, as well as speedup relative to 2nd-order occlusion and OccPOI.

We generate a sequence of binary-label $2D+1$-feature classification datasets consisting of ordered pairs $(\boldsymbol{X}, Y)$ sampled independently as follows (see Appendix E.1 for details): $Y \sim \mathcal{U}\{0,1\}$, $R \sim \mathcal{U}\{0,1\}$, $M_i \sim \mathcal{U}\{0,1\}$ for $i = 1, \ldots, D$, $X_R \sim \mathcal{N}(R, 1)$, $X_{M_i} \sim \mathcal{N}(M_i, 1)$, $X_{Y \oplus M_i} \sim \mathcal{N}(Y \oplus M_i, 1)$ for $i = 1, \ldots, D$. Here we denote by $\oplus$ the exclusive-or operation and $\boldsymbol{X} := (X_R, X_{M_1}, X_{Y \oplus M_i}, \ldots, X_{M_D}, X_{M_D \oplus Y})$. Intuitively, we can view the features X_R, X_{M_i}, and $X_{Y \oplus M_i}$ as noisy observations of R, M_i, and $Y \oplus M_i$, respectively. Note that R

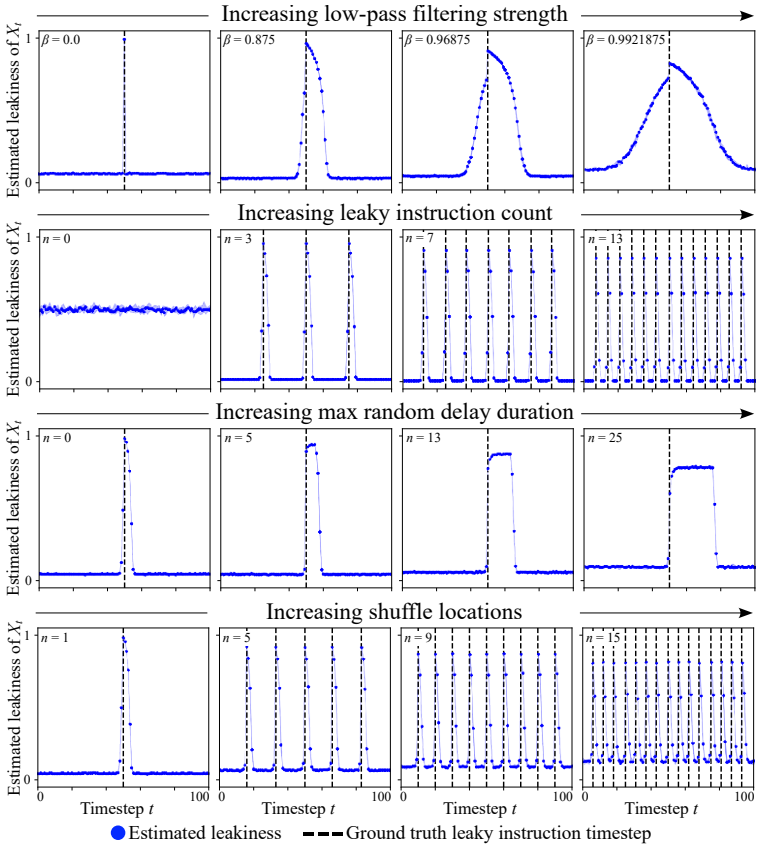

Figure 4: Output $\boldsymbol{\gamma}^* \equiv \boldsymbol{\gamma}(\tilde{\boldsymbol{\eta}}^*)$ of our ALL algorithm when applied to simulated AES-128 power trace datasets based on the Hamming weight model of (Mangard et al., 2007, ch. 4), as described in Appendix E.2. Leakiness estimated by ALL is consistent with the ground truth timestep at which leaky instruction(s) are executed across varying low-pass filtration strength, leaky instruction count, random no-op insertion, and random shuffling.

tells us nothing about Y, and while *individually* each variable M_i, $Y \oplus M_i$ is independent of Y, the *pairs* of variables $\{M_i, Y \oplus M_i\}$ allow us to determine Y through the identity $Y = Y \oplus M_i \oplus M_i$. An ideal leakage localization algorithm should indicate that each feature X_{M_i}, $X_{Y \oplus M_i}$ is leaking, while X_R is not.

In Fig. 3, we plot the performance of ALL and prior work as we sweep the value of D. First-order parametric methods *completely fail* in this setting because they are not sensitive to the second-order associations between Y and $\{X_{M_i}, X_{Y \oplus M_i}\}$ and thus indicate that all features are equally-leaky. While prior deep learning-based methods succeed for small D, they fail when D grows large because the individual contribution of each $\{X_{M_i}, X_{Y \oplus M_i}\}$ is 'drowned out' in the sense that $\mathbb{I}[Y; \{X_{M_i}, X_{Y \oplus M_i}\} \mid \{X_{M_j}, X_{Y \oplus M_j} : j \neq i\}]$ vanishes. ALL succeeds for D up to 64× larger than prior work because as the classifier becomes more-reliant on particular features they are subject to a higher occlusion probability, mitigating this effect.

Simulated AES-128 datasets As shown in Fig. 4, we apply ALL to a variety of simulated AES datasets based on the Hamming weight leakage model of (Mangard et al., 2007, ch. 4). We confirm that the outputs of ALL are consistent with ground truth 'leakiness' information while sweeping various properties of the data-generating process. See Appendix E.2 for details.

5.2 Real power and EM radiation leakage datasets

We compare our method to prior work on 6 publicly-available datasets of real recorded side-channel emissions and metadata covering diverse settings, as described in Appendix E.3.1. For fairness, all deep learning

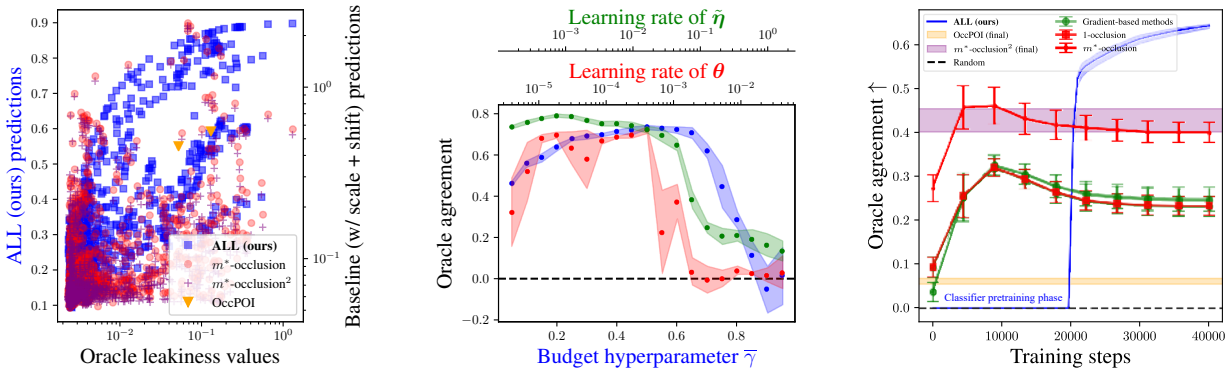

Figure 5: Selected results on ASCADv1 (variable), which is the largest and most-challenging dataset we consider. **(left)** A plot showing 'oracle' per-timestep leakiness measurements on the horizontal axis and the measurements produced by ALL (ours) and selected baselines on the vertical axis. ALL is superior to baselines because it is closer to a strictly-increasing function of the oracle leakiness values. **(center)** Performance as we individually perturb the main hyperparameters of ALL away from their best-performing configuration. Desirably, performance varies smoothly and significantly exceeds random guessing across most of the search spaces. **(right)** Performance over time during training. Performance varies smoothly for all methods, with ALL significantly beating baselines. The jump in ALL's performance is because we train only the classifier for the first 20k steps.

methods use the same neural net architecture, hyperparameter search budget, and training step count. See appendix E.3 for full results, details and ablation studies.

Evaluating performance with 'oracle agreement' For real-world datasets we lack 'ground truth' knowledge about the leakiness of each measurement and must devise a reasonable proxy. In line with previous work (Masure et al., 2019; Wouters et al., 2020; Schamberger et al., 2023; Yap et al., 2025), we evaluate performance by assessing similarity with a 'white box' leakage assessment leveraging implementation knowledge and access to the internal random variables of cryptographic implementations. As discussed in Sec. 3.1, for the ASCADv1 datasets our white box assessment consists of the average SNR for the second-order leaky internal AES variables identified by Egger et al. (2022), and for the first-order datasets we simply use the SNR of the targeted variable. These SNR values, which we refer to as *oracle leakiness values* are computed using the test dataset. We define *oracle agreement* as the Spearman rank correlation (Spearman, 1904) between the oracle leakiness values and those predicted by an algorithm.

There is no consensus about how leakage localization algorithms should be evaluated. Comparison with oracle values similar to ours is typical, though to our knowledge ours is the first work to summarize oracle agreement with a scalar and use it to quantitatively compare many methods. In appendix E.3 we consider alternate evaluation strategies which have been used in Masure et al. (2019); Hettwer et al. (2020).

Results As seen in Table 1, ALL beats prior deep learning algorithms on 5 of the 6 considered datasets, with a particular improvement on the second-order datasets which leak redundantly from 4 distinct pairs of internal AES variables. In Fig. 5 we plot predicted vs. oracle leakiness measurements as well as oracle agreement over time during training for ALL and chosen baselines, and see that ALL offers a clear improvement. We also evaluate sensitivity to the main hyperparameters of ALL and find that performance varies smoothly and significantly outperforms random noise over a wide range of values, demonstrate that it is not excessively difficult to tune.

Table 1: ALL (ours) outperforms all deep learning baselines across 5 of our 6 considered datasets in terms of oracle agreement, i.e. Spearman rank correlation of predictions with 'oracle' leakiness values computed in a white box manner. The $\boxed{\text{boxed}}$ method is best and the <u>underlined</u> methods have overlapping error bars with the best method. We report mean ± std. dev. over 5 random seeds.

	2nd-order datasets		**1st-order datasets**			
Method	ASCADv1 (fixed)	ASCADv1 (random)	DPAv4 (Zaid vsn.)	AES-HD	OTiAiT	OTP (1024-bit)
Random	-0.01 ± 0.03	-0.00 ± 0.03	0.00 ± 0.01	-0.01 ± 0.03	0.01 ± 0.02	0.01 ± 0.02
GradVis (Masure et al., 2019)	0.49 ± 0.02	0.27 ± 0.02	0.20 ± 0.01	0.07 ± 0.01	0.5 ± 0.2	0.59 ± 0.02
Saliency (Simonyan et al., 2014; Hettwer et al., 2020)	0.48 ± 0.02	0.27 ± 0.02	0.20 ± 0.01	0.07 ± 0.01	0.67 ± 0.06	0.59 ± 0.03
Input $*$ Grad (Shrikumar et al., 2017; Wouters et al., 2020)	0.49 ± 0.02	0.25 ± 0.02	0.20 ± 0.01	0.08 ± 0.01	0.71 ± 0.05	0.61 ± 0.03
LRP (Bach et al., 2015; Hettwer et al., 2020)	0.49 ± 0.02	0.25 ± 0.02	0.20 ± 0.01	0.08 ± 0.01	0.71 ± 0.05	0.61 ± 0.03
OccPOI (Yap et al., 2025)	0.05 ± 0.02	0.061 ± 0.006	0.03 ± 0.02	0.0484 ± 0.0005	0.093 ± 0.007	0.02 ± 0.04
1-Occlusion (Zeiler & Fergus, 2014; Hettwer et al., 2020)	0.49 ± 0.02	0.25 ± 0.02	0.20 ± 0.01	0.08 ± 0.01	0.71 ± 0.05	0.61 ± 0.03
m^*-Occlusion (Schamberger et al., 2023)	0.50 ± 0.02	0.41 ± 0.02	<u>0.32 ± 0.01</u>	<u>0.18 ± 0.05</u>	<u>0.73 ± 0.05</u>	0.77 ± 0.02
1-Occlusion^2 (Schamberger et al., 2023)	0.52 ± 0.02	0.27 ± 0.02	0.20 ± 0.01	0.07 ± 0.01	<u>0.74 ± 0.05</u>	0.60 ± 0.03
m^*-Occlusion^2 (Schamberger et al., 2023)	0.53 ± 0.01	0.43 ± 0.03	$\boxed{0.326 \pm 0.009}$	<u>0.19 ± 0.04</u>	0.73 ± 0.03	0.79 ± 0.01
ALL (ours)	$\boxed{0.79 \pm 0.01}$	$\boxed{0.643 \pm 0.005}$	<u>0.317 ± 0.002</u>	$\boxed{0.21 \pm 0.04}$	$\boxed{0.781 \pm 0.002}$	$\boxed{0.849 \pm 0.008}$

6 Conclusion

We have proposed a novel algorithm for localizing side-channel leakage from cryptographic implementations. Unlike prior work, ours is sensitive to arbitrary statistical associations responsible for leakage and operates in a 'black box' manner, requiring only a supervised learning-style dataset with labels denoting the secret variable under consideration. In light of the ever-increasing efficacy of deep side-channel attack algorithms and the failure of existing work to detect all leakage they may exploit, our work marks a critical step towards understanding and mitigating the emerging vulnerabilities of cryptographic hardware.

Broader Impact Statement

The goal of our work is to enhance the security of cryptographic implementations against side-channel attacks by identifying the points in time at which they reveal sensitive information, thereby facilitating targeted defenses and mitigation strategies. We foresee it as being a useful complement to widely-adopted parametric leakage localization techniques, which can identify leakage in a 'black box' manner without being limited by the domain knowledge of its users.

We believe there are limited avenues to directly exploit this work for malicious purposes. We have not introduced strategies which increase the capabilities of attackers, and we solely consider cryptographic datasets which have been made available for research purposes and are already widely studied and understood. Nonetheless, any research that contributes to understanding of side-channel attacks has the potential to be utilized for future research with malicious goals.

References

Dakshi Agrawal, Josyula R Rao, Pankaj Rohatgi, and Kai Schramm. Templates as master keys. In *Cryptographic Hardware and Embedded Systems–CHES 2005: 7th International Workshop, Edinburgh, UK, August 29–September 1, 2005. Proceedings 7*, pp. 15–29. Springer, 2005.

Cédric Archambeau, Eric Peeters, F X Standaert, and J J Quisquater. Template attacks in principal subspaces. In *Cryptographic Hardware and Embedded Systems-CHES 2006: 8th International Workshop, Yokohama, Japan, October 10-13, 2006. Proceedings 8*, pp. 1–14. Springer, 2006.

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *Journal of Cryptographic Engineering*, 10(2): 163–188, 2020.

Shivam Bhasin, Nicolas Bruneau, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Analysis and improvements of the dpa contest v4 implementation. In *Security, Privacy, and Applied Cryptography Engineering: 4th International Conference, SPACE 2014, Pune, India, October 18-22, 2014. Proceedings 4*, pp. 201–218. Springer, 2014.

Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. AES HD dataset - 50 000 traces. AISyLab repository, 2020. `https://github.com/AISyLab/AES_HD`.

Bluefin Payment Systems. Bluefin and ID TECH partner to deliver PCI validated Advanced Encryption Standard (AES) P2PE solution. Online, November 2023. URL `https://www.bluefin.com/news/bluefin-and-id-tech-partner-to-deliver-pci-validated-advanced-encryption-standard-aes-p2pe-solution/`.

Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater (eds.), *Cryptographic Hardware and Embedded Systems - CHES 2004*, pp. 16–29, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-28632-5.

Olivier Bronchain and François-Xavier Standaert. Side-channel countermeasures' dissection and the limits of closed source security evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1–25, 2020.

Elie Bursztein, Luca Invernizzi, Karel Král, Daniel Moghimi, Jean-Michel Picod, and Marina Zhang. Generic attacks against cryptographic hardware through long-range deep learning. *arXiv preprint arXiv:2306.07249*, 2023.

Suresh Chari, Charanjit S Jutla, Josyula R Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pp. 398–412. Springer, 1999.

Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, çetin K. Koç, and Christof Paar (eds.), *Cryptographic Hardware and Embedded Systems - CHES 2002*, pp. 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-36400-9.

Committee on National Security Systems. Committee on National Security Systems Policy No. 15, Fact Sheet No. 1, June 2003. URL `https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/cnss15fs.pdf`.

Jean-Sébastien Coron and Ilya Kizhvatov. An efficient method for random delay generation in embedded software. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 156–170. Springer, 2009. Dataset available at `https://github.com/ikizhvatov/randomdelays-traces`.

Joan Daemen and Vincent Rijmen. AES proposal: Rijndael document version 2. AES Algorithm Submission, September 1999. URL https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf.

Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer Berlin, Heidelberg, March 2013. URL https://link.springer.com/book/10.1007/978-3-662-04722-4.

Josef Danial, Debayan Das, Anupam Golder, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. Em-x-dl: Efficient cross-device deep learning side-channel attack with noisy em signatures. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 18(1):1–17, 2021.

Debayan Das, Anupam Golder, Josef Danial, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. X-deepsca: Cross-device deep learning side channel attack. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

Maximilian Egger, Thomas Schamberger, Lars Tebelmann, Florian Lippert, and Georg Sigl. A second look at the ascad databases. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pp. 75–99. Springer, 2022.

Guangjun Fan, Yongbin Zhou, Hailong Zhang, and Dengguo Feng. How to choose interesting points for template attacks? Cryptology ePrint Archive, Paper 2014/332, 2014. URL https://eprint.iacr.org/2014/332.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11): 665–673, 2020.

Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In Juan A. Garay and Rosario Gennaro (eds.), *Advances in Cryptology – CRYPTO 2014*, pp. 444–461, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-44371-2.

Daniel Genkin, Lev Pachmanov, Itamar Pipman, Eran Tromer, and Yuval Yarom. ECDSA key extraction from mobile devices via nonintrusive physical side channels. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1626–1638, 2016.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Anupam Golder, Ashwin Bhat, and Arijit Raychowdhury. Exploration into the explainability of neural network models for power side-channel analysis. In *Proceedings of the Great Lakes Symposium on VLSI 2022*, GLSVLSI '22, pp. 59–64, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393225. doi: 10.1145/3526241.3530346. URL https://doi.org/10.1145/3526241.3530346.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Ishaan Gulrajani and David Lopez-Paz. In search of lost domain generalization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=lQdXeXDoWtI.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825): 357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Katherine Hermann and Andrew Lampinen. What shapes feature representations? exploring datasets, architectures, and training. *Advances in Neural Information Processing Systems*, 33:9995–10006, 2020.

Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Deep neural network attribution methods for leakage analysis and symmetric key recovery. In Kenneth G. Paterson and Douglas Stebila (eds.), *Selected Areas in Cryptography – SAC 2019*, pp. 645–666, Cham, 2020. Springer International Publishing. ISBN 978-3-030-38471-5.

Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293–302, 2011.

Michael Hutter and Jörn-Marc Schmidt. The temperature side channel and heating fault attacks. In *Smart Card Research and Advanced Applications: 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers 12*, pp. 219–235. Springer, 2014.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparametrization with gumble-softmax. In *International Conference on Learning Representations (ICLR 2017)*. OpenReview. net, 2017.

Minhui Jin, Mengce Zheng, Honggang Hu, and Nenghai Yu. An enhanced convolutional neural network in side-channel attacks and its visualization. *arXiv preprint arXiv:2009.08898*, 2020.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun (eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL `http://arxiv.org/abs/1312.6114`.

Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener (ed.), *Advances in Cryptology — CRYPTO' 99*, pp. 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-48405-9.

Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz (ed.), *Advances in Cryptology — CRYPTO '96*, pp. 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-68697-2.

Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, et al. Captum: A unified and generic model interpretability library for pytorch. *arXiv preprint arXiv:2009.07896*, 2020.

Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 1–6, 2015.

Yanbin Li, Yuxin Huang, Fuwei Jia, Qingsong Zhao, Ming Tang, and Shougang Ren. A gradient deconvolutional network for side-channel attacks. *Computers & Electrical Engineering*, 98:107686, 2022.

Yanbin Li, Jiajie Zhu, Zhe Liu, Ming Tang, and Shougang Ren. Deep learning gradient visualization-based pre-silicon side-channel leakage location. *IEEE Transactions on Information Forensics and Security*, 2024.

Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

Phillip Lippe, Taco Cohen, and Efstratios Gavves. Efficient neural causal discovery without acyclicity constraints. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=eYciPrLuUhG`.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 235–274, 2021.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=S1jE5L5gl.

Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *Security, Privacy, and Applied Cryptography Engineering: 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings 6*, pp. 3–26. Springer, 2016.

Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks*. Springer New York, NY, 1st edition, March 2007. doi: 10.1007/978-0-387-38162-6. URL https://link.springer.com/book/10.1007/978-0-387-38162-6.

Loïc Masure and Rémi Strullu. Side-channel analysis against ANSSI's protected AES implementation on ARM: end-to-end attacks with multi-task learning. *Journal of Cryptographic Engineering*, 13(2):129–147, 2023.

Loïc Masure, Cécile Dumas, and Emmanuel Prouff. Gradient visualization for general characterization in profiling attacks. In *Constructive Side-Channel Analysis and Secure Design: 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3–5, 2019, Proceedings 10*, pp. 145–167. Springer, 2019.

Thomas S Messerges. Using second-order power analysis to attack dpa resistant software. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 238–251. Springer, 2000.

Nicky Mouha. Review of the Advanced Encryption Standard. NIST Interagency/Internal Report (NISTIR) 8319, National Institute of Standards and Technology, July 2021. URL https://csrc.nist.gov/pubs/ir/8319/final.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Guilherme Perin, Lichao Wu, and Stjepan Picek. I know what your layers did: Layer-wise explainability of deep learning side-channel analysis. Cryptology ePrint Archive, Paper 2022/1087, 2022. URL https://eprint.iacr.org/2022/1087.

Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. SoK: Deep learning-based physical side-channel analysis. *ACM Computing Surveys*, 55(11):1–35, 2023.

Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA): Measures and countermeasures for smart cards. In Isabelle Attali and Thomas Jensen (eds.), *Smart Card Programming and Security*, pp. 200–210, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-45418-2.

Christian Rechberger and Elisabeth Oswald. Practical template attacks. In Chae Hoon Lim and Moti Yung (eds.), *Information Security Applications*, pp. 440–456, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31815-6.

Eric Rescorla. HTTP over TLS. RFC 2818, May 2000. URL https://www.rfc-editor.org/info/rfc2818.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In Eric P. Xing and Tony Jebara (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pp. 1278–1286, Bejing, China, 22–24 Jun 2014. PMLR. URL https://proceedings.mlr.press/v32/rezende14.html.

Kotaro Saito, Akira Ito, Rei Ueno, and Naofumi Homma. One truth prevails: A deep-learning based single-trace power analysis on rsa–crt with windowed exponentiation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 490–526, 2022.

Thomas Schamberger, Maximilian Egger, and Lars Tebelmann. Hide and seek: Using occlusion techniques for side-channel leakage attribution in cnns. In Jianying Zhou, Lejla Batina, Zengpeng Li, Jingqiang Lin, Eleonora Losiouk, Suryadipta Majumdar, Daisuke Mashima, Weizhi Meng, Stjepan Picek, Mohammad Ashiqur Rahman, Jun Shao, Masaki Shimaoka, Ezekiel Soremekun, Chunhua Su, Je Sen Teh, Aleksei Udovenko, Cong Wang, Leo Zhang, and Yury Zhauniarovich (eds.), *Applied Cryptography and Network Security Workshops*, pp. 139–158, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-41181-6.

Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *Cryptographic Hardware and Embedded Systems–CHES 2005: 7th International Workshop, Edinburgh, UK, August 29–September 1, 2005. Proceedings 7*, pp. 30–46. Springer, 2005.

Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27 (3):379–423, 1948.

Alexander Shekhovtsov. Bias-variance tradeoffs in single-sample binary gradient estimators. In *DAGM German Conference on Pattern Recognition*, pp. 127–141. Springer, 2021.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3145–3153. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/shrikumar17a.html`.

K Simonyan, A Vedaldi, and A Zisserman. Deep inside convolutional networks: visualising image classification models and saliency maps. In *Proceedings of the International Conference on Learning Representations (ICLR)*. ICLR, 2014.

C Spearman. The proof and measurement of association between two things. *American Journal of Psychology*, 15:72–101, 1904.

Biaoshuai Tao and Hongjun Wu. Improving the biclique cryptanalysis of AES. In Ernest Foo and Douglas Stebila (eds.), *Information Security and Privacy*, pp. 39–56, Cham, 2015. Springer International Publishing. ISBN 978-3-319-19962-7.

George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. *Advances in Neural Information Processing Systems*, 30, 2017.

Daan van der Valk, Stjepan Picek, and Shivam Bhasin. Kilroy was here: The first step towards explainability of neural networks in profiled side-channel analysis. In Guido Marco Bertoni and Francesco Regazzoni (eds.), *Constructive Side-Channel Analysis and Secure Design*, pp. 175–199, Cham, 2021. Springer International Publishing. ISBN 978-3-030-68773-1.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pp. 1578–1585. IEEE, 2017.

Leo Weissbart, Stjepan Picek, and Lejla Batina. One trace is all it takes: Machine learning-based side-channel attack on eddsa. In *Security, Privacy, and Applied Cryptography Engineering: 9th International Conference, SPACE 2019, Gandhinagar, India, December 3–7, 2019, Proceedings 9*, pp. 86–105. Springer, 2019.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. Revisiting a methodology for efficient CNN architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 147–168, 2020.

Yuxin Wu and Justin Johnson. Rethinking" batch" in batchnorm. *arXiv preprint arXiv:2105.07576*, 2021.

Trevor Yap, Adrien Benamira, Shivam Bhasin, and Thomas Peyrin. Peek into the black-box: Interpretable neural network using sat equations in side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 24–53, 2023.

Trevor Yap, Stjepan Picek, and Shivam Bhasin. Occpois: Points of interest based on neural network's key recovery in side-channel analysis through occlusion. In Sourav Mukhopadhyay and Pantelimon Stănică (eds.), *Progress in Cryptology – INDOCRYPT 2024*, pp. 3–28, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-80311-6.

Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient CNN architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1–36, 2020.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (eds.), *Computer Vision – ECCV 2014*, pp. 818–833, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10590-1.

A Notation and variable names

See table 2 below for a list of the notation we use, and table 3 for a list of the main variables we define.

Table 2: List of the notation used in our paper.

Symbol	Description
Variable case: X, x	Upper case: random variable, lower-case: actualization
$\mathbb{R}$, $\mathbb{Z}$	Set of real numbers, set of integers
$[a \mathinner{..} b]$ where $a < b$, $a, b \in \mathbb{Z}$	Interval of integers $\{a, \dots, b\}$
Serif font: S	Other sets
$\mathsf{A} \subset \mathsf{B}$	A is a non-strict subset of B
$\mathsf{A} \setminus \mathsf{B}$	Complement of B in A, i.e. $\mathsf{A} \setminus \mathsf{B} \coloneqq \{x \in \mathsf{A} : x \notin \mathsf{B}\}$
S_+	Nonnegative elements of $\mathsf{S} \subset \mathbb{R}$
S_{++}	Positive elements of $\mathsf{S} \subset \mathbb{R}$
Bold font: $\boldsymbol{x}$	Vector in $\mathbb{R}^D$ for some $D \in \mathbb{Z}_{++}$
$\boldsymbol{a} \odot \boldsymbol{b}$	Elementwise product of $\boldsymbol{a}$ and $\boldsymbol{b}$
$f(\boldsymbol{x})$ where $f : \mathbb{R} \to \mathbb{R}$	Elementwise application of f to $\boldsymbol{x}$
$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}, \dots)$	Gradient with respect to $\boldsymbol{x}$
$\boldsymbol{x}_{\boldsymbol{\alpha}}$ where $\boldsymbol{\alpha} \in \{0, 1\}^D$	Sub-vector of $\boldsymbol{x}$ according to $\boldsymbol{\alpha}$: $(x_d : d = 1, \dots, D : \alpha_d = 1)$
$X \sim p$	X is a random variable with distribution p
p_A	The distribution of A
$p_{A,B}$	The joint distribution of A and B
$p_{A\|B}$	The conditional distribution of A given B
$\boldsymbol{X} \sim p^N$	$\boldsymbol{X}$ is a random vector with elements $X_1, X_2, \dots \overset{\text{i.i.d.}}{\sim} p$
$\mathcal{N}(\mu; \sigma^2)$	Normal distribution with mean μ, scale σ
$\mathcal{U}(\mathsf{S})$	Uniform distribution over set S
$\mathbb{E}\, f(A, B, \dots)$	Expectation w.r.t. all random variables $A, B, \dots$ in expression
$\mathbb{E}_A\, f(A, B, \dots)$	Expectation w.r.t. only to A
$\mathbb{I}[A; B]$	Mutual information between A and B
$\mathbb{I}[A; B \mid C]$	Conditional mutual information between A and B given C
$A \perp\!\!\!\perp B$, $A \not\perp\!\!\!\perp B$	A is marginally independent, dependent on B
$A \perp\!\!\!\perp B \mid C$, $A \not\perp\!\!\!\perp B \mid C$	A is conditionally independent, dependent on B given C

B Extended background

Here we provide a high-level overview of the AES algorithm and power side-channel attacks aimed at a machine learning audience. Since our algorithm views the cryptographic algorithm and hardware as a black box to be characterized with data, a deep understanding is not necessary to understand and appreciate our work. Thus, we omit many details and aim to impart an intuitive understanding of these topics. Interested readers may refer to Daemen & Rijmen (2013) for a detailed introduction to the AES algorithm, to Mangard et al. (2007) for a detailed introduction to power side-channel attacks, and to Picek et al. (2023) for a survey of supervised deep learning-based power side-channel attacks on AES implementations. Additionally, note that while for clarity of exposition we focus here on power side-channel attacks on AES implementations, our algorithm requires only a supervised learning-style dataset of side-channel emission traces and metadata which enables computation of the target variable, so is applicable in a more-general setting. We have demonstrated that it works with various target variables on AES, ECC and RSA implementations with both power and EM radiation measurements, and suspect it is relevant in far more contexts.

Table 3: List of the main variables defined in our paper.

Variable	**Description**
$T \in \mathbb{Z}_{++}$	Dimensionality of power trace
$\boldsymbol{X} \sim p_{\boldsymbol{X}}$	Power trace (random variable)
$\boldsymbol{x} \in \mathbb{R}^T$	Power trace (actualization)
X_t, x_t	Power measurement at time t, i.e. t-th element of power trace
$\mathsf{Y} \subset \mathbb{Z}$	Set of values the targeted variable may take on
$Y \sim p_Y$	Targeted variable (random variable)
$y \in \mathsf{Y}$	Targeted variable (actualization)
$\mathsf{D} \subset \mathbb{R}^T \times \mathsf{Y}$	Dataset of power traces/targeted variable pairs, sampled i.i.d. from $p_{\boldsymbol{X},Y}$
$\boldsymbol{\gamma} \in [0,1]^T$	Occlusion probabilities
γ_t	Occlusion probability at timestep t
$\boldsymbol{\gamma}^* \in [0,1]^T$	The optimal value of $\boldsymbol{\gamma}$ after solving Eqn. **??**
$\tilde{\boldsymbol{\eta}} \in \mathbb{R}^T$	Unconstrained logits which parameterize $\boldsymbol{\gamma}$
$\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}} \sim p_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}}$	Multiplicative binary noise vector parameterized by occlusion probabilities $\boldsymbol{\gamma}$
$\boldsymbol{\alpha} \in \{0,1\}^T$	Actualization of $\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}$
$c : [0,1) \to \mathbb{R}_+$	Cost function for occlusion probability elements γ_t
$C \in \mathbb{R}_{++}$	Budget for occlusion probabilities: they must satisfy $C = \sum_{t=1}^{T} c(\gamma_t)$
$\overline{\gamma} \in (0,1)$	Reparameterized version of C which is more stable w/ data dimensionality
$\Phi_\theta : \mathbb{R}^T \times [0,1]^T \to \mathbb{R}$	Noise-conditional neural net w/ weights $\boldsymbol{\theta}$; returns softmax logits for Y

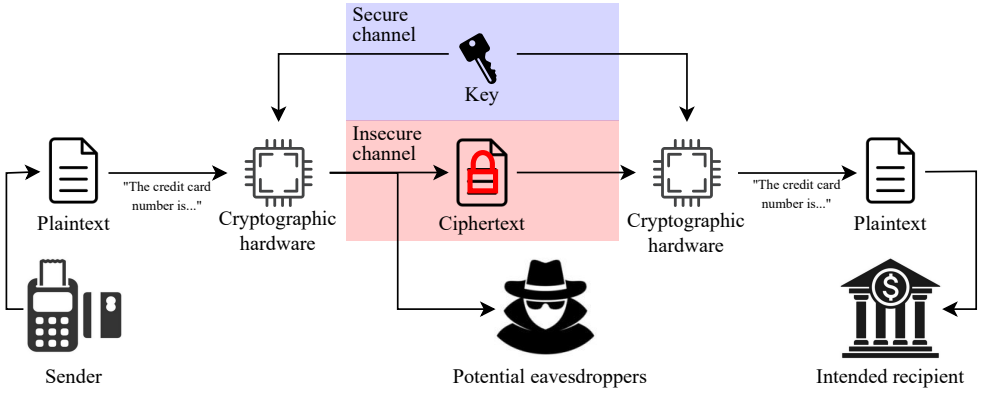

Figure 6: Diagram illustrating the main components of symmetric-key cryptographic algorithms, which enable secure transmission of data over insecure channels where it may be intercepted by eavesdroppers. The data is first partitioned and encoded as a sequence of plaintexts. Each plaintext is transformed into a ciphertext by an invertible function indexed by a cryptographic key. The key is transmitted over a secure channel to intended recipients of the data, allowing them to invert the function and recover the original plaintext. The set of functions is designed so that absent this key, the ciphertext gives no information about the plaintext. Thus, the data remains secure even if eavesdroppers have access to the ciphertext.

B.1 Cryptographic algorithms

Data is often transmitted over insecure channels which leave it accessible not only to intended recipients, but also to unknown and untrusted parties. For example, when a signal is wirelessly transmitted from one antenna to another, an eavesdropper could set up a third antenna between the two and intercept the signal. Alternately, data stored on a hard drive by one user of a computer may be accessed by a different user. Cryptographic algorithms aim to preserve the privacy of data under such circumstances by transforming it so that it is meaningful only in combination with additional data which is known to its intended recipients but not to the untrusted parties.

We focus here on the ubiquitous advanced encryption standard (AES), which is a symmetric-key cryptographic algorithm. See Fig. 6 for a diagram illustrating the important components of such algorithms. The unencrypted data to be transmitted is encoded and partitioned into a sequence of fixed-length bitstrings called *plaintexts*. The cryptographic algorithm encrypts each plaintext into a *ciphertext* by applying an invertible function from a set of functions indexed by an integer called the *cryptographic key*. This set of functions is designed so that of one were to sample a key and plaintext uniformly at random from the sets of all possible keys and plaintexts, then the plaintext and ciphertext would be marginally independent. Thus, such an algorithm may be used to securely transmit data by ensuring that the sender and recipient of the data know a shared key,[2] and that the key is kept secret from all potential eavesdroppers on the data.

B.2 Side-channel attacks

Many symmetric-key cryptographic algorithms are believed to be secure in the sense that it is not feasible to determine their cryptographic key by encrypting known plaintexts and observing the resulting ciphertexts. Any such algorithm with a finite number of possible keys is vulnerable to 'brute-force' attacks based on arbitrarily guessing and checking keys until success, but doing so requires checking half of all possible keys in the average case, which is unrealistic for algorithms such as AES which has either 2^{128}, 2^{192}, or 2^{256} possible keys. To our knowledge the best known such attack against AES reduces the required number of guesses by less than a factor of 8 compared to a naive brute force attack (Mouha, 2021; Tao & Wu, 2015).

However, while algorithms may be secure when considering only their intended inputs and outputs, *hardware executing these algorithms* will inevitably emit measurable physical signals which are statistically associated with their intermediate variables and operations. Examples of such signals include a device's power consumption over time (Kocher et al., 1999), the amount of time it takes to execute a program or instruction (Kocher, 1996; Lipp et al., 2018; Kocher et al., 2019), electromagnetic radiation it emits (Quisquater & Samyde, 2001; Genkin et al., 2016), and sound due to vibrations of its electronic components (Genkin et al., 2014). This phenomenon is called *side-channel leakage*, and can be exploited to determine sensitive data such as a cryptographic key through *side-channel attacks*.

As a simple example of side-channel leakage, consider the following Python function which checks whether a password is correct:

```
def is_correct(provided_password: str, correct_password: str) -> bool:
    if len(provided_password) != len(correct_password):
        return False
    for i in range(len(provided_password)):
        if provided_password[i] != correct_password[i]:
            return False
    return True
```

Suppose the password consists of n characters, each with c possible values. Consider an attacker seeking to determine the correct password by feeding various guessed passwords until the function returns `True`. Naively, the attacker could simply guess and check all possible m-length passwords for $m = 1, \ldots, n$. This would require $O(c^n)$ calls to the function, which would be extremely costly for realistically-large c and n. However, an attacker with knowledge of the function's implementation could dramatically reduce this cost by observing that the function's *execution time* depends on `correct_password`. Because the function exits immediately if `len(provided_password) != len(correct_password)`, the attacker can determine the length of `correct_password` in $O(n)$ time by feeding increasing-length guesses to `is_correct` until its execution time increases. Next, because `is_correct` exits the first time it detects an incorrect character, the attacker can sequentially determine each of the characters of `correct_password` by checking all c possible values of each character and noting that the correct value leads to an increase in execution time. Thus, although `is_correct` secure against attackers which use only its intended inputs and outputs, it provides *essentially no security* against attackers which measure its execution time.

[2]The key is typically shared using an asymmetric-key cryptographic algorithm such as RSA or ECC. Asymmetric-key cryptography is slow and resource-intensive, so when a sufficiently-large amount of data must be transmitted, it is more-efficient to share the key with an asymmetric-key algorithm and then transmit data using a symmetric-key algorithm than to simply transmit the data with an asymmetric-key algorithm.

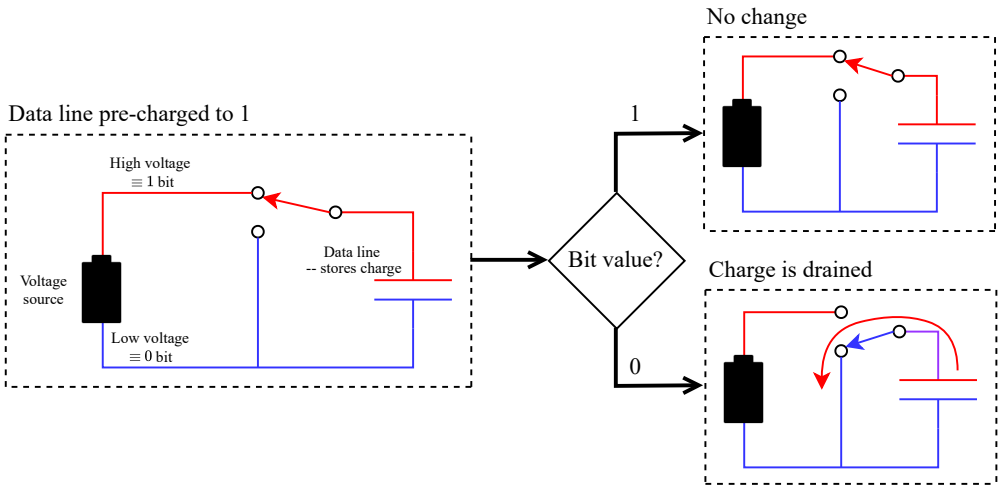

Figure 7: Diagram illustrating one reason there is power side-channel leakage in the device characterized by (Mangard et al., 2007, ch. 4). Data is transmitted over a bus consisting of multiple wires, with one wire representing each bit. Each wire represents a 0 bit as some prescribed 'low' voltage and a 1 bit as a 'high' voltage. Energy is consumed when the voltage of a wire changes from low to high because positive and negative charges, which are attracted to one-another, must be separated to create a high concentration of positive charge on the wire. When 'writing' data to the bus, this particular device first 'pre-charges' all wires to 1, then drains charge from the wires which should represent 0. Thus, because the 0's must be changed to 1's before the next write, energy is consumed in proportion to the number of 0's, thereby creating a statistical association between the device's power consumption and the data it operates on.

In this work we focus on side-channel leakage due to the power consumption over time of a device (as well as EM radiation, which is closely related due to being dominated by the time derivative of power consumption). A device's power consumption is inevitably statistically-associated with the operations it executes and the data it operates on, because these dictate which components are active and the order and manner in which they operate. There are many types of components with different functionality, and components with the same intended functionality are not identical due to imperfect manufacturing processes. These differences impact power consumption. While in general the association between power consumption and data is multifactorial and difficult to describe, in Fig. 7 we illustrate a simple relationship which accounts for a significant portion of the leakage in a device characterized by Mangard et al. (2007).

B.3 Power side-channel attacks on AES implementations

Side-channel attacks are techniques which exploit side-channel leakage to learn sensitive information such as cryptographic keys. There are many categories of attacks, but in this work we focus on a category called *profiled* side-channel attacks on symmetric-key cryptographic algorithms. These attacks assume that the 'attacker' has access to a clone of the actual cryptographic device to be attacked, and the ability to encrypt arbitrary plaintexts with arbitrary cryptographic keys, observe the resulting ciphertexts, and measure the side-channel leakage during encryption. In practice, these assumptions almost certainly overestimate the capabilities of attackers – for example, while in some cases an attacker could plausibly identify the hardware and source code of a cryptographic implementation, purchase copies of this hardware, program them with the source code, and characterize these devices, the nature of the side-channel leakage of these purchased copies would differ from those of the actual device due to PVT (pressure, voltage, temperature) variations (e.g. due to imperfect manufacturing processes, environment, and measurement setup). It has been demonstrated that profiled side-channel attacks can be effective despite this, especially when numerous copies of the target hardware are used for profiling (Das et al., 2019; Danial et al., 2021). Regardless, this type of attack provides an upper bound on the vulnerability of a device to side-channel attacks, which is a useful metric for hardware designers.

While there are diverse types of profiled side-channel attacks, at a high level the following steps encompass the important elements of these attacks:

1. Select some 'sensitive' intermediate variable of the cryptographic algorithm which reveals the cryptographic key (or part of it).
2. Compile a dataset of (side-channel leakage, intermediate variable) pairs by repeatedly randomly selecting a key and plaintext, encrypting the plaintext using the key and recording the resulting ciphertext and side-channel leakage during encryption, and computing the intermediate variable based on knowledge of the cryptographic algorithm.
3. Use supervised learning to train a parametric function approximator to predict intermediate variables from recordings of side-channel leakage during encryption.
4. Measure side-channel leakage during encryptions by the actual target device. Use the trained predictor to predict sensitive variables from side-channel leakage. Potentially, these predictions can be combined to get a better estimate of the key.

In the case of power side-channel attacks on AES, it is generally infeasible to directly target the cryptographic key because care is taken by hardware designers to prevent it from directly influencing power consumption. Instead, it is common to target an intermediate variable which the algorithm directly operates on. A common target is the `SubBytes` output, which is computed as

$$y \coloneqq \mathrm{Sbox}(k \oplus w) \tag{5}$$

where $k \in \{0,1\}^{n_{\text{bits}}}$ is the key, $w \in \{0,1\}^{n_{\text{bits}}}$ is the plaintext, $n_{\text{bits}} \in \mathbb{Z}_{++}$ is the number of bits of the key and plaintext, $\oplus$ is the bitwise exclusive-or operation, and $\mathrm{Sbox} : \{0,1\}^{n_{\text{bits}}} \to \{0,1\}^{n_{\text{bits}}}$ is an invertible function which is widely known and the same for all AES implementations. Note that if the plaintext is known, the key can be computed as

$$k = \mathrm{Sbox}^{-1}(y) \oplus w. \tag{6}$$

Additionally, it is common to independently target subsets of the bits of the cryptographic key (e.g. the individual bytes). This is reasonable because the common AES target variables are largely leaked by instructions which operate on individual bytes.

B.3.1 Template attack: example of a classical profiled side-channel attack

In order to underscore the advantage of deep learning over previous side-channel attack algorithms, we will here describe the template attack algorithm of Chari et al. (2003), variations of which are the state-of-the-art non-deep learning based attacks. The attack is based on modeling the joint distribution of power consumption and intermediate variable as a Gaussian mixture model, as described in algorithm 1.

Note that this algorithm assumes that the joint distribution is well-described by a Gaussian mixture model, which may not hold in practice. Additionally, due to the near-cubic runtime of the matrix inversion of each $\mathbf{\Sigma}_y$ required to compute the Gaussian density functions, this algorithm requires pruning power traces down to a small number of 'high-leakage' timesteps. Follow-up work (Rechberger & Oswald, 2005) proposed performing principle component analysis on the traces and modeling the coefficients of the top principle components rather than individual timesteps. Nonetheless, these constraints mean that the efficacy of this attack is contingent on simplifying assumptions and judgement of which points are 'leaky' using simple statistical techniques and implementation knowledge, limiting its usefulness as a way for hardware designers to evaluate the amount of side-channel leakage from their device.

B.3.2 Practical profiled deep learning side-channel attacks on AES implementations

Here we will give a common and concrete setting and method for performing profiled power side-channel attacks on AES implementations, which is used for all of our experiments.

Algorithm 1: The Gaussian template attack algorithm of Chari et al. (2003)

Input: Profiling (training) dataset $\mathsf{D} \coloneqq \{(\boldsymbol{x}^{(n)}, y^{(n)} : n \in [1 \,..\, N]\} \subset \mathbb{R}^T \times \{0,1\}^{n_{\text{bits}}}$, attack (testing) dataset $\mathsf{D}_{\text{attack}} \coloneqq \{(\boldsymbol{x}_{\text{a}}^{(n)}, w_{\text{a}}^{(n)}) : n \in [1 \,..\, N_{\text{a}}]\} \subset \mathbb{R}^T \times \{0,1\}^{n_{\text{bits}}}$, 'points of interest' $\boldsymbol{T}_{\text{poi}} \coloneqq \{t_m : m = 1, \ldots, \tilde{T}\} \subset [1 \,..\, T]$

Output: Predicted key k^*

Function `get_y` *(k, w)*
  return $\text{Sbox}(k \oplus w)$ `// calculate intermediate variable for given key`
for $n \in [1 \,..\, N]$ **do**
  $\tilde{\boldsymbol{x}}^{(n)} \leftarrow \left(x_{t_m}^{(n)} : m = 1, \ldots, \tilde{T}\right)$ `// prune power traces to 'points of interest'`
for $y \in \{0,1\}^{n_{\text{bits}}}$ **do**
  `// fit a multivariate Gaussian mixture model to the training dataset`
  $\mathsf{D}_y \leftarrow \left\{\tilde{\boldsymbol{x}}^{(n)} : n \in [1 \,..\, N],\, y^{(n)} = y\right\}$
  $N_y \leftarrow |\mathsf{D}_y|$
  $\boldsymbol{\mu}_y \leftarrow \frac{1}{N_y} \sum_{\tilde{\boldsymbol{x}} \in \mathsf{D}_y} \tilde{\boldsymbol{x}}$
  $\boldsymbol{\Sigma}_y \leftarrow \frac{1}{N_y - 1} \sum_{\tilde{\boldsymbol{x}} \in \mathsf{D}_y} (\tilde{\boldsymbol{x}} - \boldsymbol{\mu}_y)(\tilde{\boldsymbol{x}} - \boldsymbol{\mu}_y)^\top$
for $n \in [1 \,..\, N_a]$ **do**
  $\tilde{\boldsymbol{x}}_{\text{a}}^{(n)} \leftarrow \left(x_{\text{a},t_m}^{(n)} : m = 1, \ldots, \tilde{T}\right)$ `// prune power traces of attack dataset`
`// predict key value which maximizes log-likelihood of attack dataset`
$k^* \leftarrow \arg\max_{k \in \{0,1\}^{n_{\text{bits}}}} \sum_{n=1}^{N_{\text{a}}} \left[\log \mathcal{N}\left(\tilde{\boldsymbol{x}}^{(n)}; \boldsymbol{\mu}_{\texttt{get_y}(k, w_{\text{a}}^{(n)})}, \boldsymbol{\Sigma}_{\texttt{get_y}(k, w_{\text{a}}^{(n)})}\right) + \log N_{\texttt{get_y}(k, w_{\text{a}}^{(n)})}\right]$
return k^*

Consider an AES-128 implementation, which has a 128-bit cryptographic key and plaintext. Typically, attackers target each of the 16 bytes of the key independently rather than attacking the full key at once. This practice tacitly assumes that the bytes of the sensitive variable are statistically-independent given the power trace, which is reasonable because many AES operations (including those which are commonly targeted) are performed independently on the individual bytes. Thus, it is a convenient way to simplify the attack without significantly impacting performance.

Additionally, it is difficult and uncommon to try to directly map power traces to associated cryptographic keys, because great care is taken by hardware designers to ensure that the key does not directly impact power consumption. Instead, attackers generally target 'sensitive' intermediate variables which unavoidably directly impact power consumption and can be combined with the plaintext and ciphertext to learn the key. We consider one such intermediate variable which is referred to as the first `SubBytes` output, and is equal to

$$y \coloneqq \text{Sbox}(k \oplus w), \tag{7}$$

where $k \in \{0,1\}^8$ is one byte of the cryptographic key, $w \in \{0,1\}^8$ is the corresponding byte of the plaintext, $\oplus$ denotes the bitwise exclusive-or operation, and $\text{Sbox} : \{0,1\}^8 \to \{0,1\}^8$ is an invertible function which is publicly-available and the same for all AES implementations. Note that if w is known, as is assumed in the profiled side-channel attack setting, then k can be recovered as

$$k = w \oplus \text{Sbox}^{-1}(y). \tag{8}$$

In the context of profiled power side-channel analysis, one assumes to have a 'profiling' dataset (i.e. a training dataset) and an 'attack' dataset (i.e. a test dataset). Suppose we target n_{bytes} bytes of the sensitive variable. In our setting, the profiling dataset consists of ordered pairs of power traces and their associated sensitive intermediate variables:

$$\mathsf{D} \coloneqq \left\{(\boldsymbol{x}^{(n)}, y^{(n)}) : n \in [1 \,..\, N]\right\} \subset \mathbb{R}^T \times \{0,1\}^{n_{\text{bytes}} \times 8} \tag{9}$$

and the attack dataset consists of ordered pairs of power traces and their associated plaintexts:

$$\mathsf{D}_{\mathrm{a}} \coloneqq \left\{ (\boldsymbol{x}_{\mathrm{a}}^{(n)}, w_{\mathrm{a}}^{(n)}) : n \in [1 \mathinner{..} N_{\mathrm{a}}] \right\} \subset \mathbb{R}^T \times \{0,1\}^{n_{\mathrm{bytes}} \times 8}. \tag{10}$$

Many works prove the concept of their approaches by targeting only a single byte of the sensitive variable. When multiple bytes are targeted, it is common to either train a separate neural network for each byte of the sensitive variable, or to amortize the cost of targeting these bytes by training a single neural network with a shared backbone and a separate head for each byte. In this work we exclusively target single bytes, though it would be straightforward to extend our approach to the multitask learning setting.

Consider a neural network architecture $\Phi : \mathsf{Y} \times \mathbb{R}^T \times \mathbb{R}^P \to \mathbb{R}_+ : (y, \boldsymbol{x}, \boldsymbol{\theta}) \mapsto \Phi(y \mid \boldsymbol{x}; \boldsymbol{\theta})$, where each $\Phi(\cdot \mid \boldsymbol{x}; \boldsymbol{\theta})$ is a probability mass function over Y. In the case of a multi-headed network with each head independently predicting a single byte, we compute this probability mass of $y \in \mathsf{Y}$ as the product of the mass assigned to each of its bytes. We train the network by approximately solving the optimization problem

$$\max_{\boldsymbol{\theta} \in \mathbb{R}^P} \quad \mathcal{L}(\boldsymbol{\theta}) \coloneqq \frac{1}{N} \sum_{n=1}^{N} \log \Phi(y^{(n)} \mid \boldsymbol{x}^{(n)}; \boldsymbol{\theta}). \tag{11}$$

Given $\hat{\boldsymbol{\theta}} \in \arg\max_{\boldsymbol{\theta} \in \mathbb{R}^P} \mathcal{L}(\boldsymbol{\theta})$, we then identify the key which maximizes our estimated likelihood of our attack dataset and key as follows:

$$\hat{k} \in \operatorname*{arg\,max}_{k \in \{0,1\}^{n_{\mathrm{bytes}} \times 8}} \quad \sum_{n=1}^{N_{\mathrm{a}}} \log \Phi\left(\left(\mathrm{Sbox}(k_i \oplus w_{\mathrm{a},i}^{(n)}) : i = 1, \ldots, n_{\mathrm{bytes}} \right) \mid \boldsymbol{x}_{\mathrm{a}}^{(n)}; \hat{\boldsymbol{\theta}} \right) \tag{12}$$

where we denote by k_i and $w_i^{(n)}$ the individual bytes of k and $w^{(n)}$.

C Extended related work

Here we consider existing work which has been applied to leakage localization in the context of power or EM radiation side-channel analysis. In line with the problem framing given in Sec. 2, we view these methods as functions which map joint emission-target variable distributions $p_{\boldsymbol{X},Y}$ to vectors in $\mathbb{R}^T$ which assign to each emission measurement variable X_t a scalar 'leakiness' measurement. Prior approaches to leakage localization can largely be categorized as either 1) parametric statistics-based methods which check for pairwise associations between X_t and Y, or 2) neural net attribution methods which use standard supervised deep learning techniques to train a model $\hat{p}_{Y|\boldsymbol{X}} \approx p_{Y|\boldsymbol{X}}$, then use 'attribution' techniques to estimate the average 'importance' of each feature X_t to the predictions made by the model.

C.1 First-order parameteric statistics-based methods

In the side-channel attack literature it is common to use parametric first-order statistical methods to localize leakage. In this work we consider the signal to noise ratio (Mangard et al., 2007), the sum of squared differences (Chari et al., 2003), and correlation power analysis with a Hamming weight leakage model (Brier et al., 2004) due to their popularity and efficacy for 'point of interest' selection (Fan et al., 2014). Below we summarize and discuss these methods.

The signal to noise ratio (SNR) is a standard tool for leakage localization and is defined as

$$\mathrm{SNR}(p_{\boldsymbol{X},Y}) \coloneqq \frac{\mathrm{Var}_{Y \sim p_Y} \mathbb{E}_{\boldsymbol{X} \sim p_{\boldsymbol{X}|Y}}[\boldsymbol{X}]}{\mathbb{E}_{Y \sim p_Y} \mathrm{Var}_{\boldsymbol{X} \sim p_{\boldsymbol{X}|Y}}(\boldsymbol{X})}. \tag{13}$$

The sum of squared differences (SOSD) method was introduced as a point of interest (feature) selection technique for the Gaussian template attack, a parametric side-channel attack based on modeling emission measurements as a multivariate Gaussian mixture model with a component corresponding to each possible

value of the target variable, then using Bayes' rule to estimate the conditional distribution of the target variable given the emission measurements. The SOSD is defined as

$$\mathrm{sosd}(p_{\boldsymbol{X},Y}) \coloneqq \sum_{y \in \mathsf{Y}} \sum_{y' \in \mathsf{Y}} \left(\mathbb{E}_{\boldsymbol{X} \sim p_{\boldsymbol{X}|Y}(\cdot|y)}[\boldsymbol{X}] - \mathbb{E}_{\boldsymbol{X} \sim p_{\boldsymbol{X}|Y}(\cdot|y')}[\boldsymbol{X}] \right)^2 . \tag{14}$$

Correlation power analysis with a Hamming weight leakage model (CPA) is based on the assumption that power consumption is a noisy linear function of the Hamming weight of the target variable, which is a useful model for certain devices. It is defined as the elementwise Pearson correlation between emission measurements and the target variable's Hamming weight:

$$\mathrm{cpa}(p_{\boldsymbol{X},Y}) \coloneqq \frac{\mathbb{E}[(\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])(\mathrm{HW}(Y) - \mathbb{E}[\mathrm{HW}(Y)])]}{\sqrt{\mathrm{Var}(\boldsymbol{X})\,\mathrm{Var}(\mathrm{HW}(Y))}} \tag{15}$$

where $\mathrm{HW} : \left[0 \,..\, 2^d - 1\right] \to [0 \,..\, d]$ maps integers to the sum of their bits when writing them as an unsigned integer.

While techniques such as these are invaluable due to their simplicity, interpretability and low cost, they have major shortcomings. Notably, they consider they are sensitive only to pairwise associations between single emission measurements and the target variable and will consider the measurement X_t to be non-leaky when it is nonleaky in isolation but gives exploitable information *when combined* with X_τ for some $\tau \neq t$, i.e. when $\mathbb{I}[Y; X_t] = 0$ but $\exists \tau \neq t$ such that $\mathbb{I}[Y; X_t \mid X_\tau] > 0$.

Additionally, these techniques make strong assumptions about the nature of $p_{\boldsymbol{X},Y}$ which have generally been observed to hold in practice, but nonetheless introduce the risk of failing to detect leaking measurements. SNR and SOSD are sensitive only to the influence of Y on the mean of $\boldsymbol{X}$, and would fail to identify X_t is leaking if Y changes its distribution while leaving its mean unchanged (e.g. if the variance of X_t changes with Y). CPA is sensitive only to associations between X_t and the Hamming weight of Y, and additionally assumes a linear relationship between these variables.

While the present work concerns mainly 'black box' leakage localization algorithms which make minimal assumptions about the cryptographic implementation being evaluated, in practice these parametric methods are often employed as tools in white-box analyses of implementations. For example, in our work we consider the ASCADv1 datasets (Benadjila et al., 2020), which use a Boolean masking countermeasure and thus have mainly second-order leakage. This renders the above methods ineffective when directly analyzing leakage of their canonical target variable. However, Egger et al. (2022) identified 4 *pairs* of internal AES variables which individually have first-order leakage and may be combined to determine the target variable. Thus, if one is aware *a priori* that these variables leak and has access to the internal randomly-generated Boolean mask variables, they may individually analyze leakage of these variables with the above methods and accumulate the results. We use such an approach to compute 'ground truth' leakiness measurements when running experiments on the ASCADv1 datasets. We emphasize that this type of analysis is challenging and error-prone: multiple leaking variable identified by Egger et al. (2022) were unknown or overlooked by Benadjila et al. (2020), who introduced the dataset. This underscores the importance of black box techniques such as ours to supplement white box analysis.

C.2 Neural net attribution methods

There is a great deal of prior work on localizing leakage based on interpretability techniques to determine the relative importance of input features to a deep neural net which has been trained to model $\hat{p}_{Y|\boldsymbol{X}} \approx p_{Y|\boldsymbol{X}}$ using standard supervised deep learning techniques (Masure et al., 2019; Hettwer et al., 2020; Jin et al., 2020; Zaid et al., 2020; Wouters et al., 2020; van der Valk et al., 2021; Wu & Johnson, 2021; Golder et al., 2022; Li et al., 2022; Perin et al., 2022; Schamberger et al., 2023; Yap et al., 2023; Li et al., 2024; Yap et al., 2025). As baselines we consider the recent works Yap et al. (2025); Schamberger et al. (2023) as well as a variety of older neural net attribution techniques which were compared in Masure et al. (2019); Hettwer et al. (2020); Wouters et al. (2020). Note that our choice of deep learning baselines subsumes those of Yap et al. (2025); Schamberger et al. (2023). Since these methods all 'interpret' a trained deep neural net, we view them as

functions mapping the data distribution $p_{\boldsymbol{X},Y}$ as well as a model $\hat{p}_{Y|\boldsymbol{X}}$ s.t. $\operatorname{softmax}\hat{p}_{Y|\boldsymbol{X}} \approx p_{Y|\boldsymbol{X}}$ to a vector of leakiness estimates in $\mathbb{R}^T$. Here we summarize and discuss the works Masure et al. (2019); Hettwer et al. (2020); Wouters et al. (2020); Schamberger et al. (2023); Yap et al. (2025) which we consider as baselines.

To our knowledge Masure et al. (2019) were the first to explore neural net interpretability for leakage localization, proposing the saliency-like GradVis leakage assessment, defined as

$$\operatorname{gradvis}(p_{\boldsymbol{X},Y}; \hat{p}_{Y|\boldsymbol{X}}) \coloneqq \mathbb{E}_{\boldsymbol{X},Y} \left| -\nabla_{\boldsymbol{x}} \operatorname{logsoftmax} \hat{p}_{Y|\boldsymbol{X}}(Y \mid \boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{X}} \right| . \tag{16}$$

Hettwer et al. (2020) subsequently compared the 1-occlusion (Zeiler & Fergus, 2014), saliency (Simonyan et al., 2014), and layerwise relevance propagation (LRP) (Bach et al., 2015) as leakage localization techniques. The 1-occlusion technique is based on computing the size of change in the model's prediction as each individual input feature is 'occluded' (replaced by 0), and is defined as

$$\text{1-occl}(p_{\boldsymbol{X},Y}; \hat{p}_{Y|\boldsymbol{X}}) \coloneqq \mathbb{E}_{\boldsymbol{X},Y} \left(\left| \hat{p}_{Y|\boldsymbol{X}}(Y \mid \boldsymbol{X}) - \hat{p}_{Y|\boldsymbol{X}}(Y \mid (\boldsymbol{1} - \boldsymbol{I}_t) \odot \boldsymbol{X}) \right| : t = 1, \ldots, T \right) \tag{17}$$

where $\boldsymbol{I}_t$ denotes the vector in $\mathbb{R}^T$ with element t equal to 1 and all other elements equal to 0. Saliency is defined as

$$\operatorname{saliency}(p_{\boldsymbol{X},Y}; \hat{p}_{Y|\boldsymbol{X}}) \coloneqq \mathbb{E}_{\boldsymbol{X},Y} \left| \nabla_{\boldsymbol{x}} \hat{p}_{Y|\boldsymbol{X}}(Y \mid \boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{X}} \right| . \tag{18}$$

Layerwise relevance propagation (Bach et al., 2015) is a gradient-based explainability technique which is more-complicated than the above, and we refer readers to Bach et al. (2015) for an explanation. Wouters et al. (2020) applied the input $*$ gradient method (Shrikumar et al., 2017) to leakage localization; as its name suggests, this method is defined as

$$\operatorname{inputxgrad}(p_{\boldsymbol{X},Y}; \hat{p}_{Y|\boldsymbol{X}}) \coloneqq \mathbb{E}_{\boldsymbol{X},Y} \left| \boldsymbol{X} \odot \nabla_{\boldsymbol{x}} \hat{p}_{Y|\boldsymbol{X}}(Y \mid \boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{X}} \right| . \tag{19}$$

We find that all these techniques have a tendency to incorrectly assign low leakiness to certain measurements, particularly in scenarios where lots of features have significant leakage. We suspect that a primary reason for this is that these methods rely on perturbing the input x_t to the function $\hat{p}_{Y|\boldsymbol{X}}(y \mid x_1, \ldots, x_t, \ldots, x_T)$, but when Y is almost entirely determined by X_τ for $\tau \neq t$, it becomes nearly independent of X_t conditioned on $\{X_1, \ldots, X_T\} \setminus \{X_t\}$, i.e. $\hat{p}_{Y|\boldsymbol{X}}(y \mid x_\tau : \tau = 1, \ldots, T) \approx \hat{p}_{Y|\boldsymbol{X}}(y \mid x_\tau : \tau = 1, \ldots, t-1, t+1, \ldots, T)$. We demonstrate this phenomenon with a simple Gaussian mixture model setting in Sec. C.3.

We also consider the recent works Schamberger et al. (2023); Yap et al. (2025) which estimate leakiness by perturbing many inputs simultaneously to a classifier, and do not necessarily suffer from the same issue. Schamberger et al. (2023) presents the m-occlusion technique, which is like 1-occlusion except that it occludes m-diameter windows rather than single points. This could plausibly overcome the aforementioned issue if the 'redundant' points are temporally-local. However, it has an undesirable 'smoothing' effect where the estimated leakiness of a single point is tied to those of nearby points. Schamberger et al. (2023) also proposes to use second-order occlusion to analyze leakiness, where *pairs* of measurements are occluded rather than only individual measurements. This is computationally-expensive because it requires $\Theta(T^2)$ passes through the dataset where T is the data dimensionality. Additionally, Schamberger et al. (2023) proposes it as a means to determine whether a measurement has first-order leakage or is part of a second-order leaking pair, and does not explore its use for single-measurement leakiness estimation. In our experiments we find it only marginally-better than first-order occlusion for this task, and not worth the significantly-higher computational cost. Yap et al. (2025) proposes the OccPOI technique, which aims to identify a non-unique minimal set of measurements sufficient for a neural net to attain some chosen classification performance when all other measurements are occluded. This differs from our other considered methods in that it does not assign a leakiness estimate to every measurement, and we find that it is ill-suited for our leakage localization task and performance metrics. Additionally, it is computationally-expensive to run, requiring $\Omega(T)$ *non-parallelizable* passes through the dataset.

C.3 Numerical experiment illustrating conditional mutual information decay when many redundant leaking measurements are present

Here we provide a simple numerical experiment to illustrate conditional mutual information decay. Consider random variables $Y \sim \mathcal{U}\{-1, 1\}$ and $X_1, \ldots, X_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}(Y, \sigma^2)$. In Fig. 8 we plot the quantity $\mathbb{I}[Y; X_n \mid$

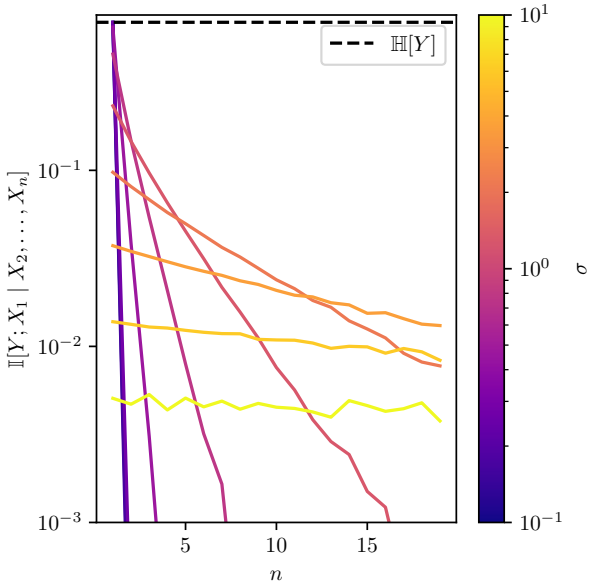

Figure 8: A numerical experiment evaluating the scaling behavior of $\mathbb{I}[Y; X_n \mid X_1, \ldots, X_{n-1}]$ vs. n for various values of σ, where $Y \sim \mathcal{U}\{-1, 1\}$ and $X_1, \ldots, X_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}(Y, \sigma^2)$. Observe that for small σ, each X_i is approximately a point mass on Y and we have $\mathbb{I}[Y; X_1] \approx \mathbb{H}[Y]$ and $\mathbb{I}[Y; X_n \mid X_1, \ldots, X_{n-1}] \approx 0$ for all $n > 1$. When σ is large, each X_i gives us little information about Y and we have $\mathbb{I}[Y; X_n \mid X_1, \ldots, X_{n-1}] \ll \mathbb{H}[Y]$ approximately constant with n sufficiently small.

$X_1, \ldots, X_{n-1}]$ vs. n. We see that for various values of σ the quantity decays rapidly with n, and appears to be well-described by the function $\mathbb{I}[Y; X_1 \mid X_2, \ldots, X_n] \approx Ik^n$ for some $I \in \mathbb{R}_+$, $k \in (0, 1)$. Informally, the fact that conditional mutual information decays with n makes sense for the following reason: each successive X_i can be viewed as an independent 'noisy observation' of Y which reduces uncertainty about it without fully determining it. For all n we have $0 \leq \mathbb{I}[Y; X_1, \ldots, X_n] = \mathbb{I}[Y; X_1] + \mathbb{I}[Y; X_2 \mid X_1] + \ldots \mathbb{I}[Y; X_n \mid X_1, \ldots, X_{n-1}] \leq \mathbb{H}[Y]$. If we assume that each $\mathbb{I}[Y; X_m \mid X_1, \ldots, X_{m-1}]$ is nonnegative, then for their sum to stay bounded they must converge to 0.

D Extended method with derivations

Given $\boldsymbol{X}, Y \sim p_{\boldsymbol{X},Y}$ as defined in section 2, where $\boldsymbol{X} := (X_1, \ldots, X_T)$, we seek to assign to each timestep t a scalar γ_t^* indicating the 'amount of leakage' about Y due to X_t. All of the quantities $\mathbb{I}[X; X_t \mid \mathsf{S}]$ for $\mathsf{S} \subset \{X_1, \ldots, X_T\} \setminus \{X_t\}$ are relevant to the 'leakiness' of X_t, but it is not clear how to weight these 2^T quantities into a single scalar measurement. Most prior work simply ignores most of these quantities: the first-order parametric methods (Mangard et al., 2007; Brier et al., 2004; Chari et al., 2003) consider only the pairwise terms $\mathbb{I}[Y; X_t]$, and GradVis (Masure et al., 2019), Saliency (Simonyan et al., 2014; Hettwer et al., 2020), 1-occlusion (Zeiler & Fergus, 2014; Hettwer et al., 2020), layerwise relevance propagation (Bach et al., 2015; Hettwer et al., 2020), input $*$ gradient (Shrikumar et al., 2017; Wouters et al., 2020) may loosely be viewed as computing proxies for $\mathbb{I}[Y; X_t \mid X_{\tau \neq t}]$. While m-occlusion, second-order occlusion (Schamberger et al., 2023), and OccPOI (Yap et al., 2025) are sensitive to more of these terms, they still ignore almost all of them in an *ad hoc* manner.

In this section we propose a constrained optimization problem which implicitly defines an intuitively-reasonable definition of γ_t^* which is sensitive to $\mathbb{I}[Y; X_t \mid \mathsf{S}]$ for *all* $\mathsf{S} \subset \{X_1, \ldots, X_T\} \setminus \{X_t\}$. We then propose an adversarial deep learning algorithm which lets us approximately solve it by modeling all the conditional distributions $p_{Y|\mathsf{S}}$ in an amortized manner which emphasizes those with a large impact on the objective. While our objective is a sum over 2^T occlusion mask-like values, in practice we find we can efficiently optimize

it using the reparameterization trick (Kingma & Welling, 2014; Rezende et al., 2014) with a CONCRETE-like relaxation (Jang et al., 2017; Maddison et al., 2017) of our objective. This lets us exploit first-order gradient information, in contrast to 'hard' occlusion-based methods such as Schamberger et al. (2023); Yap et al. (2025) which leverage only zeroth-order information.

D.1 Optimization problem

We define a vector $\boldsymbol{\gamma} \in [0,1]^T$ which we name the *occlusion probabilities*. We use $\boldsymbol{\gamma}$ to parameterize a distribution over binary vectors in $\{0,1\}^T$ as follows:

$$\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}} \sim p_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}} \quad \text{where} \quad \mathcal{A}_{\boldsymbol{\gamma},t} = \begin{cases} 1 & \text{with probability } 1-\gamma_t \\ 0 & \text{with probability } \gamma_t, \end{cases} \tag{20}$$

i.e. $\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}$ is a vector of independent Bernoulli random variables where the t-th element has parameter $p = 1-\gamma_t$. For arbitrary vectors $\boldsymbol{x} \in \mathbb{R}^T$, $\boldsymbol{\alpha} \in \{0,1\}^T$, let us denote $\boldsymbol{x}_{\boldsymbol{\alpha}} \coloneqq (x_t : t = 1, \ldots, T : \alpha_t = 1)$, i.e. the sub-vector of $\boldsymbol{x}$ containing its elements for which the corresponding element of $\boldsymbol{\alpha}$ is 1. We can accordingly use $\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}$ to obtain random sub-vectors $\boldsymbol{X}_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}}$ of $\boldsymbol{X}$. Note that γ_t denotes the probability that X_t will *not* be an element of $\boldsymbol{X}_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}}$ (hence, 'occlusion probability').

We assign to each element of $\boldsymbol{\gamma}$ a 'cost', defined as

$$c : [0,1] \to \mathbb{R}_+ : x \mapsto \begin{cases} \frac{x}{1-x} & x < 1 \\ \infty & x = 1 \end{cases}. \tag{21}$$

We seek to solve the constrained optimization problem

$$\min_{\boldsymbol{\gamma} \in [0,1]^T} \quad \mathcal{L}_{\text{ideal}}(\boldsymbol{\gamma}) \coloneqq \mathbb{I}[Y; \boldsymbol{X}_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}} \mid \boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}] \quad \text{such that} \quad \sum_{t=1}^{T} c(\gamma_t) = C \tag{22}$$

for hyperparameter $C > 0$. Note that c is strictly-increasing with $c(0) = 0$ and $\lim_{\substack{x \to 1 \\ x<1}} c(x) = \infty$ so that for finite C any optimal $\boldsymbol{\gamma}$ will be in $[0,1)^T$, and increasing some γ_t necessarily entails reducing some other γ_τ, $\tau \neq t$. Additionally, for each t we can re-write our objective as

$$\begin{aligned} \mathcal{L}_{\text{ideal}}(\boldsymbol{\gamma}) &= \sum_{\boldsymbol{\alpha} \in \{0,1\}^T} p_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma}}}(\boldsymbol{\alpha})\, \mathbb{I}[Y; \boldsymbol{X}_{\boldsymbol{\alpha}}] && (23) \\ &= \sum_{\substack{\boldsymbol{\alpha} \in \{0,1\}^T \\ \alpha_t = 0}} p_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma},-t}}(\boldsymbol{\alpha}_{-t}) \left[(1-\gamma_t)\, \mathbb{I}[Y; X_t, \boldsymbol{X}_{\boldsymbol{\alpha}}] + \gamma_t\, \mathbb{I}[Y; \boldsymbol{X}_{\boldsymbol{\alpha}}] \right] && (24) \\ &= \sum_{\substack{\boldsymbol{\alpha} \in \{0,1\}^T \\ \alpha_t = 0}} p_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma},-t}}(\boldsymbol{\alpha}_{-t}) \left[\mathbb{I}[Y; X_t, \boldsymbol{X}_{\boldsymbol{\alpha}}] - \gamma_t\, \mathbb{I}[Y; X_t \mid \boldsymbol{X}_{\boldsymbol{\alpha}}] \right], && (25) \end{aligned}$$

which implies

$$\frac{\partial \mathcal{L}_{\text{ideal}}(\boldsymbol{\gamma})}{\partial \gamma_t} = - \sum_{\substack{\boldsymbol{\alpha} \in \{0,1\}^T \\ \alpha_t = 0}} p_{\boldsymbol{\mathcal{A}}_{\boldsymbol{\gamma},-t}}(\boldsymbol{\alpha}_{-t})\, \mathbb{I}[Y; X_t \mid \boldsymbol{X}_{\boldsymbol{\alpha}}]. \tag{26}$$

D.2 Estimating mutual information with deep neural nets

We cannot solve equation 22 directly because we lack an expression for $p_{\boldsymbol{X},Y}$. Here we derive an equivalent optimization problem which uses deep learning to characterize $p_{\boldsymbol{X},Y}$ using data.

Consider the family $\{\Phi_{\boldsymbol{\alpha}}\}_{\boldsymbol{\alpha} \in \{0,1\}^T}$ with each element a deep neural net

$$\Phi_{\boldsymbol{\alpha}} : \mathsf{Y} \times \mathbb{R}^{\sum_{t=1}^{T} \alpha_t} \times \mathbb{R}^P \to [0,1] : (y, \boldsymbol{x}_{\boldsymbol{\alpha}}, \boldsymbol{\theta}) \mapsto \Phi_{\boldsymbol{\alpha}}(y \mid \boldsymbol{x}_{\boldsymbol{\alpha}}; \boldsymbol{\theta}). \tag{27}$$

We assume each $\Phi_{\boldsymbol{\alpha}}(\cdot \mid \boldsymbol{x}; \boldsymbol{\theta})$ is a probability mass function over Y (e.g. the neural net has a softmax output activation). We define the optimization problem

$$\min_{\boldsymbol{\gamma} \in [0,1]^T} \max_{\boldsymbol{\theta} \in \mathbb{R}^P} \quad \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta}) \coloneqq \mathbb{E} \log \Phi_{\boldsymbol{\mathcal{A}}_\gamma}(Y \mid \boldsymbol{X}_{\boldsymbol{\mathcal{A}}_\gamma}; \boldsymbol{\theta}) \quad \text{such that} \quad \sum_{t=1}^{T} c(\gamma_t) = C. \tag{28}$$

Proposition D.1. *Consider the objective function $\mathcal{L}_{\text{adv}}$ of equation 28. Suppose there exists some $\boldsymbol{\theta}^* \in \mathbb{R}^P$ such that $\Phi_{\boldsymbol{\alpha}}(y \mid \boldsymbol{x}_{\boldsymbol{\alpha}}; \boldsymbol{\theta}^*) = p_{Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}}(y \mid \boldsymbol{x}_{\boldsymbol{\alpha}})$ for all $\boldsymbol{\alpha} \in \{0,1\}^T$, $\boldsymbol{x} \in \mathbb{R}^T$, $y \in \mathsf{Y}$. Then*

$$\boldsymbol{\theta}^* \in \arg\max_{\boldsymbol{\theta} \in \mathbb{R}^P} \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta}) \quad \forall \boldsymbol{\gamma} \in [0,1]^T. \tag{29}$$

Furthermore, for all $y \in \mathsf{Y}$ and for all $\boldsymbol{\gamma} \in [0,1]^T$, $\boldsymbol{\alpha} \in \{0,1\}^T$ such that $p_{\boldsymbol{\mathcal{A}}_\gamma}(\boldsymbol{\alpha}) > 0$,

$$\Phi_{\boldsymbol{\alpha}}(y \mid \boldsymbol{x}_{\boldsymbol{\alpha}}; \hat{\boldsymbol{\theta}}) = p_{Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}}(y \mid \boldsymbol{x}_{\boldsymbol{\alpha}}) \quad p_{\boldsymbol{X}}\text{-almost surely} \quad \forall \hat{\boldsymbol{\theta}} \in \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^P} \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta}). \tag{30}$$

Proof. Note that since each $\Phi_{\boldsymbol{\alpha}}(\cdot \mid \boldsymbol{x}, \boldsymbol{\theta})$ is a probability mass function over Y, by Gibbs' inequality we have

$$\mathbb{E} \log \Phi_{\boldsymbol{\alpha}}(Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}; \boldsymbol{\theta}) \leq \mathbb{E} \log p_{Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}}(Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}) \quad \forall \boldsymbol{\alpha} \in \{0,1\}^T, \boldsymbol{\theta} \in \mathbb{R}^P. \tag{31}$$

Thus,

$$\mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta}^*) \geq \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta}) \quad \forall \boldsymbol{\theta} \in \mathbb{R}^P, \boldsymbol{\gamma} \in [0,1]^T, \tag{32}$$

which implies the first claim.

Next, consider some fixed $\boldsymbol{\gamma} \in [0,1]^T$ and $\hat{\boldsymbol{\theta}} \in \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^P} \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta})$. We must have $\mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \hat{\boldsymbol{\theta}}) = \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta}^*)$. Thus,

$$\begin{aligned}
0 &= \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta}^*) - \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \hat{\boldsymbol{\theta}}) && (33)\\
&= \mathbb{E}\left[\log p_{Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}}(Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}) - \log \Phi_{\boldsymbol{\alpha}}(Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}; \hat{\boldsymbol{\theta}})\right] && (34)\\
&= \sum_{\boldsymbol{\alpha} \in \{0,1\}^T} p_{\boldsymbol{\mathcal{A}}_\gamma}(\boldsymbol{\alpha}) \, \mathbb{E}\left[\log p_{Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}}(Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}) - \log \Phi_{\boldsymbol{\alpha}}(Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}; \hat{\boldsymbol{\theta}})\right]. && (35)
\end{aligned}$$

By Gibbs' inequality, each of the expectations in the summation is nonnegative, which implies that whenever $p_{\boldsymbol{\mathcal{A}}_\gamma}(\boldsymbol{\alpha}) > 0$ we must have

$$\begin{aligned}
0 &= \mathbb{E}\left[\log p_{Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}}(Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}) - \log \Phi_{\boldsymbol{\alpha}}(Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}; \hat{\boldsymbol{\theta}})\right] && (36)\\
&= \int_{\mathbb{R}^{\sum_{t=1}^{T} \alpha_t}} p_{\boldsymbol{X}_{\boldsymbol{\alpha}}}(\boldsymbol{x}_{\boldsymbol{\alpha}}) \, \mathbb{KL}\left[p_{Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}}(\cdot \mid \boldsymbol{x}_{\boldsymbol{\alpha}}) \parallel \Phi_{\boldsymbol{\alpha}}(\cdot \mid \boldsymbol{x}_{\boldsymbol{\alpha}}; \hat{\boldsymbol{\theta}})\right] d\boldsymbol{x}_{\boldsymbol{\alpha}}. && (37)
\end{aligned}$$

Since $\mathbb{KL}\left[p_{Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}}(\cdot \mid \boldsymbol{x}_{\boldsymbol{\alpha}}) \parallel \Phi_{\boldsymbol{\alpha}}(\cdot \mid \boldsymbol{x}_{\boldsymbol{\alpha}}; \hat{\boldsymbol{\theta}})\right] \geq 0$ with equality if and only if $p_{Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}}(y \mid \boldsymbol{x}_{\boldsymbol{\alpha}}) = \Phi_{\boldsymbol{\alpha}}(y \mid \boldsymbol{x}_{\boldsymbol{\alpha}}; \hat{\boldsymbol{\theta}})$ $\forall y \in \mathsf{Y}$, this must be the case except possibly for $\boldsymbol{x} \in \mathbb{R}^T$ where

$$\int_{\{\boldsymbol{x}_{\boldsymbol{\alpha}} : \boldsymbol{x} \in \mathbb{R}^T\}} p_{\boldsymbol{X}_{\boldsymbol{\alpha}}}(\boldsymbol{x}_{\boldsymbol{\alpha}}) \, d\boldsymbol{x}_{\boldsymbol{\alpha}} = 0 \implies \int_{\mathbb{R}^T} p_{\boldsymbol{X}}(\boldsymbol{x}) \, d\boldsymbol{x} = 0. \tag{38}$$

This implies the second claim. □

Corollary D.2. *Under the assumptions of Proposition D.1, equations 22 and 28 are equivalent.*

Proof. Observe that for any $\boldsymbol{\gamma} \in [0,1]^T$,

$$\begin{aligned}
\max_{\boldsymbol{\theta}\in\mathbb{R}^P} \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma},\boldsymbol{\theta}) &= \max_{\boldsymbol{\theta}\in\mathbb{R}^P} \mathbb{E}\log \Phi_{\boldsymbol{\mathcal{A}}_\gamma}(Y \mid \boldsymbol{X}_{\boldsymbol{\mathcal{A}}_\gamma}(Y \mid \boldsymbol{X}_{\boldsymbol{\mathcal{A}}_\gamma};\boldsymbol{\theta}) && (39)\\
&= \sum_{\boldsymbol{\alpha}\in\{0,1\}^T} p_{\boldsymbol{\mathcal{A}}_\gamma}(\boldsymbol{\alpha})\,\mathbb{E}\log p_{Y\mid \boldsymbol{X}_{\boldsymbol{\alpha}}}(Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}) \quad \text{by Prop. D.1} && (40)\\
&= -\sum_{\boldsymbol{\alpha}\in\{0,1\}^T} p_{\boldsymbol{\mathcal{A}}_\gamma}(\boldsymbol{\alpha})\,\mathbb{H}[Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}] && (41)\\
&\equiv \sum_{\boldsymbol{\alpha}\in\{0,1\}^T} p_{\boldsymbol{\mathcal{A}}_\gamma}(\boldsymbol{\alpha})\,[\mathbb{H}[Y] - \mathbb{H}[Y \mid \boldsymbol{X}_{\boldsymbol{\alpha}}]] \quad \text{because } \mathbb{H}[Y] \text{ is not a function of } \boldsymbol{\gamma} && (42)\\
&= \sum_{\boldsymbol{\alpha}\in\{0,1\}^T} p_{\boldsymbol{\mathcal{A}}_\gamma}(\boldsymbol{\alpha})\,\mathbb{I}[Y;\boldsymbol{X}_{\boldsymbol{\alpha}}] && (43)\\
&= \mathbb{I}[Y;\boldsymbol{X}_{\boldsymbol{\mathcal{A}}_\gamma} \mid \boldsymbol{\mathcal{A}}_\gamma] && (44)\\
&= \mathcal{L}_{\text{ideal}}(\boldsymbol{\gamma}). && (45)
\end{aligned}$$

This implies the result. □

Corollary D.3. *Suppose the assumptions of Proposition D.1 are satisfied, and let* $\hat{\boldsymbol{\theta}} \in \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^P} \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma},\boldsymbol{\theta})$ *for some* $\boldsymbol{\gamma} \in [0,1]^T$*. Consider* $\boldsymbol{\alpha} \coloneqq \boldsymbol{\alpha}' + \boldsymbol{\alpha}''$ *where* $\boldsymbol{\alpha}', \boldsymbol{\alpha}'' \in \{0,1\}^T$ *such that* $\alpha'_t = 1 \implies \alpha''_t = 0$ *and* $\alpha''_t = 1 \implies \alpha'_t = 0$*, and* $p_{\boldsymbol{\mathcal{A}}_\gamma}(\boldsymbol{\alpha}) > 0$*. For all* $y \in \mathsf{Y}$*, it follows immediately from Proposition D.1 that* $p_{\boldsymbol{X}}$*-almost everywhere we can use our classifiers to compute the pointwise mutual information quantities*

$$\begin{aligned}
\text{pmi}(y;\boldsymbol{x}_{\boldsymbol{\alpha}'} \mid \boldsymbol{x}_{\boldsymbol{\alpha}''}) &\coloneqq \log p_{Y\mid \boldsymbol{X}_{\boldsymbol{\alpha}}}(y \mid \boldsymbol{x}_{\boldsymbol{\alpha}}) - \log p_{Y\mid \boldsymbol{X}_{\boldsymbol{\alpha}''}}(y \mid \boldsymbol{x}_{\boldsymbol{\alpha}''}) && (46)\\
&= \log \Phi_{\boldsymbol{\alpha}}(y \mid \boldsymbol{x}_{\boldsymbol{\alpha}};\hat{\boldsymbol{\theta}}) - \log \Phi_{\boldsymbol{\alpha}''}(y \mid \boldsymbol{x}_{\boldsymbol{\alpha}''};\hat{\boldsymbol{\theta}}). && (47)
\end{aligned}$$

This is useful because it allows us to assess leakage from *single* power traces, as opposed to merely summarizing distributions of power traces. There are scenarios where a power measurement might leak for some traces but not for others. For example, a common countermeasure is to randomly delay leaky instructions or swap their order with another instruction so that they do not occur at a deterministic time relative to the start of encryption. One could use pmi computations to determine the timetep at which the leaky instruction has been run in a single trace.

Since it would be impractical to train 2^T deep neural networks independently, we implement the family of classifiers by a single neural net with input dropout and with the dropout mask fed to the neural net as an auxiliary input:

$$\Phi : \mathsf{Y} \times \mathbb{R}^T \times \{0,1\}^T \times \mathbb{R}^P \to [0,1] : (y,\boldsymbol{x},\boldsymbol{\alpha},\boldsymbol{\theta}) \mapsto \Phi(y \mid \boldsymbol{x}\odot\boldsymbol{\alpha},\boldsymbol{\alpha};\boldsymbol{\theta}) \tag{48}$$

where $\Phi_{\boldsymbol{\alpha}}(y \mid \boldsymbol{x}_{\boldsymbol{\alpha}};\boldsymbol{\theta}) \coloneqq \Phi(y \mid \boldsymbol{x}\odot\boldsymbol{\alpha},\boldsymbol{\alpha};\boldsymbol{\theta})$. This approach was inspired by Lippe et al. (2022).

D.3 Re-parametrization into an unconstrained optimization problem

We would like to approximately solve equation 28 using an alternating stochastic gradient descent-style approach, similarly to GANs (Goodfellow et al., 2014). Thus, it is convenient to express it as an unconstrained optimization problem. We first define a new vector $\boldsymbol{\eta} \in \Delta^{T-1}$ where $\Delta^{T-1} \coloneqq \{(\delta_1,\ldots,\delta_T) \in \mathbb{R}_+^T : \sum_{t=1}^T \delta_t = 1\}$ denotes the T-simplex. We then define $\boldsymbol{\gamma}$ to be the vector satisfying the equality

$$\begin{aligned}
& c(\gamma_t) = C\eta_t && (49)\\
\implies & \frac{\gamma_t}{1-\gamma_t} = C\eta_t && (50)\\
\implies & \log\gamma_t - \log(1-\gamma_t) = \log C + \log\eta_t && (51)\\
\implies & \gamma_t = \text{sigmoid}\left(\log C + \log\eta_t\right). && (52)
\end{aligned}$$

If we define $\boldsymbol{\eta} \coloneqq \mathrm{softmax}(\tilde{\boldsymbol{\eta}})$ for $\tilde{\boldsymbol{\eta}} \in \mathbb{R}^T$, then we can express

$$\gamma_t = \mathrm{sigmoid}\left(\log C + \log \tilde{\eta}_t - \mathrm{logsumexp}(\tilde{\boldsymbol{\eta}})\right), \tag{53}$$

which allows us to map the unconstrained vector $\tilde{\boldsymbol{\eta}}$ to $\boldsymbol{\gamma}$ or $\log \boldsymbol{\gamma}$ using numerically-stable PyTorch operations. Our constrained optimization problem 28 is thus equivalent to the following unconstrained problem:

$$\min_{\tilde{\boldsymbol{\eta}} \in \mathbb{R}^T} \max_{\boldsymbol{\theta} \in \mathbb{R}^P} \quad \mathcal{L}(\tilde{\boldsymbol{\eta}}, \boldsymbol{\theta}) \coloneqq \mathbb{E} \log \Phi\left(Y \mid \boldsymbol{X} \odot \boldsymbol{\mathcal{A}}_{\gamma(\tilde{\boldsymbol{\eta}})}, \boldsymbol{\mathcal{A}}_{\gamma(\tilde{\boldsymbol{\eta}})}; \boldsymbol{\theta}\right). \tag{54}$$

D.4 Implementation details

It is infeasible to exactly compute the expectation with respect to $\boldsymbol{\mathcal{A}}_{\gamma(\tilde{\boldsymbol{\eta}})}$ because doing so would require summing over 2^T terms. As is routine in deep learning contexts, we instead approximate the gradient with Monte Carlo integration. Note that our objective[3] takes the form $\mathcal{L}(\tilde{\boldsymbol{\eta}}) = \mathbb{E}\, f(\boldsymbol{\mathcal{A}}_{\gamma(\tilde{\boldsymbol{\eta}})})$ where $f(\boldsymbol{\alpha}) \coloneqq \mathbb{E}_{\boldsymbol{X},Y} \log \Phi(Y \mid \boldsymbol{X} \odot \boldsymbol{\alpha}, \boldsymbol{\alpha}; \boldsymbol{\theta})$ and the distribution of $\boldsymbol{\mathcal{A}}_{\gamma(\tilde{\boldsymbol{\eta}})}$ depends on $\tilde{\boldsymbol{\eta}}$.

Unbiased estimators for $\nabla_{\tilde{\boldsymbol{\eta}}} \mathcal{L}(\boldsymbol{\eta})$ of this form are usually based on the REINFORCE estimator (Williams, 1992) with control variates, and tend to have complicated implementations and high variance. We tried using the vanilla REINFORCE estimator with simple control variates as well as the more-sophisticated REBAR estimator (Tucker et al., 2017), and found that the former works poorly, while the latter works well. Subsequent ablation studies revealed that the biased CONCRETE estimator (Maddison et al., 2017) works almost as well as REBAR for our application and is considerably simpler, so in this work we use CONCRETE.

The CONCRETE estimator lets us write $\boldsymbol{\mathcal{A}}_{\gamma(\tilde{\boldsymbol{\eta}})}$ as a deterministic function of $\tilde{\boldsymbol{\eta}}$ and $\tilde{\boldsymbol{\eta}}$-independent noise and thereby use the reparameterization trick to estimate $\nabla_{\tilde{\boldsymbol{\eta}}} \mathcal{L}(\tilde{\boldsymbol{\eta}})$ using standard automatic differentiation tools. For binary random variables such as the elements of $\boldsymbol{\mathcal{A}}_{\gamma(\tilde{\boldsymbol{\eta}})}$, the estimator is built on the observation that

$$x \sim \mathrm{Bernoulli}(p) \equiv x = H(\log p - \log(1-p) + \log u - \log(1-u)) \quad \text{for} \quad u \sim \mathcal{U}(0,1) \tag{55}$$

where $H(x) \coloneqq \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$ denotes the unit step function. H is not amenable to gradient descent because its derivative is zero almost everywhere, but we can approximate it with the tempered sigmoid function $\mathrm{sigmoid}_\tau(x) \coloneqq \mathrm{sigmoid}(x/\tau)$. The temperature parameter $\tau > 0$ can be tuned to control a bias-variance tradeoff for the estimator: $\lim_{\tau \to 0} \mathrm{sigmoid}_\tau(x) = H(x)\ \forall x \in \mathbb{R} \setminus \{0\}$, but variance increases as $O(1/\tau)$ (Shekhovtsov, 2021). We find that results are reasonable when we simply leave τ fixed at 1, and we do this throughout the present work. We conjecture that this is because all our performance evaluation metrics consider only the *relative* leakiness estimates produced by our method, and the nature of the bias is not to significantly impact the relative sizes of the elements of $\boldsymbol{\gamma}^*$.

Note that while our original loss function

$$\ell(\tilde{\boldsymbol{\eta}}, \boldsymbol{\theta}, \boldsymbol{x}, y, \boldsymbol{\alpha}) \coloneqq \log \Phi(y \mid \boldsymbol{x} \odot \boldsymbol{\alpha}, \boldsymbol{\alpha}; \boldsymbol{\theta}) \tag{56}$$

is written for a 'hard' occlusion mask $\boldsymbol{\alpha} \in \{0,1\}^T$, the relaxed occlusion masks lie inside the open ball $(0,1)^T$. Thus, we must relax our loss function to accept these inputs as well. While the right-hand side of Eqn. 56 is still a valid expression for $\boldsymbol{\alpha}$ in $(0,1)^T$, its optimal value of this loss with respect to $\boldsymbol{\theta}$ does not vary smoothly with $\boldsymbol{\alpha}$ because rescaling the elements of $\boldsymbol{X}$ by nonzero constants does not change its mutual information with Y. Thus, we instead use the stochastic relaxed loss function

$$\ell_{\text{relaxed}}(\tilde{\boldsymbol{\eta}}, \boldsymbol{\theta}, \boldsymbol{x}, y, \boldsymbol{\alpha}) \coloneqq \log \Phi(y \mid \boldsymbol{x} \odot \boldsymbol{\alpha} + \boldsymbol{\varepsilon} \odot (\boldsymbol{1} - \boldsymbol{\alpha}); \boldsymbol{\theta}) \quad \text{where} \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(0,1)^T. \tag{57}$$

In Alg. 2 we provide pseudocode for a practical implementation of ALL, omitting details such as minibatch use for clarity. Also refer to this link for a minimal self-contained PyTorch (Paszke et al., 2019) implementation.

[3] ignoring its $\boldsymbol{\theta}$-dependence because differentiating with respect to $\boldsymbol{\theta}$ is straightforward here

Algorithm 2: Simplified implementation of our Adversarial Leakage Localization (ALL) algorithm.

Input: Dataset $\mathsf{D} \coloneqq \{(\boldsymbol{x}^{(n)}, y^{(n)}) : n = 1, \ldots, N\} \subset \mathbb{R}^T \times \mathsf{Y}$, mask-conditional classifier architecture $\tilde{\Phi}_{\cdot} : \mathbb{R}^T \times [0,1]^T \to \mathbb{R}^{|\mathsf{Y}|}$, initial classifier weights $\boldsymbol{\theta}_0 \sim \mathbb{R}^P$, initial pre-constraint occlusion logits $\tilde{\boldsymbol{\eta}}_0 \in \mathbb{R}^T$, occlusion budget $\overline{\gamma} \in (0,1)$

Output: Per-timestep 'leakiness' estimate $\boldsymbol{\gamma}^* \in [0,1]^T$

Function `getOcclProbLogits` ($\tilde{\boldsymbol{\eta}} \in \mathbb{R}^T$: pre-constraint logits of occlusion probabilities)
  return $\tilde{\boldsymbol{\eta}} - \operatorname{logsumexp}(\tilde{\boldsymbol{\eta}}) + \log T + \log \overline{\gamma} - \log(1 - \overline{\gamma})$

Function `sampleFromCONCRETE` ($\tilde{\boldsymbol{\gamma}} \in (0,1)^T$: logits of occlusion probabilities)
  return $\operatorname{sigmoid}(\operatorname{logsigmoid}(\tilde{\boldsymbol{\gamma}}) - \operatorname{logsigmoid}(-\tilde{\boldsymbol{\gamma}}) + \log \boldsymbol{u} - \log(\mathbf{1} - \boldsymbol{u})),\ \boldsymbol{u} \sim \mathcal{U}(0,1)^T$

Function `getMaskedCrossEntropy` ($\boldsymbol{\theta} \in \mathbb{R}^P$: classifier weights, $(\boldsymbol{x}, y) \in \mathbb{R}^T \times \mathsf{Y}$: input and label, $\boldsymbol{\alpha} \in [0,1]^T$: relaxed input mask)
  return $\operatorname{logsoftmax} \tilde{\Phi}_{\boldsymbol{\theta}}(y \mid (\mathbf{1} - \boldsymbol{\alpha}) \odot \boldsymbol{x} + \boldsymbol{\alpha} \odot \boldsymbol{\varepsilon}, \mathbf{1} - \boldsymbol{\alpha}),\ \boldsymbol{\varepsilon} \sim \mathcal{N}(0,1)^T$

for $t = 0, 1, \ldots$ until convergence **do**
  $(\boldsymbol{x}_t, y_t) \leftarrow$ `sampleDatapoint`(D)
  $\tilde{\boldsymbol{\gamma}}_t \leftarrow$ `getOcclProbLogits`$(\tilde{\boldsymbol{\eta}}_t)$
  $\boldsymbol{\alpha}_t \leftarrow$ `sampleFromCONCRETE`$(\tilde{\boldsymbol{\gamma}}_t)$
  $\ell_t \leftarrow$ `getMaskedCrossEntropy`$(\boldsymbol{\theta}_t, (\boldsymbol{x}_t, y_t), \boldsymbol{\alpha}_t)$
  $\boldsymbol{g}_t^{\theta} \leftarrow \nabla_{\boldsymbol{\theta}} \ell_t,\ \ \boldsymbol{g}_t^{\tilde{\eta}} \leftarrow -\nabla_{\tilde{\boldsymbol{\eta}}} \ell_t,\ \ \boldsymbol{\theta}_{t+1} \leftarrow$ `OptStep`$(\boldsymbol{\theta}_t, \boldsymbol{g}_t^{\theta}),\ \ \tilde{\boldsymbol{\eta}}_{t+1} \leftarrow$ `OptStep`$(\tilde{\boldsymbol{\eta}}_t, \boldsymbol{g}_t^{\tilde{\eta}})$

return $\operatorname{sigmoid}$ `getOcclProbLogits`$(\tilde{\boldsymbol{\gamma}}_{t+1})$

E Extended experimental details and results

E.1 Toy setting where our method succeeds and prior work fails

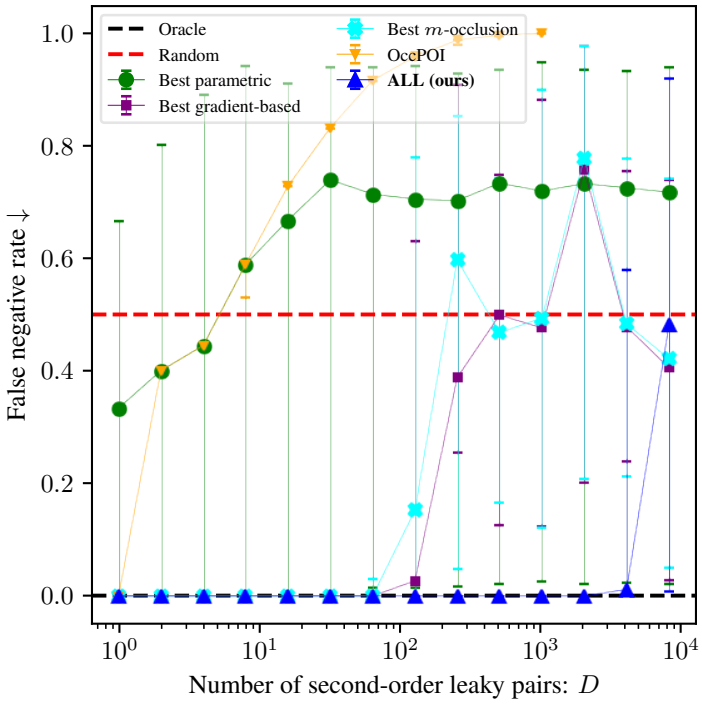

Figure 9: A toy setting where ALL (ours) significantly outperforms baselines. We sample 1 non-leaky feature and D second-order leaky pairs, then plot the false negative rate, defined as the proportion of points incorrectly assigned leakiness less than or equal to that of the non-leaky point, as we increase D. ALL (ours) succeeds for D up to 64× higher the best prior deep learning-based approach, and the first-order parametric methods completely fail in this setting. Dots denote median and error bars denote min–max over 5 random seeds.

As previously discussed, first-order parametric statistics-based methods are insensitive to associations of order 2 or higher. Prior deep learning-based leakage localization algorithms tend to exploit few of the available 2^{T-1} associations between X_t and Y given subsets of $\{X_1, \ldots, X_T\} \setminus \{X_t\}$, with many of them using only the maximal conditioning set $\{X_1, \ldots, X_T\} \setminus \{X_t\}$ itself. This creates issues when there is a large number of leaky measurements and the individual contribution of each is 'drowned out' in the sense that $\mathbb{I}[Y; X_t \mid \{X_1, \ldots, X_T\} \setminus \{X_t\}]$ vanishes. Here we construct a simple setting where both of these issues are present, and demonstrate that ALL succeeds whereas the prior approaches face issues.

We generate a sequence of binary-label $2D + 1$-feature classification datasets consisting of ordered pairs $(\boldsymbol{X}, Y)$ sampled independently as follows:

$$Y \sim \mathcal{U}\{0,1\},\ R \sim \mathcal{U}\{0,1\},\ M_i \sim \mathcal{U}\{0,1\} : i = 1, \ldots, D, \tag{58}$$

$$X_R \sim \mathcal{N}(R, 1),\ X_{M_i} \sim \mathcal{N}(M_i, 1),\ X_{Y \oplus M_i} \sim \mathcal{N}(Y \oplus M_i, 1) : i = 1, \ldots, D. \tag{59}$$

Here we denote by $\oplus$ the exclusive-or operation and $\boldsymbol{X} \equiv (X_R, X_{M_1}, X_{Y \oplus M_1}, \ldots X_{M_D}, X_{Y \oplus M_D})$. Intuitively, we can view X_R as a noisy observation of R and each X_{M_i}, $X_{Y \oplus M_i}$ as noisy observations of M_i, $Y \oplus M_i$, respectively. Here the variable Y is analogous to a targeted sensitive variable, and the values $(M_i, Y \oplus M_i)$ are analogous to the pairs of second-order leaky variables which arise in Boolean masked implementations such as Benadjila et al. (2020).

Clearly R, and thus X_R, tells us nothing about Y. Additionally, the values M_i *in isolation* tell us nothing about Y. Similarly, the values $Y \oplus M_i$ in isolation tell us nothing about Y because

$$\mathbb{P}(Y = 0 \mid Y \oplus M_i = 0) = \mathbb{P}(M_i = 0) = \tfrac{1}{2} = \mathbb{P}(M_i = 1) = \mathbb{P}(Y = 1 \mid Y \oplus M_i = 0) \tag{60}$$

and similarly

$$\mathbb{P}(Y = 0 \mid Y \oplus M_i = 1) = \mathbb{P}(M_i = 1) = \tfrac{1}{2} = \mathbb{P}(M_i = 0) = \mathbb{P}(Y = 1 \mid Y \oplus M_i = 1). \tag{61}$$

Despite this, given the *pair* of values $\{M_i, Y \oplus M_i\}$ we can recover Y via the identity

$$M_i \oplus M_i = 0 \implies (Y \oplus M_i) \oplus M_i = Y. \tag{62}$$

Thus, we would like a leakage localization algorithm to indicate that X_R is non-leaky and the values X_{M_i}, $X_{Y \oplus M_i}$ are leaky.

All experiments use the hyperparameters shown in Table 4.

Table 4: Hyperparameters used for our toy Gaussian dataset experiments. We use the default PyTorch settings except where specified. †These hyperparameters apply only to ALL. Other hyperparameters are used both for ALL and for our baseline methods.

Hyperparameter	**Value**
Classifier architecture	ReLU MLP with 1×500-neuron hidden layer
Classifier optimizer	AdamW
Classifier learning rate	10^{-4}
Classifier weight decay	0.01
Dataset size	10k
Training steps	5k
Minibatch size	800
Noise distribution learning rate†	10^{-3}
Budget $\overline{\gamma}$†	$1 - 2^{-0.1 \cdot D - 1}$

Results can be seen in Fig. 9. We measure the performance of methods by the percent of measurements in $\{X_{M_i} : i = 1, \ldots, D\} \cup \{X_{Y \oplus M_i} : i = 1, \ldots, D\}$ assigned a leakiness greater than or equal to that of X_R. For clarity we report for each D the best result out of SNR, SOSD and CPA as 'best parametric', out of GradVis, Saliency, Input $*$ Grad, and LRP as 'best gradient-based', and the best result out of $\{2m + 1 : m \in [0 \,..\, 24], m < 2D + 1\}$-occlusion as 'best m-occlusion'. Due to their high cost we use OccPOI only for D up to 256, and we do not use second-order occlusion. Note that in subsequent experiments second-order occlusion does not significantly outperform first-order occlusion, and OccPOI performs poorly due to identifying only a small number of leaky measurements rather than assigning a leakiness to every measurement.

E.2 Simulated AES datasets where we have ground truth knowledge about leakage

Here we present experiments done on synthetic AES power traces. These are a useful complement to the experiments on real datasets because 1) here we have ground truth knowledge about which timesteps are leaking, which we can use to validate our model's output, 2) we can generate infinitely-large datasets to eliminate dataset size as a confounding variable in results, and 3) we can observe the change in our technique's behavior as we individually vary particular dataset properties such as low-pass filtering strength and leaky instruction count.

Algorithm 3: Pseudocode for our synthetic data generation procedure, based on the Hamming weight leakage model of Mangard et al. (2007). For clarity we omit the random delay and shuffling procedures, but these are straightforward and may be found in our code.

Input: Dataset size $N \in \mathbb{Z}_{++}$,
Timesteps per power trace $T \in \mathbb{Z}_{++}$,
Bit count $n_{\text{bits}} \in \mathbb{Z}_{++}$,
Operation count $n_{\text{ops}} \in \mathbb{Z}_{++}$,
Data-dependent noise variance $\sigma^2_{\text{data}} \in \mathbb{R}_+$,
Operation-dependent noise variance $\sigma^2_{\text{op}} \in \mathbb{R}_+$,
Residual noise variance $\sigma^2_{\text{resid}} \in \mathbb{R}_+$,
Low-pass filtering strength $\beta \in [0, 1)$,
Leaking timestep count $n_{\text{lkg}} \in \mathbb{Z}_+$

Output: Synthetic dataset $\mathsf{D} \subset \mathbb{R}^T \times [0 \,..\, 2^{n_{\text{bits}}} - 1]$

$\{k^{(n)} : n \in [1 \,..\, N]\} \sim \mathcal{U}\left(\{0,1\}^{n_{\text{bits}}}\right)^N$ `// AES keys`

$\{w^{(n)} : n \in [1 \,..\, N]\} \sim \mathcal{U}\left(\{0,1\}^{n_{\text{bits}}}\right)^N$ `// plaintexts`

$\{o_t : t \in [1 \,..\, T]\} \sim \mathcal{U}\left([1 \,..\, n_{\text{ops}}]\right)^N$ `// operations`

$\{\tilde{x}_{\text{op},\,i} : i \in [1 \,..\, n_{\text{ops}}]\} \sim \mathcal{N}(0, \sigma^2_{\text{op}})^{n_{\text{ops}}}$ `// operation-dependent power consumption`

$\mathsf{T}_{\text{lkg}} \sim \mathcal{U}\begin{pmatrix}[1 \,..\, T] \\ n_{\text{lkg}}\end{pmatrix}$ `// timesteps of leaky instructions`

for $n \in [1 \,..\, N]$ **do**

  $y^{(n)} \leftarrow \text{Sbox}(k^{(n)} \oplus w^{(n)})$ `// targeted variable: first SubBytes output`

  $\boldsymbol{x}^{(n)}_{\text{resid}} \sim \mathcal{N}(0, \sigma^2_{\text{resid}})^T$ `// residual power consumption`

  for $t \in \mathsf{T}_{\text{lkg}}$ **do**

    $d^{(n)}_t \leftarrow y^{(n)}$ `// leaky timesteps: targeted variable is the data`

  for $t \in [1 \,..\, T] \setminus \mathsf{T}_{\text{lkg}}$ **do**

    $d^{(n)}_t \sim \mathcal{U}\left(\{0,1\}^{n_{\text{bits}}}\right)$ `// rest of data treated as random`

  for $t \in [1 \,..\, T]$ **do**

    $x^{(n)}_{\text{data},\,t} \leftarrow \sigma_{\text{data}}\left(4 - \text{HW}(d^{(n)}_t)\right)/\sqrt{2}$ `// data-dependent power consumption`

  $\boldsymbol{x}^{(n)} \leftarrow \boldsymbol{x}^{(n)}_{\text{data}} + \boldsymbol{x}_{\text{op}} + \boldsymbol{x}^{(n)}_{\text{resid}}$ `// total power consumption`

  for $t \in [2 \,..\, T]$ **do**

    $x^{(n)}_t \leftarrow \beta x^{(n)}_{t-1} + (1-\beta) x^{(n)}_t$ `// Discrete low-pass filtering of` $\boldsymbol{x}^{(n)}$.

    `// In practice we prepend 'burn-in' timesteps to allow transient effects to decay.`

return $\left\{(\boldsymbol{x}^{(n)}, y^{(n)}) : n \in [1 \,..\, N]\right\}$

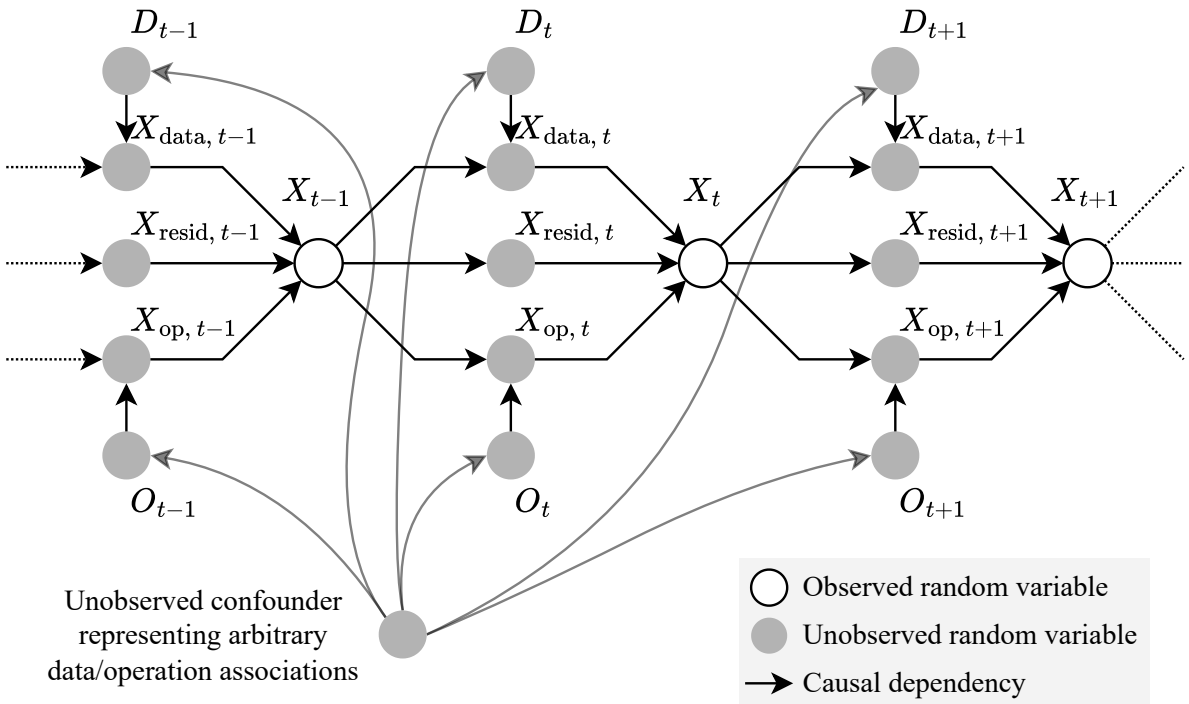

Figure 10: Causal diagram representing the assumed data-generating process for our synthetic AES datasets. We assume this is an I-map for the data-generating process – i.e. we assume independence conditions present in the diagram hold for the data-generating process, but the data-generating process might have additional independence conditions not reflected here. This is an effort to make more precise the Hamming weight leakage model of (Mangard et al., 2007, ch. 4). We represent the data of the AES algorithm by the random variables D_t, its operations O_t, and its power consumption X_t, which is further broken down into data-dependent power consumption $X_{\text{data},\, t}$, operation-dependent power consumption $X_{\text{op},\, t}$, and 'residual' power consumption $X_{\text{resid},\, t}$ (e.g. due to random noise, other processes running in parallel, etc.). We assume that only the composite power consumption X_t is observed.

E.2.1 Data generation procedure

We base our synthetic data generation procedure on the Hamming weight leakage model of (Mangard et al., 2007, ch. 4), which we will subsequently describe.[4] As above, let us represent our power/EM traces as a random vector $\boldsymbol{X} \coloneqq (X_t : t = 1, \ldots, T)$ with range $\mathbb{R}^T$. We represent the cryptographic algorithm as sequences of data $\boldsymbol{D} \coloneqq (D_t : t = 1, \ldots, T)$ and operations $\boldsymbol{O} \coloneqq (O_t : t = 1, \ldots, T)$, where each D_t has range $\{0, 1\}^{n_{\text{bits}}}$ and each O_t has range $[1 \,..\, n_{\text{ops}}]$ for some $n_{\text{bits}}, n_{\text{ops}} \in \mathbb{Z}_{++}$. For each $t \in [1 \,..\, T]$, we can decompose

$$X_t = X_{\text{data},\,t} + X_{\text{op},\,t} + X_{\text{resid},\,t} \tag{63}$$

with dependency structure illustrated in the causal diagram of Fig. 10. Here we have represented by $X_{\text{data},\,t}$ the data-dependent component of power consumption, $X_{\text{op},\,t}$ the operation-dependent component of power consumption, and $X_{\text{resid},\,t}$ the 'residual' power consumption due to random noise, other processes running simultaneously on the same hardware, etc.

Mangard et al. (2007) experimentally characterized the power consumption of a cryptographic device and found that it is reasonable to approximate $X_{\text{data},\,t}$ as Gaussian noise with D_t-dependent mean, $X_{\text{op},\,t}$ as Gaussian noise with O_t-dependent mean, and $X_{\text{resid},\,t}$ as Gaussian noise with constant mean (which we will assume to be zero because it contains no information and is thus irrelevant). For their device, they found that the mean of $X_{\text{data},\,t}$ was roughly proportional to $n_{\text{bits}} - \text{HW}(D_t)$ where $\text{HW} : \{0, 1\}^{n_{\text{bits}}} \to [0 \,..\, n_{\text{bits}}] : \boldsymbol{x} \mapsto \sum_{k=1}^{n_{\text{bits}}} x_k$. We adopt these approximations for our synthetic dataset experiments. See Alg. 3 for pseudocode giving a simplified version of our data generation procedure, and refer to our code for full details.

We simulate several factors of variation which can reasonably be expected to occur in realistic settings. We apply discrete-time low-pass filtering to the power traces, to simulate the low-pass filtering which occurs due to measurement apparatus as well as the fact that power consumption does not change instantaneously in real circuits. We allow for the presence of multiple leaky instructions. Additionally, we simulate random delays to the leaky instruction which might result from countermeasures such as random no-op insertion (Coron & Kizhvatov, 2009), and random 'shuffling' where the leaky instruction is randomly placed at one of several points in time, as done in Masure & Strullu (2023).

We emphasize that these approximations are specific to the device studied by Mangard et al. (2007) and may hold to a limited extent or not at all for other devices. For example, the Hamming weight dependence of power consumption stems from the fact that their device 'pre-charges' all its data bus lines to 1, then drains the charge from the lines which should represent 0, thereby consuming power proportional to the number of lines which represent 0. Many devices operate differently, and cryptographic hardware is often designed with the explicit goal of obfuscating this data/power consumption dependence. Thus, while we expect all real cryptographic devices to have some exploitable dependence between data and power consumption given sufficient quality and quantity of data, in many cases the nature of the dependence will likely elude a simple characterization such as this.

E.2.2 Experimental details

We run experiments on many variations of this dataset and verify that ALL produces outputs which align with our expectations. For all these experiments, our classifier $\Phi_{\boldsymbol{\theta}}$ is a 3-layer multilayer perceptron with hidden dimension 500, ReLU activations, an input dropout rate of 0.1, hidden dropout rate of 0.2, and pre-logits dropout rate of 0.3. We generate data continuously so that our dataset size is effectively infinite. Additional hyperparameters are listed in table 5, and default settings corresponding to Alg. 3 are listed in table 6, which we use except where otherwise stated.

E.2.3 Results

We vary 4 parameters of the data generation process of Alg. 3 and observe its effect on the output of ALL. We find that ALL consistently produces results consistent with our expectations given the timestep(s) of

[4]For clarity we alter the notation and explicitly define a causal structure for the data-generating process. The decomposition of power consumption in Eqn. 63 and the definition of $X_{\text{data},\,t}$ in terms of the Hamming weight of the data are based on Mangard et al. (2007), but the additional details are our own.

Table 5: List of hyperparameters used for experiments on synthetic AES datasets. We use the default PyTorch settings unless otherwise stated. Note that in general for ALL classfiers we disable input dropout, but for these experiments the dropout rate was set to 0.1 due to an oversight.

Hyperparameter	**Value**
Classifier architecture	ReLU MLP with 3×500-neuron hidden layers
Input dropout	0.1
Hidden dropout	0.2
Output dropout	0.3
Optimizer for both $\boldsymbol{\theta}$ and $\tilde{\boldsymbol{\eta}}$	`torch.optim.AdamW`
Weight decay for $\boldsymbol{\theta}$	0.01 for weights, 0 for biases
Weight decay for $\tilde{\boldsymbol{\eta}}$	0
Training steps	10^4
Minibatch size	10^3
Noise budget $\overline{\gamma}$	0.5
Weight initializer	`torch.nn.init.xavier_uniform_`

Table 6: Default synthetic AES dataset configuration, corresponding to the inputs of Alg. 3. Subsequent experiments will use these settings unless otherwise stated.

Setting	**Value**
Dataset size N	∞
Timesteps per power trace T	101
Bit count n_{bits}	8
Operation count n_{op}	32
Data-dependent noise variance σ^2_{data}	1.0
Operation-dependent noise variance σ^2_{op}	1.0
Residual noise variance σ^2_{resid}	1.0
Number of leaky instructions n_{lkg}	1
Low-pass filtering strength β	0.5
Maximum random delay size	0
Possible leaky timestep location count (i.e. shuffling)	1

the leaky instruction(s), and that the variance of its output is quite low in this context where we have an infinitely-large dataset and a long training duration. Here we describe the parameters being swept and justify the output of ALL given this.

Low pass filtering strength β Recall that we are discrete low-pass filtering traces via the recursive function $x_t^{\text{lpf}} \coloneqq (1-\beta)x_t + \beta x_{t-1}$. See the first row of Fig. 11, where from left to right β takes on the values 0, 0.5, 0.75, 0.875, 0.9375, 0.96875, 0.984375, 0.9921875. Note that the peak estimated leakiness always corresponds to the ground-truth leaky instruction timestep. As we increase β we see that measurements to the left and right are assigned high leakiness as well, which makes sense for the following reason:

Let us denote by t^* the timestep at which the leaky instruction was executed. We then have $X_{t^*} = c_1 \operatorname{HW}(Y) + c_2 U_{t^*} + c_3 X_{t^*-1}$ where U_{t^*} denotes a 'residual' random variable independent of Y, and c_1, c_2 are appropriate constants. Note that $X_{t^*+1} = \beta X_{t^*} + (1-\beta)U_{t^*+1} = \beta c_1 \operatorname{HW}(Y) + \beta c_2 U_{t^*} + (1-\beta)U_{t^*+1}$, so X_{t^*+1} also leaks Y. Recursively it is clear that the same can be said for X_{t^*+2}, $X_{t^*+3}, \ldots$. Less-intuitively, X_{t^*-1} also leaks Y. This is because although X_{t^*-1} is *marginally* independent of Y, $X_{t^*} - c_3 X_{t^*-1}$ has a higher correlation with $\operatorname{HW}(Y)$ than X_{t^*} does – i.e. X_{t^*-1} is dependent on the Y-independent noise of X_{t^*}, and can be used to reduce the noise. Recursively, since $X_{t^*-2}, X_{t^*-3}, \ldots$ are correlated with X_{t^*-1}, they also leak Y by the same mechanism.

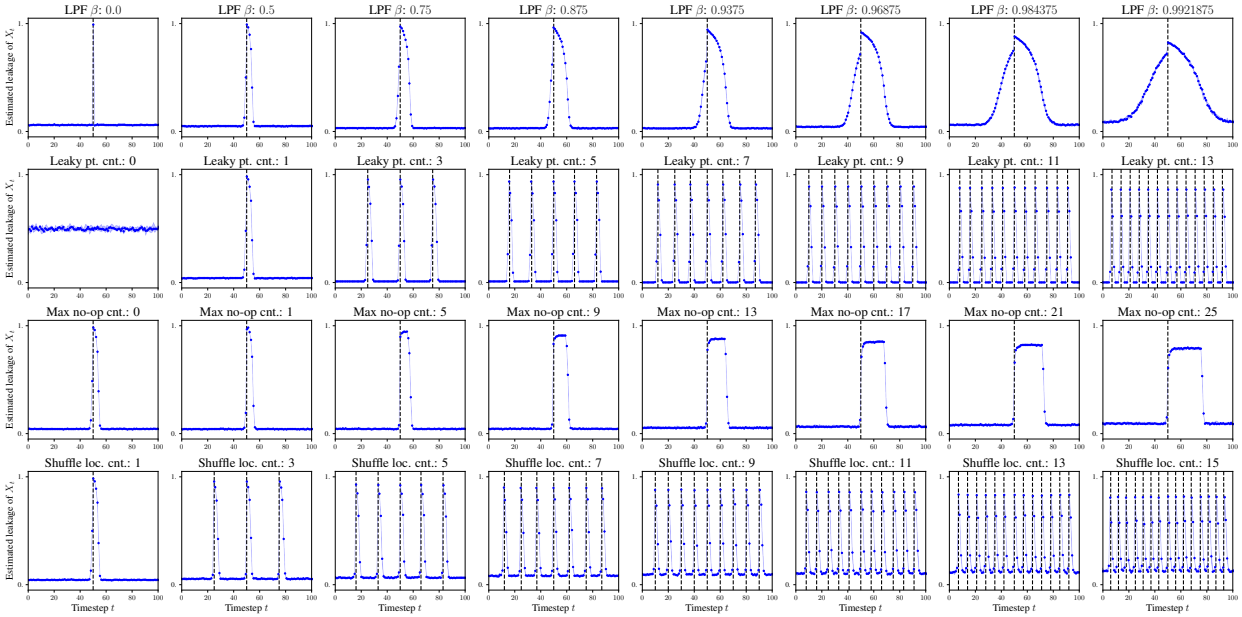

Figure 11: Results of applying ALL (ours) to synthetic AES datasets with varying parameters. The estimated leakage by ALL is denoted by blue dots and the ground truth timestep of the leaking instruction is denoted by the vertical black dotted lines. Blue dots denote mean and shading denotes median over 5 random seeds. Note that the ALL output is consistent with the ground truth leaky instruction timestep in all cases, and the variance between runs is quite low in the infinite data regime. **(first row)** Increasing low pass filtering strength β from left to right. **(second row)** Increasing number of leaky instructions from left to right. **(third row)** Increasing random delay insertion from left to right. **(fourth row)** Increasing number of possible shuffling locations for the leaky point from left to right.

Leaky point count Here we sweep the number of leaky instructions. See the second row of Fig. 11 where from left to right the leaky point count n_{lkg} takes on the values 0, 1, 3, 5, 7, 9, 11, 13. As expected, all leaky instruction timesteps correspond to a peak of ALL-estimated leakiness.

Random delay size Here we insert random delays – i.e. instead of always occurring at time t, the leaky instruction occurs at $t + u$ where $u \sim \mathcal{U}\{0, \ldots, d_{\max}$. See the third row of Fig. 11 where from left to right $d_{\max}$ takes on values 0, 1, 5, 9, 13, 17, 21, 25. We see that the estimated leakiness becomes 'spread out' over the interval $[t \,..\, t + d_{\max}]$.

Shuffle location count Here we randomly 'shuffle' the leaky instruction – i.e. instead of always occurring at time t, the leaky instruction occurs at $t' \sim \mathcal{U}\{t_1, \ldots, t_{n_{\text{shuff}}}\}$. See the fourth row of Fig. 6 where from left to right n_{shuff} takes on values 1, 3, 5, 7, 9, 11, 13, 15. We see that the leakiness becomes 'spread out' over the set of timesteps at which the leaky instruction might occur.

E.3 Experiments on real power and EM radiation leakage datasets

Here we run experiments on a variety of publicly-available side-channel attack datasets, where we attempt to localize leakage of their canonical target variable.

E.3.1 Datasets

We compare ALL with prior work on the 6 datasets described in Table 7, which consist of traces of power and EM radiation measurements and associated cryptographic variables recorded from real implementations. Note that we evaluate on AES, ECC and RSA implementations implemented on several MCUs and an FPGA

Table 7: A list of the datasets used in our paper, with a summary of their salient attributes. Note that our experiments cover a variety of settings: AES, RSA and ECC implementations on both microcontrollers (MCUs) and a field-programmable gate array (FPGA), both power and EM radiation traces, and various types of countermeasures. For all datasets we localize leakage of the canonical target variable. We denote by subscripts the targeted byte of the variable. We denote by k_n, w_n, m_n, k^*_n, c_n the n-th byte (counting from 0) of the AES key, plaintext, mask, last round key, and ciphertext, respectively. †As described below, we deviate from the canonical profiling/attack split.

Dataset	ASCADv1 (fixed key)	ASCADv1 (variable key)	DPAv4 (Zaid version)
Citation	Benadjila et al. (2020)	Benadjila et al. (2020)	Bhasin et al. (2014); Zaid et al. (2020)
Link	(here)	(here)	(here)
Algorithm	AES-128	AES-128	AES-128
Hardware	ATMega8515 (MCU)	ATMega8515 (MCU)	ATMega163 (MCU)
Emission measured	Power	Power	Power
Countermeasures	Boolean masking	Boolean masking	Rotating Sbox mask (known)
Targeted variable	$\text{Sbox}(k_3 \oplus w_3)$	$\text{Sbox}(k_3 \oplus w_3)$	$\text{Sbox}(k_0 \oplus w_0)$
Dataset size (profile/attack)	50k/10k	200k/100k	3k/500†
Feature count T	0.7k	1.4k	4k
Dataset	AES-HD	One Trace is All it Takes (OTiAiT)	One Truth Prevails (OTP) (1024-bit)
Citation	Bhasin et al. (2020)	Weissbart et al. (2019)	Saito et al. (2022)
Link	(here)	(here)	(here)
Algorithm	AES-128	EdDSA w/ Curve2559	1024-bit RSA-CRT
Hardware	XiLinx Virtex-5 (FPGA)	STM32F4 (MCU)	STM32F4 (MCU)
Emission measured	EM radiation	Power	EM radiation
Countermeasures	None	None	Dummy load
Targeted variable	$\text{Sbox}^{-1}(k^*_{11} \oplus c_{11}) \oplus c_7$	Ephemeral key nibble	Dummy load?
Dataset size (profile/attack)	50k/25k	5.12k/1.28k	100k/98.304k†
Feature count T	1.25k	1k	1k

with various target variables. ALL as well as most of our baseline algorithms are in principle agnostic to most of these details, requiring only a supervised learning-style dataset with power traces and the associated value of the targeted variable as labels. Through these experiments we demonstrate that this is true in practice across a diverse array of settings.

Note that the ASCADv1 datasets have primarily second-order leakage due to their Boolean masking countermeasure, whereas the other 4 datasets have primarily first-order leakage. Our comparisons include both deep learning methods as well as simple first-order parametric methods which are widely used due to their low cost and interpretability. We find that the latter are competitive or superior to the deep learning methods on the first-order datasets but perform significantly worse on the second-order datasets due to failing to exploit second-order leakage. Our experiments do not compellingly show that deep learning methods improve on these simpler methods for first-order datasets, but we nonetheless include them as additional points of comparison between the deep learning methods and to show that ALL works in a variety of settings.

In general we use the canonical target variable and profiling/attack dataset split (note that in the context of profiling side-channel analysis the training dataset is called the profiling dataset, and the test dataset is called the attack dataset). We deviate from the canonical dataset configuration in the following cases:

- We find that the canonical attack dataset of DPAv4 is too small to compute useful oracle leakiness assessments. We thus concatenate the canonical 4.5k-trace profiling dataset and 0.5k-trace attack dataset into a single 5k-length dataset, and use the first 3k traces for profiling and the last 2k to compute the oracle assessments. Since some of our experiments require metadata which is only available for the attack dataset, for everything other than oracle assessment computation we use the canonical attack dataset and leave the remaining 1.5k traces unused.

- We use the version of DPAv4 which was preprocessed and distributed by Zaid et al. (2020) here rather than the original version. This version has shortened traces which have been cropped around the leaky instruction, and has the rotating Sbox mask effectively 'disabled' by providing the masked SubBytes variable as the target.

- The One Truth Prevails (OTP) dataset consists of approximately 64M traces and has a high label imbalance. To save computational resources we extract a 100k-trace randomly-selected balanced subset, which we find is more than sufficient for strong supervised classification and leakage localization performance. See our code for details.

E.3.2 Implementation details for the leakage localization algorithms

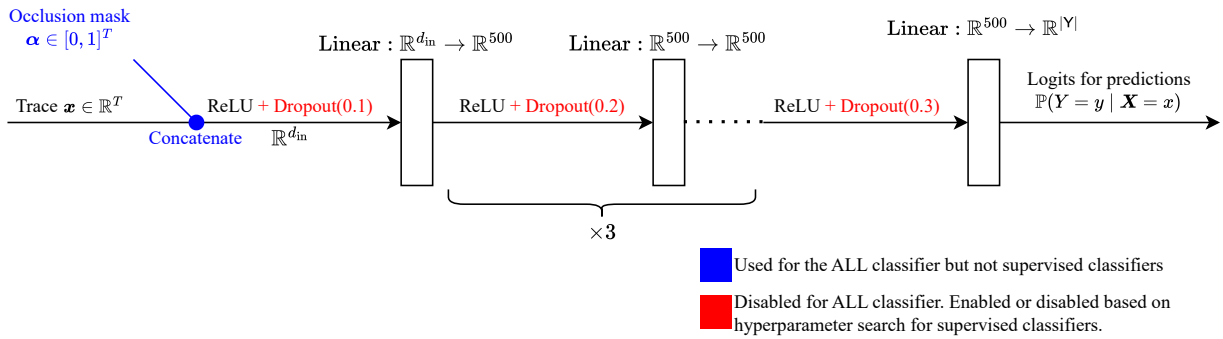

Figure 12: Diagram of the multilayer perceptron architecture used for classifiers in the deep learning methods, based on the architecture proposed in Wang et al. (2017) for time-series classification.

For ALL and all considered baseline methods the classifier uses the simple ReLU + Dropout MLP architecture of Wang et al. (2017) shown in Fig. 12, which has 3 500-neuron hidden layers, input dropout rate of 0.1, hidden dropout rate of 0.2, and output dropout rate of 0.3. For the deep learning baselines we enable or disable the input, hidden and output dropout based on our hyperparameter search outcome, and for ALL we leave it disabled. The classifier for ALL takes the occlusion mask as an auxiliary input, which we implement by concatenating it with the masked trace.

We also explored convolutional architectures, but preliminary experiments indicated that these achieved weaker classification and leakage localization performance across the board, as well as training more-slowly than the MLP architecture due to a higher layer count. We suspect that the inductive biases of convolutional layers are not useful for the datasets we consider. As a sanity check for this design choice, we run the deep learning baseline methods using both our MLP architecture and a handful of open-weight classifiers which were released with Wouters et al. (2020) on the datasets for which they are available.

All deep learning methods are implemented in PyTorch (Paszke et al., 2019). Most non-deep methods are implemented with Numpy (Harris et al., 2020), with a handful of compute-intensive methods implemented with Numba (Lam et al., 2015). We use Scipy (Virtanen et al., 2020) implementations of statistical methods where available.

We use the AdamW optimizer (Loshchilov & Hutter, 2018; Kingma & Welling, 2014) with the default PyTorch settings $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\lambda = 0.01$, $\epsilon = 10^{-8}$ and the learning rate chosen through hyperparameter search. Weight decay is applied only to the weights of the linear layers, not to the biases. We use a minibatch size of 256. Weights are initialized with the uniform Glorot initialization (Glorot & Bengio, 2010) `torch.nn.init.xavier_uniform_` (default for Keras) rather than the default PyTorch initialization, and we find that this is a critical detail – many supervised classifier runs on the ASCADv1 datasets completely fail to generalize beyond the training dataset when the default PyTorch weight initialization is used. We randomly set aside 20% of the profiling datasets for validation and use the remaining 80% for training. We standardize the traces as $\boldsymbol{x} \mapsto \frac{\boldsymbol{x}-\boldsymbol{\mu}}{\max(\boldsymbol{\sigma}, 10^{-6})}$ where μ and $\boldsymbol{\sigma}$ denote the elementwise sample mean and standard deviation computed using the profiling dataset.

In general we measure the performance of supervised classifiers with their *mean rank* rather than their accuracy, as accuracy tends to be low and too-coarse in the context of side-channel analysis. Given a label $y \in \mathsf{Y}$ and predicted label distribution $\hat{p}_Y \in \Delta^{|\mathsf{Y}|-1}$ (e.g. the softmaxed output of a classifier neural net), we define the rank as the number of possible labels assigned at least as much probability mass as the true label,

i.e.

$$\mathrm{Rank}(\hat{p}_Y; y) \coloneqq \left|\{y' \in \mathsf{Y} : \hat{p}_Y(y') \geq \hat{p}_Y(y)\}\right| . \tag{64}$$

This metric has range $[1 \,..\, |\mathsf{Y}|]$, with lower being better.

Implementation of the baseline methods We use Captum (Kokhlikyan et al., 2020) implementations of the saliency, input $*$ grad, LRP, and m-occlusion methods. We implement SNR, SOSD, CPA, GradVis, and second-order m-occlusion ourselves, and we implement a PyTorch version of OccPOI based on the Keras implementation released by the authors here.

Note the following choices we have made in implementing and evaluating OccPOI (Yap et al., 2025):

- OccPOI differs from the other methods we consider in that rather than assigning a leakiness value to every measurement, it aims to identify a non-unique subset of measurements which are sufficient for a classifier to attain some specified performance level when all other measurements are occluded. Since our evaluation metrics require a leakiness value for every measurement, we assign a leakiness of 0 to measurements not identified by OccPOI.
- Their method uses the attack dataset for probing the classifier's sensitivity to input features, which introduces data contamination in the context of our performance metrics. We cannot easily use the validation dataset instead because the attack procedure requires having many traces corresponding to a fixed AES key, which is generally only available for the attack dataset. We find that ALL significantly outperforms OccPOI despite this contamination, so as the aim of our work is to demonstrate the efficacy of ALL, we allow OccPOI to use the attack dataset.
- Since OTiAiT and OTP do not have attack datasets which facilitate this kind of multi-trace prediction, for these datasets we simply use the mean rank of the classifier on the attack dataset as our performance metric.
- Yap et al. (2025) proposes an extension of OccPOI which ranks the leakiness of the points it has identified using a 1-occlusion-like strategy. We use this extension in our implementation.
- Yap et al. (2025) proposes an extension of OccPOI where they apply it repeatedly on the residual measurements not selected during the last iteration. We do not use this extension because it is very computationally expensive, requiring $O(T)$ applications of OccPOI which each require $\Omega(T)$ *non-parallelizable* passes through the attack dataset. In preliminary experiments this extension performed better than basic OccPOI, but still far below the performance of the other considered methods despite requiring orders of magnitude more wall clock time.
- Similarly to Yap et al. (2025), to save compute we use only a subset of the attack datasets for classifier evaluation. We use 1.8k traces for ASCADv1-fixed, 2.8k for ASCADv1-variable, 140 for DPAv4, 25k for AES-HD, 100 for OTiAiT, and 100 for OTP. These are approximately 10–100$\times$ the necessary number of traces to successfully attack the AES methods.
- Yap et al. (2025) define their performance threshold to be that the classifier correctly predicts the AES key after accumulating all traces in the attack dataset, and we use the same threshold for the ASCADv1 datasets, AES-HD, and DPAv4. For the non-AES dataset OTiAiT and OTP, our threshold is that the mean rank of the classifier rises by 0.1 relative to its mean rank when no measurements are occluded.

As we will show, OccPOI attains significantly lower performance than other methods according to our performance metrics, due to not assigning a leakiness to most measurements. This reflects that the aim of Yap et al. (2025) is somewhat different from the present work: Yap et al. (2025) heavily emphasized the usefulness of OccPOI as a feature selection tool with leakage localization being an auxiliary goal, whereas our work is concerned solely with leakage localization. We consider OccPOI as a baseline because it is similar in spirit to our work, but these results demonstrate that it is ill-suited to the task considered in our paper.

We make the following choices in implementing m-occlusion and m-second-order occlusion (which we call m-occlusion^2 for brevity) (Schamberger et al., 2023):

- Schamberger et al. (2023) explore different ways to occlude measurements and conclude that it works well to replace measurements by their average value over the profiling dataset, whereas other heuristics such as replacing them with 0 or with Gaussian noise works poorly. We thus replace measurements by their mean. Since we are element-wise standardizing the traces, this is the same as replacing them by 0.
- Schamberger et al. (2023) propose second-order occlusion as a means of estimating the leakiness of *pairs* of measurements, which is useful for discerning whether a measurement has first-order leakage or is part of a second-order leaking pair of measurements. They do not propose a means of using it to estimate the leakiness of individual measurements. We choose to define the leakiness of X_t as the *average* leakiness of the pairs $\{\{X_t, X_{t'}\} : t' \in [1 \,..\, T]\}$.
- Schamberger et al. (2023) uses a large stride for second-order occlusion, and thus get leakiness values for windows of measurements rather than single measurements. We set the stride to 1 to make it more suitable for our setting.
- These methods introduce the occlusion window size m as a new hyperparameter which must be tuned. For m-occlusion we tune m by testing successive odd-numbered window sizes starting from 1 until oracle agreement performance starts decreasing, and using the window size which maximizes oracle agreement. Note that this introduces some data contamination, which we accept because our aim is to demonstrate the efficacy of ALL, and ALL outperforms m-occlusion despite the contamination.
- For m-occlusion^2 we use the optimal value of m for m-occlusion. We don't do another sweep because m-occlusion^2 is very computationally-expensive, requiring $\Theta(T^2)$ passes through the dataset.
- Like Yap et al. (2025), Schamberger et al. (2023) uses the attack dataset to evaluate the sensitivity of the classifier to occluding measurements. To avoid data contamination, and for compatibility with the OTiAiT and OTP datasets which do not have the same kind of attack dataset as the AES implementations, we track the average change in logits over the profiling dataset as we occlude measurements.

E.3.3 Hyperparameter tuning procedure

Table 8: Outcome of a 50-trial random hyperparameter search for the supervised classification models used by the deep learning baseline methods. All trials are early stopped at the point of lowest validation rank, and we choose the hyperparameter configuration which minimizes the lowest validation rank, with ties broken based on validation loss. Models are trained with AdamW and weight decay applied only to the weights of dense and conv1d layers. We use the default PyTorch settings everywhere unless otherwise stated.

Hyperparameter	**Search space**	**Selected value**					
		ASCADv1 (fixed)	ASCADv1 (variable)	DPAv4 (Zaid)	AES-HD	OTiAiT	OTP
Learning rate	$\bigcup_{m=1}^{9}\bigcup_{n=3}^{5}\{m \cdot 10^{-n}\}$	$3 \cdot 10^{-4}$	$2 \cdot 10^{-4}$	$4 \cdot 10^{-3}$	$9 \cdot 10^{-5}$	$8 \cdot 10^{-3}$	$6 \cdot 10^{-3}$
LR schedule	{constant, cos annealing}	cos annealing	cos annealing	constant	cos annealing	cos annealing	cos annealing
Input dropout	$\{0.0, 0.1\}$	0.0	0.0	0.0	0.0	0.0	0.1
Hidden dropout	$\{0.0, 0.2\}$	0.2	0.2	0.2	0.2	0.2	0.0
Output dropout	$\{0.0, 0.3\}$	0.3	0.0	0.0	0.3	0.3	0.0
Training steps	n/a	20k	40k	10k	20k	1k	1k
Classification performance of chosen model							
Test rank ↓		101 ± 1	79.1 ± 0.2	5.6 ± 0.7	124.7 ± 0.2	1.010 ± 0.007	1.00171 ± 0.00007
Test loss ↓		5.59 ± 0.04	5.380 ± 0.009	3.0 ± 0.2	5.65 ± 0.04	0.10 ± 0.08	0.0085 ± 0.0006
Traces to AES key disclosure ↓		179 ± 71	288 ± 126	1.4 ± 0.5	4355 ± 1915	n/a	n/a

All the deep learning methods we consider require hyperparameter tuning. Work in other deep learning subfields such as Gulrajani & Lopez-Paz (2021) has emphasized the importance of a fair hyperparameter tuning process and realistic model selection criterion when comparing the performance of different algorithms, and in this work we aim to follow these recommendations. Accordingly, all methods are tuned with a 50-trial random hyperparameter search. Note that while our main performance evaluation metric is the oracle agreement, it would be unrealistic to use this for model selection, even when computed on a validation dataset, because it relies on 'white box' knowledge about cryptographic implementations that we assume not to have

Table 9: Outcome of a 50-trial random hyperparameter search for adversarial leakage localization. Models are trained with AdamW and weight decay applied only to the weights of the dense and conv1d layers. We use the default PyTorch settings everywhere unless otherwise stated. For ASCADv1 (fixed), ASCADv1 (variable) and AES-HD we find it helpful to 'pretrain' the classifier with fixed $\tilde{\boldsymbol{\eta}}$ before beginning the simultaneous phase of training, but for DPAv4, OTiAiT, and OTP we find this unnecessary.

Hyperparameter	**Search space**	**Selected value**					
		ASCADv1 (fixed)	ASCADv1 (variable)	DPAv4 (Zaid)	AES-HD	OTiAiT	OTP
$\boldsymbol{\theta}$ learning rate (pretrain)	$\bigcup_{m=1}^{10}\{m \cdot 10^{-4}\}$	10^{-4}	10^{-4}	n/a	10^{-3}	n/a	n/a
$\overline{\gamma}$ (pretrain)	n/a	0.5	0.5	n/a	0.5	n/a	n/a
$\boldsymbol{\theta}$ learning rate	$\bigcup_{m=1}^{9}\bigcup_{n=4}^{6}\{m \cdot 10^{-n}\}$	$6 \cdot 10^{-6}$	$7 \cdot 10^{-5}$	$2 \cdot 10^{-5}$	10^{-4}	10^{-4}	$4 \cdot 10^{-4}$
$\frac{\tilde{\boldsymbol{\eta}} \text{ learning rate}}{\boldsymbol{\theta} \text{ learning rate}}$	$\bigcup_{m=1}^{9}\bigcup_{n=0}^{2}\{m \cdot 10^{n}\}$	50	6	9	20	3	3
$\overline{\gamma}$	$\bigcup_{m=1}^{19}\{0.05 \cdot m\}$	0.3	0.4	0.7	0.85	0.8	0.65
Training steps (pretrain)	n/a	10k	20k	0	10k	0	0
Training steps	n/a	10k	20k	10k	10k	1k	1k

at training time. Instead, for ALL we use the model selection criterion proposed in Sec. E.3.5. The prior deep learning methods are based on 'interpreting' a classifier trained with supervised learning, and in line with prior work we tune its associated hyperparameters to optimize classification performance via minimizing the early-stopped mean rank on a validation dataset. We also visualize the distribution of results over the hyperparameter sweep for each method.

ALL is sensitive to the noise budget parameter $\overline{\gamma}$ and the learning rates of the classifier weights $\boldsymbol{\theta}$ and the noise distribution parameter $\tilde{\boldsymbol{\eta}}$. We consistently find that $\tilde{\boldsymbol{\eta}}$ should have a higher learning rate than $\boldsymbol{\theta}$, so in order to focus the search on better hyperparameter configurations we tune the *ratio* of the learning rate of $\tilde{\boldsymbol{\eta}}$ to that of $\boldsymbol{\theta}$ rather than the learning rate of $\tilde{\boldsymbol{\eta}}$. For ASCADv1 (fixed, variable) and AES-HD we find that performance is much better if we pretrain the classifier for half of the training steps with fixed $\tilde{\boldsymbol{\eta}} = \mathbf{0}$ and noise budget $\overline{\gamma} = 0.5$, then proceed as normal for the next half. Thus, for these trials, we first do a 10-trial grid search of the learning rate for $\boldsymbol{\theta}$ to minimize mean rank during this pretraining phase, then tune all hyperparameters as normal using this trained classifier as the starting point for the remaining 40 trials. In preliminary experiments we explored tuning the β_1, β_2, ϵ and weight decay strength of the AdamW optimizer as well as the number of $\tilde{\boldsymbol{\eta}}$ steps per $\boldsymbol{\theta}$ step, but chose to leave these fixed for the final search because they had little impact on performance. See Table 9 for the hyperparameter search space and chosen configurations for ALL.

The deep learning baselines are based on 'interpreting' a trained supervised classifier and require tuning this classifier – we tune its learning rate, dropout rates, and decide whether to use a constant or cosine decay learning rate schedule. In preliminary experiments we also explored tuning the β_1, β_2, ϵ and weight decay strength λ of the AdamW optimizer, but chose to leave them at fixed values for the final search because they had little effect on classification performance. See Table 8 for the hyperparameter search space, chosen configurations, and resulting classification performance for the supervised classifiers.

E.3.4 Performance evaluation methods

Unlike for the experiments on synthetic datasets, here we lack ground truth knowledge about the leakiness of individual measurements. It is challenging to evaluate the performance of leakage localization algorithms in this setting, and there is currently no consensus about the best way to do so. We consider 4 quantitative performance evaluation strategies which are similar in spirit to those used in prior work. To account for the varying 'shapes' of leakage assessments returned by the compared methods, all of our evaluation metrics are sensitive only to the *relative* leakiness assigned to measurements.

'Oracle' leakiness via SNR with relevant leaking first-order variables The present work is concerned with 'black box' leakage localization algorithms which require only a supervised learning-style dataset of traces and associated target variable values, and minimal *a priori* knowledge about the cryptographic implementations being analyzed. However, it is also possible to analyze devices in a 'white box' manner which does incorporate *a priori* knowledge. In particular, while second-order datasets such as ASCADv1

(fixed and variable) are not amenable to black box analysis with the first-order parametric methods such as SNR, it is possible to study the implementation and 'decompose' the second-order leakage into first-order leakage of *pairs* of internal variables, then use parametric methods to analyze leakage of these variables individually. As our main performance evaluation strategy, we use such a white box analysis to compute per-measurement leakiness predictions, which we treat as an 'oracle' against which to compare output. This is a useful way to validate deep learning methods because 1) it is interpretable and hyperparameter-free, and 2) it lets us check whether an output is consistent with an analysis which a domain expert might do.

For the ASCADv1 datasets, we use the white box analysis of Egger et al. (2022) to compute 'oracle' leakiness estimates for each of the measurements. We use the canonical target variable for both datasets, which is $\text{Sbox}(k_2 \oplus w_2)$ where Sbox is an invertible function which is publicly-known and shared by all AES implementations, $\oplus$ denotes the bitwise exclusive-or operation and k_2 and w_2 denote byte 2 of the key and plaintext, respectively, with indexing starting from 0. The underlying AES-128 implementation of these datasets uses Boolean masking, as shown in Alg. 1 of Benadjila et al. (2020). By design, this Boolean masking prevents the algorithm from ever directly operating on $\text{Sbox}(k_2 \oplus w_2)$, making all power measurements X_t nearly marginally statistically-independent of $\text{Sbox}(k_2 \oplus w_2)$.

Alg. 1 does directly operate on the following pairs of variables: $(r_2, \text{Sbox}(k_2 \oplus w_2) \oplus r_2)$, $(r_{\text{out}}, \text{Sbox}(k_2 \oplus w_2) \oplus r_{\text{out}})$, $(r_{\text{in}}, k_2 \oplus w_2 \oplus r_{\text{in}})$. The variables r_2, r_{out}, r_{in} are called *masks* and are internal variables which are randomly generated during each encryption. Thus, each of these variables *is* marginally dependent on some measurements X_t and can be detected with first-order parametric methods. Each of these pairs is said to leak $\text{Sbox}(k_2 \oplus w_2)$ because given both, one can calculate $\text{Sbox}(k_2 \oplus w_2)$ using the identity $a \oplus b \oplus b = a$. In addition to these pairs of variables, Egger et al. (2022) identified that the variables $\text{Sbox}(S_{\text{prev}} \oplus r_{\text{in}}) \oplus r_{\text{out}}$ and a 'security load' $S_{\text{prev}} \oplus \text{Sbox}(w_2 \oplus k_2) \oplus r_{\text{out}}$ also have a strong first-order association with power consumption and might contribute to leakage, where for byte 2 $S_{\text{prev}} = \text{Sbox}(k_{11} \oplus w_{11}) \oplus r_{11}$.

All 8 of these internal variables may be computed using the metadata published with the ASCADv1 datasets. As our oracle assessment for the ASCADv1 datasets, we compute the SNR of each of these 8 variables using the attack dataset and average them together. We can then qualitatively assess agreement between the output of leakage localization algorithms and these oracle assessments. For the 4 first-order datasets, we use as our oracle assessment the SNR of the target itself from the attack dataset, which amounts to an assumption that there is no leakage of order 2 or higher.

We use the Spearman rank correlation coefficient as a scalar summary of this agreement. This quantity is defined as the Pearson correlation coefficient between the *ranks* of a pair of sequences, and is useful for our purposes because it tells us the extent to which leakage localization algorithms assign the same *relative* leakiness to measurements as the oracle, while being insensitive to differences in their 'shape'.

Most prior work (Masure et al., 2019; Wouters et al., 2020; Schamberger et al., 2023; Yap et al., 2025) has used white box assessments similar to this for qualitative evaluation of leakage localization algorithms. To our knowledge, ours is the first work to summarize agreement with a scalar and use it for large-scale comparison between a large number of methods.

We refer to this performance evaluation strategy as *oracle agreement*. Note that we use the word 'oracle' for clarity of exposition, and we believe this is the least-flawed of the evaluation metrics we consider, but it does not give us genuine ground truth leakiness measurements. It is sensitive only to first-order leakage of variables which can be identified *a priori* as leaky, and will ignore any other exploitable variables. Additionally, SNR is not perfectly sensitive even to first-order leakage: it relies on changes in the expected values $\mathbb{E}[X_t \mid Y = y]$ with y and will not detect dependencies which do not influence the mean (e.g. if X_t is a Gaussian random variable with Y-independent mean but Y-dependent variance).

DNN occlusion tests Hettwer et al. (2020) proposed a variety of tests based on plotting the performance of a trained supervised classifier as its input features are successively occluded in order of their leakiness. The intuition is that leakier features should have a larger impact on the performance of the classifier, so the rate at which its performance changes as we successively occlude its inputs tells about the extent to which these inputs were leaky. In a similar spirit, we propose 2 evaluation metrics which we name the *forward* and *reverse DNN occlusion tests*.

Algorithm 4: Pseudocode for the DNN occlusion tests.

Input: Trained supervised classifier $\Phi^* : \mathbb{R}^T \times \mathsf{Y} \to [0, 1]$, attack dataset $\mathsf{D}_{\text{attack}} \subset \mathbb{R}^T \times \mathsf{Y}$, leakiness estimates $\boldsymbol{\ell} \in \mathbb{R}^T$, direction $d \in \{\text{'forward'}, \text{'reverse'}\}$

Output: Area under DNN occlusion curve $\overline{r}$

$\boldsymbol{m}_0 \leftarrow \mathbf{0}$ `// occlusion mask`

$\boldsymbol{\ell}_{\text{idx}} \leftarrow \texttt{argsort}(\boldsymbol{\ell})$ `// indices of sorted leakiness values, from low-high`

if $d = \text{'forward'}$ **then**

  $\boldsymbol{\ell}_{\text{idx}} \leftarrow \texttt{reverseOrder}(\boldsymbol{\ell}_{\text{idx}})$ `// sort from high-low instead`

else if $d = \text{'reverse'}$ **then**

  `pass`

for $t = 1, \ldots, T$ **do**

  $\boldsymbol{m}_t \leftarrow \boldsymbol{m}_{t-1} + \boldsymbol{I}_{\ell_{\text{idx},\,t}}$ `// un-occlude` t`-th least (reverse) or most (forward)-leaky feature`

  $r_t \leftarrow \frac{1}{|\mathsf{D}_{\text{attack}}|} \sum_{\boldsymbol{x}, y \in \mathsf{D}_{\text{attack}}} \text{Rank}\left(\Phi^*(\cdot \mid (\mathbf{1} - \boldsymbol{m_t}) \odot \boldsymbol{x}); y\right)$ `// record average classifier performance on attack dataset under this occlusion mask`

return $\frac{1}{T} \sum_{t=1}^{T} r_t$ `// area under the DNN occlusion curve`

See Alg. 4. For the *forward DNN occlusion test* we initially occlude all the input features of a trained classifier, then successively un-occlude one feature at a time from most- to least-leaky as predicted by the leakage localization algorithm under test. At each occlusion level we measure the performance of the classifier on the attack dataset in terms of mean rank. We then report the average performance across all occlusion levels. For a 'good' leakage assessment we expect the average performance to be better (lower) because useful features are un-occluded at a greater proportion of occlusion levels. Conversely, for a 'bad' leakage assessment we expect the average performance to be worse because useful features stay occluded for longer. The *reverse DNN occlusion test* is identical except that we un-occlude features from least- to most-leaky. For this test we expect the average performance to be worse (higher) for a 'good' leakage assessment and better (lower) for a 'good' leakage assessment. In general we expect the forward test to be sensitive to the extent to which the predicted-leakiest measurements are truly among the leakiest (similar to true/false positives), and the reverse test to be sensitive to the extent to which the predicted-nonleaky features are truly nonleaky (similar to true/false negatives).

A major limitation of the DNN occlusion tests is that they rely on an imperfect DNN classifier, and are only sensitive to associations insofar as the classifier exploits them. Additionally, we use the same architecture, training procedure and hyperparameters for these classifiers as for those 'interpreted' by the neural net attribution baseline methods, so the test may be 'biased' in favor of these. Nonetheless, we consider them a useful supplement to the oracle agreement metric because they do not suffer from the same restrictive assumptions about the nature of associations.

Feature selection efficacy for Gaussian template attack Similarly to Masure et al. (2019); Yap et al. (2023), we also evaluate leakage localization assessments based on their ability to do feature selection for Gaussian template attacks. To carry out this test, we first select the top 20 measurements with the highest estimated leakiness. We then perform a Gaussian template attack (Chari et al., 2003) using these measurements. The Gaussian template attack is a well-known parametric side-channel attack based on modeling $p_{\boldsymbol{X}|Y}$ with a Gaussian mixture model with one component per value Y may take on, then using Bayes' rule to estimate $p_{Y|\boldsymbol{X}}$. The leakier these features are, the more-performant we expect the attack to be.

For AES datasets accuracy is often low when predicting Y from a single value of $\boldsymbol{X}$. Thus, attack datasets typically consist of many traces $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_M$ recorded with a *fixed* AES key but varying plaintext. The target variable Y is typically chosen so that given the corresponding plaintext and ciphertext, Y is a known invertible function of the key. One can thereby make many predictions about the key using these traces, then 'accumulate' these predictions through the identity $\log p_{\boldsymbol{X}_1,\ldots,\boldsymbol{X}_M|K} = \sum_{m=1}^{M} \log p_{\boldsymbol{X}_m|K}$ where K denotes the key. A common performance metric for attacks is the *mean traces to disclosure (MTTD)*, given by the mean number of traces one must accumulate before the true key has the highest predicted probability mass (lower

is better). We use this metric to measure the performance of Gaussian template attacks on AES datasets. For OTiAiT and OTP, which are not AES datasets, we simply use the mean rank of the target variable on the attack dataset.

Note that for the second-order ASCADv1 datasets, the algorithms we consider do not reveal which of the leaky internal AES variables a measurement leaks. This is problematic when selecting features for a template attack, because a successful attack must have features corresponding to both of a pair of second-order leaky variables. If we simply used the top 20 predicted-leakiest features for an attack, this would be left to chance and make the performance metric unreliable. To address this issue, for the second-order datasets we instead segment the T measurements into 10 bins each containing $\lfloor \frac{T}{10} \rfloor$ consecutive measurements, then select the top 2 leakiest measurements from each bin.

The main shortcoming of this performance metric is that it is only sensitive to the 20 predicted-leakiest measurements and ignores all others (e.g. it cannot detect that leaky measurements have been assigned spuriously-low leakiness). Additionally, it assumes that the relationship between measurements and the target variable is well-described by a Gaussian mixture model, which may not hold in practice. However, unlike the oracle agreement metric it does not rely on human-identified first-order leaky variables, and unlike the DNN occlusion tests it is hyperparameter-free and may be biased towards different associations than the DNN classifiers. Thus, it is also a useful supplement to the aforementioned metrics.

E.3.5 Model selection criterion

While we consider the oracle agreement metric our most straightforward and useful performance metric, we cannot use it for model selection (e.g. choosing hyperparameters, or early-stopping runs). This is because it relies on 'white box' knowledge of the internal first-order leaky variables of second-order algorithms and knowledge of their random masks, which we assume not to have at training time. Thus, we must devise a model selection criterion which does not rely on this information.

Note that we can freely use the forward and reverse DNN occlusion tests and the template attack feature selection test for model selection by running them on our validation dataset rather than the attack dataset. Additionally, we find that because 'good' runs typically converge to similar leakiness assessments whereas 'bad' runs resemble random noise, a reasonable model selection heuristic is to 1) compute the average leakiness value for each measurement over all hyperparameter tuning runs, then 2) use the Spearman rank correlation coefficient between the average leakiness assessments and those for a particular run as a proxy for the run's performance. We refer to this model selection strategy as the *mean agreement criterion*.

In Fig. 13 we visualize the relationship between the oracle agreement and the forward and reverse DNN occlusion tests as well as the mean agreement criterion. We find that the forward and reverse DNN occlusion criterion are weakly correlated with oracle agreement, and that the mean agreement is strongly correlated for every dataset apart from OtiAiT. However, while we can consistently discard bad runs using these criteria, they typically lead to selection of suboptimal models. In this work we select ALL models using a 'composite criterion' based on ranking runs according to the forward and reverse DNN occlusion tests and mean agreement criterion, then selecting the model with the highest mean ranking across these 3 criteria. More research into model selection strategies for leakage localization algorithms is warranted.

E.3.6 Summary of experiments

We report and visualize our results in a variety of ways, which we list and summarize here.

Visualization of the best-performing leakage localization results found by ALL In Fig. 14 we visualize the oracle leakiness measurements and qualitatively compare them with the best ALL runs found during our hyperparameter sweeps. For the ASCADv1 datasets we draw distinct curves for the individual leaky variables – note the similarity between these plots and (Egger et al., 2022, Fig. 3a). We see a strong visual resemblance between the ALL outputs and the oracle leakiness assessments, despite the nonlinear relationship between them. Additionally, there are few points assigned a low oracle leakiness but high *relative* ALL-predicted leakiness or vice-versa. Most of the disagreement appears to lie in the predicted relative leakiness within groups of high-leakiness or low-leakiness measurements.

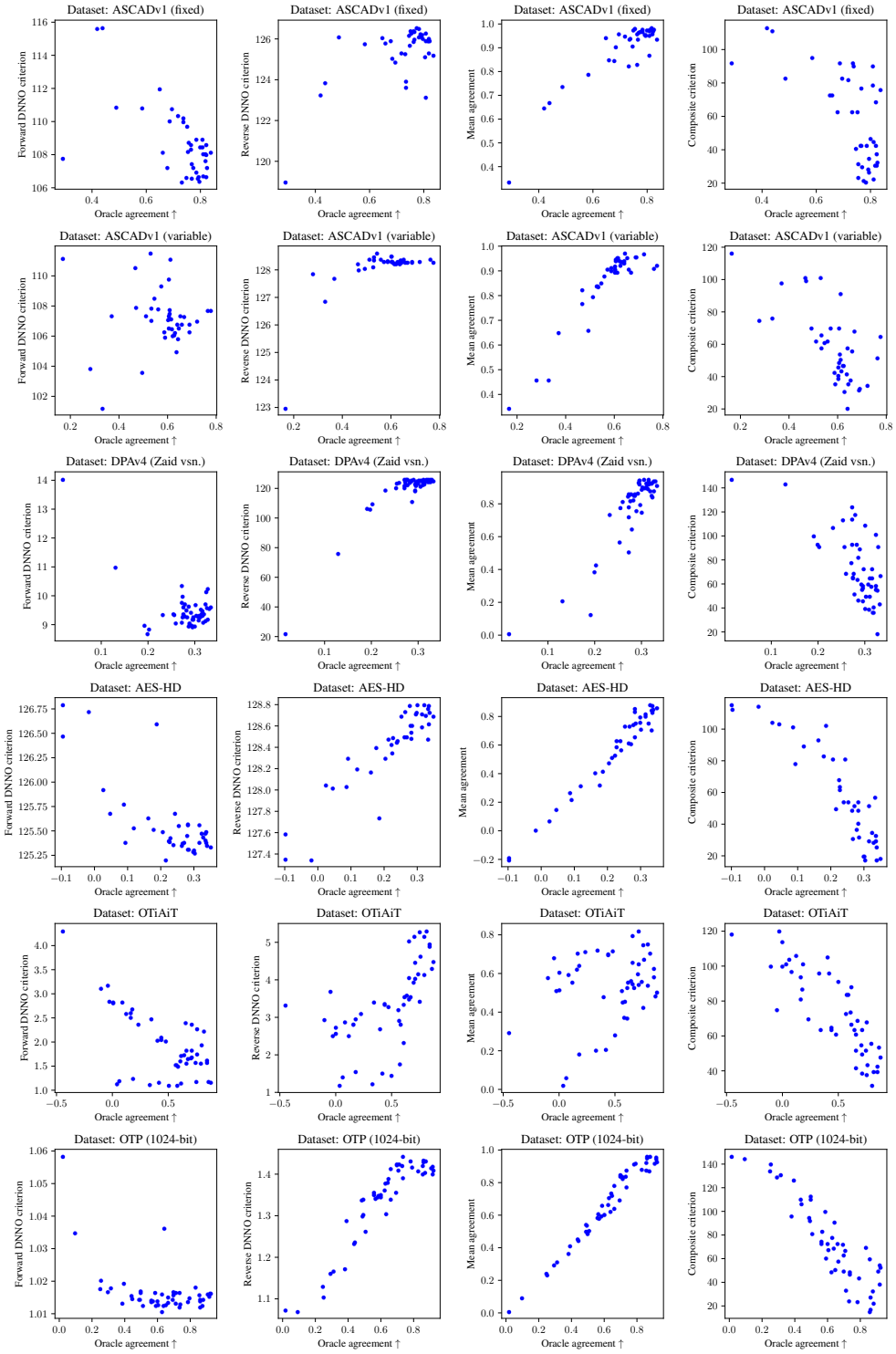

Figure 13: Visualization of the relationship between various model selection criteria from Sec. E.3.5 and the oracle agreement for ALL runs produced during hyperparameter search. We find consistently across datasets that the forward/reverse DNN occlusion and mean agreement criteria are consistently positively or negatively correlated with the oracle agreement, though this correlation is often weak. We achieve slightly better results using a composite criterion which uses considers the 'votes' according to all these criterion, and we adopt this composite criterion when selecting ALL models for comparison with baselines. However, this criterion often selects suboptimal models, and future research on leakage localization model selection strategies is warranted.

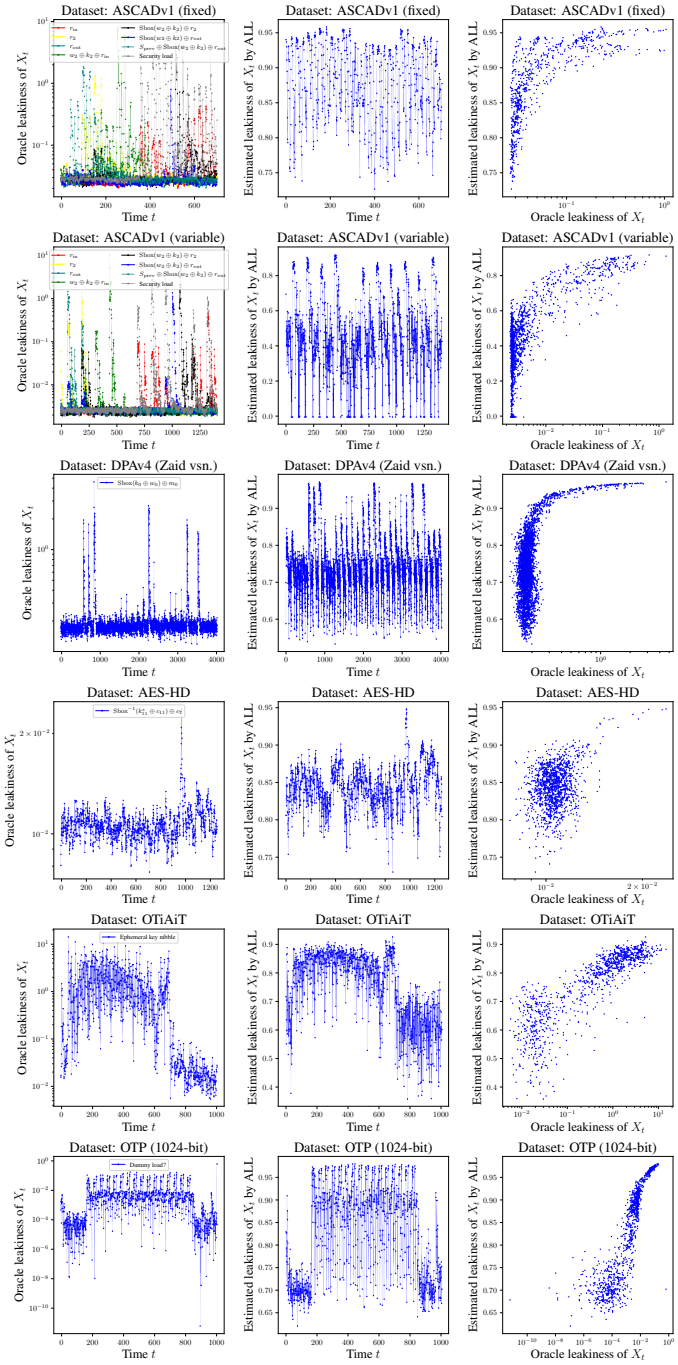

Figure 14: A plot which qualitatively compares the estimated leakiness of ALL with the oracle leakiness values. (**left column**) A plot of oracle leakiness of X_t vs. timestep t. For the second-order ASCADv1 datasets (top two rows), note that our plots are similar to (Egger et al., 2022, Fig. 3a) as they are based on the same first-order variables; differences are because we measure leakiness with SNR whereas they use CPOI. (**middle column**) A plot of estimated leakiness of X_t vs. timestep t according to ALL, for the best-performing ALL run as measured by oracle agreement. We want this column to look like the left column, up to a strictly-increasing nonlinear transform. (**right column**) A plot of the ALL-estimated leakiness of X_t vs. the oracle leakiness of X_t. These curves show that the ALL estimates are good in the sense that they tend to apply similar *relative* leakiness values to the measurements as the oracle.

ALL training curves In Fig. 15 we plot the evolution of various metrics over time during the ALL training procedure for the runs chosen using the model selection criterion of Sec. E.3.5. In general we observe that for successful ALL runs the classifier validation rank drops significantly below the random-guessing threshold at some point in training, though it may begin to rise again as the noise distribution trains adversarially against it. These curves are generally smooth and we do not observe significant training instability. Unfortunately, we are not aware of a reliable way to predict oracle agreement performance from the training curves.

Sensitivity of ALL to hyperparameters The main hyperparameters of ALL are the noise budget $\overline{\gamma}$ and the learning rates of the classifier weights $\boldsymbol{\theta}$ and the noise distribution parameter $\tilde{\boldsymbol{\eta}}$. In Fig. 16 we evaluate the sensitivity of ALL to these hyperparameters by varying them individually using the optimal configuration with respect to oracle agreement as a starting point. We find in general that performance generally varies smoothly with these hyperparameters and stays significantly above random-guessing over a large search space, which is desirable from a standpoint of hyperparameter tuning.

Attack performance and training curves of supervised classifiers All of the deep learning-based baseline methods are based on 'interpreting' a fixed classifier which has been trained using supervised learning to predict the target variable Y from the trace $\boldsymbol{X}$. In Fig. 17 we plot the cross-entropy loss and mean rank over time for these classifiers during training. Additionally, for the AES datasets we plot the rank of the correct key as we accumulate traces from the attack dataset. For reference we superimpose the correct key rank during trace accumulation for the following publicly-available open-weight classifiers: the CNN_{Best} and MLP_{Best} model of Benadjila et al. (2020), both of which are available here, and the models of Zaid et al. (2020) and their simplified versions from Wouters et al. (2020) distributed here. Also note that in Table 8 we list the early-stopped validation cross-entropy loss, rank, and mean traces to disclosure (MTTD) for these models. We find that our classifiers are able to 'successfully' attack all datasets (i.e. they can successfully predict the key by accumulating all traces in the provided attack dataset), and they achieve comparable MTTD to these open-weight models on ASCADv1-fixed, ASCADv1-variable, and DPAv4, and somewhat-worse MTTD on the AES-HD dataset.

m-occlusion window size sweep and smoothing effect We consider as baselines m-occlusion and m-second-order-occlusion (or m-occlusion^2). These baselines introduce the occlusion window size as an additional hyperparameter which must be tuned. In Fig. 18 we plot the oracle agreement performance of m-occlusion as we sweep m, and qualitatively show how the resulting leakiness vector is smoothed out with increasing m. In subsequent experiments, we denote by m^* the optimal value of m found in these experiments, and report results for 1-occlusion, m^*-occlusion, 1-occlusion^2 and m^*-occlusion^2 (due to the high cost of second-order occlusion we do not sweep its window size). Note that this introduces some data leakage into the results, as we are doing validation with our test metric which incorporates implementation knowledge that we assume not to have at training time. Because the goal of this work is to demonstrate the efficacy of ALL, and ALL generally outperforms m^*-occlusion and m^*-occlusion^2 despite the data leakage, we consider this acceptable.

Compared to 1-occlusion, m-occlusion has two major differences: it occludes multiple inputs simultaneously to increase the influence on classifier predictions, and it has a 'smoothing' effect which causes nearby measurements to be assigned similar leakiness values. The latter effect can be easily simulated using average-pooling, so we also plot the performance of ALL as we average-pool it with stride 1 and kernel size m. We find that while m-occlusion significantly improves performance over 1-occlusion on the DPAv4 and AES-HD datasets, on these same datasets we can significantly improve the performance of ALL by average-pooling, and ALL convincingly outperforms m-occlusion when accounting for this. We conjecture that the smoothing effect provides a useful inductive bias for these datasets, and emphasize that it can easily be applied to any other leakage localization technique.

Quantitative comparison between ALL and baseline methods using oracle agreement, forward and reverse DNN occlusion tests, and template attack feature selection test In Tables 10, 11, 12, 13 we compare the performance of ALL with our baseline methods according to the oracle agreement, forward DNN occlusion, reverse DNN occlusion, and template attack feature selection tests, respectively. As a sanity check against our supervised neural net architecture and training procedure, we also include results

Table 10: Performance comparison between various leakage localization algorithms according to the oracle agreement metric (**larger is better**) described in Sec. E.3.4. Results are reported as mean ± standard deviation over 5 random seeds. The best result is boxed and the best deep learning result is underlined.

	2nd-order datasets		**1st-order datasets**			
Method	ASCADv1 (fixed)	ASCADv1 (random)	DPAv4 (Zaid vsn.)	AES-HD	OTiAiT	OTP (1024-bit)
Random	-0.00 ± 0.04	-0.02 ± 0.01	0.01 ± 0.01	-0.01 ± 0.02	0.02 ± 0.03	0.03 ± 0.03
SNR	0.031	−0.092	0.344	0.185	0.989	0.944
SOSD	−0.253	0.272	0.259	0.063	0.886	0.803
CPA	0.521	−0.095	0.420	0.303	0.630	0.945
GradVis	0.48 ± 0.02	0.27 ± 0.01	0.198 ± 0.009	0.07 ± 0.01	0.55 ± 0.05	0.57 ± 0.02
Saliency	0.47 ± 0.02	0.26 ± 0.01	0.198 ± 0.008	0.07 ± 0.01	0.67 ± 0.06	0.58 ± 0.02
Input $*$ Grad	0.47 ± 0.02	0.25 ± 0.01	0.202 ± 0.009	0.08 ± 0.02	0.71 ± 0.05	0.60 ± 0.02
LRP	0.47 ± 0.02	0.25 ± 0.01	0.202 ± 0.009	0.08 ± 0.02	0.71 ± 0.05	0.60 ± 0.02
OccPOI	0.07 ± 0.01	0.064 ± 0.004	0.030 ± 0.008	0.044 ± 0.009	0.07 ± 0.02	0.01 ± 0.02
1-Occlusion	0.47 ± 0.02	0.25 ± 0.01	0.202 ± 0.009	0.08 ± 0.01	0.71 ± 0.05	0.60 ± 0.02
m^*-Occlusion	0.49 ± 0.02	0.41 ± 0.01	0.32 ± 0.01	0.18 ± 0.05	0.72 ± 0.04	0.77 ± 0.01
1-Occlusion^2	0.51 ± 0.01	0.27 ± 0.01	0.206 ± 0.009	0.08 ± 0.01	0.74 ± 0.05	0.60 ± 0.02
m^*-Occlusion^2	0.52 ± 0.01	0.42 ± 0.01	0.330 ± 0.009	0.19 ± 0.05	0.75 ± 0.04	0.788 ± 0.007
WoutersNet 1-Occlusion	0.18 ± 0.03	n/a	0.21 ± 0.02	0.11 ± 0.03	n/a	n/a
WoutersNet m^*-Occlusion	0.20 ± 0.03	n/a	0.29 ± 0.02	0.21 ± 0.04	n/a	n/a
WoutersNet GradVis	0.19 ± 0.03	n/a	0.21 ± 0.02	0.11 ± 0.03	n/a	n/a
WoutersNet Input $*$ Grad	0.18 ± 0.03	n/a	0.21 ± 0.02	0.11 ± 0.03	n/a	n/a
WoutersNet Saliency	0.19 ± 0.03	n/a	0.21 ± 0.02	0.11 ± 0.03	n/a	n/a
ZaidNet 1-Occlusion	0.24 ± 0.03	n/a	0.19 ± 0.01	0.13 ± 0.02	n/a	n/a
ZaidNet m^*-Occlusion	0.25 ± 0.04	n/a	0.273 ± 0.009	0.21 ± 0.05	n/a	n/a
ZaidNet GradVis	0.25 ± 0.03	n/a	0.19 ± 0.01	0.13 ± 0.02	n/a	n/a
ZaidNet Input $*$ Grad	0.25 ± 0.04	n/a	0.19 ± 0.01	0.13 ± 0.02	n/a	n/a
ZaidNet Saliency	0.25 ± 0.03	n/a	0.19 ± 0.01	0.13 ± 0.02	n/a	n/a
ALL (ours)	0.794 ± 0.006	0.60 ± 0.01	0.317 ± 0.002	0.22 ± 0.03	0.782 ± 0.001	0.848 ± 0.003

for GradVis, Saliency, LRP, Input $*$ Grad, 1-occlusion, and m^*-occlusion computed using the models of Zaid et al. (2020) and their simplified versions from Wouters et al. (2020) distributed here.

We find that ALL outperforms all prior deep learning-based leakage localization algorithms on all datasets except for DPAv4 according to the oracle agreement metric. The first-order parametric methods outperform the deep learning methods on the first-order datasets but generally do poorly on the second-order ASCADv1 datasets due to not being sensitive to second-order associations. Unsurprisingly, SNR and SOSD achieve near-random performance on ASCADv1-fixed and SNR and CPA achieve near-random performance on ASCADv1-variable. Surprisingly, CPA performs fairly well on ASCADv1-fixed and SOSD performs fairly well on ASCADv1-variable. We are not sure why this is the case, but conjecture it is because these methods have some natural proclivity to 'rule out' measurements which are not at useful points in time for these particular datasets (e.g. those which are not close to a clock edge). Note that this surprisingly-strong performance compared to the deep learning baselines does not appear to carry over to the evaluations with the DNN occlusion tests or the template attack feature selection test.

On average, ALL is the best method on the majority of datasets according to the reverse DNN occlusion test and template attack test, but results are mixed according to the forward DNN occlusion test. However, we find that the DNN occlusion tests have high variance, and in general there is a large overlap in error bars (note the large number of boxed and underlined methods in Tables 11 and 12).

Distribution of performance during hyperparameter sweeps In Fig. 19 we show the distribution of performance during the random hyperparameter searches for ALL and selected baselines. We see that ALL convincingly outperforms all baselines besides m^*-occlusion on all datasets. Additionally, it has a higher peak performance than m^*-occlusion on every dataset except for DPAv4, and a higher median performance on ASCADv1-fixed, ASCADv1-variable, and AES-HD. As previously noted and considered again in subsequent ablation studies, we can further improve the performance of ALL relative to m^*-occlusion by mimicking its smoothing effect with stride-1 average-pooling.

Table 11: Performance comparison between various leakage localization algorithms according to the forward DNN occlusion test (**smaller is better**) described in Sec. E.3.4. Results are reported as mean ± standard deviation over 5 random seeds. The best result is boxed and the best deep learning result is underlined.

	2nd-order datasets		**1st-order datasets**			
Method	ASCADv1 (fixed)	ASCADv1 (random)	DPAv4 (Zaid vsn.)	AES-HD	OTiAiT	OTP (1024-bit)
Random	111 ± 7	112 ± 10	20 ± 20	126.8 ± 0.7	1.3 ± 0.8	1.1 ± 0.1
SNR	118 ± 7	118 ± 10	11 ± 5	125.1 ± 0.8	1.2 ± 0.5	1.02 ± 0.04
SOSD	116 ± 8	106 ± 20	11 ± 20	125.4 ± 0.6	1.2 ± 0.6	1.03 ± 0.06
CPA	111 ± 8	117 ± 10	12 ± 9	125.3 ± 0.7	2 ± 2	1.02 ± 0.04
GradVis	109 ± 8	97 ± 10	9 ± 5	125.6 ± 0.7	2 ± 2	1.01 ± 0.03
Saliency	109 ± 8	96 ± 10	9 ± 5	125.6 ± 0.7	2 ± 2	1.01 ± 0.03
Input $*$ Grad	109 ± 8	97 ± 10	9 ± 5	125.6 ± 0.7	2 ± 1	1.01 ± 0.03
LRP	109 ± 8	97 ± 10	9 ± 5	125.6 ± 0.7	2 ± 1	1.01 ± 0.03
OccPOI	122 ± 6	121 ± 10	58 ± 50	127 ± 1	3 ± 1	1.1 ± 0.2
1-Occlusion	108 ± 8	97 ± 10	9 ± 5	125.6 ± 0.7	2 ± 1	1.01 ± 0.03
m^*-Occlusion	108 ± 7	96 ± 10	9 ± 8	125.3 ± 0.8	2 ± 2	1.01 ± 0.04
1-Occlusion^2	108 ± 7	97 ± 10	9 ± 5	125.5 ± 0.7	2 ± 1	1.01 ± 0.03
m^*-Occlusion^2	108 ± 7	96 ± 10	9 ± 8	125.3 ± 0.8	2 ± 1	1.01 ± 0.03
WoutersNet 1-Occlusion	109 ± 8	n/a	9 ± 6	125.4 ± 0.8	n/a	n/a
WoutersNet m^*-Occlusion	109 ± 8	n/a	9 ± 8	125.4 ± 0.9	n/a	n/a
WoutersNet GradVis	109 ± 9	n/a	9 ± 6	125.4 ± 0.8	n/a	n/a
WoutersNet Input $*$ Grad	109 ± 9	n/a	9 ± 6	125.4 ± 0.8	n/a	n/a
WoutersNet Saliency	109 ± 9	n/a	9 ± 6	125.4 ± 0.8	n/a	n/a
ZaidNet 1-Occlusion	110 ± 8	n/a	9 ± 5	125.5 ± 0.8	n/a	n/a
ZaidNet m^*-Occlusion	110 ± 8	n/a	9 ± 8	125.4 ± 0.8	n/a	n/a
ZaidNet GradVis	110 ± 8	n/a	9 ± 5	125.5 ± 0.8	n/a	n/a
ZaidNet Input $*$ Grad	110 ± 8	n/a	9 ± 5	125.5 ± 0.8	n/a	n/a
ZaidNet Saliency	110 ± 8	n/a	9 ± 5	125.5 ± 0.8	n/a	n/a
ALL (ours)	107 ± 5	104 ± 9	10 ± 5	125.5 ± 0.8	2 ± 2	1.01 ± 0.03

Table 12: Performance comparison between various leakage localization algorithms according to the reverse DNN occlusion test(**larger is better**) described in Sec. E.3.4. Results are reported as mean ± standard deviation over 5 random seeds. The best result is boxed and the best deep learning result is underlined.

	2nd-order datasets		**1st-order datasets**			
Method	ASCADv1 (fixed)	ASCADv1 (random)	DPAv4 (Zaid vsn.)	AES-HD	OTiAiT	OTP (1024-bit)
Random	112 ± 7	114 ± 10	25 ± 30	126.7 ± 0.7	1.3 ± 0.8	1.1 ± 0.1
SNR	111 ± 7	123 ± 6	126 ± 20	128.6 ± 0.5	5 ± 3	1.4 ± 0.1
SOSD	112 ± 8	127 ± 5	108 ± 30	128.2 ± 0.6	5 ± 3	1.4 ± 0.2
CPA	120 ± 6	118 ± 10	115 ± 20	128.4 ± 0.4	4 ± 2	1.4 ± 0.1
GradVis	126 ± 4	128 ± 2	122 ± 20	128.0 ± 0.6	4 ± 2	1.3 ± 0.1
Saliency	126 ± 4	127 ± 2	122 ± 20	128.0 ± 0.6	5 ± 2	1.3 ± 0.1
Input $*$ Grad	126 ± 4	127 ± 2	122 ± 20	128.1 ± 0.6	5 ± 2	1.3 ± 0.1
LRP	126 ± 4	127 ± 2	122 ± 20	128.1 ± 0.6	5 ± 2	1.3 ± 0.1
OccPOI	122 ± 8	125 ± 7	43 ± 40	127 ± 1	4 ± 2	1.1 ± 0.1
1-Occlusion	126 ± 4	127 ± 2	122 ± 20	128.1 ± 0.6	5 ± 2	1.3 ± 0.1
m^*-Occlusion	126 ± 4	127 ± 3	121 ± 20	128.5 ± 0.6	5 ± 2	1.3 ± 0.1
1-Occlusion^2	126 ± 4	127 ± 2	122 ± 20	128.1 ± 0.6	5 ± 2	1.3 ± 0.1
m^*-Occlusion^2	126 ± 4	127 ± 3	121 ± 20	128.5 ± 0.6	5 ± 3	1.3 ± 0.1
WoutersNet 1-Occlusion	122 ± 4	n/a	116 ± 20	128.2 ± 0.6	n/a	n/a
WoutersNet m^*-Occlusion	122 ± 5	n/a	119 ± 20	128.4 ± 0.5	n/a	n/a
WoutersNet GradVis	122 ± 4	n/a	116 ± 20	128.2 ± 0.6	n/a	n/a
WoutersNet Input $*$ Grad	122 ± 4	n/a	116 ± 20	128.2 ± 0.6	n/a	n/a
WoutersNet Saliency	122 ± 4	n/a	116 ± 20	128.2 ± 0.6	n/a	n/a
ZaidNet 1-Occlusion	122 ± 5	n/a	116 ± 20	128.1 ± 0.6	n/a	n/a
ZaidNet m^*-Occlusion	123 ± 6	n/a	119 ± 20	128.4 ± 0.6	n/a	n/a
ZaidNet GradVis	122 ± 6	n/a	117 ± 20	128.1 ± 0.6	n/a	n/a
ZaidNet Input $*$ Grad	122 ± 5	n/a	116 ± 20	128.1 ± 0.6	n/a	n/a
ZaidNet Saliency	122 ± 6	n/a	117 ± 20	128.1 ± 0.6	n/a	n/a
ALL (ours)	126 ± 4	128 ± 4	125 ± 20	128.3 ± 0.5	6 ± 2	1.4 ± 0.2

Table 13: Performance comparison between various leakage localization algorithms according to the template attack feature selection test (**smaller is better**) described in Sec. E.3.4. Results are reported as mean $\pm$ standard deviation over 5 random seeds. The best result is $\boxed{\text{boxed}}$ and the best deep learning result is underlined.

	2nd-order datasets		**1st-order datasets**			
Method	ASCADv1 (fixed)	ASCADv1 (random)	DPAv4 (Zaid vsn.)	AES-HD	OTiAiT	OTP (1024-bit)
Random	1293 ± 1000	56421 ± 40000	449 ± 100	25000.000	1.2 ± 0.1	1.44 ± 0.02
SNR	5496.010	54834.990	2.590	17159.690	$\boxed{1.058}$	1.385
SOSD	7733.870	3237.250	91.410	17520.530	1.059	1.398
CPA	3826.440	100000.000	10.540	18974.510	1.101	1.385
GradVis	686 ± 100	1162 ± 1000	2.7 ± 0.1	20014 ± 6000	1.4 ± 0.3	1.378 ± 0.007
Saliency	726 ± 100	1412 ± 2000	2.7 ± 0.1	19438 ± 6000	1.14 ± 0.02	1.379 ± 0.005
Input $*$ Grad	675 ± 100	1194 ± 2000	2.6 ± 0.1	19893 ± 6000	1.14 ± 0.02	1.378 ± 0.003
LRP	675 ± 100	1194 ± 2000	2.6 ± 0.1	19893 ± 6000	1.14 ± 0.02	1.378 ± 0.003
OccPOI	787 ± 100	942 ± 200	71 ± 30	25000.000	$\underline{1.08 \pm 0.03}$	1.47 ± 0.05
1-Occlusion	667 ± 100	1376 ± 2000	2.65 ± 0.08	20011 ± 6000	1.14 ± 0.02	1.379 ± 0.003
m^*-Occlusion	673 ± 70	727 ± 400	9 ± 1	$\boxed{\underline{16283 \pm 10}}$	1.17 ± 0.02	1.382 ± 0.007
1-Occlusion^2	709 ± 100	1086 ± 1000	2.65 ± 0.08	20222 ± 6000	1.14 ± 0.02	1.378 ± 0.003
m^*-Occlusion^2	642 ± 60	710 ± 400	9 ± 1	$\boxed{\underline{16033 \pm 700}}$	1.16 ± 0.03	1.381 ± 0.008
WoutersNet 1-Occlusion	6454 ± 4000	n/a	2.9 ± 0.6	20278 ± 3000	n/a	n/a
WoutersNet m^*-Occlusion	4408 ± 4000	n/a	11.7 ± 0.6	$\boxed{\underline{16124 \pm 300}}$	n/a	n/a
WoutersNet GradVis	6230 ± 4000	n/a	2.9 ± 0.5	19539 ± 5000	n/a	n/a
WoutersNet Input $*$ Grad	4988 ± 4000	n/a	3.0 ± 0.6	20546 ± 4000	n/a	n/a
WoutersNet Saliency	5878 ± 4000	n/a	2.8 ± 0.5	20151 ± 4000	n/a	n/a
ZaidNet 1-Occlusion	2236 ± 2000	n/a	2.6 ± 0.3	20696 ± 2000	n/a	n/a
ZaidNet m^*-Occlusion	2485 ± 2000	n/a	9 ± 2	$\boxed{\underline{16124 \pm 300}}$	n/a	n/a
ZaidNet GradVis	2560 ± 3000	n/a	3.0 ± 0.8	22790 ± 3000	n/a	n/a
ZaidNet Input $*$ Grad	3234 ± 3000	n/a	2.5 ± 0.3	21600 ± 3000	n/a	n/a
ZaidNet Saliency	2295 ± 2000	n/a	2.4 ± 0.4	22561 ± 3000	n/a	n/a
ALL (ours)	$\boxed{\underline{459 \pm 40}}$	$\boxed{\underline{394 \pm 20}}$	$\boxed{\underline{2.22 \pm 0.01}}$	17582 ± 5000	1.11 ± 0.02	$\boxed{\underline{1.363 \pm 0.007}}$

E.3.7 Ablation studies

ALL has many differences from prior work, so here we run ablation studies to evaluate the impact of some of the important individual differences. For each ablated design decision, we run a new 50-trial hyperparameter sweep and plot the distribution of performance in terms of oracle agreement. Results are shown in Fig. 20, with the salient results from Fig. 19 copied over for reference. We ablate the following design decisions:

Heavy input dropout for the supervised classifiers used by baselines See the distributions labeled 'm^*-Occl + heavy dropout'. The ALL classifier is trained on occluded inputs with the possible 'heaviness' of the occlusion chosen spanning a wide range and chosen as a hyperparameter to optimize our proxy for oracle agreement. In contrast, the baseline methods are based on 'interpreting' fixed classifiers which have been trained with input dropout chosen from $\{0.0, 0.1\}$ to optimize classification performance. A plausible conjecture is that ALL has strong performance because the heavy input corruption 'encourages' the classifier to compensate by leveraging a wider variety of input-output associations, whereas because the supervised classifiers train on uncorrupted or lightly-corrupted inputs, they have no such 'incentive'. We test this assumption by tuning the classifiers with input dropout chosen from $\{0.05, 0.1, \ldots, 0.95\}$ (the same search space that ALL uses for $\overline{\gamma}$). We plot the performance distribution for m^*-occlusion and 1-occlusion. For clarity we omit GradVis, Saliency, LRP and Input $*$ Grad, but these have similar trends as 1-occlusion. We did not test OccPOI or second-order occlusion because they are costly and in our prior experiments second-order m-occlusion performs similarly to m-occlusion and OccPOI performs poorly compared to baselines.

We see that this heavy input dropout leads to significant improvements to the maximum and median performance on all datasets, though ALL still convincingly outperforms 1-occlusion and m-occlusion except on DPAv4 and OTiAiT. To our knowledge no prior work has explored the effect of regularization strategies on leakage localization performance of methods which 'interpret' supervised classifiers, and this result suggests that such research may be fruitful.

Adversarial → 'Cooperative' leakage localization See the distributions labeled 'ALL (cooperative)'. The intuition behind ALL is that we train a noise distribution to distribute a fixed amount of noise to minimize the performance of a classifier, and we then interpret noisier measurements as leakier. Along the lines of this intuition, we could also train it to distribute noise to *maximize* the performance of a classifier, then interpret less-noisy measurements as leakier. We try this approach and find that it typically degrades performance relative to the adversarial version of the algorithm. We conjecture that this is because the adversarial approach encourages the classifier to rely on a diverse assortment of input-output associations, whereas the cooperative approach does the opposite.

Omitting the noise conditioning to the classifier See the distributions labeled 'ALL (unconditional)'. We feed the occlusion mask as an auxiliary input to the ALL classifiers, motivated by our theory which views it as a family of classifiers trained in an amortized manner via conditioning a single neural net. We test omitting this auxiliary input and find mixed results. On ASCADv1-fixed and DPAv4, ALL achieves stronger performance without this auxiliary input, whereas on AES-HD it achieves stronger performance with this input and on ASCADv1-variable, OTiAiT, and OTP results are approximately the same. For future work it is likely justifiable to simplify ALL by omitting this conditioning.

Average pooling ALL to mimic the smoothing effect of m^*-occlusion See the distributions labeled 'ALL + AvgPool(m^*)'. Recall that we have chosen the occlusion window size m^* by sweeping it over successive odd numbers until finding a local maximum in oracle agreement. We find that this consistently improves performance. One plausible explanation for the performance improvement is that large-window occlusion has a 'smoothing' effect which amounts to an assumption that temporally-close measurements have similar leakiness. This is likely true for some datasets, and as 1-occlusion is always included in the window size search space, m^* occlusion will never degrade performance relative to 1-occlusion. To test this explanation we average pool ALL with stride 1, kernel size m^*, and zero-padding to preserve dimensionality, which creates a similar smoothing effect. Our results seem to support this explanation, with average pooling significantly improving the performance of ALL on the DPAv4 and AES-HD datasets, where m^*-occlusion also has significantly stronger performance than 1-occlusion. Note that using oracle agreement to choose an occlusion/pooling

window size causes data contamination, so while it is fair to compare results for m^*-average-pooled ALL to those of m^*-occlusion, they cannot fairly be compared to any other baseline.

E.3.8 Theoretical and empirical computational cost of deep learning methods

In table 14 we list the theoretical computational complexity of the considered methods, as well as the measured wall clock time to run them. While for fairness our experiments use an equal number of training steps for ALL and supervised learning, note that in general we suspect the latter will converge in fewer training steps; thus, these measurements likely overestimate the practical wall-clock time of supervised training. Additionally, note that OccPOI takes significantly more time than m-occlusion despite having the same computational complexity; this is because it requires $\Omega(T)$ *sequential* forward passes through the model, whereas all forward passes may be done in parallel for m-occlusion. There is additional variance for the runtime of this method because we choose the attack dataset size to be as small as possible while comfortably allowing the classifier to attain a correct-key rank of 0 – e.g. AES-HD takes the longest because we use the full attack dataset. We omit the parametric statistics-based methods from consideration, but these are done on the CPU and take a negligible amount of time compared to the deep learning methods.

Table 14: Comparison of the computational cost of methods considered in our work. We denote by C_F and C_B the cost of a forward and backward pass through our neural net respectively, N the dataset size, n_{sup} and n_{all} the number of epochs required for supervised learning and ALL respectively, and T the data dimensionality. We omit the parametric statistics-based baseline methods because they are done on the CPU and take a negligible amount of time relative to the deep learning methods. Runtimes are reported as mean $\pm$ standard deviation over 5 runs. †These methods are used to 'interpret' a trained supervised classifier, but for comparison we report the cost of running them *after* training the classifier. In practice one would also incur the cost of supervised training (first row).

Method	**Total FLOPS**	**A6000 minutes per trial**					
		ASCADv1 (fixed)	ASCADv1 (random)	DPAv4 (Zaid)	AES-HD	OTiAiT	OTP
Supervised training	$\Theta(Nn_{\text{sup}}(C_F + C_B))$	2.09 ± 0.01	3.68 ± 0.02	1.98 ± 0.04	1.78 ± 0.03	0.17 ± 0.02	0.448 ± 0.009
GradVis†	$\Theta(N(C_F + C_B))$	0.0675 ± 0.0008	0.263 ± 0.002	0.0080 ± 0.0001	0.0384 ± 0.0003	0.0080 ± 0.0001	0.0666 ± 0.0005
Saliency†	$\Theta(N(C_F + C_B))$	0.078 ± 0.001	0.300 ± 0.003	0.0086 ± 0.0002	0.048 ± 0.002	0.0085 ± 0.0002	0.0881 ± 0.0004
Input $*$ Grad†	$\Theta(N(C_F + C_B))$	0.080 ± 0.002	0.3023 ± 0.0007	0.0086 ± 0.0002	0.049 ± 0.002	0.0086 ± 0.0002	0.086 ± 0.003
LRP†	$\Theta(N(C_F + C_B))$	0.080 ± 0.001	0.3045 ± 0.0008	0.0088 ± 0.0002	0.050 ± 0.002	0.00861 ± 0.00007	0.087 ± 0.005
m-Occlusion†	$\Theta(NC_FT)$	0.1235 ± 0.0004	0.952 ± 0.003	0.0644 ± 0.0003	0.1781 ± 0.0007	0.018 ± 0.002	0.247 ± 0.002
2nd-order m-Occlusion†	$\Theta(NC_FT^2)$	16.6 ± 0.1	327 ± 1	81.1 ± 0.5	68.4 ± 0.6	4.42 ± 0.02	70.6 ± 0.3
OccPOI†	$\Omega(NC_FT)$	2.29 ± 0.09	7.6 ± 0.4	0.709 ± 0.009	36.5 ± 0.6	0.042 ± 0.002	0.0437 ± 0.0004
ALL (Ours)	$\Theta(Nn_{\text{all}}(C_F + C_B))$	3.2 ± 0.2	4.7 ± 0.2	2.44 ± 0.05	2.28 ± 0.05	0.200 ± 0.005	0.323 ± 0.004

F Limitations

For real side-channel leakage datasets we lack ground truth knowledge about the leakiness of each measurement, and all evaluation metrics considered in our paper have limitations. In particular:

- The 'oracle' leakage assessments used in the main paper ignore leakage of order greater than 1 except where higher-order leakage may be decomposed into first-order leakage of multiple variables, similarly to the analysis of Egger et al. (2022). This relies on careful analysis of implementations by humans, and is subject to error and oversights. Additionally, the SNR is not guaranteed to detect even first-order leakage – it is sensitive only to the influence of the secret variable Y on the *mean* of each $X_t \mid Y$, and will not detect cases where the distribution of $X_t \mid Y$ changes with the mean remaining fixed (e.g. if X_t is Gaussian distributed with Y-dependent variance and Y-independent mean).
- The DNN occlusion tests and similar metrics proposed by Hettwer et al. (2020) are sensitive only to the associations between $\boldsymbol{X}$ and Y that the neural net exploits. The superior performance of ALL compared to prior deep learning-based algorithms, as well as work in other domains such as Geirhos et al. (2020); Hermann & Lampinen (2020), suggests that DNNs are prone to exploiting some but not all of the associations at their disposal. Additionally, such metrics may be biased in favor of deep learning methods due to both leveraging SGD-trained DNNs.

- Evaluation via feature selection for a Gaussian template attack as done by Masure et al. (2019); Yap et al. (2023) is sensitive only to the top ≈ 10 leakiest measurements identified by an algorithm and ignores all others, which is a major limitation. Additionally, for second-order datasets a template attack will be unsuccessful unless *both* of a leaking pair of variables (e.g. r_3 and $\text{Sbox}(w_3 \oplus k_3) \oplus r_3$ for ASCADv1) are leaked through the selected measurements, and because no method considered in our work except for second-order occlusion has any ability to discern the particular variable leaking through a measurement, this would rely on random chance.

We believe it is important for work in this domain to use a variety of evaluation strategies to compensate for these individual limitations. This is similar to image synthesis research where it is common to use precision, recall, and FID score to avoid the individual limitations of each of these metrics.

ALL (alongside the other deep learning-based method we consider) has major advantages over manual analysis and simpler parametric methods. However, it also comes with its own limitations:

- Deep learning methods such as ours have a far larger computational cost than parametric methods. Additionally, ALL is more computationally-expensive than the prior deep learning algorithms apart from OccPOI and second-order occlusion. This increased cost is not fully reflected in our reported runtimes because while for fairness we have run both ALL and supervised learning-based methods for the same number of training steps and post-hoc early-stopped the latter, in practice we find that supervised learning usually converges to good solutions in fewer training steps than ALL.
- Deep learning methods such as ours require hyperparameter tuning and will give incorrect leakiness estimates if poorly tuned. We view ALL as *complementary* with traditional approaches rather than as a replacement. Traditional approaches can cheaply exploit domain/implementation knowledge and detect many types of leakage for a low computational cost. ALL can then be used to search for additionally leaking measurements not detected by these approaches.

Additionally, a limitation of the literature on deep side-channel leakage localization which we do not address in this work is that experiments are done at a smaller scale than the state of the art in deep side-channel attacks, and the considered algorithms would likely need to be significantly scaled up for practical use. The largest-scale experiments in this and prior work consider the ASCADv1-variable dataset, which consists of 300k 1400-measurement power traces and can be attacked with simple MLP and CNN architectures. The 1400-length power traces are extracted from longer 250k-length traces via downsampling and cropping in the general vicinity of known leaky instructions which were themselves located with a white box analysis. We believe that a critical direction for future work in this area is scaling existing methods to larger-scale architectures applied to uncropped high-dimensionality datasets, such as the transformer architecture of Bursztein et al. (2023) and the raw ASCADv2 dataset (Masure & Strullu, 2023), which consists of 800k 1M-measurement power traces.

Our experiments consider mainly temporally-synchronized power trace datasets, which we believe is reasonable because in practice implementation designers are likely able to collect synchronized traces, as the dataset authors have done (e.g. by using the clock line of the hardware to trigger an oscilloscope). In practice a violation of this condition would mean that leakage appears 'spread out' (e.g. see row 3 of Fig. 4), making it more-challenging for designers to identify its source.

A limitation of ALL, alongside the parametric statistical methods and OccPOI, is that they produce only a single vector summarizing leakage over the entire dataset, rather than for individual traces. As observed by Wouters et al. (2020), the neural net attribution methods can also be used to assign leakiness estimates to individual traces, though to our knowledge no work has systematically studied this ability. This would be a useful direction for future work, and would likely require innovation in performance evaluation strategies beyond those introduced in this paper or prior work.

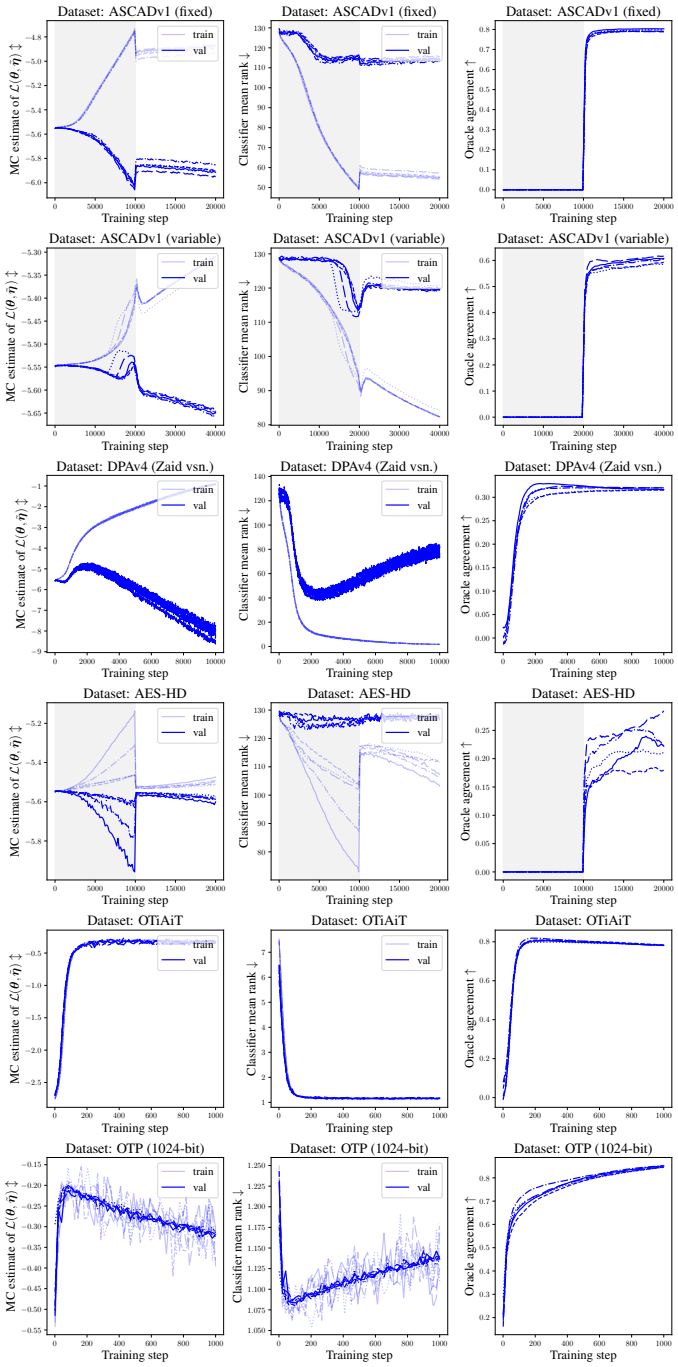

Figure 15: Training curves for ALL with the hyperparameter configuration chosen using our composite model selection criterion. Grey shaded regions denote 'pretraining' phases where we optimize $\boldsymbol{\theta}$ but not $\tilde{\boldsymbol{\eta}}$. Note that jumps in the traces happen because we set $\overline{\gamma} = 0.5$ for pretraining and change it to the setting chosen through hyperparameter search for training. (**left column**) The per-minibatch estimates of our objective function $\mathcal{L}(\boldsymbol{\theta}, \tilde{\boldsymbol{\eta}})$ (i.e. negative cross-entropy classification loss of the classifier) during training. We are optimizing $\boldsymbol{\theta}$ to maximize this value and $\tilde{\boldsymbol{\eta}}$ to minimize it. (**center column**) The mean rank of the correct label in the logits of the classifier (lower corresponds to higher classifier performance). We use this instead of accuracy for its finer granularity. (**right column**) The performance in terms of oracle agreement over the course of training; higher is better, as it indicates that the current ALL leakiness estimates are closer to being a strictly-increasing function of the oracle assessments.

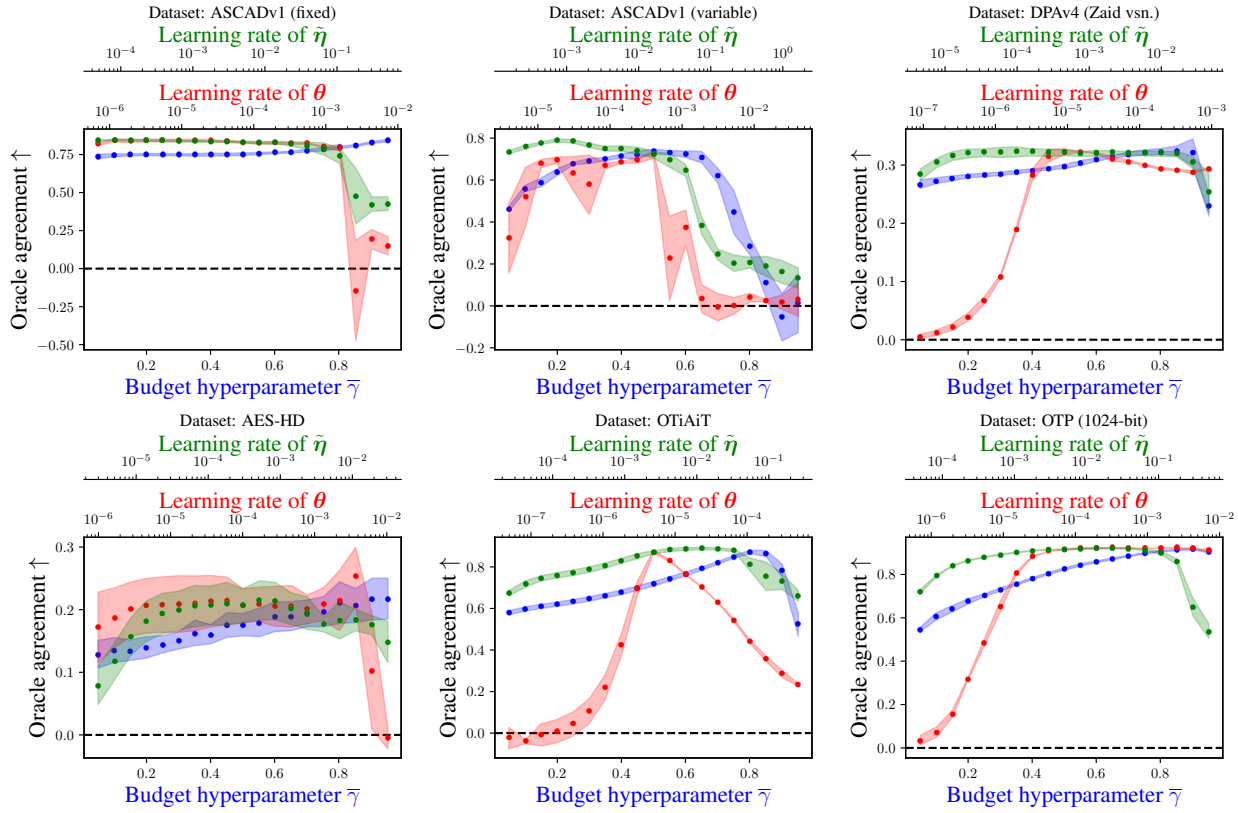

Figure 16: A plot of the performance of ALL as we perturb its 3 main hyperparameters: the noise budget $\overline{\gamma}$ and the learning rates of the classifier weights $\boldsymbol{\theta}$ and the noise distribution parameter $\tilde{\boldsymbol{\eta}}$. We find that performance varies smoothly with these hyperparameters and stays significantly better than random guessing over a large region of the search space, which is useful from a standpoint of hyperparameter tuning. We consider $\overline{\gamma}$ values in `np.arange(0.05, 1.0, 0.05)` and learning rates scaled by values in `np.logspace(-2, 2, 19)` relative to their optimal values according to the oracle agreement metric. All hyperparameters other than those being perturbed are left at their optimal values according to the oracle agreement metric. We repeat trials for 5 random seeds and report their mean with dots and $\pm$ 1 standard deviation with shading.

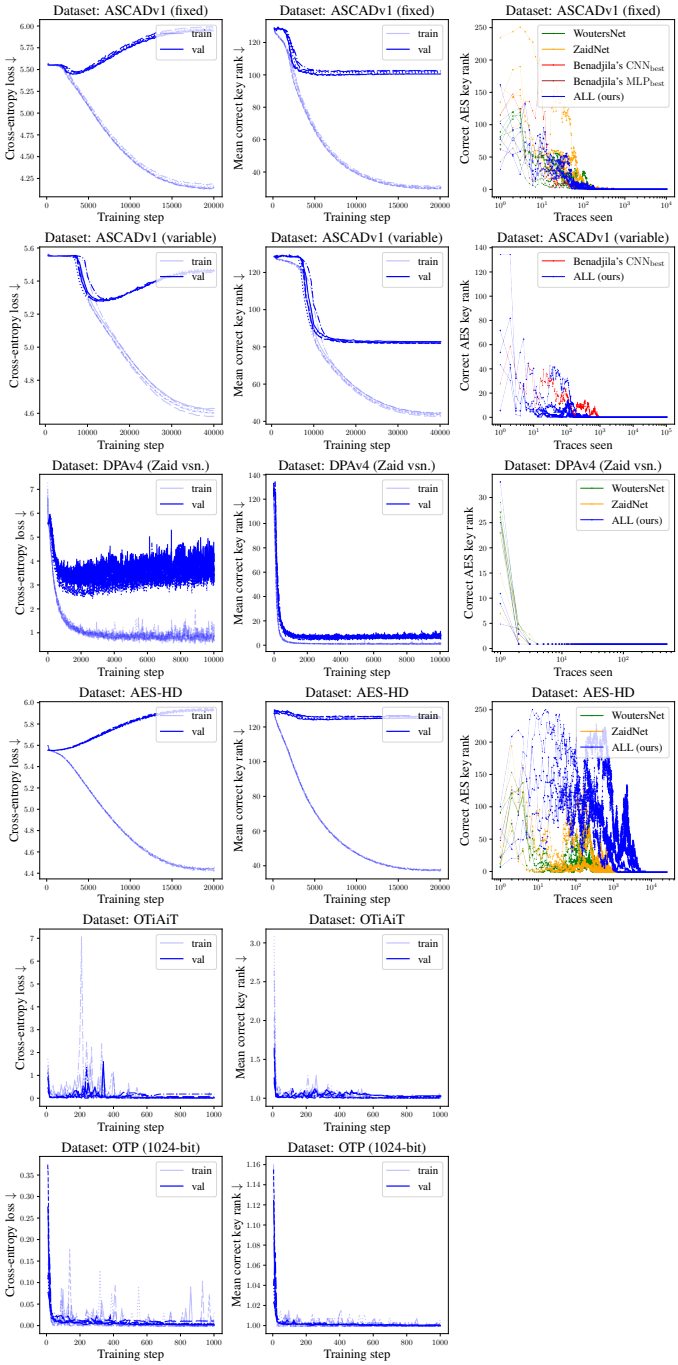

Figure 17: Training curves and attack performance of the supervised classifiers which are 'interpreted' by the deep learning baseline methods. (**Left column**) The training + validation cross-entropy loss vs. training steps for the supervised classifiers. (**center column**) The training + validation rank vs. training steps for the supervised classifiers. (**right column**) The rank of the correct key as we accumulate predictions on the attack dataset for the supervised classifiers, which is a common way of evaluating attack performance in the side-channel literature. For reference, we superimpose the results using open-weight classifiers provided by Benadjila et al. (2020) and Wouters et al. (2020). Note that our classifiers can successfully attack all datasets. They get comparable performance to the open-weight classifiers on ASCADv1-fixed, ASCADv1-variable and DPAv4, and somewhat worse performance on AES-HD.

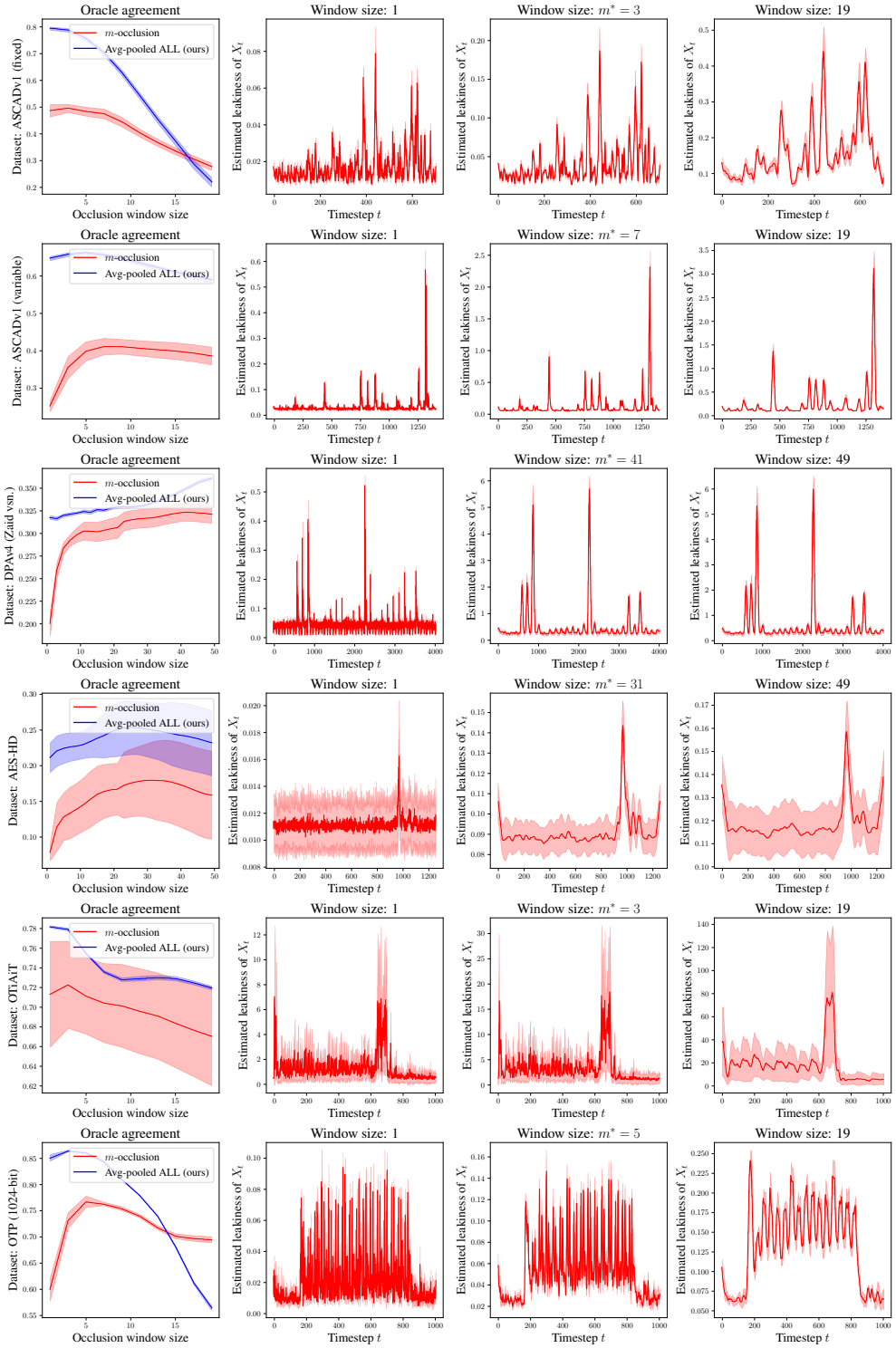

Figure 18: A sweep of the window size m for m-occlusion. In the **first column** we plot the oracle agreement vs. window size (red curve). For reference we also plot the output of ALL (blue curve) as we average-pool it with stride 1 and kernel size m. Note that ALL consistently outperforms m-occlusion for a wide range of window sizes. In the **second, third and fourth columns** we plot the m-occlusion leakiness assessment for $m = 1$, optimal m, and maximum considered m. Observe that there is a smoothing effect as we increase m.

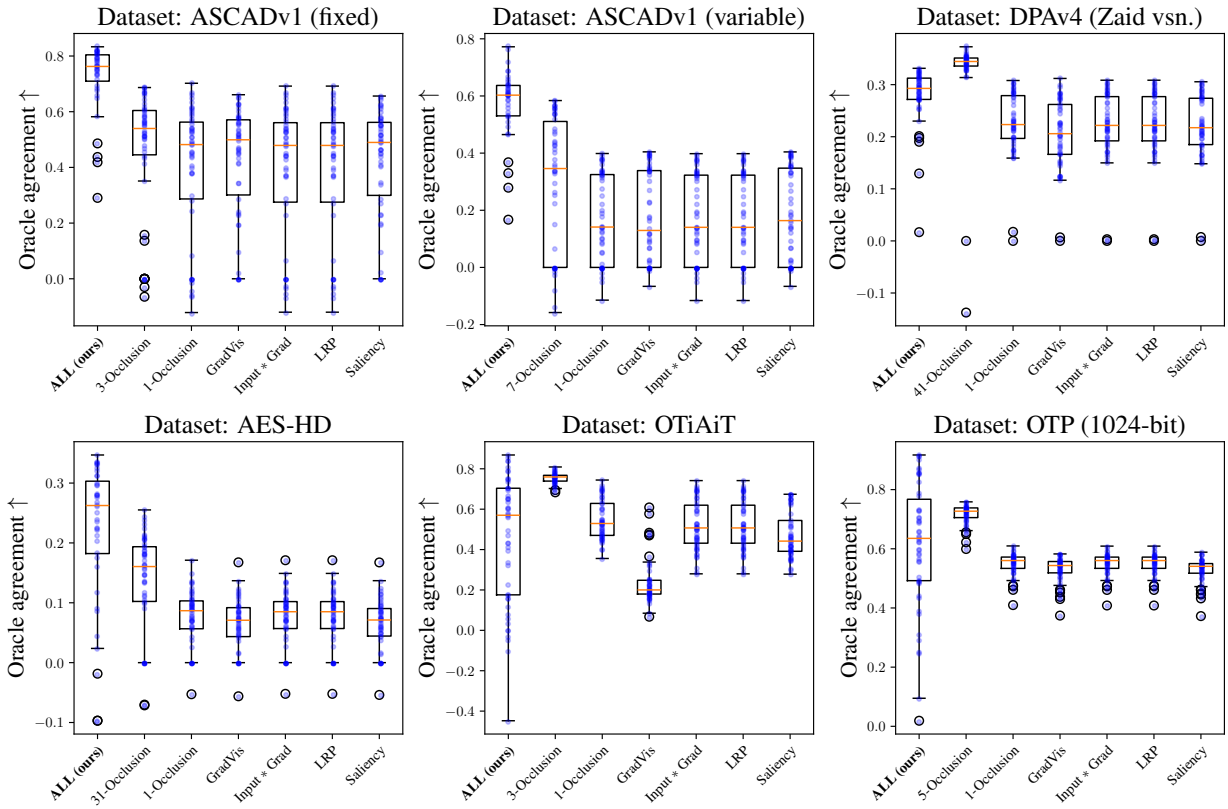

Figure 19: Distribution of performance of ALL and selected baseline methods during a 50-trial random hyperparameter search. Note that ALL generally outperforms baselines over a wide range of configurations. Blue dots denote individual samples, and boxes extend from first quartile to third quartile with a line at the median and whiskers extending to the furthest dot lying within 1.5× the interquartile range from the box.

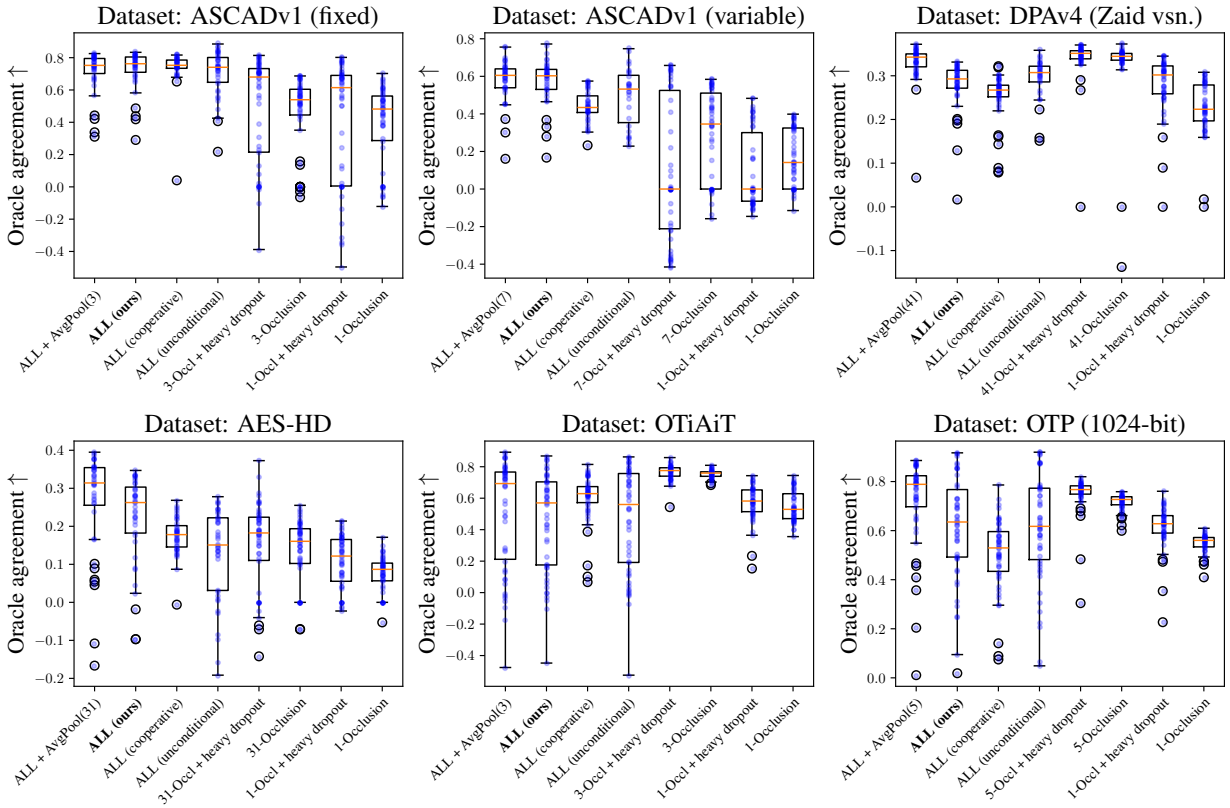

Figure 20: Ablation studies where we evaluate the influence of individual aspects of ALL on its performance gains relative to prior work, as described in Sec. E.3.7. These are plots of the distribution of oracle agreement values after a 50-trial random hyperparameter search with each ablation in place. 'ALL (ours)' denotes our method without modification. 'ALL + AvgPool(m^*)' denotes an average-pooled version of ALL to mimic the smoothing effect of m^*-occlusion. 'ALL (cooperative)' denotes ALL with the adversarial objective replaced by a 'cooperative' objective where both the classifier and noise distribution are trained to maximize the performance of the classifier. 'ALL (unconditional)' denotes ALL without the occlusion masks being fed as an auxiliary input to the classifier. 'm-Occl + heavy dropout' denotes m-occlusion with the input dropout to the classifier chosen from $\{0.05, 0.1, \ldots, 0.95\}$ rather than $\{0.0, 0.1\}$ as in our other experiments. 'm-occlusion' denote the results with the unmodified m-occlusion techniques.