Robust Single-Trace Full-Key Extraction from Million-Point Traces With Cross-Implementation Transfer

Aron Gohr[1], Friederike Laus[2][0000−0003−3709−0292], and Gregor Leander[3][0000−0002−2579−8587]

[1] cryptosolutions
[2] Independent Researcher
[3] Ruhr University Bochum

Abstract. End-to-end deep-learning side-channel attacks on public-key implementations have recently become possible even for million-sample traces. However, existing methods require large computational resources and extract only partial key shares, which means that dedicated post-processing is required to turn detected leakage into demonstrations of successful key recovery attacks. We present an end-to-end sequence-to-sequence prediction approach to recover complete 256-bit key shares from single raw traces on the SCAAML ECC datasets recently studied by Bursztein et al. [3].

Our solution combines aggressive trace compression for dimensionality reduction with a 1-D U-Net trained using Connectionist Temporal Classification loss. The key idea is to decouple detecting leakage from mapping each leakage site to the correct part of the secret: the network outputs an annotated map of the trace marking likely leakage sites, and a greedy decoder reconstructs the ordered key bits from that map. Using synthetic tasks, we show that this division of labor circumvents a fundamental problem that causes neural network architectures and training methods commonly used in side-channel analysis to struggle with massive multi-target or misaligned extraction tasks.

As a result, we are able to train a single extractor that achieves high accuracy on all four SCAAML ECC datasets in a single training run that takes minutes on a single GPU. The resulting extractors are robust, essentially maintaining their performance under large misalignment (we empirically tested rotations up to 61% of trace length), while degrading gracefully under a variety of trace corruptions, and even time reversal. They transfer across key shares and datasets with little degradation and no retraining. The U-Net outputs also yield prediction maps that localize leakage along the trace prior to decoding.[4]

Keywords: Side-Channel Analysis · Robustness · Deep Learning · U-Net · ECC

[4] Supplementary code and data for this work are available in the repository `https://github.com/agohr/seq2seq-sca`.

1 Introduction

1.1 Motivation and Context

Side-channel analysis (SCA) extracts secret information from cryptographic devices by measuring physical properties such as power consumption or electromagnetic emissions during execution. Deep learning has been widely adopted for partially automating these attacks. However, end-to-end learning of attacks remains challenging, especially for asymmetric targets, due to extremely long traces containing millions of samples. Although recent work has made progress [3,24], existing end-to-end solutions extract only partial keys, require significant computational resources for hyperparameter tuning and training, and face enormous dataset sizes – for example, the full SCAAML ECC CM1 training set is 1.8 *terabytes* [3].

The problems encountered in the processing of long traces using machine-learning methods can be viewed as a manifestation of the *curse of dimensionality* that is widely known in machine learning. Usually, curse of dimensionality problems in machine learning based SCA are addressed by a mix of four different techniques: segmentation, Point of Interest (POI) selection, dimensionality reduction by techniques such as Principal Component Analysis (PCA), and handling of the remaining high-dimensional input space by neural networks. While these techniques do work, they have the following drawbacks:

- *Segmentation, POI selection, and dimensionality reduction* each require significant expertise and make strong simplifying assumptions. Segmentation must be near-perfect for attacks to *work*, and often relies on ad-hoc methods or instrumentation of the profiling target. POI selection and PCA-based dimensionality reduction discard potentially leakage-relevant information without considering the target task, and do not preserve the temporal structure of the data, which conflicts with the inductive biases of the convolutional architectures commonly used in SCA.
- *Classical preprocessing creates brittleness.* These preprocessing steps are far less capable than the neural networks they feed, yet errors propagate through the entire pipeline. In particular, the preprocessing steps cannot learn complex landmarks in the input space, which potentially makes everything that follows brittle against misalignment. It is often hard to see how even a very good model could repair problems that were introduced by the preprocessing steps.
- *Practical workflows accumulate ad-hoc steps.* Extracting multiple targets typically requires training separate models per target and applying manual alignment heuristics. While CNNs can handle some misalignment [4], this tolerance is typically limited to a few percent of a pre-selected window of interest. This is far less than the feature mobility routinely handled in computer vision, and therefore it is unclear why side-channel models cannot be more flexible with respect to misalignment.

In this paper, we address the aforementioned problems with a complete end-to-end solution for single-trace full-key extraction from long traces.

1.2 Contributions

Our central observation is that one main obstacle to scaling deep-learning side-channel attacks to long, unsegmented, multi-target traces is not the difficulty of recognizing leakage in noisy measurement data. Rather, it is an *indexing problem*: standard neural network architectures used in SCA struggle to route many simultaneously detected leakage sites to the correct output positions when feature locations vary between traces. We support this claim through carefully designed synthetic tasks that strip away all noise and leakage-extraction difficulty yet reproduce the same failure modes observed on real data, exposing the architectural bottleneck in isolation. To overcome this bottleneck, we provide the following contributions:

U-Net with CTC Loss and Aggressive Trace Compression We reformulate key extraction as a sequence-to-sequence prediction problem. A 1-D U-Net produces a temporal prediction map that localizes and classifies leakage along the trace; a classical CTC decoder then recovers the ordered key bit sequence from this map. Because the network is only asked to annotate *where* each bit leaks – not to solve the problem of mapping leakage sites to output positions – the architectural limitation that defeats standard approaches is removed entirely. The training pipeline needs no operation-level labels or segmented traces, and can work with misaligned data. Aggressive, leakage-preserving trace compression via low-pass filtering enables up to $\sim 5000\times$ dataset size reduction, making the pipeline computationally lightweight, while robustness to misalignment, trace corruptions, and implementation changes emerges as a natural consequence of the division of labor between the network and the decoding step. Our main training pipeline for the CM1 target also needs roughly 24 times fewer traces than the state-of-the-art attack in [3].

Robustness, Cross-Implementation Transfer, and Test-Time Augmentation Beyond performance, the resulting extractors exhibit surprising robustness and transfer properties: they remain effective under severe misalignment (including large rotations up to $\sim 61\%$ of trace length), realistic trace corruptions (jitter, interrupts, and frequency changes), and even time-reversal of traces. This robustness yields cross-implementation transfer across CM0–CM3 and between key shares without retraining, and it enables test-time augmentation by averaging predictions over perturbed inputs to further improve performance.

Built-in Explainability via Prediction Maps Finally, the U-Net naturally provides built-in explainability: it outputs an explicit temporal prediction map that localizes likely leakage sites along the trace before decoding. In our setting, these maps suggest, for instance, that the implementation processes the key in 4-bit windows, without any prior knowledge of such structure.

Synthetic Tasks for Side-Channel Analysis To highlight the inherent advantage of our method, we demonstrate that standard VGG-like networks fail to learn

how to extract most of the key information from the trace. Furthermore, we illustrate that a specialized architecture discussed in previous multi-target extraction work is able to extract the full first key share of the CM1 target, but struggles with learning or extracting the second key share. Both observations are explained by considering a simple synthetic learning problem that corresponds closely to the decoding stage of the CTC pipeline. Standard SCA neural network architectures fail at this synthetic task in a similar way as they fail on the real data. This shows that at least in some natural settings where leakage is in fact abundant and readily learnable, traditional neural-network based side-channel architectures fail to do so, and motivates the use of techniques that separate the part of the problem that neural networks can easily learn (feature processing and localization) from the part of the problem that is best solved by fixed algorithms (decoding of the leakage map into a representation of the secret).

1.3 Related Work

Previous Work on the SCAAML ECC Datasets The most closely related work to ours addresses end-to-end key extraction from multi-million sample traces in public-key cryptography. Bursztein et al. [3] demonstrated that transformer-based models can extract key bytes directly from full SCAAML ECC traces without manual segmentation, but they extract only partial keys (requiring dedicated post-processing to recover full keys) and require computing-intensive hyperparameter tuning for each target along with high training and inference costs. Our work reuses their main datasets, but extracts complete 256-bit key shares from single traces at much lower computational cost and obtains more robust models.

Long Trace Processing In symmetric SCA, other approaches have been used to successfully extract full keys from long traces. Gohr et al. [7] trained directly on AES traces downsampled by decimation from 650,000 to 65,000 points, using an architecture designed for compression invariance. In [8], this approach was later extended to traces of approximately 219,000 points. It has low computational requirements and works well when the traces are very well-aligned, but is sensitive to misalignment and has difficulties with extracting all key shares already for the SCAAML CM1 target (which has two 256-bit key shares; the approach by Gohr et al. [7,8] is able to extract the first key share, but struggles with the second key share).

Segmentation Without Manual Annotation Several recent works consider automated segmentation of cryptographic traces. Wang et al. [24] use reinforcement learning (Pulse Tailor) to segment traces into operations based on inter-segment similarity, and afterwards apply unsupervised clustering for key extraction and verification that the segmentation worked. Galli et al. [6] (Hound) train a CNN to classify sliding windows as operation boundaries with the help of an instrumented profiling device. Both methods require either specific structural assumptions or instrumented devices for training data collection.

In contrast, our U-Net-with-CTC approach performs sequence-to-sequence extraction directly from raw traces to key bits without explicit segmentation or specially annotated training data, requiring only trace-key pairs for training. The CTC loss automatically learns to localize and extract leakage without operation-level labels, making the method applicable retroactively to existing datasets and robust to variable-length non-cryptographic segments, unknown operation structures, and implementation variations.

1.4 Organization of the Paper

The remainder of the paper is organized as follows: In Section 2, we describe the neural network architectures and in Section 3 the datasets we study. Section 4 details our preprocessing methods, and Section 5 shows the experimental results. Finally, in Section 6 we discuss possible extensions and reflect on the results.

2 Neural Network Architectures

Network Types We use three neural network types in our experiments: VGG-like networks as a standard SCA baseline following [25], subsamplingbased networks specialized for multi-target extraction following [7], and a 1-D U-Net with CTC decoding as our main contribution for end-to-end training without segmentation and minimal preprocessing of the data.

Methodological Choices We deliberately avoid hyperparameter tuning for two main reasons. First, our extractors already achieve accuracy sufficient for single-trace attacks, so gains from hyperparameter tuning would not be cryptographically meaningful in the settings we study. Additionally, hyperparameter tuning is orthogonal to our discussion: one can probably gain *some* performance on our targets by optimizing hyperparameters, but it is unlikely to fix the qualitative problems we find for traditional neural network architectures used in side-channel analysis for instance on the very simple SIFT synthetic task proposed in the present paper. In the context of this paper, hyperparameter tuning would in our view only introduce a confounding additional variable into our experimental setup.

Accordingly, we default to simple standard choices with regard to picking activation functions and main architectural components: all our networks use simple rectified linear units for all internal activations, batch normalization where it seems appropriate, shortcut connections/residual towers for deep architectures, and sigmoid outputs to predict output bit values directly.

Likewise, all of our networks use weak weight regularization by default (L_2-regularization with a factor of 10^{-5}) and the Adam optimizer with a cyclic learning rate schedule. The cyclic learning rate schedule cycles between a warm-up phase and a learning rate decay phase over 20 training epochs, linearly interpolating between maximal and minimal learning rates of 10^{-3} and 10^{-6}.

Implementation Details Our networks are implemented in a TensorFlow-based Python library that we call the `model_library`. Open-source code is available in the supplementary data archive, which contains also the code for the downsampling and segmentation methods used in this paper. Jupyter notebooks within the archive show how the main experiments here reported were run. The datasets themselves are not included in the archive and must be retrieved separately from the source indicated in [3].

In the following, we briefly describe each network architecture used in our experiments.

2.1 1D U-Net with CTC

Overview This architecture combines a 1D U-Net [18] with Connectionist Temporal Classification (CTC) loss [9] for end-to-end key extraction from unsegmented traces. The U-Net's encoder progressively downsamples the input while increasing feature channels; its decoder upsamples back to the original resolution via skip connections from corresponding encoder levels, combining local and global information to produce an output of the same shape as the input. Here, this output is a per-sample probability map over key bit values and a non-leaking (blank) class. U-nets have previously been used in side-channel analysis to denoise trace data [11,27], but to the best of our knowledge not for extraction of secrets. U-nets were originally invented for the segmentation of biomedical images, where segmentation means the creation of an annotation of an image that shows with pixel-level accuracy which objects appear where in the image.

Motivation Intuitively, what image segmentation has in common with side-channel tasks, especially with side-channel tasks on implementations with secret-sharing countermeasures, is the fact that a large number of similar features needs to be extracted from the data with high precision: in the case of image segmentation, the features in question are one object-type label for each point in the image, and in the case of side-channel extraction, it is leakage for each of many internal variables that are of interest to an attacker. Figure 1 illustrates these considerations by showing a plausible segmentation mask layered on top of a photo of a frog.

Training a segmentation network (independently of the architecture used) normally requires detailed per-sample label maps, which in our setting would require annotating each trace point with its leakage contribution or lack of leakage. CTC loss, which was originally developed for sequence-to-sequence speech transcription [9], circumvents this by requiring only traces and target keys. It marginalizes over all possible alignments between the predicted probability map and the target sequence.



Figure 1: Segmentation mask on a frog photo.

At inference time, a decoder assigns symbols to each sample point according to the predicted probabilities, collapses consecutive identical symbols, and removes non-leaking symbols. Training maximizes the log-likelihood of the target sequence

under this decoding process. The next two paragraphs explain the network architecture and training method in detail.

U-nets The 1D U-Net has depth d (default $d = 4$) with a symmetric encoder–decoder and skip connections. Let F denote the base filter count ($F = 64$ in all experiments). Encoder level $i \in \{0, \ldots, d-1\}$ applies a double convolutional block—two sequential $\mathsf{Conv1D}(F \cdot 2^i, k{=}3, \textit{same})$ layers, each followed by batch normalization and ReLU, with dropout between them—and then halves the temporal dimension via $\mathsf{MaxPool1D}(2)$. The bridge applies a double convolutional block with $F \cdot 2^d$ filters. The decoder mirrors the encoder in reverse order: at each level the temporal dimension is doubled by $\mathsf{UpSampling1D}(2)$, the result is concatenated with the corresponding encoder skip connection, and a double convolutional block (without dropout) is applied. Finally, a 1×1 convolution followed by $\mathsf{softmax}$ maps to C output channels, where $C = |\Sigma| + 1$ comprises one channel for each target class in the alphabet Σ plus one blank channel. Because integer floor-division in each $\mathsf{MaxPool1D}(2)$ step can leave the decoder output length slightly below T when T is not divisible by 2^d, a linear interpolation step is applied afterwards to restore the length to exactly T. The overall architecture is illustrated in Figure 2.

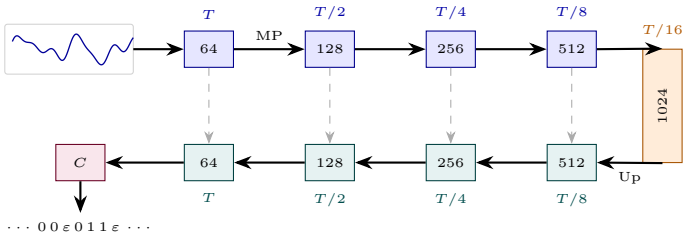


Figure 2: 1D U-Net architecture (d=4, F=64). Numbers give filter counts per double-conv block ($\mathsf{Conv1D} \times 2 + \mathsf{BN} + \mathsf{ReLU}$). The left box shows a stylized raw power trace (network input); the symbol sequence below the purple block shows the decoded per-sample CTC output (ε: blank); blue: encoder; teal: decoder; orange: bridge (1024 filters). Solid arrows: data flow ($\mathtt{MP} = \mathsf{MaxPool1D}(2)$; $\mathtt{Up} = \mathsf{UpSampling1D}(2)$). Dashed arrows: skip connections (feature-map concatenation before each decoder double-conv).

CTC-Decoding and CTC Loss Let Σ denote the label alphabet (in our setting, the two key bit values $\{0, 1\}$) and $\varepsilon \notin \Sigma$ the blank symbol. For a trace of length T, the U-Net outputs a sequence of probability vectors $\mathbf{p}_t \in \Delta^{|\Sigma|+1}$, $t = 1, \ldots, T$. A *path* $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_T) \in (\Sigma \cup \{\varepsilon\})^T$ is mapped to a label sequence by the collapse operator $\mathcal{B}$: merge consecutive repeated symbols, then remove blanks.

The CTC loss for target sequence y is given by

$$\mathcal{L}_{\mathrm{CTC}}(y \mid x) = -\log \sum_{\boldsymbol{\pi}\colon\, \mathcal{B}(\boldsymbol{\pi})=y} \prod_{t=1}^{T} p_t(\pi_t),$$

summing over all paths that collapse to y. This sum is computed in $\mathcal{O}(T \cdot |y|)$ time by the forward–backward algorithm [9]. Only trace–key pairs are required for training; no alignment labels are needed. At inference time, greedy decoding sets $\hat{\pi}_t = \arg\max_c \{p_t(c)\}$ independently at each step and applies $\mathcal{B}$ to obtain the decoded key bit sequence. Note that this greedy decoding method does *not* in general yield the output sequence that most likely explains the observations; but it is in widespread use in areas where sequence-to-sequence models trained with CTC loss are used.

2.2 VGG-like Network

Our VGG-like network follows the standard architecture template used in other side-channel works as well. Specifically, we follow the architecture of Weissbart et al. [25], consisting of a configurable sequence of 1D-convolutional blocks (each with Conv1D, BatchNormalization, and MaxPooling1D with stride 2) followed by fully connected layers and a sigmoid or softmax output layer. VGG-like networks are a classical and widely used architecture for SCA [14,16,12,25], where the progressive downsampling followed by a dense prediction head encourages learning of translation-equivariant features while also allowing the network to capture position dependencies. In our experiments, we generally work with networks that are slightly shallower than those in [25], as our aggressive trace compression makes wide receptive fields within the convolutional layers unnecessary. Full architectural details are given in Appendix A.

2.3 Subsampling-based Network

The subsampling-based network is specifically designed for multi-target extraction from long traces, and directly reuses the architecture of Gohr et al. [7,8]. The main underlying idea of the architecture is built-in invariance under trace compression, achieved by splicing the input trace into q_1 decimated parts of length q_2 that are processed separately but with some information flow between neighboring subnetworks. A deep residual tower is used to enable training of very deep networks. Unlike deep CNNs, this architecture does not have a translation-equivariance bias, which is intentional since translation of features can change their meaning in side-channel analysis [7].

In our experiments on the SCAAML datasets, this architecture shows – as in previous work [7,8] – excellent performance in massively multi-target extraction, but is very sensitive to misalignment. Full architectural details are given in Appendix A.

3 Datasets

In this section we describe the datasets used in our study. Our experiments focus on two types of datasets: the SCAAML ECC hardware implementations (CM0–CM3), which provide real-world traces from ECDSA implementations using a cryptographic accelerator, and two synthetic tasks (SIFT and PICK) that we designed to isolate specific learning problems that arise in SCA. The SCAAML datasets serve as our primary target for demonstrating end-to-end key extraction from million-sample traces, while the synthetic tasks help to explain why standard architectures struggle with certain aspects of multi-target extraction and misalignment.

The SCAAML datasets were previously studied in [3], who succeeded in creating partial key extraction attacks using a transformer-based model, albeit at high computational cost (roughly one GPU month for CM1–CM3), with most of the time spent on hyperparameter tuning (but also ≈ 150 GPU hours on attack training for CM3). This means that SCAAML represents a challenging target for side-channel analysis from long raw traces. The synthetic tasks we designed isolate specific problems that standard SCA neural network architectures struggle with, and help to explain why they fail on the SCAAML datasets.

3.1 SCAAML ECC Hardware Implementations

Dataset Origins The SCAAML datasets were created by Bursztein et al. [3] to study fully automated end-to-end side-channel attacks. The target is the NXP K82F's dedicated cryptographic accelerator performing ECDSA signature generation (more specifically, elliptic curve scalar multiplication) with NIST P-256. Power traces were captured asynchronously, i.e. without clock synchronization between the capture equipment and the target device. This introduces natural timing jitter and makes the datasets realistic and challenging targets for end-to-end attacks from raw traces. For full details on the capture setup and target hardware, we refer to [3].

Raw Data Characteristics The SCAAML datasets contain full power traces covering complete secret-shared ECC scalar multiplication operations. Trace lengths vary significantly by target considered; see Table 1 for details. Each dataset is split into training, test, and holdout sets, with the holdout set captured on a physically different chip to enable evaluation of cross-device transfer. Dataset sizes range from approximately 200 GB (CM0) to 3.7 TB (CM3) in uncompressed form, making them among the largest public side-channel datasets available.

The raw traces are challenging targets for several reasons: first, the high trace length combined with asynchronous capture means that the location of cryptographic operations can drift between traces. Second, the traces contain substantial periods of non-cryptographic activity (e.g. initialization, communication overhead, result transmission) intermixed with the target scalar multiplications. Third, the implementations use several different secret-sharing methods, and the CM3 dataset combines both additive and multiplicative masking. The

length of the shares differs by masking method (multiplicative masking uses 128-bit shares, additive masking uses 256-bit shares), implying that the format of the potentially leaking values changes between datasets.

Table 1 summarizes the key characteristics of each dataset.

Table 1: SCAAML ECC dataset characteristics. All datasets have 8192 hold-out traces (captured on a separate chip); CM1 has 2048 test traces, all others have 8192. Add, Mult, Combined are additive, multiplicative, combined secret sharing.

	CM0	**CM1**	**CM2**	**CM3**
Countermeasure	None	Add	Mult	Combined
Trace length	1.6M	5M	10M	17.5M
Train traces	57 344	194 544	122 880	122 880

Cryptographic Target and Secret Sharing Countermeasures The cryptographic operation captured in all four datasets is elliptic curve scalar multiplication during ECDSA signature generation: computing $k \times G$ where k is a 256-bit ephemeral secret (the signature nonce) and G is the generator point of the NIST P-256 curve. The four datasets differ in their side-channel countermeasures, which progressively increase the complexity of the leakage pattern:

1. **CM0 (Constant-time execution):** The baseline dataset relies on the hardware accelerator's constant-time and approximately constant-power execution. The scalar multiplication processes the full 256-bit nonce k directly. While the hardware provides some protection, the nonce bits still leak through power variations during processing (observation based on our analysis of the leakage). This is the simplest case: a single 256-bit target where leakage positions align very well across traces.
2. **CM1 (Additive masking):** The nonce is split into two additive shares: a random 256-bit mask r and the masked value $km = k - r \bmod n$, where n is the curve order. The hardware accelerator computes $P_{km} = km \times G$ and $P_r = r \times G$ separately, then combines them as $P_k = P_{km} + P_r = k \times G$. The two share computations appear at different trace locations separated by hundreds of thousands of samples, and are statistically independent of each other and of k. In practice, the first share can be recovered straightforwardly by a single network (e.g., [7,8]), whereas the second is much harder for traditional architectures without precise trace segmentation to align both shares beforehand.
3. **CM2 (Multiplicative masking):** The nonce is split using Euclidean division with a 128-bit random mask r: compute quotient $km = \lfloor k/r \rfloor$ and remainder $rem = k \bmod r$. The accelerator then computes three scalar multiplications: $P_{km} = km \times G$, $P_r = rem \times G$, and $P_{km2} = r \times P_{km}$, combining

them as $P_k = P_r + P_{km2} = k \times G$. This scheme involves approximately 384 bits of secret material across three operations of different lengths. Multiplicative masking introduces more complex error propagation: mistakes in km amplify when scaled by r, and the unknown base point in the third multiplication (P_{km} rather than G) may randomize power signatures. Furthermore, due to the different bit length of the shares, the power signatures of the entire multiply chains are statistically quite different from the unprotected case.

4. **CM3 (Combined masking):** This dataset combines additive and multiplicative masking by applying them in sequence. The nonce is first split additively into two 256-bit shares, $k = k_1 + k_2 \bmod n$ (with k_2 a random mask and $k_1 = k - k_2$). Each additive share is then independently masked multiplicatively, exactly as in CM2. Applying this to both additive shares yields six scalar multiplications in total, whose results are recombined by point additions into $k \times G$. Because every one of these six scalars is the operand of a multiplicative split, they are all effectively 128-bit values, even though the hardware processes them as 256-bit inputs with the high-order bits set to zero.

3.2 Further Exploration via Synthetic Tasks

Overview

Motivation To understand why standard neural network architectures struggle with the SCAAML datasets, we designed two synthetic tasks called SIFT and PICK that isolate specific aspects of the multi-target extraction problem. One of them (SIFT) will be discussed here; for the other (PICK), see Appendix B. Basically, the idea is that in SIFT, neural networks have to isolate a trivially extractable but lengthy signal under poor alignment, while in PICK, *convolutional* neural networks have to extract a trivially extractable, *perfectly* aligned, noiseless but lengthy signal from noise that itself looks locally like exactly the signal to be extracted. Both problems have simple algorithmic solutions.

Observations on SCAAML In our experiments with SCAAML, we observed that standard VGG-like and ResNet-based architectures could extract only a few bytes of the CM1 key at a time. When we attempted to train them to extract the full key sequence, they did not converge, particularly for bytes in the second key share of CM1. The networks learned bytes near clear landmarks (e.g., the first and last bytes of the second share), but failed on bytes in between. Similarly, the subsampling-based architecture of Gohr et al. [7,8], while successful on single-share CM0, had difficulties with the second share in CM1.

The above observations suggested two possible fundamental obstacles for convolutional architectures:

(1) Difficulty in locating specific targets among many similar-looking features when positions vary (the "seventeenth cat problem"),

(2) Difficulty in learning mappings that are not translation-equivariant when many inputs and outputs must be handled simultaneously.

We test the above hypotheses using tasks that have trivial algorithmic solutions but preserve the structural challenges we suspected were causing standard architectures to fail on SCAAML. By making the leakage extraction problem itself extremely easy (no noise, very simple signals), we can study these problems in isolation.

SIFT-n-k: Simple Integer Filtering Task

Task Definition and Data Format The Simple Integer Filtering Task (SIFT-n-k) isolates the alignment and landmark-based localization problem in SCA. The task is defined as follows:

- **Input:** Fixed-length sequences of length n over the alphabet $\{0, 1, 2\}$. Each position is first assigned 0 or 1 with equal probability, then independently replaced by 2 with probability p.
- **Target output:** Extract the subsequence consisting of the first k occurrences of symbols $\{0, 1\}$, discarding all 2s.

For example, with $n = 20$ and $k = 8$, an input sequence might be:

`2 0 2 1 2 2 0 1 2 0 2 1 1 0 2 2 1 0 2 2`

The target output would be: `0 1 0 1 0 1 1 0` (the first eight symbols that are not 2).

The raw data is simply sequences of integers. There is no noise, no trace alignment issues, no need to learn what leakage looks like. The *only* challenge is localization: finding the k target symbols among n positions where 2-symbols act as "non-leaking" operations that must be skipped.

Relation to SCA SIFT models the structure of side-channel traces where:

- Symbols $\{0, 1\}$ represent operations that leak secret bits (analogous to e.g. scalar multiplication steps processing key bits).
- Symbol 2 represents non-leaking operations or noise (analogous to e.g. dummy operations, communication overhead, or other non-cryptographic activity).
- The network must learn to count: "find the first target, then the second target, then the third..." rather than "find targets near position 10, position 15, etc."

We think that this directly parallels the challenge in SCAAML CM1, where the network has to extract 256 bits from the first scalar multiplication and 256 bits from the second, but the exact positions of these bits vary due to jitter and the presence of variable amounts of non-cryptographic activity between and around the two operations.

Expected Difficulties for CNNs Standard convolutional architectures face a fundamental difficulty: they learn translation-equivariant features ("this pattern indicates a leaked 1"), but cannot cope with ordinal localization ("this is the *seventeenth* leaked bit"). For an analogy, while it is straightforward to train a CNN to detect whether *any* cat appears in an image, it is much harder to train one to classify the fur color of *specifically the seventeenth cat from the left* in an image containing many similar cats. SIFT tests exactly this capability: distinguishing between the first, second, third, etc. occurrences of very similar features when their absolute positions are not fixed.

Why SIFT is hard for direct gradient training. There are two main reasons why SIFT is expected to be hard for direct gradient-descent based learners when the proportion p of blank tokens in the input stream satisfies $p \gg 0$.

First, the ℓ-th coordinate of the output is by definition x_{I_ℓ}, where I_ℓ is the smallest index i for which $\sum_{j \le i} \mathbf{1}[x_j \neq 2] = \ell$. Since the input coordinates x_i are independent, I_ℓ cannot be determined with certainty without checking the content of all indices $\le I_\ell$. Without I_ℓ, we cannot determine x_{I_ℓ} unless the contents at the remaining candidate locations of the input string are very biased one way or another. This means that there are no good correlates of algebraic order $\ll \ell$ for SIFT-n-n. Non-existence of useful low-order features is often described as a serious obstacle for gradient-descent learning, see e.g. [1].

Second, there is a simple reduction from PARITY to SIFT-n-n: encoding an arbitrary bit string $b \in \{0,1\}^n$ as a SIFT input $b' \coloneqq 2 - b$ yields a SIFT output $\mathrm{SIFT}(b')$ of the form $(1, 1, 1, \ldots, 1)$, and an alternating sum over this output gives the parity of b. PARITY is notoriously hard to learn from random samples via gradient descent [20], which suggests that SIFT should also be hard.

It is worth noting that this type of reduction argument does not *prove* hardness of the learning problem. SIFT, unlike PARITY, is a multi-output function, and the first few coordinates are low-order functions of the input. Learning a multi-output function can be substantially easier than learning the components. For instance, the bitwise representation of the Hamming weight of a binary vector is in practice much easier to learn than parity computation directly, even though parity is one of the output bits (but for large n, the parity bit becomes hard to learn even in that setting). A structurally related setup to SIFT is the synthetic-data experiment in [8], where a network is trained to predict $(x_1,\, x_1 \oplus x_2,\, \ldots,\, x_1 \oplus \cdots \oplus x_n)$ from the inputs $(x_1, x_2, \ldots, x_n)$. In this case, the last coordinate turns out to be difficult to learn directly even for modest n.

Taken together with the experimental evidence in Section 5, the two arguments provide convincing evidence that SIFT should indeed be difficult for direct gradient training from random samples irrespective of network architecture.

PICK-*n*-*k*: Predetermined Index Content Keeping The Predetermined Index Content Keeping task (PICK-n-k) complements SIFT by testing whether CNNs can perform massive multi-target extraction when alignment is *perfect*, but the extraction map is not translation-equivariant. In PICK, the input is a

random binary sequence of length n, and the network must extract the bits at k predetermined fixed positions. While it is trivially solvable by a linear model after $O(n)$ examples, CNNs struggle with PICK to a not insignificant extent because their translation-equivariance bias conflicts with the position-dependent indexing the task requires. A full description of the PICK task and experimental results are given in Appendix B.

A discussion of why SIFT and PICK together explain the observed failure modes of standard SCA architectures, and to what extent the U-Net with CTC loss avoids them, is given in Appendix C.

4 Data Preprocessing

In the following, we detail the preprocessing of the traces, in particular our downsampling strategy.

Our preprocessing pipeline combines aggressive trace compression with optional segmentation to make end-to-end attacks on million-sample traces computationally feasible. We briefly describe the preprocessing algorithms we employ and their application to different datasets and network architectures. Furthermore, we give a qualitative preview of the results obtained with different preprocessing settings on the SCAAML datasets. Finally, we discuss a methodological choice that we had to make in our experiments due to limited storage and computational resources.

4.1 Dealing with the Size of the SCAAML Datasets

The most important dataset for our experiments is CM1. To the best of our knowledge, the Google Drive repository for the SCAAML datasets given in [3] only allows us to download the datasets in their original uncompressed form. Apart from the CM0 dataset, the training sets are all larger than 1 TB, which meant that a download of the training sets exceeded the storage capacity of our analysis PC or the remaining storage capacity of the Google One account that we used instead of the analysis PC to download the datasets.

To circumvent the storage problem, we chose to train on the holdout sets of the SCAAML datasets and test on the test sets. This replicates the setting of the original paper [3] insofar as the holdout set is captured on a physically different chip than the test and training sets. As a consequence, we have much less training data available for training our networks than the authors of [3]. However, it should not make the training task *easier*, so we consider a comparison with the original paper as fair.

The aforementioned point becomes even stronger when considering the fact that we can train our U-Net on *one key share* of CM1 and attack the other key share or the other datasets without retraining. Also, our method can train on unsegmented CM1 and directly attack full unsegmented traces from CM0, CM2, and CM3 in addition to the CM1 test set without retraining.

4.2 Hard Low-Pass Downsampling

Our primary dimensionality reduction method applies hard low-pass downsampling followed by resampling. To this aim, we compute the discrete Fourier transform (DFT) of the input trace, apply a hard low-pass filter at the Nyquist frequency corresponding to the target resolution, and reconstruct the signal via inverse DFT at the reduced length.

4.3 Prediction-Error Segmentation

When coarse-grained segmentation is needed, we employ a prediction-error method that identifies segment boundaries by detecting points where local signal predictability drops. The instance of our method that we use here simply applies the hard low-pass filter method to downsample the trace by a factor of ten, interpolates back to the original length, and computes the reconstruction error. Peaks in the error signal correspond to operation boundaries where the low-pass filtered signal fails to capture abrupt transitions between different computational modes. Standard peak detection with non-maximum suppression (when there is a cluster of peaks, keep only the highest one) yields the final boundary points.

4.4 Preprocessing for SCAAML Datasets

For the SCAAML ECC datasets, we apply different preprocessing strategies depending on the dataset and neural network architecture used:

- **CM0 (Unmasked):** Dataset CM0 contains a single 256-bit secret key without masking. For single-byte extraction experiments using VGG-like networks, we apply hard low-pass downsampling from ∼1.6M to 2000 samples. For full-key extraction, the subsampling-based network uses the same resolution successfully extracts all 256 bits in one forward pass. The U-Net-CTC approach was not directly trained on CM0, but demonstrates zero-shot transfer from CM1: when tested on CM0, traces are downsampled to 2000 points, then segmented and resampled to 1000 points using the same pipeline developed for CM1. No effort was made to find out how far we would be able to compress the traces further without losing the leakage.
- **CM1 (Additive Masking):** Dataset CM1 contains two 256-bit secret shares in separate scalar multiplications. We apply 5000-fold hard low-pass downsampling (from ∼5M to ∼1000 samples) followed by coarse-grained segmentation that removes non-cryptographic operations of variable length occurring between the two share operations. This segmentation does not isolate individual point additions within the scalar multiplication chain (which would be difficult to do simply on account of the fact that we do not know the implementation); it only excises regions without leakage and improves alignment. After separating the shares by segmentation, we upsample each share again to 1000 points. This upsampling serves two purposes. First, it provides a fixed input size that is independent of the precise segmentation

points chosen for a particular trace. Second, it increases the number of processing nodes in the network, allowing each forward pass to use more computational power than the same architecture would have at a smaller input size.

The resulting preprocessed data allows extraction of all 512 bits from 1000 trace points. This pipeline works well for both the subsampling-based network and the U-Net-CTC approach. At a slightly higher trace resolution of 5000 points, the U-Net-CTC is able to learn to extract all 512 bits in one forward pass from a single trace preprocessed only with the hard low-pass downsampling. The VGG-like and the subsampling-based network were in our experiments unable to extract the second key share in this and similar settings that we have tested.

– **CM2 and CM3:** We only analyze CM2 and CM3 by training the CTC-U-Net on the CM1 dataset, followed by performing zero-shot transfer to CM2 and CM3 after compressing the traces to a comparable resolution to the CM1 dataset. The same picture as for CM1 applies here, i.e. we achieve excellent zero-shot transfer and can see all key bits with high fidelity. Direct attacks on the full unsegmented traces use a higher-resolution network (equivalent to 5000 points per trace on CM1).

5 Experimental Results

In this section, we present a detailed summary of our main experimental results. We start with the VGG-like network and the subsampling-based network on CM0 and CM1 and discuss observed limitations of our attacks. We then compare them to similar problems encountered with the synthetic tasks SIFT and PICK. Finally, we present the results obtained with the U-Net-CTC on the SCAAML datasets.

5.1 Results on CM0 and CM1 Using VGG-like and Subsampling-Based Networks

CM0 Analyzing CM0, we found that we could easily extract single bytes of the key after performing hard low-pass downsampling to 2000 points. Various methods worked: a VGG-like network, the subsampling-based architecture, and even a Random Forest. However, while we were able to see individual bytes with almost 100% accuracy, it proved difficult to make the VGG-like network extract the full key in one go, i.e. to train a version of the VGG-like network to extract *all* 256 labels corresponding to all key bits in one inference step. However, the subsampling-based architecture, having had previous successes with massively multi-target prediction tasks in SCA in [7,8], was immediately able to extract the full key.

Table 2 summarizes the network architectures and training parameters used in our CM0 experiments.

Table 2: Network architectures and training parameters for CM0 experiments at 2000-point resolution.

Parameter	**VGG-like**	**Subsampling-based**
Architecture		
Input length	2000	2000
Conv filters	[32,64,128,128,64,32,16,8,4]	–
Dense layer sizes	[512,512,512]	–
Dense batch norm	Yes	–
Reshape parameters (q_1, q_2)	–	(20, 100)
Residual depth	–	20
Width (conv filters)	–	256
Output units	8 (single byte)	256 (full key)
Output activation	Sigmoid	Sigmoid
Training		
Optimizer	Adam	Adam
Loss function	Binary cross-entropy	Binary cross-entropy
Batch size	50	32
Epochs	100	100
Learning rate schedule	Cyclic sawtooth	Cyclic sawtooth
LR range	10^{-6} to 10^{-3}	10^{-6} to 10^{-3}

CM1 When working with CM1, we decided to *train* on the *holdout* set, and to *evaluate* on the *validation/test* set. This kept the effort required for downloading the data manageable, while preserving the natural train-test split of the data and in particular taking into consideration the fact that the holdout set was acquired from a different device than the training and validation sets, see also the discussion in Section 4.1.[5] We again found that we could see some parts of the key – or, to be more precise, of the key shares – extremely well, but seeing the whole key was difficult. The VGG-like network could not extract the full key, and failed to train to more than about 70% accuracy on the bits of the *second* key share. The subsampling-based architecture converged more slowly, and showed no degradation in performance when predicting all targets jointly, but still failed to extract the second key share.

Targeting Byte Windows We downsampled the CM1 traces to 1000 points, corresponding to a downsampling factor of 5000, and still obtained better than 95% accuracy on most of the first key share bits using a VGG-like network with 8 sigmoid outputs (which would thereby predict one byte at a time, as a sequence of bits). Figure 3 shows the final accuracy of the VGG-like network as a function of the targeted byte.

[5] It is natural to ask why we did not train on the validation set used in [3] and test on their holdout set, which would have mirrored their setup more closely. The reason is that the CM1 validation set is relatively small, at 2048 traces.

One interesting observation we made in this experiment was that not the *entire* second key share was difficult to extract: our VGG-like network could see the first byte almost perfectly, and the last byte also reasonably well. The middle bytes, however, were very difficult to predict with enough accuracy for a key extraction attack. Targeting byte windows worked also with other methods – for instance, a random forest worked reasonably well – but the middle bytes of the second key share were difficult for all methods we tried. The subsampling network performed in qualitatively the same way on the task of predicting all of the second key share bits.

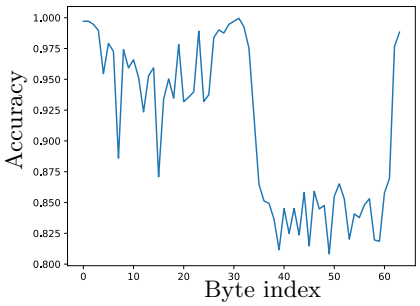


Figure 3: Validation accuracy of the VGG-like network as a function of the targeted byte, bytes 0-31 for the first key share and bytes 32-63 for the second key share.

Segmentation Based on the observation that the active regions of the traces looked very similar to the CM0 active region, we used our segmentation algorithm to perform a rough hard low-pass downsampling segmentation of the traces into their main regions. Our plan was to cut out the active regions and train our subsampling-based network to attack the regions separately, predicting 256 bits in one go for each key share. For each segment, we included a small buffer around the edges, as it turned out that the buffer regions were important for our networks.

We visually identified the two largest segments as the cryptographically active areas of the traces, cut them out and resampled to 1000 points (see Figure 10 in Appendix D for a visualization of this segmentation process). We then labeled each of these segments with the key share associated to it, thereby doubling the size of our training and testing sets. We used a subsampling-based network with 256 output units, 5 internal branches each treating a 200-point slice of the trace, a residual tower depth of 20, and sigmoid output activations. Training this network for 1001 epochs using a cyclic learning rate schedule with a peak value of 10^{-3} and a lowest learning rate of 10^{-5}, we obtained a network that reached 99% accuracy for the first key share and 98.3% accuracy for the second key share.

Estimating the Cost of Full Key Extraction Our subsampling network outputs a probability for each bit of either key share after training, which allows key rankings to be computed in standard ways for both shares, see e.g. [23,15,10,26]. When k is the ephemeral scalar used in ECDSA signature generation, the point $k \times G$ is leaked in the signature up to sign. As a consequence, a meet-in-the-middle collision search can recover both shares k_1 and k_2 at a cost of $\leq r_1 + r_2$ point multiplications, where r_1 and r_2 are the individual key rank estimates for the two shares. This idea generalizes straightforwardly to the cases of a single or

of multiple key shares (via decomposition of one of the shares, e.g. into upper and lower half bits). Note that $r_1 + r_2$ is an *upper bound* on the actual residual work, so the projected success rates below are *lower bounds*. The bound can be quite loose: when many high-ranked candidates for k_1 and k_2 all satisfy $k_1 + k_2 = k$, finding *any* valid decomposition is a random-walk problem over those candidates, solvable in $\approx \sqrt{r_1 + r_2}$ steps rather than $r_1 + r_2$. For a concrete scenario where this applies, if the upper half of both shares leaks perfectly while the lower half does not leak at all, the naive upper bound estimate given reports a remaining work factor of $r_1 + r_2 \approx 2^{128}$, but the 2^{128} valid decompositions $k = k_1 + k_2$ form a set which a random walk will hit in $\approx 2^{64}$ steps.

Allowing for a brute force work effort of up to 2^{41} point multiplications/hash table lookups, the upper bound estimate predicts that our network would be successful in extracting the key in roughly 87% of the cases (note that in this projection, we are working with the data in the 5000-fold downsampled traces; at higher resolution, success rates are expected to be substantially higher). The full predicted success rate curve as a function of work factor is shown in Figure 11 in Appendix D.

5.2 Results on SIFT Using VGG-like and Subsampling-Based Networks

Network Performance on SIFT The VGG-like and subsampling-based networks used for CM1 failed on SIFT-512-32 with parameters comparable to those used for CM1. Both architectures correctly extracted the first few bits, but produced high error rates for subsequent positions in the sequence.

Position-Dependent Performance Figure 4 shows the binary cross-entropy loss of the VGG-like network on SIFT-512-32 as a function of bit position. The loss curve shows a highly predictable pattern of increasing difficulty with advancing bit position. The synthetic task lacks landmarks near the sequence end, preventing the performance recovery observed at the end of the second CM1 key share.

5.3 Results on the SCAAML Datasets Using U-Net-CTC and Segmentation into Active Regions

Main Ideas As described in Section 2, our approach uses a 1-D U-Net [18] trained with CTC loss [9]. The network outputs a per-sample probability distribution over three classes: *blank* (non-leaking region or boundary), *zero* (0-bit of the key), and *one* (1-bit of the key). CTC training requires only trace-key pairs without per-sample annotations, and automatically learns to align predicted probability

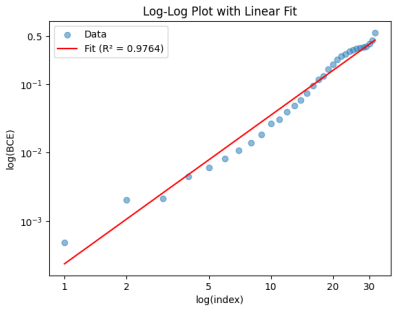


Figure 4: Cross-entropy loss of the VGG-like network on SIFT-512-32 as a function of bit position.

maps with target key sequences. At the inference stage, we apply standard greedy CTC decoding: select the most likely class at each time step, collapse consecutive duplicates, and remove blanks to yield the predicted key bit sequence.

Data Preprocessing The CM1 dataset was preprocessed as follows: the traces were hard low-pass downsampled to 2000 points. By visual inspection of one trace, we set cut points for the two key shares. For the first key share, we selected the region between index 150 and 800, for the second key share the region between index 800 and 1500. These regions were uniformly cut out and resampled to 1000 points by Fourier interpolation. The traces were labeled with the correct key share, resulting in 256 labels for each trace. Note that this way of segmenting the traces into key shares is *much less precise* than the segmentation into active regions that we used previously for work with the CM1 dataset. In particular, the segmentation by fixed cut points does nothing to align the traces consistently with the start of the sensitive operation beyond whatever alignment the original traces may have already achieved. This means our preprocessing is much simpler than before, and our networks solve a harder task.

We ran two sets of experiments. In the first set of experiments, which is described in the current subsection, we used the traces corresponding to the *first* key share on CM1 on the *holdout* set for training, which implies that also our training data is reduced by half compared to our previous experiments with the CM1 dataset. Testing was performed primarily on the *validation* set, although also the second key share of the holdout set could be used for additional testing as we do not use it in training. Results did not significantly differ between the different parts of the overall dataset that were not seen in training.

In the second set of experiments, which will be described in the next subsection, we perform no segmentation and directly attack full test traces while also demonstrating excellent cross-implementation transfer of the trained model.

Training Training was performed for the U-Net with 21 and 201 epochs, respectively, using a cyclic learning rate schedule (period 20 epochs, initial warm-up from 10^{-5} to 10^{-3}, subsequently cycling between 10^{-3} and 10^{-5}) using the Adam optimizer. A training/validation split of 90:10 of the training data was used to monitor progress. The best model by validation loss was saved to disk after each epoch. All networks were trained on the first key share of the CM1 holdout set. No further training was performed for any of the networks to adapt them for other conditions (validation set, second key share, or various corruptions we tried). For comparison, we also trained the subsampling-based network with the same architecture and hyperparameters as previously described for 201 epochs in the same setting.

Decoding the Predictions The predictions of the U-Net yield a segmentation map of the trace into three parts, which correspond to regions of the trace that do not contain leakage (or are at the boundary of two different leakage sites), regions that emit a one-bit of the target key share, and regions that emit a zero-bit of the target key share. For each time step, the network predicts a probability distribution over these three classes. We use standard greedy decoding: we take the most likely class for each time step, and then collapse consecutive duplicate classes into a single emission. For the subsampling-based network, we decode the vector of 256 output predictions into a bitstring by simple thresholding at 0.5.

Success Metrics We used three different metrics to evaluate the success of key extraction:

- *Bitwise accuracy:* fraction of the bits in each key share that were correctly predicted on average.
- *String similarity:* similarity of the predicted key share compared to the ground truth, when both are interpreted as 256-bit bitstrings. Mathematically, for two strings s_1 and s_2, we used the insert-delete distance to compute the string similarity $s(s_1, s_2) \coloneqq 1 - \frac{d(s_1,s_2)}{|s_1|+|s_2|}$. We refer to this value as the *similarity ratio* of the two strings and use [19] with `python-Levenshtein` [2] installed for its computation. Note that the library provided in [19] always rounds the computed similarity values to full percentages; this is adequate for most of our study, but in Section 5.4, where we need more precise measurements because we work in a regime with higher trace resolution and higher performance, we use `python-Levenshtein` directly.
- *Remaining work factor/key rank:* the expected number of operations that need to be performed to recover the key share from the predictions.

Rationale for the Choice of Metrics Accuracy alone as a success metric for side-channel extractors generally has the problem that it is not strongly correlated with the remaining work factor: two attacks that have the same accuracy per operation label can have vastly different work factors [17], and in particular *low* accuracy does not prove that an attack does not work. In our setting of direct end-to-end attacks based on sequence-to-sequence learning, this phenomenon becomes much worse than in the general literature, because a single *missing* bit near the beginning of a predicted secret bit string suffices to lower the bitwise accuracy of the prediction to $\approx 50\%$. On the other hand, standard techniques for computing key rank are not straightforwardly portable to the asymmetric setting, and also not easy to apply to sequence-to-sequence attacks. However, when any of these metrics becomes *very good*, this shows that a large amount of leakage has been extracted from the side-channel. We therefore consider string similarity alongside the two other metrics (as it deals gracefully with misalignment of prediction and ground truth) and report for each experiment the metrics that seem to be the most relevant given the experimental setting.

Test Conditions To evaluate the robustness and generalization capabilities of our extractors, we additionally considered the following experiments beyond the base case (extraction of the first key share from CM1 validation traces):

- *Alignment robustness:* Small rotation (1 sample), large rotation (10% of trace length, i.e., 100 samples), and random large rotation (61% of trace length, i.e., 614 samples).
- *Zero-shot cross-share transfer:* Extraction of the second key share from CM1 without retraining. For the subsampling network baseline, we also report results with optimal alignment: for each trace, we test 200 circular shifts (from -100 to $+100$ samples) and select the shift that maximizes similarity with the ground truth. The maximization is performed on a trace-by-trace basis. The reason for running the experiment with optimal alignment is that otherwise negative results could be due to misalignment of the features between the two key shares and therefore just a repetition of the rotation experiments.
- *Zero-shot cross-dataset transfer:* Extraction from CM0 (unmasked, single 256-bit key), and from CM2 (multiplicative masking, three key shares of 128 bits each). We expect the same workflow to work also on CM3, but we did not test this. Instead, we ran an attack on CM3 in the setting that will be described in the next section, where we do away entirely with the segmentation into active regions.
- *Time reversal:* Traces reversed in time to test zero-shot adaptation to an out-of-distribution test condition.

The rotation and time reversal experiments for the U-Net are run by performing the desired manipulation on the trace, obtaining the prediction map, and reversing the manipulation on the prediction map to obtain the final pre-decoded prediction. This prediction is then decoded into a bitstring and compared to the ground truth. For the subsampling network baseline, the prediction is directly the vector of 256 probabilities and the decoding into binary outputs is performed by a simple thresholding at 0.5; the back-rotation step does not make sense in this case and is therefore skipped.

We also tested robustness to trace corruptions including simulated jitter, interrupts, and frequency changes. Briefly, the U-Net maintained some predictive power (85% similarity ratio, compared to a random baseline of $\approx 80\%$) even under combined corruptions that substantially distorted trace morphology, and proved much more robust than the subsampling network (73%), which fell below the random baseline. A detailed description of the corruption settings, the corruption parameters, and example visualizations is given in Appendix E.

Baseline Results with Subsampling Network We first show the results of the subsampling-based network on the CM1 dataset. Qualitatively, the network learned to extract the validation first key share perfectly, but largely failed to learn a useful extractor in all other tested conditions. Table 3 shows a selection of the quantitative results using similarity ratio as the metric.

In the second key share condition with *optimal alignment*, subsampling network similarity ratio reached 89% on average and 95% on the best trace, showing

Table 3: Similarity ratio of the subsampling network (*Sub*), the CTC-loss U-Net trained for 21 epochs (*U-Net-21*), and for 201 epochs (*U-Net-201*) under various conditions. All models were trained solely on the first key share of CM1. Values are rounded to the nearest integer; dashes indicate conditions not evaluated for that model. The random large rotation test rotates each trace by 614 steps (61.4% of trace length) and reverses the rotation after computing the prediction map. Time reversal reverses the entire trace and reverses back after the prediction map is computed. The second key share result for Sub uses optimal alignment; all U-Net results are zero-shot.

Condition	**Sub**	**U-Net-21**	**U-Net-201**
Base (first key share)	100%	96%	98%
Second key share (zero-shot)	89%	95%	97%
CM0 (zero-shot)	74%	95%	98%
CM2 first key share (zero-shot)	–	–	98%
CM2 second key share (zero-shot)	–	–	98%
CM2 third key share (zero-shot)	–	–	97%
Small rotation (1 sample)	53%	96%	98%
Large rotation (10% of trace)	78%	96%	98%
Random large rotation (61% of trace)	–	95%	97%
Corrupted traces	73%	85%	85%
Time reversal	–	–	92%

that some predictive power could be restored by optimizing alignment. However, even these high values are likely insufficient for practical key recovery. On the whole, the subsampling network failed to adapt to all conditions except for the base case for which it was trained.

CTC-Loss Based U-Net: 21 Epochs We now turn our attention to the CTC-loss based U-nets trained on the CM1 first key share on the CM1 holdout set. Qualitatively, the network trained for 21 epochs shows reduced performance compared to the subsampling baseline in the base condition, but massively outperforms it in all other scenarios. Table 3 summarizes the performance results.

We see stable performance across all conditions. The training of the 21-epoch U-Net was very efficient, taking only roughly 8 minutes on a single NVIDIA RTX 4070 GPU.

CTC-Loss Based U-Net: 201 Epochs We trained a second U-Net model for 201 epochs on the first key share of CM1, using the same setup and hyperparameters as above. The U-Net-201 column of Table 3 summarizes the similarity ratio results of this model under various out-of-distribution conditions. Compared to the 21-epoch model, the 201-epoch model achieves slightly higher performance in the *base* condition (increasing similarity ratio from $\sim$ 96% to $\sim$ 98%) and remains similarly robust under other conditions.

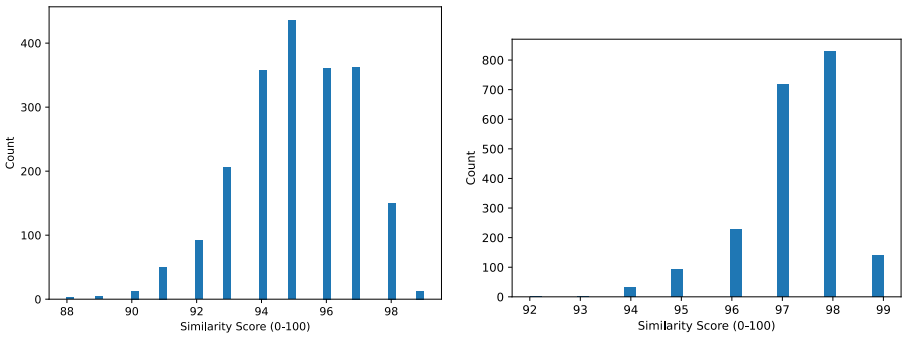
We ran an additional *time reversal* experiment on the U-Net-201, in which we reversed each trace and then reversed the resulting prediction map back before decoding. This experiment is interesting in so far as it tests the network's ability to extract key bits in a setting that is not represented at all in the training data, but where one might expect a sufficiently intelligent extractor to succeed easily.

Attacking CM2 Given the observed robustness of our approach, one natural question was whether it would be possible to use the same network to attack the CM2 dataset as well. As described in Section 3, CM2 uses multiplicative masking with a 128-bit random mask r, where the secret key k is split into a quotient km and the remainder rem from division by r, requiring recovery of about 384 total secret bits across three scalar multiplications.

To attack the CM2 implementation using our U-Net model, we first downloaded the validation part of the dataset. We downsampled the traces from originally 10^7 samples each to 5000 samples. We then eyeballed a segmentation of the trace into three parts, corresponding to the multiplications with km, rem, and r. We picked sample points 600 to 1400 for km, 1500 to 2300 for rem, and 2500 to 3300 for r. We used these fixed cut points to define three target segments, upsampled each to 1000 points, and used the U-Net-201 model to predict the key share. As the shares are 128-bit values, we decoded the 1000-point prediction maps greedily as before, but discarded all but the last 128 bits of the decoded prediction (the discarded bits were overwhelmingly zeroes anyway). Overall, extraction of all key shares was excellent, as shown in Table 3.

Test-Time Augmentation We have already seen that the predictions of the CTC-loss U-Net degrade gracefully (or in some settings not at all) when input traces are shifted or otherwise perturbed. The observed robustness suggests that *averaging* predictions from multiple, slightly perturbed versions of the same trace might improve final key extraction performance. We implemented this idea as a *test-time augmentation* procedure in which, for each test trace, we generated multiple *rolled* (circularly rotated) versions of the trace, obtained per-sample output probability maps from the model, and then reverse-rolled these maps back before summation. The final probability map is an average of all these augmented inferences. We then performed standard CTC greedy decoding on the averaged map.

In experiments with the U-Net 201 epochs model, test-time augmentation further boosted similarity ratio results on mildly out-of-distribution data (e.g. the second key share of CM1) to about 99%. A similar improvement (from 95% to 97%) was observed for the 21-epoch model. The distribution of the similarity ratio results for the 21-epoch model before test-time augmentation is shown in Figure 5a, and the distribution after test-time augmentation is shown in Figure 5b. One clearly sees that in particular bad predictions are removed by the test-time augmentation and that the mode of the distribution shifts significantly towards higher values; also the variance is reduced.



(a) Without TTA **(b)** With TTA

Figure 5: Distribution of similarity ratio results for the 21-epoch U-Net on the second key share of CM1, before and after test-time augmentation (TTA).

5.4 Attacking all Targets without Segmentation

Main Idea We have seen that the U-Net can extract the key shares of CM0, CM1 and CM2 without the need for retraining or any other adaptation on the network level. However, both training and testing still require segmentation of the traces into key shares. While the segmentation is very simple and efficient to perform – one just needs to eyeball one sample trace, set fixed cut points, and apply them uniformly to all captures – it is still natural to ask whether it is possible to succeed by plain sequence-to-sequence learning and prediction. In the following, we will answer this question in the affirmative by showing that the U-Net can be trained on unaltered CM1 traces at a resolution of 5000 samples per trace and extract both CM1 and CM3 key shares in one unified forward pass and decoding step.

We go on to show that the approach has a further significant benefit: the prediction of the U-Net is an annotated map of the trace, and that map can be used to identify active regions of the trace both at the macro- and microscale.

Data Preprocessing We start by downsampling the CM1 traces to 5000 samples per trace and the CM3 target traces to 17 500 samples per trace. The goal of the CM3 downsampling step is to match the resolution of the CM1 traces. Note that both downsampling steps are just compression by a factor of 1000, so we are simply matching the same compression ratio that we used for the CM1 traces. Besides downsampling, no further preprocessing is performed on the traces.

Training and Inference We train a U-Net-CTC model using the Adam optimizer with a cyclic sawtooth learning rate schedule. Training was done using the compressed CM1 holdout set. The learning rate cycles between 10^{-5} and 10^{-3} over periods of 20 epochs. The model is trained for 101 epochs with a batch size of 32 and a validation split of 10%. The U-Net architecture has an input length of 5000

samples. Inference was performed on the CM1 and CM3 validation sets. At this point, we use the CM1 validation set to test self-explanatory properties of the U-Net output, and the CM3 validation set to test cross-dataset generalization.

Self-Explanatory Properties of the U-Net Output

Overview The first property of the U-Net that we are interested in is whether the predicted probability map is aligned with expectations on the location of the cryptographic operations in the trace. This question can be asked both on a macroscopic level, where one aims at locating the scalar multiplications with the key shares within the trace, as well as on a microscopic level, where one wants to learn exactly where each part of the key is leaking. We consider both experiments here.

On a high level, these experiments show that the raw output of our U-Net based extractor yields a considerable level of immediate insight into a black-box cryptographic implementation, without applying separate explainability tools. The generation of similar insights is difficult for most other neural side-channel setups also simply because they do not consider whole raw traces but rely on an expert to set up relevant windows of interest and perform some preprocessing in advance.

Locating the Scalar Multiplications To this end, we run inference on the CM1 validation set without decoding, scale the power values of the traces to the 0-1 interval, and plot the blank output after $\mathsf{softmax}$ $p(\text{blank})$ alongside the trace data scaled accordingly for one randomly chosen sample trace. Figure 6 shows the result for one such sample trace.

In Figure 6, we see that the blank output maxes out in the presumed inactive regions of the trace and oscillates very rapidly between 0 and 1 in the active regions. This is consistent with the expectation that the active regions will contain blank outputs that act as separators between emitted bit values, and that the inactive regions will be uniformly blank. The overall alignment of the blank output with cryptographic operations across the rest of the trace is abundantly clear.

Explaining the Third Non-Blank Region There is a small non-blank region towards the end of the trace that at first sight does not seem aligned with cryptographic operations. We believe this is the U-Net detecting the share recombination step, in which the crypto accelerator combines the results of the two separate scalar multiplications via point addition. As we shall see shortly, the target operations within the scalar multiplications are likely themselves point additions performed as part of a windowed scalar multiplication scheme; the recombination would therefore be difficult for the network to distinguish from the target operations, causing a short burst of non-blank emissions that do not correspond to meaningful key bits. This hypothesis is also supported by the observation that directly following the supposed recombination step, we see a

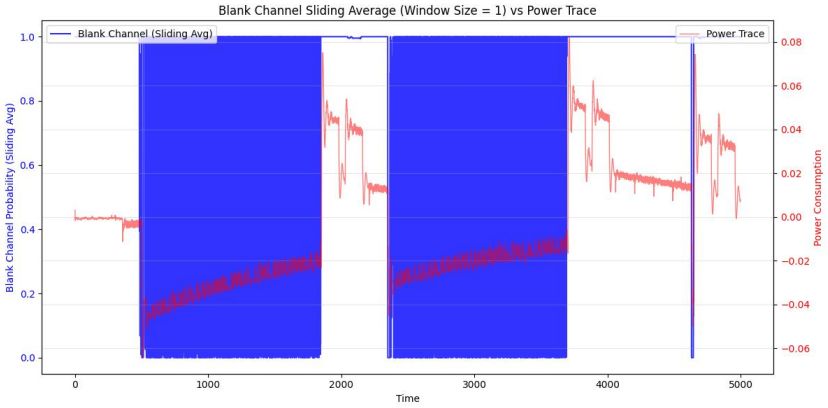


Figure 6: Blank channel output $p(\text{blank})$ plotted together with the scaled power trace for a sample from the CM1 validation set. The alignment between regions of rapidly oscillating blank probability and periods of cryptographic activity in the power trace demonstrates the U-Net's ability to localize relevant features without explicit supervision.

sequence of power spikes very similar to what is observable right after the scalar multiplications, which might be communication from the crypto accelerator to the MCU.

Locating Single-Bit Leakages To perform an analogous analysis on the microscopic level, we look at random locations *within* the active regions and plot the blank, 0 and 1 output probabilities for that small trace segment. Figure 7 shows a representative example.

In Figure 7, clusters of four rapidly consecutive non-blank emissions are followed by longer blank regions. Blank regions within these clusters are shorter and are expected because the CTC decoding requires that identical bits be separated by blanks, whereas non-identical consecutive bits do not need a separator; this fits the observed patterns in Figure 7. The observed pattern is consistent under the assumption that the underlying cryptographic hardware performs a windowed scalar multiplication with a window size of four bits, although we cannot confirm this hypothesis with certainty without access to the implementation details, which are to the best of our knowledge not public for the K82F chip that was used to record the data.

Cross-Dataset Generalization

Running Inference To test cross-dataset generalization, we downsample the CM3 validation traces to 17 500 samples per trace and run inference on the downsampled traces with the U-Net-CTC model trained on the CM1 holdout set. Since

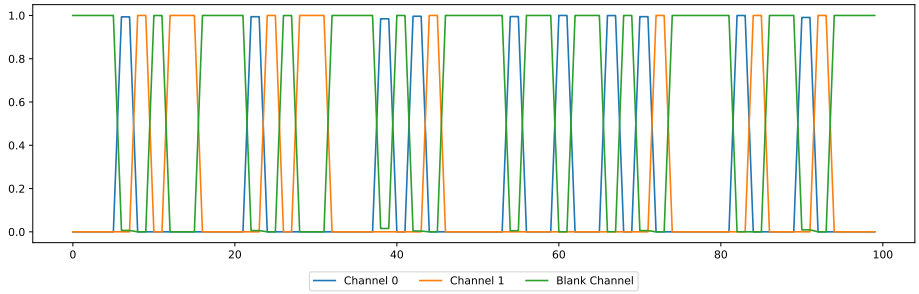


Figure 7: U-Net output probabilities for channels 0, 1, and blank for a segment (indices 1000-1100) within an active region of a CM1 validation trace. The alternating pattern between channels 0 and 1, separated by brief blank periods, demonstrates the model's ability to detect and distinguish individual bit operations at the microscopic level. The blank regions between consecutive identical bits are required by CTC decoding constraints.

our network architecture fixes the size of the input layer, but is fully convolutional in design, we do this simply by creating an instance of our architecture with a suitably wider input layer and loading the pre-trained weights for the 5000-sample U-Net model from disk. This suffices to obtain predictions for the CM3 validation set.

Assessing Predictions The predictions obtained are bit vectors that correspond to six scalar multiplications, each of which generates 256 bits of key share output. This is correct even for the 128-bit scalars that are used in the multiplicative part of the sharing scheme (their high-order bits are simply zeros, which is natural given that all scalar multiplications are run on the same hardware, and that hardware presumably does not accommodate variable-length inputs and outputs).

One difficulty that arises in this context is that it is a priori not clear *which* key share corresponds to which part of the prediction bit sequence of roughly $6 \cdot 256 = 1536$ bits length. Normally, in a white-box setting, this would be resolved by inspecting the implementation details. We resolve this problem by choosing one reference trace at random, testing all $6! = 720$ possible permutations of the key shares, and selecting the permutation that yields the highest similarity ratio with the decoded reference trace prediction. Our reported mean similarity ratio is then calculated using this permutation of the ground truth bit strings for all traces.[6]

Testing Large-Scale Segmentation Producing a similar visualization of the blank output for a CM3 validation trace as was done in Figure 6 for a CM1 validation

[6] Note that e.g. randomly permuting the key share order would not yield significant protection against our attack, because the attacker can brute-force the 720 permutations for each trace separately at a roughly 2^9 additional computational cost.

trace shows six scalar multiplications and three recombination steps. This is perfectly in line with expectations.

Similarity to Ground Truth Global average similarity ratio of predicted bits after straightforward greedy CTC decoding is $\approx 96.6\%$. However, if our hypothesis about the share recombination step is correct, this figure includes short bursts of non-blank nonsense emissions. We can suppress them by first segmenting the raw output map into active regions separated by long stretches of blank emissions (we use a minimum gap of 100 blank-only steps), and then discarding any active region whose greedy CTC decode is too short (we use a minimum of 150 decoded symbols). Further, the CM3 key shares are all 128-bit values with 128 bits of leading zeros, so we can accept only the last 128 bits of each decoded region prediction. The six scalar multiplications are labeled `km2a`, `rem2a`, `r2a` (the multiplicative split of the first additive share) and `km2b`, `rem2b`, `r2b` (the split of the second additive share), where for each branch `km` denotes the quotient, `rem` the remainder, and `r` the multiplicative mask. Figure 8 shows the average similarity ratio of the regions corresponding to the six scalar multiplications before and after discarding the leading bits. We see that the custom decoding produces almost perfect results.

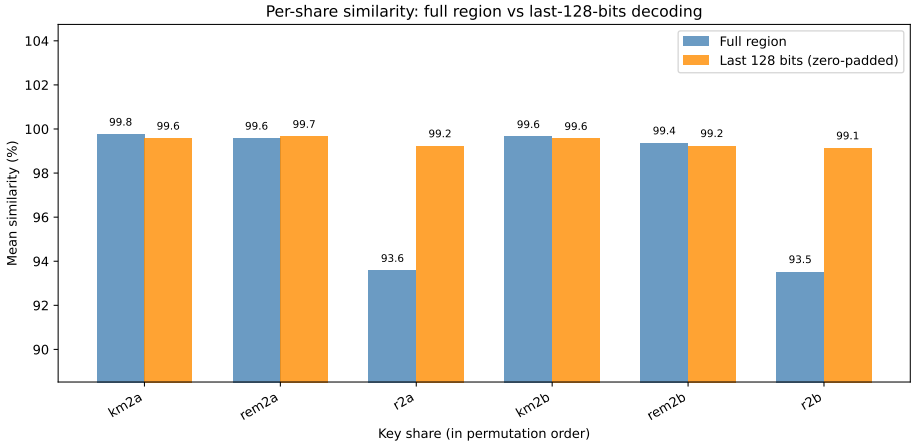


Figure 8: Per-share mean similarity ratio for the six CM3 scalar multiplications. Blue bars show the similarity when comparing the full decoded region to the 256-bit ground truth; orange bars show the similarity after keeping only the last 128 decoded bits and zero-padding to 256 bits. Because all six scalars are effectively 128-bit values, this custom decoding brings even the previously weaker `r2a` and `r2b` shares to near-perfect recovery.

6 Conclusions and Future Work

6.1 Conclusions

Chaining aggressive trace compression, a 1-D U-Net, and CTC loss yields a simple, end-to-end pipeline for full-key extraction from single raw power traces containing millions of samples. A single model trained on one key share of CM1 for a few minutes (21-epoch version) to less than two hours (201-epoch version) on a consumer GPU recovers complete key shares with high fidelity across all four SCAAML ECC datasets with excellent robustness properties, given only mild pre- and post-processing.

The SIFT experiment explains why standard convolutional architectures struggle with multi-target extraction from long traces: the bottleneck is not recognizing leakage patterns but routing them to the correct output positions. CTC removes this bottleneck by requiring only a correctly ordered prediction map, not an explicit input-to-output position assignment. This insight should generalize: whenever many targets must be extracted from a long, potentially misaligned trace, decoupling localization from classification via sequence-to-sequence learning is a natural alternative to manually segmenting the target and extracting the leakages one-by-one.

Finally, the U-Net prediction maps provide built-in explainability by localizing leakage at both macroscopic and microscopic scales. In our experiments they revealed a 4-bit windowed processing pattern that has high plausibility from the point of view of common ECC implementation patterns. The prediction maps also show a failure mode of the network itself, in that it fails to ignore share recombination. Similar failure modes could plausibly surface in standard attacks and cause evaluators to underestimate attack power.

6.2 Future Work

Several lines of future work suggest themselves. We discuss a few of them below.

Test-Time Augmentation for Side-Channel Analysis Test-time augmentation is an established technique in image processing [13], but to our knowledge has not been applied to side-channel analysis before our work. Classical side-channel models are extremely brittle against input transformations, which likely explains this gap; our attack is not, and our experiments confirm that TTA can improve side-channel performance. We expect future work to identify more useful augmentations and better inference combiners [21], and more broadly, that robust extractors will open the door to other methods of trading online computation for accuracy, such as test-time training [22].

Synthetic Tasks for Side-Channel Analysis The present work shows that using synthetic tasks that model one part of the side-channel extraction problem in isolation can be quite useful in order to demonstrate limitations of current approaches. We think that it would be desirable to have more limitations results

of this type, both based on mathematical analysis of current profiling strategies and based on experimentation with suitably constructed simple tasks. Our work shows that quite simple obstacles to good performance in a side-channel exploitation pipeline can hide large amounts of leakage from evaluators. A better understanding of the whole range of such obstructions would be of great practical utility.

More General Sequence-to-Sequence Extractors Our sequence-to-sequence extractors gain performance and training efficiency in part from richer annotations than previous end-to-end pipelines: unlike classical end-to-end attacks, which are agnostic to the temporal order of leakages, we use that ordering at profiling time, though not precise leakage locations. It would be desirable to relax this assumption, even at the cost of sample efficiency, or conversely to study what efficiency gains are possible when traces are labeled with leakage locations rather than just their order.

Acknowledgement A significant part of the results reported here were obtained during the KISKA project (2023-2025) run by cryptosolutions under commission from BSI.

References

1. Abbe, E., Boix-Adserà, E., Brennan, M.S., Bresler, G., Nagaraj, D.: The staircase property: How hierarchical structure can guide deep learning. In: Advances in Neural Information Processing Systems 34 (NeurIPS 2021). pp. 26989–27002 (2021), https://proceedings.neurips.cc/paper/2021/hash/e2db7186375992e729165726762cb4c1-Abstract.html
2. Bachmann, M.: python-Levenshtein: Python extension for computing string edit distances and similarities, https://pypi.org/project/python-Levenshtein/, pyPI version 0.27.3 (2025-11-01). Accessed 2026-02-11
3. Bursztein, E., Invernizzi, L., Král, K., Moghimi, D., Picod, J.M., Zhang, M.: Generalized power attacks against crypto hardware using long-range deep learning. IACR TCHES **2024**(3), 472–499 (2024). https://doi.org/10.46586/tches.v2024.i3.472-499
4. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 45–68. Springer, Cham (Sep 2017). https://doi.org/10.1007/978-3-319-66787-4_3
5. Chvátal, V., Sankoff, D.: Longest common subsequences of two random sequences. Journal of Applied Probability **12**(2), 306–315 (1975). https://doi.org/10.2307/3212444
6. Galli, D., Chiari, G., Zoni, D.: Hound: Locating cryptographic primitives in desynchronized side-channel traces using deep-learning. In: 2024 IEEE 42nd International Conference on Computer Design (ICCD). pp. 114–121 (2024). https://doi.org/10.1109/ICCD63220.2024.00027

7. Gohr, A., Jacob, S., Schindler, W.: Subsampling and knowledge distillation on adversarial examples: New techniques for deep learning based side channel evaluations. In: Dunkelman, O., Jacobson, Jr., M.J., O'Flynn, C. (eds.) SAC 2020. LNCS, vol. 12804, pp. 567–592. Springer, Cham (Oct 2020). https://doi.org/10.1007/978-3-030-81652-0_22
8. Gohr, A., Laus, F., Schindler, W.: Breaking masked implementations of the clyde-cipher by means of side-channel analysis A report on the CHES challenge side-channel contest 2020. IACR TCHES **2022**(4), 397–437 (2022). https://doi.org/10.46586/tches.v2022.i4.397-437
9. Graves, A., Fernández, S., Gomez, F., Schmidhuber, J.: Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: Proceedings of the 23rd International Conference on Machine Learning. pp. 369–376 (2006). https://doi.org/10.1145/1143844.1143891
10. Grosso, V.: Scalable key rank estimation (and key enumeration) algorithm for large keys. Cryptology ePrint Archive, Report 2018/175 (2018), https://eprint.iacr.org/2018/175
11. Karayalcin, S., Perin, G., Picek, S.: Diffuse some noise: Diffusion models for measurement noise removal in side-channel analysis. In: Boura, C., Mashatan, A., Miri, A. (eds.) SAC 2025. LNCS, vol. 16207, pp. 724–752. Springer, Cham (Aug 2026). https://doi.org/10.1007/978-3-032-10536-3_27
12. Kim, J., Picek, S., Heuser, A., Bhasin, S., Hanjalic, A.: Make some noise: Unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR TCHES **2019**(3), 148–179 (2019). https://doi.org/10.13154/tches.v2019.i3.148-179, https://tches.iacr.org/index.php/TCHES/article/view/8292
13. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems 25 (NeurIPS 2012). pp. 1106–1114 (2012), https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html
14. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: International Conference on Security, Privacy, and Applied Cryptography Engineering. pp. 3–26 (2016). https://doi.org/10.1007/978-3-319-49445-6_1
15. Poussier, R., Standaert, F.X., Grosso, V.: Simple key enumeration (and rank estimation) using histograms: An integrated approach. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 61–81. Springer, Berlin, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53140-2_4
16. Prouff, E., Strullu, R., Benadjila, R., Cagli, E., Dumas, C.: Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. Cryptology ePrint Archive, Report 2018/053 (2018), https://eprint.iacr.org/2018/053
17. Robissout, D., Zaid, G., Colombier, B., Bossuet, L., Habrard, A.: Online performance evaluation of deep learning networks for profiled side-channel analysis. In: Bertoni, G.M., Regazzoni, F. (eds.) COSADE 2020. LNCS, vol. 12244, pp. 200–218. Springer, Cham (Apr 2020). https://doi.org/10.1007/978-3-030-68773-1_10
18. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention. pp. 234–241. Springer (2015). https://doi.org/10.1007/978-3-319-24574-4_28

19. SeatGeek: TheFuzz: Fuzzy string matching in Python, https://github.com/seatgeek/thefuzz, version 0.22.1 (commit a1a8cde), tagged 2023-10-30. Accessed 2026-02-11
20. Shalev-Shwartz, S., Shamir, O., Shammah, S.: Failures of gradient-based deep learning. In: International Conference on Machine Learning. pp. 3067–3075. PMLR (2017)
21. Shanmugam, D., Blalock, D.W., Balakrishnan, G., Guttag, J.V.: Better aggregation in test-time augmentation. In: 2021 IEEE/CVF International Conference on Computer Vision (ICCV). pp. 1194–1203. IEEE (2021). https://doi.org/10.1109/ICCV48922.2021.00125
22. Sun, Y., Wang, X., Liu, Z., Miller, J., Efros, A.A., Hardt, M.: Test-time training with self-supervision for generalization under distribution shifts. In: Proceedings of the 37th International Conference on Machine Learning (ICML 2020). pp. 9229–9248. Proceedings of Machine Learning Research, PMLR (2020), http://proceedings.mlr.press/v119/sun20b.html
23. Veyrat-Charvillon, N., Gérard, B., Renauld, M., Standaert, F.X.: An optimal key enumeration algorithm and its application to side-channel attacks. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 390–406. Springer, Berlin, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-35999-6_25
24. Wang, Z., Ding, Y., Wang, A., Zhang, Y., Wei, C., Sun, S., Zhu, L.: SPA-GPT: General pulse tailor for simple power analysis based on reinforcement learning. IACR TCHES **2024**(4), 40–83 (2024). https://doi.org/10.46586/tches.v2024.i4.40-83
25. Weissbart, L., Chmielewski, Ł., Picek, S., Batina, L.: Systematic side-channel analysis of Curve25519 with machine learning. Journal of Hardware and Systems Security **4**, 314–328 (2020). https://doi.org/10.1007/s41635-020-00106-w
26. Young, R., Mather, L., Oswald, E.: Comparing key rank estimation methods. In: International Conference on Smart Card Research and Advanced Applications. pp. 188–204. Springer (2022). https://doi.org/10.1007/978-3-031-25319-5_10
27. Yu, H., Wang, M., Song, X., Shan, H., Qiu, H., Wang, J., Yang, K.: Noise2clean: Cross-device side-channel traces denoising with unsupervised deep learning. Electronics **12**(4), 1054 (2023)

A Baseline Neural Network Architectures

This appendix provides full architectural details for the two baseline neural network architectures used in our experiments. The U-Net architecture, the application of which to side-channel analysis is a central point of our main contribution, is described in Section 2 of the main text.

A.1 VGG-like Network

Overview The VGG-like network architecture in our implementation follows the design principles of state-of-the-art CNNs for SCA, specifically the VGG-like architecture of Weissbart et al. [25].

Parameters Our implementation takes as input a one-dimensional trace of length `input_length` and produces outputs for `num_classes` target classes. The network structure is defined by two parameter lists:

- `conv_filters`: A list specifying the number of filters in each convolutional layer. Each convolutional block consists of a Conv1D layer followed by BatchNormalization and MaxPooling1D with stride 2, which progressively reduces the temporal dimension while increasing feature depth.
- `dense_sizes`: A list defining the sizes of fully connected layers that follow the convolutional feature extractor.

Our implementation includes several regularization mechanisms to prevent overfitting, including L_2 weight regularization (controlled by the `l2_reg` parameter) and optional batch normalization for the dense layers. The kernel size for convolutional operations defaults to 3, which we found to be effective for the SCAAML datasets and synthetic tasks. On the whole, our architecture is a straightforward reimplementation of the network described in [25], except that we offer a flexible number of convolutional blocks, flexible regularization mechanisms, and flexible activation functions.

Network Structure For the output layer, we use a sigmoid activation for binary classification tasks and softmax for multi-class problems, although this can be overridden with the `output_activation` parameter. The complete network structure can be represented as:

1. Input layer: 1D trace of length `input_length`
2. Reshape trace to add channel dimension: $(input_length, 1)$
3. For each filter size in `conv_filters`:
   - Conv1D with specified number of filters, kernel size 3, and activation function
   - BatchNormalization
   - MaxPooling1D with pool size and stride of 2
4. Flatten operation
5. For each size in `dense_sizes`:
   - Dense layer with specified size and activation function
   - Optional BatchNormalization (if `dense_batch_norm=True`)
6. Output layer: Dense with `num_classes` units and appropriate activation

A.2 Subsampling-based Network

Overview The subsampling-based network architecture is specifically designed for multi-target extraction from long traces, based directly on Gohr et al. in [7]. We slightly generalize their network architecture by adding optional support for dropout regularization and custom activation functions inside the residual layers.

Subsampling invariance is built into the network by making the network splice the input trace into q_1 decimated parts of length q_2 right after the input layer.

Each decimation is offset from the previous by one sample. Each splice is first compressed to an internal width nodes wide representation by a dense layer, and then further processed by a residual tower of fully connected feed-forward layers. Inside the residual tower, each slice of the state is connected to the previous and the next slice, which means that the network can propagate information between slices, compensating thereby for the loss of information caused by the decimation of the input data performed in each slice.

Parameters Our implementation accepts several parameters that customize the network's behavior. We list below the main ones:

- `outputs`: Number of target classes or secret bits to be extracted simultaneously from the trace.
- `q1, q2`: Reshape parameters specifying how the input trace of length $p = q_1 \times q_2$ is reshaped into a 2D representation (q_1, q_2). The parameter q_1 effectively determines the subsampling factor.
- `depth`: Number of residual convolutional blocks, typically set to 2 for most applications to balance expressiveness with computational efficiency.
- `width`: Number of convolutional filters used throughout the network. If not specified, it defaults to `outputs`, establishing a direct correspondence between network capacity and the number of targets.
- `activation`: Activation function used in the residual layers.
- `output_activation`: Activation function used in the output layer.

Dropout regularization is optionally supported, but in our experiments, we found it to be ineffective.

Network Structure The network structure is identical to the one proposed by Gohr et al. [7], except that we allow for custom activation functions and dropout regularization. The complete network structure can be represented as:

1. Input layer: 1D trace of length $p = q_1 \times q_2$
2. Reshape operation: Transform input to (q_1, q_2) to create a structured representation
3. Initial processing:
   - BatchNormalization
   - Kernel-size 1 Conv1D with `width` filters (projection layer)
4. Residual processing blocks (repeated `depth` times):
   - BatchNormalization
   - Kernel-size 3 Conv1D with `width` filters
   - Dropout (optional)
   - Residual connection via Add layer
5. Subsampling and output:
   - AveragePooling1D with pool size q_1 (averages over the q_1 subsequences, reducing the sequence dimension to 1)
   - Flatten operation
   - Dropout (optional)
   - Dense output layer with `outputs` units

B The PICK Synthetic Task

This appendix provides the full description and experimental results for the PICK synthetic task introduced in Section 3.2. Together with the SIFT task described in the main text, PICK helps explain why standard neural network architectures struggle with multi-target extraction from long traces.

B.1 Task Definition and Data Format

The Predetermined Index Content Keeping task (PICK-n-k) tests whether CNNs can perform massive multi-target extraction when alignment is perfect, but the extraction map is not translation-equivariant. The task is defined as follows:

- **Input:** Fixed-length binary sequences of length n, drawn uniformly at random from $\{0,1\}^n$.
- **Target output:** Extract the bits at k predetermined fixed positions $p_1, \ldots, p_k \in \{0, \ldots, n-1\}$.

For example, with $n = 100$ and $k = 20$, the positions $p_1, \ldots, p_{20}$ might be $\{5, 12, 19, 27, \ldots, 94\}$. Given an input sequence of 100 bits, the network must output the 20 bits at exactly these positions, ignoring the other 80 bits.

The data is again noise-free symbolic sequences. Unlike SIFT, there is no alignment problem here: position p_i is always at index p_i in the input. The *only* challenge is learning a position-dependent map: "output bit i should come from input position p_i".

B.2 Connection to SCA

PICK models side-channel scenarios where:

- The trace is perfectly aligned (no jitter).
- Multiple target variables leak at known fixed positions (e.g., key bytes from an unrolled AES round, or bits from perfectly aligned windowed scalar multiplication).
- The goal is that the network extract all targets simultaneously in one forward pass.

The goal of PICK is to provide a massively simplified model of these situations.

B.3 Expected Difficulties for CNNs

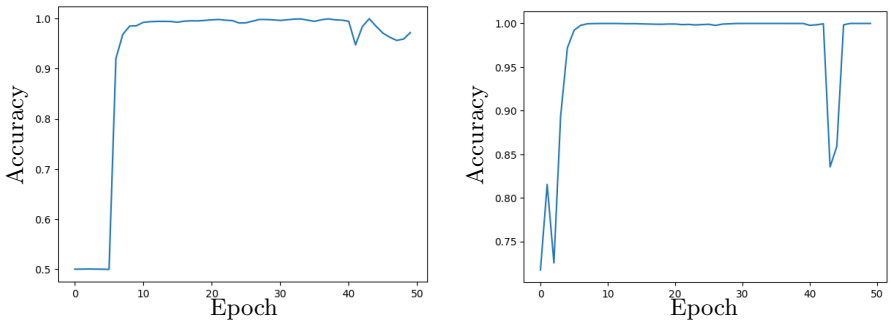
A single-layer multilayer perceptron (MLP) with n input nodes, k output nodes, and linear activations can solve PICK perfectly after seeing n linearly independent training examples (the learning problem reduces to solving a linear system). The task is trivial for architectures without structural biases.

However, CNNs have a built-in bias toward translation-equivariant representations: a feature detector that activates for a certain pattern should activate the same way regardless of where that pattern appears in the input. PICK explicitly violates this assumption: the network must learn that a 1 at position p_i is important (and should be output) while a 1 at position $p_i + 1$ might be irrelevant (and should be ignored). With many targets (in our experiments, we use $k = 32$) spread across hundreds of input positions (in our experiments, $n = 512$), the convolutional architecture we tried *does* learn to extract the targets, but faces significant difficulty in doing so. We expect that these difficulties get much worse for real targets, because then, the extraction task itself is already non-trivial.

B.4 Experimental Results on PICK

We tested the VGG-like and subsampling-based architectures on PICK-512-32. For the VGG-like network, we used 4 convolutional layers (32, 64, 128, 128 filters), a prediction head with three layers of 512 units each, sigmoid outputs, and batch normalization in the prediction head.

Figure 9a shows the accuracy of the VGG-like network as a function of training epoch. The network does converge, but goes through an initial phase of accuracy stuck at 50%, and never manages to reach 100% accuracy. By comparison, a subsampling-based network with comparable settings reached 100% test accuracy within a few epochs (Figure 9b), which is expected since the subsampling architecture is not constrained by translation-equivariance biases.



(a) VGG-like network **(b)** Subsampling-based network

Figure 9: Accuracy on PICK-512-32 as a function of training epoch.

These results confirm that PICK is indeed easy for architectures without translation-equivariance biases, but at least somewhat challenging for standard CNNs. This parallels the difficulties observed on the SCAAML datasets, where VGG-like networks fail at multi-target extraction but the subsampling-based architecture (which lacks translation-equivariance bias) succeeds in the well-aligned case.

C Discussion: Why Synthetic Tasks Matter

Together, the SIFT and PICK tasks explain the observed failure modes of standard SCA architectures on the SCAAML datasets. SIFT is trivially solvable algorithmically, yet standard VGG-like and residual CNN architectures have some difficulties with it when n is large (in the hundreds) and k is large (≥ 32). This directly explains their failure on SCAAML: it is not that they cannot learn to recognize leaked key bits in noisy power traces (in fact they can, given segmented data and single-target tasks), or even that the networks themselves lack the capacity to represent good solutions, but rather that biases of gradient descent learning and architectural biases of the networks themselves make it fundamentally difficult to solve the *indexing* and *localization* subproblems inherent in unsegmented multi-target extraction.

The U-Net with CTC loss succeeds on both SIFT-n-n and SCAAML because the network is unburdened from the need to learn to route leakage sites to specific output bits; that part is handled by CTC decoding. Instead, the network simply outputs a map of the leakage in the trace at the same resolution as the input trace, which is decoded by a classical algorithm. We propose the hypothesis that this architectural difference in handling misalignment, rather than any improvement in learning to extract complicated leakage from noise, explains why our approach succeeds where standard architectures fail.

D Additional CM1 Extraction Results

D.1 Trace Segmentation Visualization

Figure 10 illustrates the coarse-grained segmentation of a CM1 trace into its two active cryptographic regions, as described in Section 5. The prediction-error segmentation algorithm identifies the boundaries between inactive and active regions, and the two largest active segments are cut out and resampled as described in the main text.

D.2 Projected Full Key Extraction Success Rate

Figure 11 shows the projected success rate of the subsampling network as a function of the work factor of the meet-in-the-middle attack on CM1. This complements the analysis in Section 5: at a work factor of 2^{41} point multiplications or hash table lookups, the upper bound estimate predicts success in roughly 87% of cases. Note that $r_1 + r_2$ is an upper bound on the actual residual work, so the plotted success rates are lower bounds; the actual attack success rate is at least as high.

E Trace Corruptions

This appendix provides details on the trace corruption experiments that are briefly summarized in the main text.

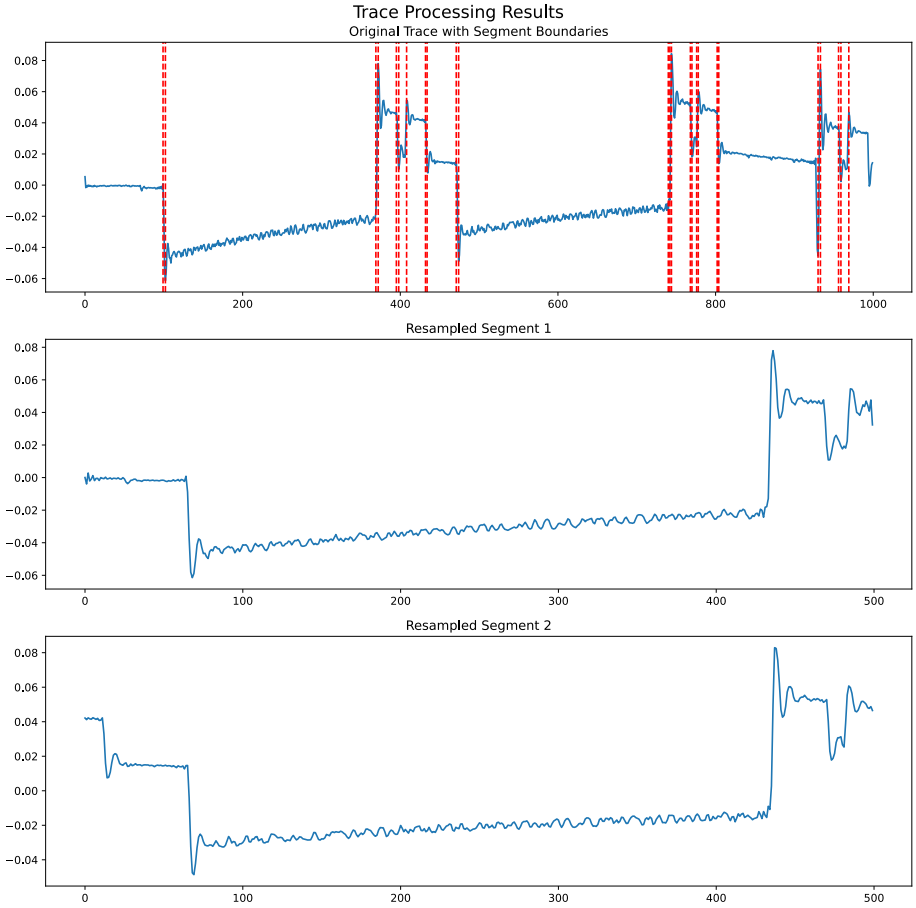


Figure 10: Segmentation of a CM1 trace into active regions using hard low-pass downsampling segmentation before non-maximum suppression. Horizontal axis is time in samples, vertical axis is power. **Top:** Full trace with candidate segment boundaries highlighted in red. **Middle:** First active segment with buffer region, x-axis relative to the start of the region. **Bottom:** Second active segment with buffer region, x-axis relative to the start of the region.

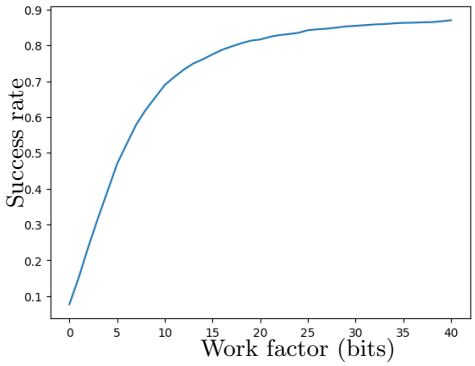


Figure 11: Projected success rate of the subsampling network as a function of the work factor of the meet-in-the-middle attack on CM1. Work factor is given on a $\log_2$-scale in terms of the number of bits that need to be tested for each key share.

E.1 Corruption Settings

To test robustness to realistic trace distortions, we applied several types of simulated corruptions to the preprocessed CM1 traces. Table 4 lists the corruption parameters used.

Table 4: Configuration parameters for trace corruptions used in robustness testing. These corruptions qualitatively simulate real-world effects such as system interrupts, clock frequency variations, and timing jitter.

Corruption Type	Parameter	Value
Interrupts	Number of interrupts	5
	Maximum duration	10 samples
Frequency Changes	Number of changes	3
	Maximum duration	50 samples
	Maximum slowdown	1.5x
Jitter	Strength	1.0

Each corruption type is implemented as follows:

– **Interrupts** are simulated by adding a constant offset to a short contiguous segment at a random position in the trace. The offset amplitude is drawn uniformly between 0.5× and 2× the trace mean, and the segment duration is drawn uniformly up to the maximum duration. This models the elevated power draw of an interrupt service routine briefly preempting the cryptographic computation, while preserving the underlying trace shape within the affected region.

– **Frequency changes** simulate dynamic frequency scaling by selecting random segments, stretching them in time by a factor drawn uniformly from $[1, \text{max_slowdown}]$ via linear interpolation, and then resampling the full trace back to its original length. This introduces local timing distortions analogous to clock frequency transitions.
– **Jitter** is modeled as a random time-axis perturbation: each sample index is shifted by a value drawn from $\mathcal{N}(0, \sigma)$ where σ is the jitter strength, and the trace is reconstructed by interpolating onto the original uniformly spaced time axis.

The corruptions listed in Table 4 were applied jointly to the preprocessed CM1 traces.

E.2 Corruption Visualization

Figure 12 compares an original CM1 power trace (first key share) with the same trace after applying the corruption setting. The corruptions visibly distort the trace, although the large-scale structure of the cryptographic operations remains recognizable.

E.3 Results Under Trace Corruptions

Table 5 summarizes the performance of both the subsampling-based network and the CTC-loss U-Net under trace corruptions, alongside the clean base condition for reference.

Table 5: Performance (similarity ratio) under trace corruptions, compared to the clean base condition. All networks were trained on clean first-key-share CM1 holdout traces only; no corruption-augmented training was performed.

Condition	**Subsampling**	**U-Net (21 ep.)**	**U-Net (201 ep.)**
Base (clean)	100%	96%	98%
Corrupted traces	73%	85%	85%

To calibrate these numbers, note that the expected similarity ratio between two independent uniformly random binary strings of equal length is approximately 80% (the Chvátal–Sankoff constant for alphabet size 2 [5]). The U-Net results of 85% under corruptions are therefore meaningfully above the random baseline, indicating that some modest amount of predictive power is retained. The subsampling-based network's 73% falls *below* the random baseline, which can occur when the predictions have a biased structure (for instance, a strong bias towards emitting one-bits) that reduces the length of the longest common subsequence with the ground truth.

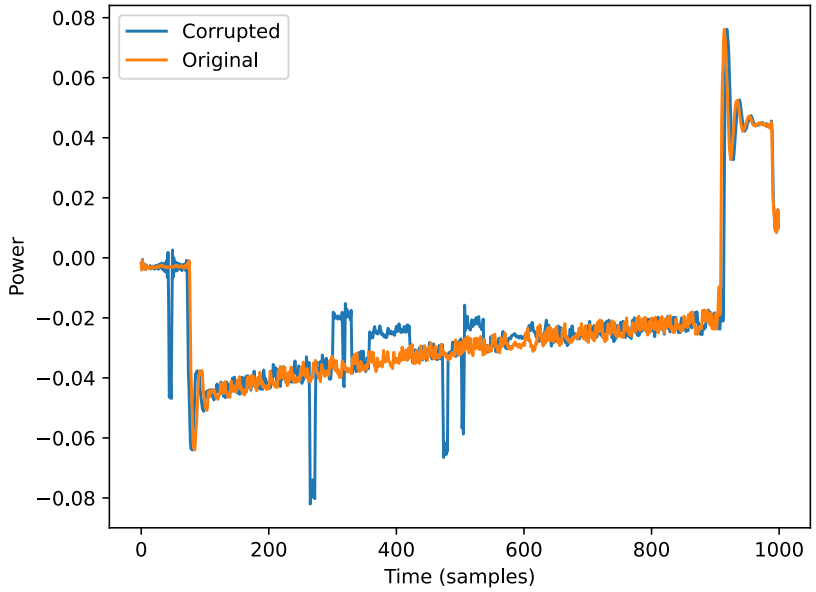


Figure 12: Comparison between an original CM1 power trace (first key share) and the same trace after applying corruptions. The corrupted trace includes 5 interrupts (maximal duration 10 samples), 3 frequency changes (maximal duration 50 samples, maximal slowdown factor 1.5), and jitter (strength 1.0, which means individual samples are displaced randomly with a standard deviation of 1.0 sample).

Neither model was trained on corrupted data – the robustness of the U-Net emerges purely from its architecture and the CTC training objective. The subsampling-based network, in contrast, relies on exact positional correspondence between input features and output predictions, which is disrupted by the jitter and frequency changes present in the corrupted traces.