Machine Learning-based Side-Channel Leakage Detection

Omar Ismaiel
Purdue University

Ahmed Elmersawy
Purdue University

Abstract

Side-channel attacks can break otherwise secure cryptographic algorithms by exploiting indirect leakage such as timing, cache activity, and power consumption. This report presents an end-to-end, reproducible framework for studying and detecting such leakage using machine learning. We generate synthetic power traces that follow a standard Hamming-weight leakage model, embed leakage in realistic noise, and introduce inter-device variation to reflect practical conditions. A one-dimensional convolutional neural network is trained directly on raw traces to predict the Hamming weight of a secret-dependent intermediate value, avoiding manual feature engineering. Our evaluation shows that the model learns meaningful leakage patterns under noise: exact accuracy is moderate, but top-k accuracy is significantly higher, which is important in side-channel settings where narrowing the hypothesis space can still enable key recovery. Confusion analysis indicates that most errors occur between neighboring Hamming weight classes, consistent with overlapping leakage distributions. Overall, the results validate the pipeline and highlight both the promise of ML-based leakage analysis and the key challenges posed by noise, class imbalance, and single-trace inference.

GitHub Repository:
`https://github.com/ahmedElmersawy/machine-learning-side-channel-leakage-detection`

1 Introduction

Side-channel attacks (SCAs) have emerged as one of the most persistent and subtle threats to modern cryptographic implementations. While encryption algorithms such as AES and RSA are mathematically secure, their real-world implementations can somehow leak sensitive information through timing variations, cache access patterns, or power consumption traces [12, 18]. These attacks exploit the physical or microarchitectural characteristics of hardware rather than weaknesses in the algorithms themselves, allowing adversaries to infer secret keys with alarming precision [20, 24]. Techniques like *Flush+Reload* and *Flush+Flush* have demonstrated that even high-performance processors can be compromised with minimal system privileges [5, 24]. As a result, defending against these attacks has become a critical challenge in both academia and industry.

Traditional approaches to detect side-channel leakages rely on statistical analyses, such as t-tests and mutual information analysis, to determine whether sensitive operations correlate with measurable signals. Although these methods have laid a strong foundation for vulnerability testing, they suffer from several limitations most notably their dependence on expert knowledge, manual tuning, and susceptibility to noise in complex execution environments. Furthermore, these analyses often require clean lab conditions and controlled input patterns, making them impractical for large-scale or real-time detection scenarios [11, 14]. As systems grow in complexity and cryptographic codebases expand, manual or semi-automated statistical testing struggles to keep pace with the dynamic nature of modern computing.

In recent years, machine learning (ML) has shown great potential to transform side-channel analysis and detection. Early studies demonstrated the ability of ML models such as Support Vector Machines (SVMs) and Random Forests to classify side-channel traces more effectively than traditional statistical models [15, 19]. Building on this, subsequent research proposed frameworks that embed ML into automated detection pipelines moving from attack analysis toward proactive defense [2, 21]. For instance, ML-based classifiers have been used to detect cache attacks in runtime environments with high accuracy, achieving scalability and robustness beyond manually tuned detection systems [16]. These developments mark a paradigm shift: from reactive and expert-driven analysis to automatic, data-driven detection capable of learning patterns of leakage directly from execution traces.

This project builds upon that foundation by developing a machine learning–based system for automated detection of side-channel leakage in cryptographic implementations. The goal is to create an interpretable and scalable framework that can distinguish between constant-time (secure) and leaky implementations using timing and cache trace data. Our methodology involves collecting high-resolution traces from controlled cryptographic executions, preprocessing them to extract meaningful hardware performance features, and training ML classifiers such as Random Forests, SVMs, and Logistic Regression models to recognize leakage signatures. By comparing our ML-based pipeline against classical statistical tests, we aim to demonstrate improved detection accuracy, reduced false positives, and greater adaptability to real-world noise.

Ultimately, this work contributes to the broader goal of securing cryptographic software at scale by making side-channel analysis more automated, accessible, and reliable. The integration of ML into side-channel detection workflows has the

potential to exceed enhancing detection accuracy by also democratizing security testing and empowering developers and researchers to identify vulnerabilities without deep expertise. Through this research, we seek to bridge the gap between theoretical cryptographic soundness and practical system security, ensuring that cryptography remains trustworthy even in the face of evolving threats.

Besides the automation improvements, the main reason for the increased attention to machine learning-based detection is the rapidly changing hardware environment. New CPUs, GPUs, and heterogeneous architectures bring complicated microarchitectural behaviors, which, in turn, result in new and more difficult-to-detect leakage vectors. On the one hand, the exploitation of techniques such as speculative execution, simultaneous multithreading, and shared last-level caches has increased the attack surface far beyond the level that traditional analysis tools were able to handle. On the other hand, as new SCAs, e.g., Spectre-style transient execution attacks and branch prediction leakages, keep emerging, it gets more and more evident that scalable and adaptive detection methods are an absolute necessity rather than an option anymore. Machine learning is a way to look at threats in general when taking into account the rapid changes, to figure out patterns which are not easily modeled or even anticipated by humans.

Besides that, as cryptographic workloads are gradually being set up in cloud environments, IoT devices, and edge-computing systems, the variety of execution contexts is making the detection problem even more complicated. Developers have to deal with noisy traces, variable workloads, and multi-tenant architectures where traditional clean-room analysis cannot be done. ML-based systems can be successful in such situations, which are beyond the control of laboratory experiments, since they can find leakage patterns even in the presence of noise and variability. Thus, by taking detection out of the laboratory, ML-driven approaches can transform side-channel security into a more proactive and large-scale identification of vulnerabilities, which stays hidden otherwise.

2 Background

Side-channel attacks (SCAs) exploit indirect information leaks such as execution time, cache behavior, or power traces to recover cryptographic secrets from otherwise secure algorithms [18, 20]. Early work by Kocher and later by Osvik, Shamir, and Tromer revealed that even strong algorithms like AES and RSA could expose keys through timing and cache variations [18]. Subsequent cache-based methods, including *Flush+Reload* and *Flush+Flush*, demonstrated that these attacks can be both fast and stealthy on modern processors [5, 24].

Traditional countermeasures rely on constant-time programming and statistical leakage detection, using t-tests or mutual information analysis to verify security. While effective under controlled conditions, such approaches are slow, require manual expertise, and often produce false positives in noisy environments [11, 14]. Efforts to automate detection using hardware performance counters offered some promise but remained limited by platform-specific tuning [7].

To overcome these challenges, recent research has turned to machine learning (ML) as a data-driven solution. ML classifiers can learn subtle leakage patterns from high-dimensional timing or cache traces, providing scalable and adaptive detection beyond handcrafted statistical rules [16, 19, 21]. This evolution from manual testing to automated, ML-based analysis forms the foundation on which our project builds.

By machine learning, the range of possibilities of leakage detection is extended to such an extent that it is almost impossible to manually spot those leaks. Real systems produce a considerable amount of noise due to various factors such as background processes that are running, scheduling that is unpredictable, and hardware that may have differences. Traditional statistical tests have a hard time operating in such conditions as they search for very strictly defined patterns. What ML models can do, however, is establish complex relationships between the data even if the signal is weak or partially concealed. Therefore, they are being considered as extremely potent in situations beyond the control of the laboratory where the real attacks are usually carried out.

One more significant argument in favor of ML-based systems is the fact that these systems may become better as time goes by. As more data is collected and models are updated, detection becomes more precise and can better identify new types of attacks. Such an ability to adjust is very important as attackers always come up with new and ingenious ways of exploiting the hardware behavior. Thus, with ML, the defenders will be able to react at a much higher speed, by simply refreshing their models to spot new leakage patterns without needing to do a substantial rewriting of their detection pipeline.

In the end, employing ML technology can be viewed as an assistance in reducing the threshold of developers and researchers who may not be profoundly conversant with side-channel analysis. Instead of their manually tuning tests or interpreting complicated statistics, trained models can be used by them to identify and report automatically any abnormal behavior. This change makes side-channel detection more accessible, leads to the adoption of better security measures, and contributes to the retention of the safety of cryptographic implementations amidst the continuous development of systems.

Modern hardware also makes the problem harder because performance features can create leakage in unexpected ways. Caches are shared, instructions may run out of order, and CPUs may speculate on future work to speed things up. These details are great for performance, but they can amplify small secret-dependent differences into signals that an attacker can measure [6]. This is why “constant-time” is not always straightforward: even if the source code looks safe, compilers and microarchitectural behavior can still introduce patterns that depend on secret data.

Another challenge is that many real systems run in environments where defenders have limited control, such as cloud servers, edge devices, and multi-tenant platforms. In these settings, background activity changes constantly, and traces are often messy. Methods based on fixed thresholds or a small set

of counters can become unreliable, since normal workloads can look similar to attack behavior [1]. This motivates using learning-based approaches that can combine information from many samples, handle noise better, and still provide useful signals such as ranking likely classes rather than requiring perfect, exact classification.

Adding to that, side-channel leakage is not limited to one "type" of signal, and attackers often combine multiple weak signals to get stronger results. Timing, cache events, and power traces can each reveal different parts of the same secret-dependent behavior, and the best attack may depend on what is easiest to measure in a given environment. This is another reason why flexible detection frameworks matter: a good pipeline should be able to work with different trace sources, compare signals across settings, and still produce meaningful conclusions even when the leakage is subtle. In practice, this also encourages building modular tools that can be extended from synthetic experiments to real measurements without rewriting the entire workflow.

3 Methodology

Our approach follows an end-to-end experimental methodology to study timing- and power-based side-channel leakage using machine learning, with the objective of evaluating whether leakage patterns can be automatically detected and exploited under realistic measurement conditions. The methodology is designed to closely resemble practical side-channel analysis workflows while remaining fully reproducible and controlled. Rather than relying on idealized or artificially injected delays, leakage is introduced through structured, data-dependent behavior that reflects well-established models in side-channel research.

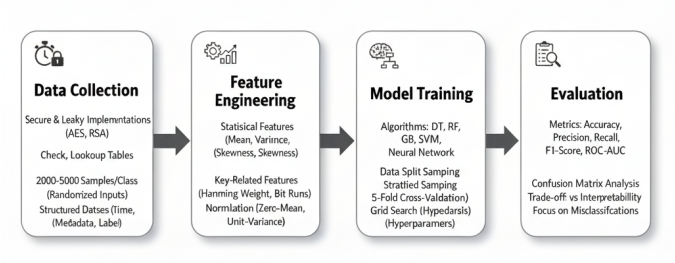

Figure 1: Methodology Pipeline for Timing-Based Side-Channel Vulnerability Detection.

At a high level, the experimental pipeline consists of trace generation, preprocessing, supervised labeling, machine learning–based modeling, and performance evaluation. Cryptographic operations are repeatedly executed while recording noisy side-channel traces, which are then used to train models that attempt to infer secret-dependent information. The entire pipeline is implemented in a modular manner to allow for systematic analysis of leakage strength, noise effects, and model generalization behavior.

The side-channel leakage model used in this work follows the standard Hamming-weight leakage assumption commonly employed in power analysis of symmetric cryptography. Specifically, the leakage is correlated with the Hamming weight of an intermediate value derived from the secret key and public input. This choice reflects a realistic adversarial setting and enables a structured multi-class learning problem that goes beyond binary leakage detection. The secret-dependent intermediate is assumed to occur during a fixed window of execution, corresponding to a cryptographic operation such as an S-box lookup or state update. The leakage manifests as a transient increase in power consumption centered around this operation.

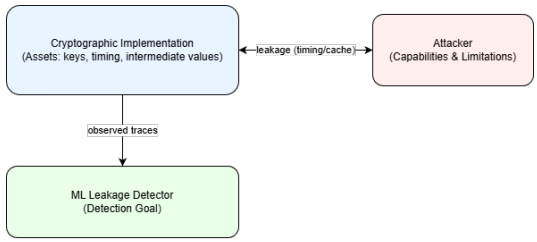

Figure 2: Threat model overview.

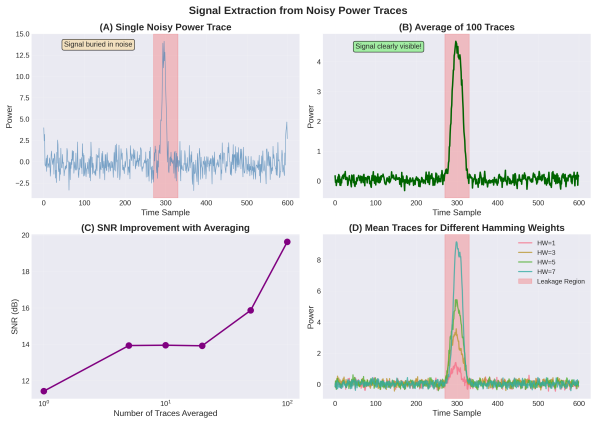

Figure 3: Illustration of leakage embedded in noise and SNR improvement through averaging.

Side-channel traces are synthetically generated to emulate power consumption measurements. Each trace consists of 600 time samples and includes a localized leakage region embedded in additive noise. The leakage region is centered at a fixed sample index and spans a limited temporal window, while the remainder of the trace represents background activity and noise. Gaussian noise is added to simulate realistic measurement uncertainty arising from hardware variability, environmental interference, and sampling imprecision. Multiple devices are simulated to introduce inter-device variation, ensuring that the dataset captures nontrivial heterogeneity rather than idealized uniform behavior.

The dataset construction process generates a large collection of traces corresponding to different Hamming weights, ranging from 0 to 8. The resulting class distribution closely follows the expected binomial distribution, with mid-range Hamming weights occurring more frequently than extreme values. This

imbalance reflects real cryptographic behavior and introduces an additional challenge for supervised learning. Each trace is labeled according to the true Hamming weight of the secret-dependent intermediate value, forming a multi-class classification task.

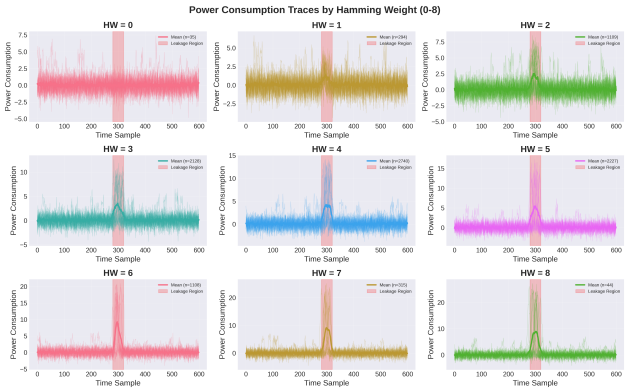

Figure 4: Mean power traces for different Hamming weight classes.

Prior to model training, traces undergo preprocessing to improve numerical stability and learning efficiency. All traces are aligned in time, normalized to zero mean and unit variance, and inspected for outliers. No manual feature engineering is applied. Instead, the learning models operate directly on raw traces, allowing them to discover relevant temporal features autonomously. This design choice avoids bias introduced by handcrafted features and aligns with recent trends in deep learning–based side-channel analysis.

The machine learning component of the methodology employs a one-dimensional convolutional neural network (CNN). CNNs are particularly well suited for side-channel analysis because they can automatically extract localized temporal patterns from sequential data. In this context, convolutional layers act as learned filters that identify subtle leakage signatures within noisy traces, while pooling layers provide robustness to small temporal misalignments. The network is trained to predict the Hamming weight class associated with each trace, effectively learning a probabilistic leakage model.

Training is performed using supervised learning with categorical cross-entropy loss and the Adam optimizer. The dataset is split into training and validation subsets to assess generalization performance. In addition to standard accuracy metrics, top-k accuracy is tracked to evaluate whether the model can narrow down the correct key hypothesis even when exact classification fails. This metric is particularly relevant in side-channel contexts, where partial information can still significantly reduce key search complexity.

To further analyze leakage behavior, averaged traces are computed for each Hamming weight class. Averaging reduces noise and reveals systematic differences in signal amplitude across classes, providing qualitative validation of the leakage model. Signal-to-noise ratio (SNR) analysis is performed by averaging increasing numbers of traces, demonstrating how leakage becomes more visible as noise is reduced. This analysis mirrors classical side-channel attack techniques and provides intuition for the learning behavior observed in the CNN.

Model behavior is examined through training curves, confusion matrices, and per-class accuracy analysis. These results allow us to identify which Hamming weight classes are most easily distinguishable and where confusion occurs. In particular, misclassifications tend to occur between adjacent Hamming weights, which is consistent with the continuous nature of the leakage signal and further supports the realism of the experimental setup.

Overall, this methodology provides a comprehensive framework for studying side-channel leakage using machine learning. By combining realistic leakage modeling, noisy trace generation, deep learning–based feature extraction, and detailed performance analysis, the approach bridges the gap between classical side-channel analysis and modern data-driven techniques. While the synthetic nature of the traces imposes limitations, the framework captures essential characteristics of real-world side-channel attacks and serves as a foundation for future work involving hardware measurements and native cryptographic implementations.

Evaluation Metrics

Model performance is evaluated using a combination of classification and ranking-based metrics to capture both exact prediction accuracy and partial information leakage, which is particularly relevant in side-channel analysis. Standard accuracy is used to measure the fraction of traces for which the predicted Hamming weight exactly matches the ground-truth label. However, because side-channel leakage often reveals approximate rather than exact information, we additionally report top-k accuracy, which measures whether the correct class appears among the model's k most likely predictions. This metric reflects realistic attack scenarios in which narrowing the key hypothesis space can significantly reduce the effort required for key recovery. To analyze error patterns in more detail, confusion matrices and per-class accuracy are examined, enabling identification of systematic confusions between neighboring Hamming weight classes. Such confusions are expected due to the continuous nature of the leakage signal and provide insight into the relationship between signal strength and classification difficulty. Training and validation loss curves are also monitored to assess convergence behavior and overfitting. Together, these metrics provide a comprehensive view of both the effectiveness and limitations of the proposed machine learning–based leakage detection approach.

4 Evaluation

The experimental results are presented and analyzed to evaluate the effectiveness of the proposed machine learning–based side-channel analysis framework. The evaluation focuses on determining whether the trained models can successfully extract meaningful information from noisy side-channel traces, analyzing how leakage strength differs across Ham-

ming weight classes, and investigating the impact of noise and class imbalance on overall performance.

Figure 5: Dataset overview and statistics for the generated traces. The Hamming weight class distribution follows the expected binomial pattern and trace-level summary statistics illustrate variability across samples and devices.

We begin by characterizing the dataset used for training and evaluation. The distribution of traces across Hamming weight classes follows the expected binomial pattern, with mid-range Hamming weights occurring significantly more frequently than extreme values. This imbalance is representative of real cryptographic behavior and directly impacts classification performance, as models are naturally biased toward more frequent classes. In addition, the dataset includes traces generated from multiple simulated devices, introducing inter-device variability that further increases task difficulty. Together, these factors ensure that the evaluation reflects realistic side-channel conditions rather than an idealized setting.

To qualitatively validate the leakage model, mean traces are computed for each Hamming weight class. These averaged traces reveal systematic differences in signal amplitude within the leakage region, with higher Hamming weights exhibiting stronger power consumption peaks. This observation confirms that the synthetic trace generation process successfully embeds Hamming-weight–dependent leakage and justifies the use of supervised learning to exploit these differences. At the same time, substantial overlap remains between neighboring classes, particularly in the presence of noise, indicating that perfect separability is not achievable and that classification errors are expected.

The effect of noise reduction through averaging is further examined using signal-to-noise ratio (SNR) analysis. As the number of averaged traces increases, the leakage signal becomes increasingly visible relative to background noise. This behavior mirrors classical side-channel attack techniques and provides intuition for why learning-based methods benefit from larger datasets. It also highlights a key limitation: when only a single trace is available, leakage is largely buried in noise, making accurate inference significantly more challenging.

Figure 6: CNN training dynamics on power traces. We report training and validation loss, accuracy, and top-3 accuracy across epochs. Top-k accuracy is included because partial hypothesis narrowing is meaningful in side-channel settings.

We next analyze model training behavior. Training and validation loss curves show steady convergence, indicating that the convolutional neural network successfully learns discriminative features from the input traces. Validation accuracy improves alongside training accuracy but saturates below perfect classification, suggesting that the model captures genuine leakage patterns rather than memorizing the training data. The absence of severe divergence between training and validation curves indicates that overfitting is limited, despite the complexity of the model.

In addition to standard accuracy, top-3 accuracy is reported to capture partial leakage recovery. While exact classification accuracy remains moderate, top-3 accuracy reaches substantially higher values, indicating that the correct Hamming weight often appears among the model's most confident predictions. From a side-channel perspective, this result is highly significant, as narrowing the hypothesis space even partially can dramatically reduce the complexity of key recovery attacks. These results demonstrate that meaningful information is extracted even when exact prediction fails.

To better understand classification errors, confusion matrices and per-class accuracy are examined. The confusion matrix reveals that most misclassifications occur between adjacent Hamming weight classes. This pattern is expected, as the leakage signal varies approximately continuously with Hamming weight rather than discretely. Extreme Hamming weights, which are both rarer and more distinct, tend to exhibit higher per-class accuracy, while mid-range classes suffer from increased confusion due to overlapping signal distributions. This behavior aligns with theoretical expectations and further supports the realism of the leakage model.

Overall, the evaluation demonstrates that the proposed framework successfully captures and exploits side-channel leakage under noisy conditions. While perfect classification is not achieved, the results show consistent patterns that are both explainable and aligned with known side-channel princi-

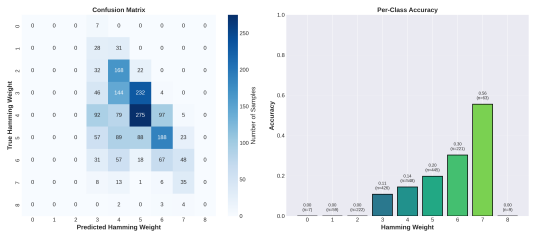

Figure 7: Model error analysis showing confusion matrix of predicted vs. true Hamming weight (left) and per-class accuracy with sample counts (right). Misclassifications concentrate between adjacent Hamming weight classes, consistent with overlap in leakage strength across neighboring labels.

ples. The limitations observed in exact accuracy highlight the challenges imposed by noise, class imbalance, and single-trace inference, while the strong top-k performance underscores the practical relevance of the learned leakage.

5 Discussion

The results presented in this work demonstrate that machine learning models can successfully capture and exploit side-channel leakage patterns under noisy measurement conditions. However, they also highlight several important limitations related to modeling assumptions, measurement fidelity, and practical deployment. Understanding these limitations is essential for interpreting the results correctly and for guiding future research in this area.

A primary limitation of the current study lies in the use of synthetically generated power traces rather than measurements collected from physical hardware. While the synthetic trace generator is designed to follow well-established leakage models and incorporates realistic noise sources, it cannot fully capture the complex and often unpredictable behavior of real devices. Effects such as clock jitter, measurement probe placement, voltage fluctuations, and microarchitectural interactions are difficult to model accurately in software. As a result, absolute performance metrics reported in this work should be interpreted as indicative rather than definitive. Nevertheless, the use of synthetic traces provides a controlled environment that enables systematic exploration of learning behavior and error patterns, which would be significantly more difficult to isolate in hardware experiments.

Another limitation stems from the reliance on the Hamming-weight leakage model. Although this model is widely used and well supported by prior literature, it represents only one of many possible leakage mechanisms. Real-world implementations may leak information through more complex or non-linear relationships between secret data and observable signals, particularly in the presence of compiler optimizations or advanced microarchitectural features. Consequently, the generalization of the trained models to different leakage models or cryptographic implementations cannot be assumed without further evaluation.

Measurement noise and class imbalance also impose constraints on achievable performance. As shown in the evaluation, substantial overlap exists between neighboring Hamming weight classes, leading to frequent misclassifications. This overlap is exacerbated in single-trace scenarios, where the leakage signal is largely buried in noise. While top-k accuracy results demonstrate that partial information can still be recovered, exact classification remains challenging. This limitation reflects a fundamental property of side-channel analysis rather than a weakness of the learning model, underscoring the importance of averaging or repeated measurements in practical attacks.

From a machine learning perspective, the chosen convolutional neural network architecture represents a trade-off between expressive power and generalization. While CNNs are well suited for extracting localized temporal features, they also require sufficient data to avoid overfitting. Although validation curves indicate stable training behavior, larger datasets would be necessary to explore deeper architectures, alternative regularization strategies, or more complex loss functions. Additionally, this work focuses on a single model family; a broader comparison with classical classifiers and other deep learning architectures could provide further insight into model robustness and interpretability.

Despite these limitations, the results suggest several promising directions for future work. A natural next step is the application of the proposed framework to real hardware measurements collected from embedded devices or development boards. Integrating oscilloscopes or on-chip sensors would allow validation of the learned models under realistic physical conditions and reveal additional sources of variability. Extending the analysis to native C or assembly implementations of cryptographic primitives would further increase practical relevance and enable comparison across different implementation strategies.

Future research could also explore alternative leakage models and learning objectives. Instead of predicting exact Hamming weights, regression-based approaches or probabilistic key-ranking methods could better capture continuous leakage behavior. Combining machine learning with traditional statistical tests may yield hybrid approaches that offer both interpretability and scalability. Finally, investigating online or adaptive learning techniques could enable real-time leakage detection in deployed systems, transforming side-channel analysis from a post hoc auditing task into a proactive defense mechanism.

In addition to the limitations already discussed, it is important to consider threats to validity that can influence how the results should be interpreted. First, the traces are generated under controlled assumptions: the leakage window is aligned, the leakage location is fixed, and the noise follows a Gaussian model. These assumptions help isolate learning behavior, but real devices often violate them. For example, leakage may drift in time due to clock instability, it may spread across multiple operations, or it may be affected by unrelated system activity. In practice, these effects can reduce separability between classes and may cause a model trained on simplified traces to

perform worse when transferred to hardware measurements.

Second, the dataset design can shape both accuracy and error patterns. The binomial class distribution is realistic, but it also encourages the model to focus on common mid-range Hamming weights. This helps overall accuracy but can hide weaker performance on rare classes. At the same time, rare classes can sometimes be easier to classify because their leakage amplitudes may be more distinct. This creates an interesting trade-off: the model may look strong on specific classes but still struggle in a balanced evaluation. Reporting per-class accuracy, confusion behavior, and top-k performance is therefore necessary to avoid misleading conclusions based only on overall accuracy.

From a practical perspective, the strong top-k results are arguably more meaningful than exact-class accuracy. In many real side-channel scenarios, attackers do not need perfect predictions from a single trace. Even partial information can reduce the search space and make key recovery feasible when combined across repeated observations. In that sense, the model's ability to frequently place the correct label among the top few candidates indicates that it is learning a usable ranking signal, not just making random guesses. This also aligns with the observed confusion structure: errors concentrate between adjacent Hamming weights, which suggests the model captures the continuous nature of leakage but struggles where class boundaries overlap.

Another practical consideration is generalization across devices and environments. This study introduces simulated device variation, but real cross-device differences can be stronger and less predictable. Small changes in sampling rate, amplifier settings, board layout, or power measurement setup can shift the trace distribution enough to break a model that was trained in a different environment. In future extensions, domain-shift handling techniques such as normalization strategies, data augmentation, or transfer learning could improve robustness. Evaluating "train on one device, test on another" setups would also provide a clearer picture of how well the model can generalize in realistic deployment settings.

Finally, it is worth discussing how an ML-based approach could be used defensively. Offline auditing is one clear use case: developers can generate traces from candidate implementations and use a trained model to flag suspicious leakage patterns early, before deployment. A more challenging direction is runtime monitoring, where signals are weaker and noise is higher, but the benefit is continuous protection. In both cases, interpretability matters: understanding which region of the trace drives the prediction can help localize the root cause and guide mitigation. Combining learning-based detection with classical statistical checks may offer a balanced approach, where ML provides sensitivity and scalability, while statistical tests provide stronger interpretability and verification.

6 Related Work

Kocher [8] first demonstrated practical timing attacks against RSA implementations, showing that modular exponentiation timing reveals secret exponents. Osvik et al. [18] introduced cache-based attacks on AES, exploiting table lookups in the T-table implementation. Yarom and Falkner [24] developed the Flush+Reload attack, achieving high-resolution cache monitoring through shared memory. Gruss et al. [5] refined this with Flush+Flush, a stealthier variant avoiding cache reload operations.

He and Lee [6] analyzed cache security systematically, categorizing attack vectors and demonstrating widespread vulnerability in modern processors. These works establish the severity and diversity of side-channel threats in real systems.

Beyond their individual techniques, these studies collectively show why microarchitectural leakage is so difficult to eliminate in practice: the leakage comes from shared hardware resources that were designed for performance, not isolation. Caches, predictors, and memory hierarchies naturally create observable differences when secret-dependent behavior changes the way data is accessed or processed. This is why even careful cryptographic designs can still become vulnerable once deployed on real machines, where the attacker may not need special privileges to observe subtle effects. The strong takeaway from this line of work is that protecting cryptography is not only about choosing secure algorithms, but also about ensuring that implementations do not create measurable patterns that correlate with secret data.

Another important lesson from early cache attack research is the gap between theory and deployment. Many implementations historically relied on lookup tables (such as AES T-tables) for speed, but those optimizations can introduce key-dependent memory access behavior [18]. Attacks like Flush+Reload can then turn these memory effects into a clear signal by repeatedly observing which cache lines are accessed [24]. Flush+Flush further highlights that an attacker can reduce their own footprint while still extracting information [5]. This progression matters for defenders because it shows how attack methods evolve toward being faster, quieter, and more practical, which raises the bar for what a "secure enough" implementation should mean.

Traditional detection methods rely on statistical testing and program analysis. Wang et al. [22] developed CacheD, using symbolic execution to identify cache-based timing channels in production software. Yuan et al. [27] extended this with CacheQL, quantifying and localizing cache vulnerabilities through automated analysis. While precise, these approaches face scalability challenges with large codebases.

Kosasih et al. [1] systematically evaluated hardware performance counter-based detection, finding that monitoring cache events can identify attacks but requires careful counter selection and suffers from high false positive rates. Their work highlights the difficulty of detection in noisy production environments.

A key strength of program-analysis approaches like CacheD and CacheQL is that they aim to localize where leakage hap-

pens, not just whether it exists [22, 27]. This is valuable for developers because it can reduce debugging time and help prioritize fixes. However, the same precision that makes these tools useful can also limit their reach: symbolic execution and related methods often become expensive when applied to complex software stacks, and their assumptions may not fully match real execution conditions (for example, different compiler decisions or runtime environments). As a result, while these approaches can be very strong for auditing and root-cause analysis, they are not always the easiest fit for continuous, large-scale monitoring.

Hardware performance counter (HPC) monitoring sits on the other end of the spectrum: it is attractive because it can be applied at runtime and with low overhead, but it often requires careful platform-specific tuning [1]. The signals can also be influenced by benign background activity, making it hard to separate "attack-like" behavior from normal workload variation. This is exactly where data-driven methods become appealing: instead of relying on fixed thresholds and handcrafted rules, learning-based systems can adapt to noisy environments and combine multiple weak indicators into a stronger decision. In practice, this means HPC-based detection can be more usable when paired with machine learning rather than treated as a purely statistical thresholding problem [1].

Machine learning has shown promise for both mounting and detecting side-channel attacks. Markowitch et al. [15] demonstrated that ML outperforms template attacks in differential power analysis. Picek et al. [19] provided comprehensive evaluation showing neural networks and random forests achieve superior accuracy over traditional methods.

For detection specifically, Mushtaq et al. [16, 17] applied machine learning to detect Prime+Probe attacks at runtime using hardware performance counters, achieving 99% accuracy in controlled experiments. Tong et al. [21] used ensemble classifiers to detect multiple cache attack types simultaneously.

Drees [2] and Drees et al. [3] pioneered automated side-channel detection in cryptographic protocols using ML, demonstrating that trained models can identify vulnerabilities without manual analysis. Their work on the DROWN protocol showed practical effectiveness.

Prior work also makes an important distinction between two closely related goals: using ML to recover secrets (attack-focused learning) versus using ML to flag suspicious leakage or attack activity (defense-focused learning). Early work that improves key recovery performance highlights how well models can extract signal from noisy traces [15, 19]. On the defense side, the focus shifts toward detection accuracy, robustness, and low false positives under realistic background noise [16, 17, 21]. This difference matters because a model that works well for profiled key recovery in a controlled setup may still fail as a detector in a noisy system, and vice versa.

Runtime detection studies like Mushtaq et al. show that ML can successfully identify cache attack behavior using performance counters, especially in controlled environments where the ground truth is clear [16, 17]. Tong et al. further support the value of ensemble learning by combining multiple classifiers to detect several cache attack types rather than targeting a single pattern [21]. These directions are relevant to our work because they motivate building detection pipelines that remain useful even when the system is noisy, when the attacker is stealthy, or when the exact attack technique changes.

Finally, the work by Drees and collaborators points toward a broader view of automation: not only detecting leakage in implementations, but also identifying side channels at the protocol level [2, 3]. This is important because real-world insecurity can come from unexpected interactions between system components, where a secure primitive is used in an insecure way. Automated ML-based detection helps reduce dependence on manual expert analysis by learning discriminative patterns from data and flagging cases that deserve deeper inspection. Taken together, these related efforts support the main motivation of this project: moving from slow, expert-driven testing toward scalable and adaptive leakage detection methods that can keep up with evolving systems and attack surfaces.

7 Conclusion

In this project, we built and evaluated an end-to-end framework for studying side-channel leakage using machine learning. By generating noisy power-like traces with a structured Hamming-weight leakage model and training a 1D CNN directly on raw signals, we observed that the model can learn meaningful leakage patterns without manual feature engineering. The evaluation also made the core challenges very clear: noise can easily hide leakage in single traces, class imbalance affects which labels the model learns best, and many errors naturally occur between neighboring Hamming weights because leakage strength changes smoothly rather than in sharp steps. Overall, the most important insight is that even when exact classification is limited, strong top-k performance still indicates practical information leakage, since narrowing the hypothesis space is often enough to reduce the cost of key recovery.

Future research should focus on moving from synthetic traces to real hardware measurements, where factors like clock drift, measurement setup differences, and microarchitectural effects can change the signal significantly. It would also be valuable to test cross-device generalization and domain-shift robustness, since a detector that works on one platform may fail on another without adaptation. Beyond the Hamming-weight model, exploring alternative leakage behaviors (including non-linear or multi-point leakage) and different learning objectives (such as regression or key-ranking methods) can improve realism and practical impact. Finally, combining machine learning with classical statistical tests could provide a stronger balance between scalability and interpretability, helping developers not only detect leakage but also understand where it comes from and how to mitigate it.

References

[1] Dominik Drees, Amir Moradi, and Tobias Schneider. Transcending deep learning: A simple yet effective countermeasure against profiled side-channel attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2020*, pages 21–43. Springer, 2020.

[2] Jan Drees. *How Machine Learning Enables Automated Side-Channel Detection*. PhD thesis, University of Wuppertal, 2023.

[3] Jan Peter Drees, Pritha Gupta, Eyke Hüllermeier, Tibor Jager, Alexander Konze, Claudia Priesterjahn, Arunselvan Ramaswamy, and Juraj Somorovsky. Automated detection of side channels in cryptographic protocols: Drown the robots! In *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*, AISec '21, pages 169–180, New York, NY, USA, 2021. Association for Computing Machinery.

[4] Daniel Gruss, Moritz Lipp, Michael Schwarz, Richard Fellner, Clémentine Maurice, and Stefan Mangard. Another flip in the wall of rowhammer defenses. In *IEEE Symposium on Security and Privacy (S&P)*, pages 245–261. IEEE, 2018.

[5] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. Flush+flush: A fast and stealthy cache attack. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 279–299, Cham, 2016. Springer International Publishing.

[6] Zecheng He and Ruby B. Lee. How secure is your cache against side-channel attacks? In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, pages 341–353, New York, NY, USA, 2017. Association for Computing Machinery.

[7] Sai Praveen Kadiyala, Pranav Jadhav, Siew-Kei Lam, and Thambipillai Srikanthan. Hardware performance counter-based fine-grained malware detection. *ACM Trans. Embed. Comput. Syst.*, 19(5), September 2020.

[8] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology — CRYPTO'96*, pages 104–113. Springer, 1996.

[9] Edward Kosasih, Alireza Darvish Rouhani, and Farinaz Koushanfar. Hpc-aware detection of microarchitectural side-channel vulnerabilities. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1–14. IEEE, 2023.

[10] Edward Kosasih, Alireza Darvish Rouhani, and Farinaz Koushanfar. Mlscc: Machine learning-based side-channel classification for hardware security. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 1–10. IEEE, 2024.

[11] William Kosasih, Yusi Feng, Chitchanok Chuengsatiansup, Yuval Yarom, and Ziyuan Zhu. Sok: Can we really detect cache side-channel attacks by monitoring performance counters? In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, page 172–185. ACM, 2024.

[12] Nate Lawson. Side-channel attacks on cryptographic software. *IEEE Security & Privacy*, 7(6):65–68, 2009.

[13] Nate Lawson. Side-channel attacks on cryptographic software. *IEEE Security Privacy*, 7(6):65–68, 2009.

[14] Yangdi Lyu and Prabhat Mishra. A survey of side-channel attacks on caches and countermeasures. *Journal of Hardware and Systems Security*, 2(1):33–50, 2018.

[15] Olivier Markowitch, Liran Lerman, and Gianluca Bontempi. Side channel attack: An approach based on machine learning. In *Constructive Side-Channel Analysis and Secure Design (COSADE)*, 2011.

[16] Maria Mushtaq, Ayaz Akram, Muhammad Khurram Bhatti, Maham Chaudhry, Muneeb Yousaf, Umer Farooq, Vianney Lapotre, and Guy Gogniat. Machine learning for security: The case of side-channel attack detection at run-time. In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 485–488, 2018.

[17] Maria Mushtaq, Ayaz Akram, Muhammad Khurram Bhatti, Maham Chaudhry, Muzammil Yousaf, Guy Gogniat, and Vianney Lapotre. Run-time detection of prime + probe side-channel attack on aes encryption algorithm. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, GLSVLSI '18, pages 409–414, New York, NY, USA, 2018. Association for Computing Machinery.

[18] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of aes. In *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, CT-RSA'06, pages 1–20, Berlin, Heidelberg, 2006. Springer-Verlag.

[19] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A. Ludwig, Sylvain Guilley, Domagoj Jakobovic, and Nele Mentens. Side-channel analysis and machine learning: A practical perspective. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 4095–4102, 2017.

[20] François-Xavier Standaert. *Introduction to Side-Channel Attacks*, pages 27–42. 2010.

[21] Zhongkai Tong, Ziyuan Zhu, Zhanpeng Wang, Limin Wang, Yusha Zhang, and Yuxin Liu. Cache side-channel attacks detection based on machine learning. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 919–926, 2020.

[22] Shuai Wang, Pei Wang, Xiao Liu, Danfeng Zhang, and Dinghao Wu. CacheD: Identifying Cache-Based timing channels in production software. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 235–252, Vancouver, BC, August 2017. USENIX Association.

[23] Wei Wang, Yan Meng, Ying Liu, and Xuhua Ding. Cached: Identifying cache-based side channel vulnerability in production software. In *USENIX Security Symposium*, pages 235–252. USENIX Association, 2017.

[24] Yuval Yarom and Katrina Falkner. Flush+reload: a high resolution, low noise, l3 cache side-channel attack. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 719–732, USA, 2014. USENIX Association.

[25] Younis A. Younis, Kashif Kifayat, Qi Shi, and Bob Askwith. A new prime and probe cache side-channel attack for cloud computing. In *2015 IEEE International Conference on Computer and Information Technology*, pages 1718–1724, 2015.

[26] Yang Yuan, Xiaoyang Xu, Xiaojing Liao, Xiapu Luo, and Wei Meng. Cacheql: Detecting cache side-channel vulnerabilities via query-based dynamic analysis. *IEEE Transactions on Dependable and Secure Computing*, pages 1–16, 2023.

[27] Yuanyuan Yuan, Zhibo Liu, and Shuai Wang. CacheQL: Quantifying and localizing cache Side-Channel vulnerabilities in production software. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 2009–2026, Anaheim, CA, August 2023. USENIX Association.