UNIDLE: A Unified Framework for Deep Learning-based Side-channel Analysis

Suvadeep Hajra*, Soumi Chatterjee* and Debdeep Mukhopadhyay*
*Indian Institute of Technology Kharagpur, India

***Abstract*—Side-channel analysis (SCA) exploits the dependency of a cryptographic device's power consumption or electromagnetic (EM) emissions on the data being processed to extract secret keys. While deep learning (DL) has advanced SCA, most existing models struggle to scale to long traces and tend to be effective only against specific countermeasures. Therefore, their applicability is limited in implementations that employ multiple countermeasures simultaneously.**

This paper introduces UNIDLE, a UNIfied framework for Deep LEarning-based SCA, for building scalable and robust models for long traces. UNIDLE employs a hierarchical divide-and-conquer strategy: it segments each trace into smaller parts, which are independently processed by base models, and then aggregates their outputs using a top-level model. This approach enables efficient information extraction from long traces while making the model robust against combinations of masking and desynchronization-based countermeasures. UNIDLE also naturally supports multi-task learning, a useful DL approach for attacking countermeasures such as shuffling, while being less susceptible to the capacity bottlenecks observed in existing multi-task models.

Leveraging this framework, we develop HierNet, a novel DL model that integrates a shift-invariant base model and a transformer-based top-level model. HierNet demonstrates strong empirical performance across three datasets of masked implementations, reaching the guessing entropy 1 with fewer than 50 attack traces–where several state-of-the-art benchmark models fail even with up to $5K$ attack traces. Experiments on traces protected by shuffling show that HierNet can reach the guessing entropy 1 using 95% fewer attack traces than a state-of-the-art multi-task benchmark while requiring only one-tenth of the model size.

***Index Terms*—SCA, Deep Learning, Transformer Network**

I. Introduction

A CMOS device's power consumption, or EM emanation depends on the data being processed within the device. This characteristic is leveraged by SCAs to extract the secret key used in a cryptographic device. SCAs involve collecting power/EM profiles (commonly referred to as power/EM traces) from one or more devices during their execution and recovering the secret key by analyzing these traces using statistical techniques or machine learning methods. Template attack [1] and stochastic attack [2] are two classical approaches to SCAs. More recently, various machine learning (ML) techniques [3], including deep learning (DL) [4], are introduced in SCA.

In many practical scenarios, raw power/EM traces can span hundreds of thousands of features, with the leakages of relevant cryptographic operations occurring over a few small segments, known as Points of Interest (PoIs). However, most of the existing works in SCA literature conduct attacks on some pre-selected high signal-to-noise ratio (SNR) features or on a pre-selected attack window containing the PoIs. For example, classical SCA like template and stochastic attacks select a few (in the order of 100) high SNR features prior to the actual attack. Similarly, most of the ML and DL-based methods are investigated on some pre-selected attack window of size in the order of $1K$. However, in implementations protected by masking countermeasures [5], such feature selection requires the knowledge of additional intermediate variables of the cryptographic implementation, which might not always be available. Moreover, even with such knowledge, the feature selection might not be possible in the presence of jitter-based countermeasures (like random delay [6], clock jitter [7], and shuffling [8]). Therefore, the success of the most of the existing SCAs depends on the feasibility of feature selection.

Several deep learning (DL)-based research efforts [9]–[13] have focused on improving the scalability and effectiveness of DL models for long attack windows or long raw traces, thereby reducing their sensitivity to the attack window selection. Those models have demonstrated good performance even without selecting any smaller attack window. Moreover, several works, such as [10], [11], have demonstrated significantly improved performance when attacking long traces compared to attacking a smaller, selected attack window.

However, the performance of most of these methods significantly deteriorates when trace lengths increase or when multiple countermeasures are combined. For instance, the study by Perin et al. [11] demonstrates that their methods targeting masking countermeasures may require over 30 times more traces to achieve a successful attack (i.e., reach guessing entropy 1) on desynchronized traces compared to synchronized ones. Similarly, [13] has shown that the DL models proposed in [10] may fail to reach guessing entropy 1 even with up to $5K$ attack traces when the traces are misaligned using clock jitter. In contrast, only around 10 traces suffice in the absence of jitter. In [14], Bursztein et al. have not shown the success of their DL model, GPAM, against jitter-based countermeasures, though they demonstrated its success on traces with lengths more than $10M$. In [13], Hajra et al. demonstrated the excellent performance of their model, EstraNet, against several combinations of masking, random delay, and clock jitter countermeasures. However, their experiments were conducted on relatively small trace windows, with up to $40K$ features. The selection of such a small attack window may not be feasible in the presence of countermeasures like masking or desynchronization.

These findings suggest that different DL models tend to

be effective only on relatively short traces or against specific countermeasures, limiting their applicability in scenarios where selecting a small attack window is infeasible or multiple countermeasures are combined in a single implementation. Moreover, most DL models developed for long-trace SCA are not readily compatible with multi-task learning–a common strategy for attacking countermeasures like shuffling. As a result, novel multi-task DL models are often developed from scratch for attacking shuffling countermeasures [15], increasing both development time and cost.

In this work, we propose UNIDLE, a UNIfied framework for Deep LEarning-based SCA on long traces, for building DL models scalable to long traces while being effective against various combinations of countermeasures. UNIDLE adopts a divide-and-conquer strategy: each trace is segmented into smaller parts, which are independently processed by base models. The outputs from these base models are then aggregated using a top-level DL model. This divide-and-conquer approach enables hierarchical information extraction from long traces while retaining computational efficiency. Moreover, the two-stage design of UNIDLE allows independent design choices at the local (base model) and global (top-level model) levels, enhancing its effectiveness against combined masking and desynchronization-based countermeasures. Additionally, UNIDLE naturally supports multi-task learning, while being less susceptible to the capacity bottlenecks typically observed in existing multi-task DL models.

This work also leverages the UNIDLE framework to develop a DL model, HierNet, for SCA on long traces. HierNet employs a shift-invariant base model along with a transformer layer [16] with absolute positional encoding as the top-level model. This architecture enables robust performance against combinations of masking, random delays, clock jitter, and shuffling countermeasures, while remaining effectively scalable to long traces (exceeding $200K$ features in length). Furthermore, HierNet addresses the capacity bottleneck of existing multi-task models against shuffling-based countermeasures, achieving superior results with model sizes nearly one-tenth of an existing benchmark model.

Contributions:

The major contributions of the work are the following:

- We propose a unified framework, UNIDLE, for DLSCA on long traces. The framework adopts a hierarchical divide-and-conquer approach for effective and efficient information extraction, while enabling independent design choices at the local and global levels, enhancing robustness against combinations of masking and desynchronization-based countermeasures. Moreover, the framework naturally supports multi-task learning, while being less susceptible to the capacity bottlenecks observed in existing multi-task approaches.
- We leverage the UNIDLE framework to develop a DL model, HierNet, for SCA on long traces. HierNet employs a shift-invariant base model, offering strong robustness against bounded desynchronizations introduced by countermeasures such as random delay and clock jitter, while maintaining effectiveness against masking. Additionally, the use of a transformer layer as the top-level module further enhances the model's effectiveness against shuffling countermeasures.
- We empirically evaluated HierNet on three datasets with masked implementations and compared its performance against four existing benchmark models. The results indicate that HierNet performs significantly better on longer traces or in the presence of clock jitter countermeasures, reaching the guessing entropy 1 with fewer than 50 attack traces, whereas none of the other methods reached the same with up to $5K$ attack traces. Furthermore, in evaluations against shuffling countermeasures, HierNet reaches the guessing entropy 1 using over 95% fewer attack traces compared to a state-of-the-art multi-task benchmark model, while using only one-tenth the number of parameters.

The paper is organized as follows. Section II introduces the notations and provides the necessary background. In Section III, we briefly describe the UNIDLE framework. Section IV presents the HierNet architecture. Section V reports the empirical evaluation of HierNet. In Section VI, we discuss several potential future research directions. Finally, we conclude the paper in Section VII.

II. Preliminaries

A. Notations

The notational conventions used in the paper are as follows. A letter in the capital (like X), the corresponding small letter (like x), and the corresponding calligraphic letter (like $\mathcal{X}$) are respectively used to represent a random variable, an instantiation, and the domain of the random variable. Similarly, a capital letter in bold (like $\mathbf{X}$) and the corresponding small letter in bold (like $\mathbf{x}$) are respectively used to represent a random vector and its instantiation. We use a capital letter in Roman style (like M) to represent a matrix. The i-th element of a vector $\mathbf{x}$ is represented by $\mathbf{x}[i]$. Similarly, the element of i-th row and j-th column of a matrix M is represented by $\mathrm{M}[i, j]$. The notations $\mathbb{P}[\cdot]$ and $\mathbb{E}[\cdot]$ are respectively used to represent a random variable's probability distribution function and expectation.

B. Side-Channel Analysis

The power consumption and EM emanation of a semiconductor device are influenced by the values manipulated within the device. SCA takes advantage of this behavior of semiconductor devices to extract information about the sensitive variables in a cryptographic implementation, revealing the device's secret key. To achieve this, in an SCA, an attacker gains control of the target device, also known as Device Under Test (DUT). Then he/she collects power/EM measurements, known as traces, by repeatedly executing the encryption/decryption algorithm with various plaintexts/ciphertexts. Subsequently, the attacker uses statistical tests to deduce the device's secret key from the traces.

SCA can be categorized into two types: profiling SCA and non-profiling SCA. In profiling SCA, the attacker possesses a

clone of the DUT under control. With this clone device, he/she can create a profile of the DUT's power consumption or EM emission characteristics and employs this profile to carry out the attack on the DUT. On the other hand, in non-profiling SCA, the attacker lacks a clone device and, therefore, cannot build a power/EM profile a priory. Instead, he/she attempts to recover the secret key solely from the traces of the DUT. This paper considers profiling SCA.

C. Deep Learning-based Profiling SCA

Like any profiling SCA, DL-based profiling SCA also involves a two-phase process. In the initial phase, known as the profiling phase, the attacker configures a predefined key in the clone device and acquires a large number of traces by executing encryption (or decryption) operations on well-known plaintexts (or ciphertexts) using the device. For each trace, the attacker calculates the value of an intermediate secret variable denoted as $Z = F(X, K)$. Here, X represents a component of the random plaintext (or ciphertext), K represents a component of the known key, and $F(\cdot, \cdot)$ denotes a cryptographic primitive. Then, the attacker uses the traces and the values of the intermediate variable Z to train a DL model to predict Z from the $\mathbf{L}$, where $\mathbf{L}$ represents a trace. More specifically, the DL model takes a trace $\mathbf{L}$ as its input and outputs a probability distribution over the values of Z. Let $f(\cdot; \theta^*)$ denote the trained DL model, with θ^* denoting the model parameters. The output of the DL model for a trace, $\mathbf{l}$, can be given as $\mathbf{p} = f(\mathbf{l}; \theta^*)$ where $\mathbf{p} \in \mathbb{R}^{|\mathcal{Z}|}$ with $\mathbf{p}[j]$, for $j = 0, \cdots, |\mathcal{Z}| - 1$, denoting the predicted probability of the intermediate variable $Z = j$.

In the attack phase, attacker collect multiple traces $\{\tilde{\mathbf{l}}^i\}_{i=0}^{T_a-1}$ by executing the DUT. Let $\{\tilde{p}^i\}_{i=0}^{T_a-1}$ be the corresponding plaintexts (or ciphertexts). Then, he/she uses the set of attack trace-plaintext pairs $\{(\tilde{\mathbf{l}}^i, \tilde{p}^i)\}_{i=0}^{T_a-1}$ to compute the score of each key $k \in \mathcal{K}$ as

$$\hat{\delta}_k = \sum_{i=0}^{T_a-1} \log \mathbf{p}^i[F(\tilde{p}^i, k)] \tag{1}$$

where $\mathbf{p}^i = f(\tilde{\mathbf{l}}^i; \theta^*)$ is the output of the DL model for the i-th trace. The key with the highest score, i.e., $\hat{k} = \text{argmax}_k\, \hat{\delta}_k$, is considered as the predicted key. The prediction is said to be correct if $\hat{k} = k^*$ holds. Alternatively, the rank of the correct key within the list of all possible keys, sorted based on their respective scores $\hat{\delta}_k$, serves as a metric to determine the success level of the attack. The rank of the correct key averaged over multiple repetitions of the attack, referred to as guessing entropy, is a widely used metric to evaluate the success of an attack.

D. Threat Model

Many countermeasures, such as masking and shuffling, use internal random values to implement their protections. Depending on whether the attacker has knowledge of these internal random values during profiling, the attacker's threat model can be classified as either white-box or black-box. In a white-box threat model, the attacker is assumed to have access to the internal random variables during profiling. Conversely, in a black-box threat model, the attacker is not granted access to these variables even during profiling. In this paper, we assume a black-box threat model.

III. Unified Framework for DLSCA

In this section, we first outline the key challenges in DLSCA that this work aims to address. We then present the proposed UNIDLE framework, designed to overcome these challenges.

A. Challenges of DLSCA

Capturing Long Rang Dependency: When protected by a masking countermeasure, each sensitive intermediate variable of a cryptographic implementation is split into multiple shares such that any proper subset of the shares remains independent of the sensitive variable. Therefore, to extract any information about the sensitive variable, the adversary must combine the leakages of all the secret shares. In many implementations of masking countermeasures – particularly in software – the points of interest (PoIs) corresponding to the secret shares of a sensitive variable are scattered across the trace. Moreover, a single intermediate variable may leak at multiple PoIs within a trace. However, the distances between such PoIs can be large, especially in long traces. Consequently, a DL model must be capable of integrating information from PoIs distributed over a long input sequence.

Although several prior works such as [10], [11], [13], [14] have attempted to improve the ability of DLSCA to capture long-range dependencies, their performance tends to degrade either with increasing trace length or in the presence of other countermeasures. For instance, it has been observed that the performance of [10], [11] significantly deteriorates under countermeasures such as random delay [11] or clock jitter [10]. Furthermore, as we will demonstrate in this paper, the performance of [13] significantly degrades as the trace length increases. These observations highlight the need for novel DL models that remain robust against both long traces and combinations of diverse countermeasures.

Positional Ambiguity in Attacking Parallel Operations: Cryptographic algorithms often contain multiple identical parallel operations. For instance, each round of the AES algorithm includes 16 SubByte operations. In many implementations, each of these parallel operations leaks in different but identical segments of the trace. As an example, Figure 1 illustrates 16 distinct trace segments corresponding to the 16 SubByte operations in the first round of AES in the ASCADv2 dataset [15]. When training a DL model to attack a specific SubByte operation, the PoI must be selected from one of these 16 trace segments. However, since all these segments exhibit nearly identical patterns, DL models that do not utilize explicit or implicit positional information often struggle – even when attacking unprotected traces. Furthermore, in the presence of random delay or clock jitter, the exact position of the target PoI might shift across the traces. Therefore, the DL model should be able to tolerate bounded shifts in the positions of the PoI (i.e., local desynchronizations) introduced by such countermeasures, while also being capable of identifying the PoI corresponding to the target operation

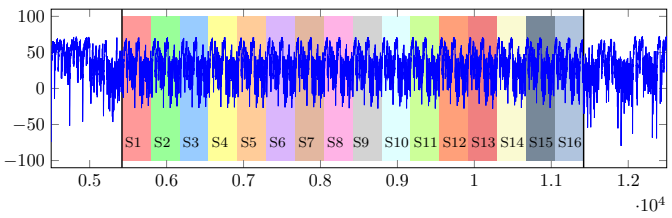


Fig. 1. Sixteen SubByte operations of AES on the traces of ASCADv2 dataset.

among PoIs of all parallel operations. Additionally, in the presence of shuffling countermeasures, the trace segment corresponding to a particular operation (e.g., one of the SubByte operations) varies randomly across traces. For example, the first SubByte operation may correspond to one segment in one trace and a completely different segment in another. Thus, DL models must be able to ignore positional information and instead leverage alternative cues – such as leakage from the permutation index – when attacking implementations protected by shuffling countermeasures.

Multi-task Learning: Secret key recovery using SCA typically follows a divide-and-conquer strategy, where the full key is partitioned into multiple subkeys, and each subkey is recovered independently by repeatedly applying an SCA algorithm. In multi-task DLSCA, a DL model is trained to recover all subkeys in parallel [15], [17]–[19]. In addition to enabling parallel recovery of multiple subkeys, multi-task DLSCA has also been leveraged to simultaneously target multiple intermediate variables associated with a single subkey, thereby improving attack performance [17], [20].

However, a major drawback of existing multi-task DLSCA models (e.g., [15]) lies in their susceptibility to capacity bottlenecks, especially against countermeasures such as shuffling. To facilitate simultaneous information flow across multiple tasks, such models require significantly wider architectures compared to single-task models. This increased model size not only raises the risk of overfitting, but also often leads to deteriorated performance.

B. The UNIDLE Framework

The overall UNIDLE framework is illustrated in Figure 2. It adopts a divide-and-conquer strategy to effectively process long side-channel traces. Specifically, each long trace is divided into multiple (possibly overlapping or discontinuous) segments. Each segment is processed independently by a base DL model, producing an output vector. Let $\mathbf{x}_i$ be the output from the i-th segment. Then $[\mathbf{x}_0, \cdots, \mathbf{x}_{m-1}]$, serves as the input to a top-level DL model, which then produces the final prediction. In the case of T distinct tasks, the framework employs T parallel top-level modules, one dedicated to each task. The core features of the UNIDLE framework and their roles in addressing the discussed challenges are detailed below.

Hierarchical Feature Learning: While the ability to capture long-distance dependencies is critical for DLSCA on long traces – especially in the presence of countermeasures like masking – not all DL architectures are equally effective in this regard. For example, Convolutional Neural Networks (CNNs)

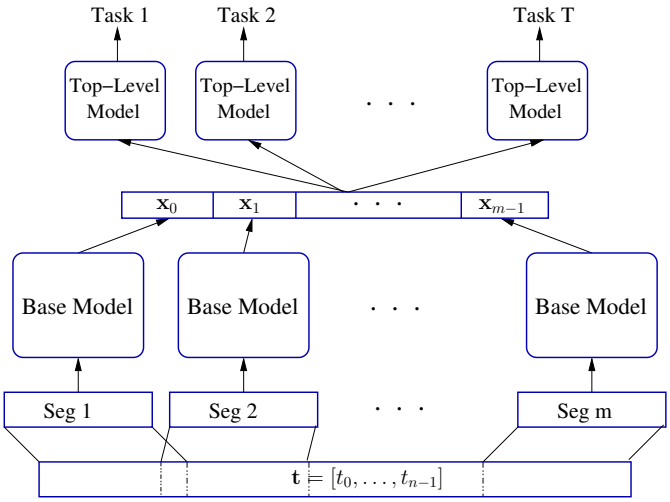


Fig. 2. UNIDLE Framework.

and Recurrent Neural Networks (RNNs) struggle to capture long-distance dependencies due to their limited receptive fields or vanishing-and-explding gradient problem. In contrast, Feed-Forward Networks (FFNs) can capture long-distance dependencies easily but lack shift-invariance, making them ineffective when masking is combined with desynchronization-based countermeasures. Transformer Networks (TNs) offer a compelling alternative, as they are both capable of modeling long-range dependencies and can be made shift-invariant [13], [21]. However, their quadratic complexity in both computation and memory with respect to input length makes them impractical for long traces. Although EstraNet [13], a linear-complexity Transformer variant, has been proposed to address this limitation, our experiments show that it performs poorly on longer traces. The UNIDLE framework seeks to overcome these limitations through a hierarchical approach. It first uses base models to extract features from smaller trace segments, and then combines their outputs using a top-level DL model. This design balances computational efficiency with the ability to integrate information across long traces, enabling robust performance against masking countermeasures on long traces.

Adaptability to Composit Positional Encoding: One of the key advantages of DL is its ability to process structured data by implicitly or explicitly incorporating positional information of features. For example, FFNs implicitly encode absolute positional information through fixed input ordering. CNNs capture relative positional patterns via their local receptive fields, while TNs can encode either absolute or relative positional information through explicit positional encoding mechanisms. In the context of SCA, such positional information plays a critical role in determining both data efficiency and robustness against desynchronization-based countermeasures, such as random delays, clock jitter, or shuffling. The two-stage architecture of the UNIDLE framework offers a flexible design for selectively incorporating positional information at local and global levels. For instance, encoding relative positional information in the base model and absolute positional information in the top-level model can make the overall model robust against bounded desynchronization while improving data

efficiency. Conversely, using relative encoding locally and no positional encoding globally can make the model resilient to combinations of random delay and shuffling countermeasures.

Overcoming the Capacity Bottleneck: In multi-task DL models, the output of the shared trunk network is typically reduced to a single vector, say $\mathbf{x} \in \mathbb{R}^d$, which is then passed to multiple task-specific heads for different tasks. To enable sufficient information flow for all tasks, the dimensionality d of this shared vector must be significantly increased compared to its single-task counterpart. However, this increased dimensionality leads to a larger model size – with the number of parameters often scaling as $\mathcal{O}(d^2)$ – which makes the model more susceptible to overfitting, poorer performance and leads to a higher training cost.

In contrast, the UNIDLE framework avoids collapsing the outputs of all segments into a single vector. Instead, it maintains a distributed representation through multiple output vectors $\mathbf{x}_0, \ldots, \mathbf{x}_{m-1}$, each produced by different base models applied on different segment of the trace. If a particular task – say, task-i – depends predominantly on a subset of features $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_k}$, the top-level model corresponding to task-i can assign higher weights to these vectors using attention mechanisms, while down-weighting less relevant ones. A different task might depend on a different subset of the features. Therefore, such a distributed representation of the input to the task-specific heads helps overcome the capacity bottleneck typically observed in multi-task models.

In the next section, we develop a DL model, named HierNet, that follows the UNIDLE framework.

IV. HierNet

The goal of designing the HierNet architecture is to ensure effectiveness against various combinations of masking, random delay, clock jitter, and shuffling countermeasures while maintaining scalability to long traces. To achieve this, we employ a shift-invariant model as the base, which makes HierNet inherently shift-invariant within each segment. Additionally, we use a transformer layer with absolute positional encoding as the top-level module. The use of transformer layers at the top-level makes the parameterization independent of both the trace and segment lengths, enabling the model to generalize effectively across a wide range of trace configurations. Furthermore, the use of a transformer layer makes the model adaptable to shuffling countermeasures.

Details of the top-level and base models, along with the trace segmentation and parameter-sharing strategies, are provided below.

A. Designing the Top-level DL Model

When designing the top-level DL model, we primarily consider three criteria. First, we aim to ensure that the model's parameterization remains independent of the trace lengths and segmentations. The number of segments, denoted as m, can vary depending on the trace length and chosen segment length, thereby affecting the input length to the top-level model. Therefore, we select a model whose parameterization is independent of the input length. Second, the model should effectively capture long-distance dependencies, enabling it to combine leakages from all segments using a minimal number of layers. Finally, we require the model to be able to easily exploit or ignore the segment ordering information, depending on the application. Based on these criteria, we selected a transformer layer [16] as the top-level DL model, as it meets all three requirements. More specifically, given an input sequence $\mathrm{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_{m-1}]^T \in \mathbb{R}^{m \times d}$, a transformer layer uses a self-attention layer and a position-wise feed-forward layer to generate the output sequence $\mathrm{Y} = [\mathbf{y}_0, \ldots, \mathbf{y}_{n-1}]^T \in \mathbb{R}^{m \times d}$ as follows:

$$\hat{\mathrm{Y}} = f_{SA}(\mathrm{X}) + \mathrm{X}$$
$$Y = f_{PFF}(\hat{\mathrm{Y}}) + \hat{\mathrm{Y}}$$

where $f_{SA} : \mathbb{R}^{n \times d} \mapsto \mathbb{R}^{n \times d}$ is the function representing the self-attention layer and $f_{PFF} : \mathbb{R}^{n \times d} \mapsto \mathbb{R}^{n \times d}$ is the function representing the position-wise feed-forward layer. The position-wise feed-forward layer independently applies a non-linear transformation to each feature vector, $\hat{\mathbf{y}}_i$, increasing the non-linearity (and, thus, its representation power) of the transformer layer. On the other hand, the self-attention layer captures the inter-dependencies among different features [12], [13], [21]. It is worth mentioning that the number of parameters in both the self-attention and position-wise feed-forward layer is independent of the sequence length n. Therefore, it can scale to a very long sequence without being prone to overfitting. More details about the transformer Network are given in Appendix A.

In HierNet, we incorporate positional encoding of the trace segments into the self-attention mechanism, enabling the model to exploit or ignore segment ordering information depending on the application. For instance, in the presence of shuffling countermeasures, the segment containing the PoI for a particular operation may vary from trace to trace. In such cases, we want the model to ignore segment ordering. Conversely, in the absence of shuffling, the segment containing the PoI for a particular operation remains fixed across traces. Therefore, the attention mechanism should leverage positional information to focus more on the segment at that fixed position in the segment order. We incorporate positional information into the self-attention operation as follows:

Self-attention with Absolute Positional Encoding: In vanilla self-attention, the i-th output feature $\hat{\mathbf{y}}_i$, $i = 0, 1, \ldots, m-1$, is computed as $\hat{\mathbf{y}}_i = \sum_{j=0}^{m-1} p_{ij} \phi(\mathbf{x}_j)$ with p_{ij}, the attention weight from the i-th feature (the output from the i-th segment) to the j-th feature, is given as

$$p_{ij} = \text{softmax}\left(\frac{\langle f(\mathbf{x}_i), g(\mathbf{x}_j)\rangle}{\sqrt{d_k}}\right)_j = \frac{\exp\left(\langle f(\mathbf{x}_i), g(\mathbf{x}_j)\rangle/\sqrt{d_k}\right)}{\sum_{l=0}^{n-1} \exp\left(\langle f(\mathbf{x}_i), g(\mathbf{x}_l)\rangle/\sqrt{d_k}\right)},$$

where $f(\cdot), g(\cdot)$ and $\phi(\cdot)$ are three vector valued functions of the input feature $\mathbf{x}_i$ ($i = 0, \ldots, m-1$), $\langle \cdot, \cdot \rangle$ denotes the dot product between two vectors and d_k be the dimension of the outputs of f (or g). We incorporate the absolute positional encoding into the of attention probabilities as follows:

$$p^a_{ij} = (1-\gamma)\,\text{softmax}\left(\frac{\langle f(\mathbf{x}_i), g(\mathbf{x}_j)\rangle}{\sqrt{d_k}}\right)_j + \gamma\,\text{softmax}\left(\frac{\langle \mathbf{v}_i, \mathbf{v}_j\rangle}{\sqrt{d_k}}\right)_j \tag{2}$$

$$= (1-\gamma)\frac{\exp\left(\langle f(\mathbf{x}_i), g(\mathbf{x}_j)\rangle/\sqrt{d_k}\right)}{\sum_{l=0}^{n-1} \exp\left(\langle f(\mathbf{x}_i), g(\mathbf{x}_l)\rangle/\sqrt{d_k}\right)} + \gamma\frac{\exp\left(\langle \mathbf{v}_i, \mathbf{v}_j\rangle/\sqrt{d_k}\right)}{\sum_{l=0}^{n-1} \exp\left(\langle \mathbf{v}_i, \mathbf{v}_l\rangle/\sqrt{d_k}\right)}, \tag{3}$$

where $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{m-1}$ are m distinct vectors and $\gamma \in [0, 1]$ is a hyper-parameter. The values of vectors $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{m-1}$ are learned along with other parameters of the model during training. Note that the first component of the R.H.S. of Eq. 2 or 3 computes the attention probability based on the similarity (or inter-dependency) between the two features, $\mathbf{x}_i$ and $\mathbf{x}_j$. The second component computes the probability based on their absolute positions or indices i, j. The hyper-parameters γ controls the relative weights of the two components in the attention probability p^a_{ij}.

B. Selection of the Base Model

The length of each trace segment is expected to be bounded. DLSCA on shorter traces has been extensively studied in the literature. Therefore, we explored existing DLSCA models from prior work as potential base models in HierNet. Since one of the primary objectives of HierNet is to ensure robustness against random delay and clock jitter countermeasures, we specifically considered shift-invariant DL models for the base architecture. Ultimately, we selected EstraNet from [13] due to its demonstrated superior performance against combinations of masking, random delay, and clock jitter countermeasures.

C. Segmentation

HierNet intends to inherit the robustness of EstraNet against random delay and clock jitter countermeasures by employing EstraNet models to process each trace segment. However, segmenting the input trace in HierNet can potentially compromise this robustness. For example, misalignments in traces may cause some PoI near a segment boundary to oscillate between segments. In one trace, the PoI might belong to one segment, while in another trace, it could be shifted to a different one, resulting in the PoI not being processed in any segment. Such issue can be mitigated by making the segment length reasonably large. Additionally, we ensure significant overlap between consecutive segments. Specifically, if one segment ends at feature t, the next segment starts from feature $t - o$, ensuring that the span from $t-o$ to t belongs to both segments. It's important to note that if the displacements are bounded by length o, each PoI will always remain within one of the two consecutive segments. However, if displacements exceed o, increasing its value (by setting a hyper-parameter) can enhance HierNet's robustness against larger desynchronizations, albeit at the cost of increased computational overhead.

D. Parameter Sharing among Base Models

To keep HierNet's model size small and to prevent it from overfitting, the parameters of all base models are shared. However, during our initial experiments, we observed that sharing the parameters of the last layer (which is a softmax attention layer) of the base models resulted in a deceleration of training convergence for HierNet. We hypothesize that the parameter-sharing of the softmax attention layer was forcing each segment's output towards a unified representation, thereby slowing the training progress. Hence, we opted not to share the parameters of last layer of the base models.

Next section experimentally evaluates HierNet.

V. Experimental Results

The experimental evaluation is divided into three distinct parts. First, we assess HierNet's ability to capture long-range dependencies by conducting experiments on datasets with masking countermeasures and long traces. Second, we examine its robustness to positional ambiguity arising from identical trace patterns of different parallel operations, using a dataset from an unprotected software implementation of AES. Finally, we evaluate HierNet's effectiveness against shuffling and its ability to overcome capacity bottlenecks.

A. Benchmark Models

We compare the performance of HierNet with four benchmark models, referred to as EffCNN [22], PolyCNN [9], LSTMNet [10], and EstraNet [13]. The EffCNN models are CNN models with a flattening layer, whereas the PolyCNN model is a CNN model with a global average-pooling layer. On the other hand, LSTMNet uses softmax attention after a bi-directional LSTM layer to accumulate information spread over a long trace. EstraNet [13] is a transformer network-based shift invariant model with linear time and memory complexity with respect to trace length. Further details of the models can be found in Appendix C.

B. Details of Hyper-parameters and Training

HierNet uses EstraNet model as a component in its architecture. The hyper-parameters related to EstraNet in HierNet are adopted from [13], with a few exceptions. Notably, following the recommendation of [13], the $\hat{\beta}_2$ hyper-parameter of the EstraNet model has been tuned for each dataset. In the top-level DL model, we set γ hyper-parameter to 1. We set the number of heads of the self-attention layers to 8, though setting it to some other value like 1 also provides similar results. HierNet introduces two new hyper-parameters: segment length and the overlap length between two consecutive segments. Across all datasets, the segment length has been set to $5K$, and the overlap length is set to $2K$.

We train HierNet for $4M$ training steps utilizing the Adam optimizer. Following the previous literature [13], [21] of transformer network for SCA, it has been trained using a cosine decay with a linear warmup learning rate schedule. More precisely, the learning rate is linearly increased from 0 to the maximum value of 2.5e-4 over t_{warmup} training steps. Subsequently, it is gradually reduced to $0.004 \times$2.5e-4 over the remaining $4M - t_{warmup}$ steps, following a cosine learning rate schedule. The default value for t_{warmup} is set to $1M$; however, it was observed that, sometimes, the model did not train effectively for this default value. In such cases, t_{warmup} was increased to either of $2M$ or $4M$.

C. Experimental Results Against Masking Countermeasures

In this section, we evaluate the effectiveness of the models in capturing long-distance dependencies under masking countermeasures on long traces. To this end, we conduct experiments on three datasets derived from software implementations of masking countermeasures. The trace lengths in these

TABLE I
STATISTICS OF THE DATASETS

	ASCADf	ASCADr	CHES20
# of Profiling traces	50K	200K	200K
# of Validation traces	5K	10K	10K
# of Attack traces	5K	10K	10K
Trace Length	100K	250K	62.5K
Mask order	1	1	2

datasets are $100K$, $250K$, and $62.5K$ features, respectively – significantly large and therefore well-suited for assessing the models' ability to learn long-distance dependencies. We first perform evaluations with added random delay, followed by evaluations with added clock jitter.

Datasets Used: We used two datasets of first-order masked AES implementation, namely ASCAD Fixed Key[1] (ASCADf) and ASCAD Random Key[2] (ASCADr) datasets, and a second-order masked implementation of Clyde-128 Tweakable Block Cipher, namely CHES 2020 CTF dataset[3] (CHES20). The first two datasets were collected from an 8-bit ATMega8515 microcontroller, whereas the third dataset was collected from ARM Cortex-M0 microcontroller. We target first round's third sbox for the first two datasets and 17th column from the right of the state matrix after the first subbyte operation for the third dataset. The datasets's details are given in Appendix B. Table I summarizes the dataset statistics.

1) Experiments with Random Delay: In this section, we present our experimental results with random delay.

Experimental Setup: To introduce random delay effect, we added random shifts in the traces. In the profiling traces, the random delay has been introduced using a profiling desync of 600. In other words, we shifted each profiling traces by a random displacement in the range $[0, 600)$. The data augmentation has been further performed using a maximum desync of 400. Evaluations of the trained models were conducted for two attack desyncs: 600 and 1000. As before, we use the minimum number of attack traces required to reach the guessing entropy 1 (denoted by T_{GE1}) as the evaluation metric.

Results: The results are detailed in Table II. From the table, it is evident that HierNet demonstrates notable performance, requiring approximately 2 to 3 traces to achieve the guessing entropy 1 across all datasets. In contrast, the PolyCNN model performs poorly across the datasets, with relatively better performance on the ASCADr dataset. Similarly, the EffCNN model shows poor performance on all datasets. Specifically, our investigation revealed that training EffCNN on the ASCADr and CHES20 datasets proceeds too slowly to reduce the training loss significantly below the level of random guessing, even after 60 hours of training (see Appendix D). In comparison, HierNet completes training within 24 to 40 hours. The LSTMNet model performs significantly worse on the ASCADf dataset, although it achieves performance comparable to HierNet on the ASCADr and CHES20 datasets.

[1]https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_fixed_key

[2]https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_variable_key

[3]https://ctf.spook.dev/

TABLE II
THE T_{GE1} VALUES (LOWER IS BETTER) FOR VARIOUS DL MODELS AGAINST THE RANDOM DELAY COUNTERMEASURE ON FULL-LENGTH TRACES. EACH MODEL WAS TRAINED THREE TIMES, AND THE TABLE PRESENTS THE MEDIAN OF THE THREE T_{GE1} VALUES.

Dataset	Model	Attack Desync	
		600	1000
ASCADf	PolyCNN	$> 5K$	$> 5K$
	EffCNN	$> 5K$	$> 5K$
	LSTMNet	208	248
	EstraNet	13	7
	HierNet	**2**	**2**
ASCADr	PolyCNN	286	326
	EffCNN	–	–
	LSTMNet	7	6
	EstraNet	–	–
	HierNet	**2**	**3**
CHES20	PolyCNN	$> 5K$	$> 5K$
	EffCNN	–	–
	LSTMNet	4	2
	EstraNet	5	6
	HierNet	**2**	**2**

Meanwhile, the EstraNet model performs close to HierNet on the ASCADf and CHES20 datasets but fails to train properly on ASCADr. We attribute EstraNet's poor performance on ASCADr to its larger trace length – $250K$ features – compared to $100K$ and $62.5K$ features in the other two datasets.

2) Experiments with Clock Jitter: In this section, we present our experimental results with clock jitter.

Generation of Clock Jitter Effect: We employed the methodology outlined in [7], [13], [23] to introduce the clock jitter effect into the traces. Specifically, we generated a noisy trace for each trace of the dataset using the following procedure. For every feature in the original trace, we performed one of the following three actions with equal probabilities: a) excluding the feature from the noisy trace, b) including the feature in the noisy trace, and c) including the feature along with an additional feature, where the additional feature is the average of the current and following features in the source trace. Notably, this method aligns with the application of the clock jitter effect using the algorithm proposed by Wu et al. [23], with the parameter `clock_jitters_level` set to 1. It is pertinent to mention that Hajra et al. [13] also employed the same algorithm for introducing the clock jitter effect. However, prior to applying the clock jitter effect, they doubled the trace length by duplicating each feature in the source trace twice to avoid the loss of any informative features. In contrast, we do not perform such preprocessing. Therefore, our attack setup is more challenging compared to that of [13].

TABLE III
SELECTED ATTACK WINDOWS FOR THE EXPERIMENTS WITH ADDED CLOCK JITTER EFFECT.

	Dataset		
	ASCADf	ASCADr	CHES20
Attack Window	$[40K, 50K]$	$[78K, 88K]$	$[46K, 56K]$

Selection of Attack Window: To reduce the compute cost, we did not conduct experiments on the full-length traces. Instead, the experiments were carried out on attack windows of

TABLE IV
THE T_{GE1} VALUES (LESSER IS BETTER) FOR DIFFERENT DL MODELS ON THE THREE DATASETS WITH ADDED CLOCK JITTER EFFECT. WE HAVE PERFORMED THE ATTACKS ON A SELECTED ATTACK WINDOW OF SIZE $10K$ FOR THESE EXPERIMENTS.

Dataset	Model				
	PolyCNN	EffCNN	LSTM	EstraNet	**HierNet**
ASCADf	$> 5K$	$> 5K$	$> 5K$	$> 5K$	**55**
ASCADr	$> 5K$	$> 5K$	$> 5K$	209	**20**
CHES20	$> 5K$	$> 5K$	$> 5K$	$> 5K$	$> 5K$

size $10K$. It should be noted that, earlier, Hajra et al. [13] also conducted attacks on selected attack windows with a length of $10K$. We adopt the same attack windows for the experiments in this section. The details of these attack windows are outlined in Table III.

Training Details: All benchmark models have been trained as outlined in Appendix C. The HierNet models were trained according to the procedure detailed in Section V-B. As in Sections V-C1, data augmentation was applied to all models during training. Specifically, each training trace was shifted by a random displacement within the range $[0, 200)$ on-the-fly during the training.

Results: The attack results of the DL models are presented in Table IV. As in Section V-C1, the metric used to evaluate model effectiveness is T_{GE1} – the minimum number of attack traces required to reach a guessing entropy of 1. Each DL model was independently trained three times, and the table reports the median T_{GE1} value across these runs.

The CHES20 dataset presents a particularly challenging scenario for all models, as none are able to reach the guessing entropy 1 with fewer than $5K$ traces. On the other two datasets, HierNet performs well, $T_{GE1} = 55$ for ASCADf and $T_{GE1} = 20$ for ASCADr. In contrast, the PolyCNN, EffCNN, and LSTMNet models fail to reduce T_{GE1} below $5K$ on either dataset. EstraNet performs moderately on ASCADr with $T_{GE1} = 209$ – though this is significantly worse than HierNet – and fails to bring T_{GE1} below $5K$ on ASCADf.

Summary: The results in Section V-C1, which evaluates the models under masking countermeasures on long traces, highlight HierNet's strong capability in capturing long-distance dependencies while the benchmark models have been found to struggle in many of those scenarioes. Additionally, the results in Section V-C2 demonstrate HierNet's exceptional robustness against clock jitter countermeasures, a particularly challenging scenario discussed in [13], [23]. We attribute this robustness to HierNet's use of EstraNet as its base model. However, as evidenced by the results in Table IV, HierNet not only inherits EstraNet's strengths but also improves upon them, outperforming EstraNet across the evaluated datasets. We believe this improvement stems from HierNet's divide-and-conquer approach, which enhances its performance on longer traces compared to EstraNet.

D. Evaluation of Robustness to Positional Ambiguity

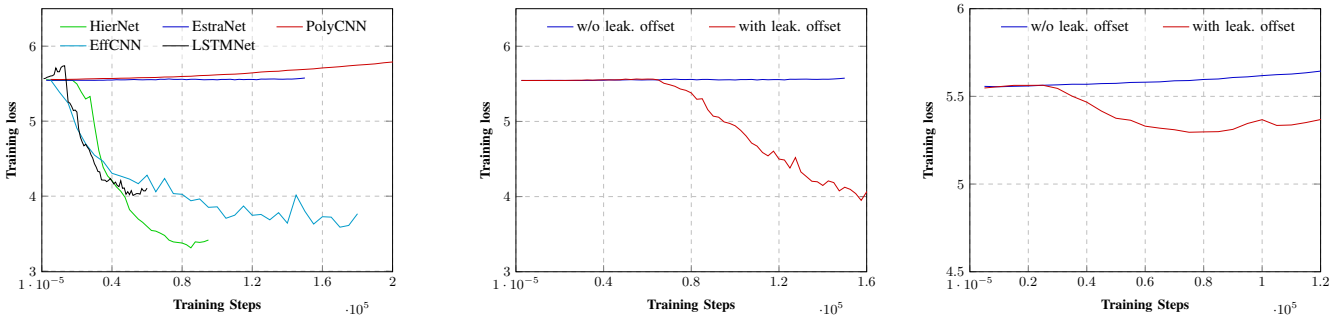
As stated in Section III, identifying the correct PoI for a target operation becomes challenging due to identical trace patterns across parallel operations – particularly for DL models that do not (or weakly) utilize any form of positional information (implicit or explicit). In this section, we evaluate the robustness of HierNet against such positional ambiguity and compare its performance with that of the benchmark models.

a) Dataset Details: We implemented an AES encryption scheme without any countermeasures on an Arduino Uno R3 board, which features an ATmega328P microcontroller. A total of $100K$ traces were collected and randomly split into training, validation, and attack sets in a 90:5:5 ratio. Each trace contains approximately $68K$ features. The peak SNR for the SBox output is around 0.4, which is significantly lower than the SNRs of the ASCADf and ASCADr datasets, where the peak SNR exceeds 8. We select the output of the first SBox as the target variable for the attack.

b) Results: We trained HierNet and the benchmark models on the collected dataset. Their training curves are shown in Figure 3a. From the figure, we observe that the training losses of HierNet, EffCNN, and LSTMNet decrease significantly as training progresses. In contrast, EstraNet and PolyCNN fail to reduce the training loss significantly from the level of random guessing. We attribute this poor performance to their limited ability to exploit positional information of the features. Note that HierNet segments the traces before processing. This segmentation allows the model to infer a unique signature based on the trace pattern of each segment. Additionally, HierNet explicitly introduces absolute positional encoding for each segment. As a result, the training loss of HierNet decreases faster than that of the other models (see Figure 3a). LSTMNet uses an LSTM layer, which implicitly encodes positional information due to its recurrent structure. EffCNN internally uses fully connected layers, which are sensitive to the absolute positions of input features. Consequently, EffCNN also performs well in this scenario.

However, PolyCNN and EstraNet struggle to be trained effectively due to their inability to capture strong positional cues for the target PoI. To further validate this hypothesis, we introduced constant leakage offsets to the trace segments corresponding to the first round's first SubBytes operation (which contains the target PoI). Since the offsets were uniformly added across all traces, they did not affect the SNR of the target variable. However, applying them only to the target SubBytes operation made the corresponding segment distinguishable from others. We then retrained EstraNet and PolyCNN using these modified traces. Figures 3b and 3c compare the training curves of both models with and without the added offsets. From Figure 3b, we observe that EstraNet becomes trainable after the inclusion of the leakage offsets. Similarly, Figure 3c shows an improvement in PolyCNN's training performance, though it still lags behind the other models.

Summary: In summary, by leveraging both implicit and explicit positional information, HierNet effectively overcomes the positional ambiguity problem – a challenge where some benchmark models like EstraNet and PolyCNN struggle. This is a particularly significant achievement because other models that manage to overcome positional ambiguity, such as LSTMNet and EffCNN, perform poorly against the clock jitter countermeasure (as shown Section V-C2) due to their



(a) Training loss vs. training steps for all models. (b) Training loss vs. training steps for EstraNet. (c) Training loss vs. training steps for PolyCNN.

Fig. 3. Evaluation of robustness of different DL models to positional ambiguity. (a) Training curve of all models. (b) Training curve of EstraNet with and without adding leakage offset at the PoI. (c) Training curve of PolyCNN with and without adding leakage offset at the PoI.

weaker shift-invariance. In contrast, HierNet not only achieves robustness to positional ambiguity but also maintains strong shift-invariance.

E. Experimental Results against Shuffling Countermeasure

In this section, we evaluate HierNet in a multi-task setting against the shuffling countermeasure. Specifically, we assess its robustness under the joint application of random delay and shuffling protections. Notably, the two-stage architecture of HierNet enables it to achieve shift-invariance within each segment and permutation-invariance at the segment level. These properties make HierNet well-suited for attacking implementations protected by both random delay and shuffling.

a) Dataset Details: To generate the dataset, we used a software implementation of AES with a shuffling countermeasure on an Arduino Uno R3 board. As before, we collected $100K$ traces and split them into training, validation, and test sets using a 90:5:5 ratio. We used only the traces from the first round of the cipher, which consist of approximately $17K$ features. Similar to previous datasets, this one also exhibits a peak SNR of around 0.4.

b) Benchmark Model: The benchmark models used in the eariler experiments failed against shuffling countermeasures. Therefore, we used a different model, namely MultiResNetSCA-2 [15], as the benchmark model. In [15], Masure et al. have proposed two models, namely MultiResNetSCA-1 and MultiResNetSCA-2, for performing SCA against shuffling countermeasure. Both models are based on a ResNet architecture and incorporate multi-task learning. However, the first model, MultiResNetSCA-1, assumes that the permutations used for shuffling are known during the profiling phase, whereas the second model, MultiResNetSCA-2, assumes they are unknown during both the profiling and attack phases. Since our study considers the second scenario, we compare HierNet with the second model.

c) Hyperparameter and Experimental Setup: In all the experiments in this section, we targeted the outputs of all 16 SBox operations in parallel using multi-task models. Accordingly, we used HierNet with 16 task-specific heads, where each task corresponds to attacking one SBox output. In all the single-task HierNet models used in the previous sections, the model width in the base models was set to 128. However, we found that with the same model width, HierNet becomes untrainable against the shuffling countermeasure. Therefore, we increased the model width to 512. We set the γ hyperparameter of HierNet's top-level module to 0. This disables the use of segment ordering information, making the top-level module permutation-invariant and thereby rendering HierNet suitable for handling the shuffling countermeasure.

d) Segmentation: As stated previously, HierNet internally segments the traces into multiple segments. When applied against shuffling countermeasures, it is important that the PoIs for different SubByte operations belong to different segments. This allows the attention mechanism in HierNet's top-level model – targeting a particular SubByte output – to attend specifically to the segment where that SubByte operation occurs. To ensure that the PoIs for different SubByte operations fall into distinct segments, we used smaller segment lengths in HierNet. Specifically, we tuned the segment length over the set $\{200, 250, 300\}$, and the overlap length between two consecutive segments over the set $\{100, 200\}$. Note that the approximate segment length can be calculated based on the details of the implementation code [15].

e) Training Details: For evaluation against the joint protection of shuffling and random delay, we introduced random displacements into both profiling and attack traces. Additionally, we applied data augmentation during training by randomly displacing each training trace on-the-fly by a value sampled from the range $[0, 5)$. Note that this data augmentation was applied during the training of both models. Each experiment is performed twice, and we report the results of both the runs.

f) Results: As in Section V-C, we report the T_{GE1} – the minimum number of attack traces required to reach guessing entropy 1 – for MultiResNetSCA-2 and HierNet in Table V. From the table, it can be observed that even when there is no random delay (i.e., $desync = 0$), HierNet significantly outperforms MultiResNetSCA-2, achieving a T_{GE1} of approximately 3 compared to more than 23 for MultiResNetSCA-2, representing an 85% reduction in the number of required attack traces. Furthermore, as the magnitude of $desync$ increases, the performance of MultiResNetSCA-2 deteriorates further, while the performance of HierNet remains nearly stable. In particular, for $desync = 200$, HierNet requires more than

TABLE V
THE T_{GE1} VALUES (LOWER IS BETTER) ACHIEVED BY MULTIRESNETSCA-2 AND HIERNET AGAINST JOINT SHUFFLING AND RANDOM DELAY COUNTERMEASURES. EACH EXPERIMENT WAS PERFORMED TWICE, AND THE RESULTS FROM BOTH RUNS ARE REPORTED.

Desync	MultiResNetSCA-2		HierNet	
	run 1	run 2	run 1	run 2
0	23	30	3	3
100	86	132	4	4
200	141	158	5	4

95% fewer traces than MultiResNetSCA-2 to reach guessing entropy 1. Therefore, the results demonstrate superior shift-invariance of HierNet while also being effective against the shuffling countermeasure.

g) Capacity Bottleneck: As stated in Section III, multi-task models generally require larger model width and size to facilitate information flow across a greater number of tasks. However, the hierarchical architecture of the UNIDLE framework–adopted by HierNet–enables distributed representations of segment-level features. This design helps overcome the capacity bottleneck in multi-task learning without significantly increasing the model width or size. Table VI presents the maximum model width and number of parameters for both models. As shown in the table, HierNet can be successfully trained with a smaller model width and significantly fewer parameters (almost one-tenth), contributing to its improved performance.

TABLE VI
COMPARISON OF THE MAXIMUM MODEL WIDTH AND NUMBER OF PARAMETERS FOR THE TWO MULTI-TASK DL MODELS.

Model	Max. Width	# of Params
MultiResNetSCA-2	4048	$79M$
HierNet	512	$7.7M$

F. Training Time

In our experimental environment (Google Colab TPU), the training time of LSTMNet can be more than five times slower for traces with lengths around $10K$ features, while it is about two times slower for longer traces (with lengths in the order of $100K$ features). The training time of the PolyCNN and EstraNet models varies between 3 to 12 times compared to HierNet on shorter traces, but their training becomes much slower on longer traces. In some cases, the training loss of the EffCNN model did not decrease significantly below the level of random guessing, even after training for more than two times the total duration of HierNet (see Appendix D). While an improved learning rate schedule might accelerate their training on longer traces, we still expect them to be significantly slower than HierNet.

While comparing the training time of HierNet with EstraNet, it's worth noting that due to the overlapping segmentation in HierNet, its per-step training time is slightly higher than EstraNet's. Specifically, in our experiments, HierNet's per-step training time was between 1 and 1.5 times that of EstraNet across the datasets. However, HierNet requires fewer training steps to converge (thanks to its divide-and-conquer approach), making the total training time comparable to EstraNet.

VI. FUTURE WORKS

a) Reducing the Number of Parameters in FFN: UNIDLE presents a unified framework for DLSCA against combinations of countermeasures on long traces. Building upon the UNIDLE framework, HierNet specifically targets software implementations of masking combined with various desynchronization-based countermeasures. In the absence of desynchronization-based countermeasures, a Feed-Forward Network (FFN) can be used for DLSCA. However, as the trace length increases, the number of parameters in such FFN models may grow superlinearly with the trace length, making them prone to overfitting and increasing the training cost. Alternatively, the UNIDLE framework, which applies a divide-and-conquer approach, can be explored to reduce the number of parameters and the training cost of FFN-based SCA on long traces as future research.

b) Attacking RDI Countermeasure: The Random Delay Interrupt (RDI) countermeasure inserts random delays at multiple intermediate positions within traces, causing significant variation in the distances between the PoIs across traces. Since HierNet segments the traces and is shift-invariant within each segment, it implicitly assumes that the intermediate distances between PoIs within a segment remain approximately constant. This assumption may not hold in the presence of RDI, making HierNet potentially unsuitable for direct application against RDI countermeasures. Hence, alternative base models – other than EstraNet – that do not rely on the assumption of fixed PoI distances can be explored within the UNIDLE framework to better handle RDI. As another possibility, a HierNet-like model can be combined with various preprocessing techniques aimed at mitigating the effects of RDI. For instance, previously proposed methods such as [23]–[25] can be applied to reduce RDI-induced desynchronization prior to model training. It should be noted, however, that these preprocessing techniques may not completely eliminate the misalignment caused by RDI. The preprocessed traces may still exhibit residual desynchronization, albeit to a lesser extent. Therefore, a model like HierNet – which has demonstrated robustness to bounded desynchronization – could still be effective in such scenarios. Investigating this direction remains an interesting avenue for future research.

VII. CONCLUSIONS

In this work, we introduced UNIDLE, a unified framework for building scalable and robust DL models for SCA. UNIDLE's hierarchical divide-and-conquer strategy enables efficient processing of long traces, offers inherent robustness against combinations of countermeasures, and naturally facilitates multi-task learning. Based on this framework, we developed HierNet, a concrete instantiation that integrates a shift-invariant base model with a transformer-based top-level model, making it effective against combinations of masking, random delay, clock jitter, and shuffling countermeasures.

Empirical evaluations across three datasets of masked implementations demonstrate HierNet's superior performance compared to four state-of-the-art models, especially on long traces or in the presence of clock jitter, demonstrating it's ability to reach the guessing entropy 1 with fewer than 50 attack traces, while other models fail even with up to $5K$ traces. Against shuffling countermeasures, HierNet achieves similar success using over 95% fewer traces than a state-of-the-art multi-task benchmark, while requiring only one-tenth of the model size.

In conclusion, the UNIDLE framework and the HierNet model represent a significant advancement in DLSCA, paving the way for the development of more powerful, scalable, and versatile models for SCA.

REFERENCES

[1] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *CHES, USA, 2002*, ser. LNCS, B. S. K. Jr., Ç. K. Koç, and C. Paar, Eds., vol. 2523. Springer, 2002, pp. 13–28.

[2] W. Schindler, K. Lemke, and C. Paar, "A stochastic model for differential side channel cryptanalysis," in *CHES, UK, 2005*, ser. LNCS, J. R. Rao and B. Sunar, Eds., vol. 3659. Springer, 2005, pp. 30–46.

[3] S. Picek, A. Heuser, A. Jovic, S. Bhasin, and F. Regazzoni, "The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations," *TCHES*, vol. 2019, no. 1, pp. 209–237, 2019.

[4] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *SPACE, India, 2016*, ser. LNCS, vol. 10076. Springer, 2016, pp. 3–26.

[5] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *CRYPTO, USA, 1999*, ser. LNCS, vol. 1666. Springer, 1999, pp. 398–412.

[6] J. Coron and I. Kizhvatov, "An efficient method for random delay generation in embedded software," in *CHES, Switzerland, 2009*, ser. LNCS, vol. 5747. Springer, 2009, pp. 156–170.

[7] E. Cagli, C. Dumas, and E. Prouff, "Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing," in *CHES, Taiwan, 2017*, ser. LNCS, vol. 10529. Springer, 2017, pp. 45–68.

[8] C. Herbst, E. Oswald, and S. Mangard, "An AES smart card implementation resistant to power analysis attacks," in *ACNS, Singapore, 2006*, ser. LNCS, J. Zhou, M. Yung, and F. Bao, Eds., vol. 3989, 2006, pp. 239–252.

[9] L. Masure, N. Belleville, E. Cagli, M. Cornelie, D. Couroussé, C. Dumas, and L. Maingault, "Deep learning side-channel analysis on large-scale traces - A case study on a polymorphic AES," in *ESORICS, UK, 2020*, ser. LNCS, vol. 12308. Springer, 2020, pp. 440–460.

[10] X. Lu, C. Zhang, P. Cao, D. Gu, and H. Lu, "Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks," *TCHES*, vol. 2021, no. 3, pp. 235–274, 2021.

[11] G. Perin, L. Wu, and S. Picek, "Exploring feature selection scenarios for deep learning-based side-channel analysis," *Cryptology ePrint Archive*, 2021.

[12] E. Bursztein, L. Invernizzi, K. Král, D. Moghimi, J. M. Picod, and M. Zhang, "Generic attacks against cryptographic hardware through long-range deep learning," *CoRR*, vol. abs/2306.07249, 2023.

[13] S. Hajra, S. Chowdhury, and D. Mukhopadhyay, "EstraNet: An efficient shift-invariant transformer network for side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 1, p. 336–374, 2023.

[14] E. Bursztein, L. Invernizzi, K. Král, D. Moghimi, J. Picod, and M. Zhang, "Generalized power attacks against crypto hardware using long-range deep learning," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2024, no. 3, pp. 472–499, 2024.

[15] L. Masure and R. Strullu, "Side channel analysis against the ANSSI's protected AES implementation on ARM," *IACR Cryptol. ePrint Arch.*, p. 592, 2021.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS, USA, 2017*, 2017, pp. 5998–6008.

[17] H. Maghrebi, "Deep learning based side-channel attack: a new profiling methodology based on multi-label classification," *IACR Cryptol. ePrint Arch.*, p. 436, 2020.

[18] T. Marquet and E. Oswald, "A comparison of multi-task learning and single-task learning approaches," in *ACNS 2023 Satellite Workshops, ADSC, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, SiMLA, Japan, Proceedings*, ser. LNCS, vol. 13907. Springer, 2023, pp. 121–138.

[19] ——, "Exploring multi-task learning in the context of masked AES implementations," in *COSADE 2024, France, Proceedings*, ser. LNCS, R. Wacquez and N. Homma, Eds., vol. 14595. Springer, 2024, pp. 93–112.

[20] L. Zhang, X. Xing, J. Fan, Z. Wang, and S. Wang, "Multi-label deep learning based side channel attack," in *AsianHOST, China, 2019*. IEEE, 2019, pp. 1–6.

[21] S. Hajra, S. Saha, M. Alam, and D. Mukhopadhyay, "TransNet: Shift invariant transformer network for side channel analysis," in *AFRICACRYPT 2022, Fes, Morocco, 2022, Proceedings*, ser. LNCS, L. Batina and J. Daemen, Eds. Springer Nature Switzerland, 2022, pp. 371–396.

[22] G. Zaid, L. Bossuet, A. Habrard, and A. Venelli, "Methodology for efficient CNN architectures in profiling attacks," *TCHES*, vol. 2020, no. 1, pp. 1–36, 2020.

[23] L. Wu and S. Picek, "Remove some noise: On pre-processing of side-channel measurements with autoencoders," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 4, pp. 389–415, 2020.

[24] F. Durvaux, M. Renauld, F. Standaert, L. van Oldeneel tot Oldenzeel, and N. Veyrat-Charvillon, "Efficient removal of random delays from embedded software implementations using hidden markov models," in *CARDIS, Austria*, ser. Lecture Notes in Computer Science, S. Mangard, Ed., vol. 7771. Springer, 2012, pp. 123–140.

[25] G. Chiari, D. Galli, F. Lattari, M. Matteucci, and D. Zoni, "A deep-learning technique to locate cryptographic operations in side-channel traces," 2024.

[26] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas, "Deep learning for side-channel analysis and introduction to ASCAD database," *J. Cryptogr. Eng.*, vol. 10, no. 2, pp. 163–188, 2020.

[27] D. Bellizia, F. Berti, O. Bronchain, G. Cassiers, S. Duval, C. Guo, G. Leander, G. Leurent, I. Levi, C. Momin, O. Pereira, T. Peters, F. Standaert, B. Udvarhelyi, and F. Wiemer, "Spook: Sponge-based leakage-resistant authenticated encryption with a masked tweakable block cipher," *IACR Trans. Symmetric Cryptol.*, vol. 2020, no. S1, pp. 295–349, 2020.

APPENDIX A
TRANSFORMER NETWORK

A Transformer Network (TN) is a kind of DL model composed of multiple transformer layers. The input trace is fed into the first transformer layer, and the output of each layer serves as the input for the subsequent layer. The final output of the last transformer layer is considered the output of the TN model.

Each transformer layer consists of a self-attention layer and a position-wise feed-forward layer. It takes a sequence $\mathrm{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}]^T \in \mathbb{R}^{n \times d}$ of n feature vectors as input and outputs another sequence $\mathrm{Y} = [\mathbf{y}_0, \ldots, \mathbf{y}_{n-1}]^T \in \mathbb{R}^{n \times d}$ where n is the sequence length (trace length in the context of SCA) and d is the dimension of the feature vectors. The output Y is given by

$$\hat{\mathrm{Y}} = f_{SA}(\mathrm{X}) + \mathrm{X}$$
$$\mathrm{Y} = f_{PFF}(\hat{\mathrm{Y}}) + \hat{\mathrm{Y}}$$

where $f_{SA} : \mathbb{R}^{n \times d} \mapsto \mathbb{R}^{n \times d}$ is the function representing the self-attention layer and $f_{PFF} : \mathbb{R}^{n \times d} \mapsto \mathbb{R}^{n \times d}$, is the function representing the position-wise feed-forward layer. The position-wise feed-forward layer independently applies a non-linear transformation to each feature vector, increasing the non-linearity (and, thus, its representation power) of the TN model. On the other hand, the self-attention layer captures the inter-dependencies among different features [12], [13],

[21]. It is worth mentioning that the number of parameters in both the self-attention and position-wise feed-forward layer is independent of the sequence length n. Therefore, it can scale to a very long sequence without being prone to overfitting.

The self-attention layer is a key component of a Transformer Network (TN). Different variations of this layer lead to vastly different types of TN models, each varying significantly in performance and computational cost. In the next part of this section, we briefly describe the self-attention layer and some of its variations, setting the groundwork for the following discussion.

1) Self-attention Layer: Let the sequence $\mathrm{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}]^T \in \mathbb{R}^{n \times d}$ of n feature vectors is the input of a self-attention layer and $\hat{\mathrm{Y}} = [\hat{\mathbf{y}}_0, \ldots, \hat{\mathbf{y}}_{n-1}]^T \in \mathbb{R}^{n \times d}$ be the corresponding output sequence. Then, in a self-attention layer, each output feature vector $\hat{\mathbf{y}}_i$, for $i = 0, \ldots, n-1$, is computed as

$$\hat{\mathbf{y}}_i = \sum_{j=0}^{n-1} \frac{k(f(\mathbf{x}_i), g(\mathbf{x}_j))}{\sum_{l=0}^{n-1} k(f(\mathbf{x}_i), g(\mathbf{x}_l))} \phi(\mathbf{x}_j) \tag{4}$$

where $f(\mathbf{x}_i), g(\mathbf{x}_i),$ and $\phi(\mathbf{x}_i)$ are some vector valued functions of input feature $\mathbf{x}_i$, and $k : \mathbb{R}^{d_k} \times \mathbb{R}^{d_k} \mapsto \mathbb{R}^+$ is called the kernel function. The kernel function outputs a high score, $k(f(\mathbf{x}_i), g(\mathbf{x}_j))$, if it finds the features $\mathbf{x}_i,$ and $\mathbf{x}_j$ to be related and a low score otherwise. In summary, in self-attention, each output feature, $\hat{\mathbf{y}}_i$, is computed by taking the weighted sum of the input features $\phi(\mathbf{x}_0), \ldots, \phi(\mathbf{x}_{n-1})$ where the weight for the j-th input feature is given by the normalized score $k(f(\mathbf{x}_i), g(\mathbf{x}_j))/\sum_{l=0}^{n-1} k(f(\mathbf{x}_i), g(\mathbf{x}_l))$. Therefore, if the i-th feature is related to (or dependent on) the j-th feature, the weight of the j-th input feature in the i-th output feature can be significantly greater than weights of the other features, increasing the component of the j-th feature (in the input) in the i-th feature (in the output). In that way, self-attention layer can combine the information of the i and j-th features.

One of the most common kernel used in TN literature is obtained by setting the kernel as $k(f(\mathbf{x}_i), g(\mathbf{x}_j)) = \exp(\langle f(\mathbf{x}_i), g(\mathbf{x}_j)\rangle/\sqrt{d_k})$, where $\langle \cdot, \cdot \rangle$ represents the dot product between two vectors and d_k is the dimension of the outputs of $f(\cdot)$ and $g(\cdot)$. The above kernel leads to following self-attention known as softmax self-attention:

$$\hat{\mathbf{y}}_i = \sum_{j=0}^{n-1} \frac{\exp\left(\langle f(\mathbf{x}_i), g(\mathbf{x}_j)\rangle/\sqrt{d_k}\right)}{\sum_{l=0}^{n-1} \exp\left(\langle f(\mathbf{x}_i), g(\mathbf{x}_l)\rangle/\sqrt{d_k}\right)} \phi(\mathbf{x}_j) \tag{5}$$

$$= \sum_{j=0}^{n-1} \mathrm{softmax}\left(\langle f(\mathbf{x}_i), g(\mathbf{x}_j)\rangle/\sqrt{d_k}\right)_j \phi(\mathbf{x}_j) \tag{6}$$

In the self-attention mechanism of the form of Eq. (4), the attention score from feature $\mathbf{x}_i$ to feature $\mathbf{x}_j$ does not depend on their positions, i or j, in the input sequence. Therefore, the actual order of the features in the input sequence does not affect the attention scores. In other words, permuting the input sequence also permutes the output sequence in the same way.

In many tasks, we want the attention a feature gives to another feature to depend on their absolute positions or the distance between them. For instance, in the presence of random delay or clock jitter countermeasures, the distances between the informative features remain almost the same across all traces. These properties are exploited in the TN models proposed in [13], [21] by making the self-attention shift-invariant. This approach is described as follows:

Relative Positional Encoding: To make the self-attention shift-invariant, one can incorporate relative positional encoding into kernel function by generalizing it to $k_r(f(\mathbf{x}_i), g(\mathbf{x}_j), e(i-j))$, where $e(i-j)$ is a vector space encoding of the relative distance $i - j$. Therefore, the output of the resultant self-attention takes the form

$$\hat{\mathbf{y}}_i = \sum_{j=0}^{n-1} \frac{k_r(f(\mathbf{x}_i), g(\mathbf{x}_j), e(i-j))}{\sum_{l=0}^{n-1} k_r(f(\mathbf{x}_i), g(\mathbf{x}_l), e(i-l))} \phi(\mathbf{x}_j) \tag{7}$$

With relative positional encoding, the kernel function returns a high score not only based on the similarity between $\mathbf{x}_i,$ and $\mathbf{x}_j$, but also considering their relative distance $i - j$.

Another scenario where incorporating position information into the attention scores is beneficial is when there is no jitter-based countermeasure in the implementation. In the absence of such countermeasures, the positions of the informative features in the input sequence remain consistent across all traces. To exploit this property, one approach is to make the attention scores dependent on the absolute positions of the features. In this work, we achieve this by incorporating absolute positional encoding into the self-attention layer. We introduce the self-attention with absolute positional encoding in Section IV-A.

APPENDIX B
DETAILS OF DATASET

This section provides a detailed description of the datasets used.

ASCAD Fixed Key (ASCADf): The ASCAD fixed key dataset, referred to as the ASCADf dataset, is a collection of EM traces collected from a masked implementation of AES executed on the 8-bit microcontroller ATMega8515. This dataset comprises a total of $60K$ traces. Out of these, $50K$ traces were used for training, while two sets of $5K$ traces each were used for validation and testing. Each trace in the dataset is $100K$ features long. In line with earlier literature [10], [13], [22], [26], we aim to attack the third key byte of the first-round key and use the identity leakage model.

ASCAD Random Key (ASCADr): Similar to the ASCADf dataset, the ASCAD random key dataset, referred to as the ASCADr dataset, also corresponds to a masked implementation of AES executed on an 8-bit ATMega8515 microcontroller. The dataset consists of $200K$ profiling traces and $100K$ attack traces. Unless stated otherwise, all profiling traces have been utilized for training the DL models, while two disjoint random samples of $10K$ attack traces were used for validation and testing, respectively. Each trace in the dataset spans $250K$ features. To reduce memory usage, the traces were compressed to a length of $125K$ by replacing each consecutive pair of features of the original traces with their average. Therefore, the attacks were conducted on the compressed traces of length $125K$. Similar to the ASCADf dataset, we attack the third key byte of the first round key using identity leakage model.

CHES 2020 CTF SW3 (CHES20): The CHES 2020 CTF presents a SCA challenge targeting the masked implementation of the Clyde-128 Tweakable Block Cipher (TBC) used in the

Spook AEAD encryption scheme [27]. The challenge encompasses various datasets derived from distinct implementations of the cipher. In this study, we utilize the dataset corresponding to a second-order masked implementation executed on an ARM Cortex-M0 microcontroller. The dataset comprises $200K$ profiling traces and $500K$ attack traces. Unless stated otherwise, all profiling traces are employed for training the DL models, while for validation and testing, two separate and randomly selected sets of attack traces, each containing $10K$ traces, are utilized. It should be noted that Clyde-128 employs a (4×32)-bit state, with 4 representing the size of the non-linear S-box and 32 indicating the size of the linear L-box. Consistent with the approach in [13], we attack the four bits of the 17th column (from the right) of the state after the first round sbox operation.

APPENDIX C
DETAILS OF THE BENCHMARK MODELS

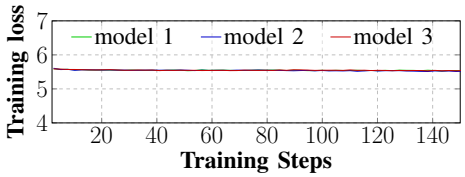
This section provides short descriptions of the benchmark DL models.

LSTMNet: LSTMNet [10] is a family of DL models that incorporate LSTM layers to aggregate information from multiple Points of Interest (PoIs). These models exhibit linear scalability with respect to input length, making those well-suited for processing very long traces. In their work [10], Lu et al. demonstrated LSTMNet's proficiency in attacking available full-length raw traces. For the ASCADf and ASCADr datasets, we trained the respective models available in their online repository[4] to conduct the attack. Since no model is available for the CHES20 dataset, following [13], we used the model designed for the ASCADr dataset to train on this dataset. The models were trained for a maximum of $4K$ epochs, employing Adam optimizer with a learning rate of 1e-4.

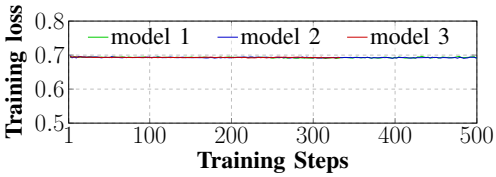
PolyCNN: The PolyCNN model [9], introduced by Masure et al. [9], is designed for attacking implementations protected by code polymorphism. The model is comprised of multiple convolutional blocks succeeded by a global average-pooling layer. [9] has demonstrated that the model can perform very well on very long traces. Consequently, we employ PolyCNN as a benchmark model in our study. The model underwent training for a maximum of $5K$ epochs, utilizing the Adam optimizer with a learning rate of 1e-5.

EffCNN: In their work [22], Zaid et al. proposed a methodology for constructing CNN-based models for SCA. They successfully demonstrated the applicability of their methodology in creating compact CNN models suitable for both synchronized and desynchronized trace scenarios. However, it is worth noting that when applying their methodology to very long traces with large trace desynchronizations, the resultant models become large and exhibit slow training speeds. Despite these considerations, we utilize models constructed through their methodology as benchmark models in our study. It is important to emphasize that distinct models have been constructed based on their methodology for different datasets. The training process involved a maximum of $2K$ epochs, utilizing the Adam optimizer with a learning rate of 2.5e-5.

[4]https://github.com/lxj-sjtu/TCHES2021_Pay_attention_to_the_raw _traces



(a) The ASCADr Dataset.



(b) The CHES20 Dataset.

Fig. 4. Training loss of three independent training of EffCNN model on the ASCADr and CHES20 datasets.

a) EstraNet: EstraNet is a Transformer-based, shift-invariant model introduced in [13]. It employs a novel self-attention mechanism called GaussiP attention, which incorporates relative positional encoding to achieve shift invariance. Hajra et al. experimentally demonstrated that EstraNet performs well against various combinations of masking, random delay, and clock jitter countermeasures, on traces of length up to 40K. In our experiments, we use the same hyper-parameters as described in [13] to train EstraNet.

APPENDIX D
PLOTS OF TRAINING LOSS OF EFFCNN MODEL ON THE ASCADR AND CHES20 DATASETS

In Figure 4, we illustrate the EffCNN model's training loss convergence in the experiments with the ASCADr and CHES20 datasets. Figure 4a displays the training loss for the ASCADr dataset, indicating negligible improvement even after 150 epochs. Notably, training for 150 epochs on this dataset took approximately 50 hours, whereas HierNet models achieved full convergence within 24 to 40 hours. Similarly, Figure 4b shows that even after 500 epochs (equivalent to 80 hours of training), the training loss did not significantly decrease compared to random guessing.