

THE UNIVERSITY OF CHICAGO

PRIVACY, ERROR, AND ENERGY TRADEOFFS IN EMBEDDED SYSTEMS AND
INTERNET-OF-THINGS DEVICES

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
TEJAS KANNAN

CHICAGO, ILLINOIS
DECEMBER 2023

Copyright © 2023 by Tejas Kannan

All Rights Reserved

CONTENTS

LIST OF FIGURES	vii
LIST OF TABLES	ix
ACKNOWLEDGMENTS	xi
ABSTRACT	xii
1 INTRODUCTION	1
2 BUDGET RNNs	6
2.1 Introduction	6
2.2 Background	9
2.2.1 Examples of Inference under Energy Budgets	9
2.2.2 Recurrent Neural Networks and Sequential Classification	10
2.2.3 Inference under an Energy Budget	11
2.2.4 Adaptive Sampling in Recurrent Neural Networks	13
2.3 Budget RNN Model Architecture	14
2.3.1 Leveled Architecture	15
2.3.2 Merging Memory States Across Subsequences	17
2.3.3 Halting Signals	19
2.3.4 Loss Function	21
2.4 Control Policy and Subsampling Algorithm	21
2.4.1 Adaptive Inference Algorithm	22
2.4.2 Optimizing Halting Thresholds	23
2.4.3 Runtime Controller	26
2.4.4 Budget RNN Selection	29
2.5 Evaluation	29
2.5.1 Experimental Setup	30
2.5.2 Inference Accuracy	33
2.5.3 Energy Comparison	36
2.5.4 Accuracy on Unseen Energy Profiles	36
2.5.5 Evaluation of Budget RNN System Design	37
2.5.6 Training Time	39
2.5.7 Performance in the Hardware Environment	40
2.5.8 Overhead Analysis	40
2.6 Related Work	41
2.6.1 Adapting Neural Network Inference	41
2.6.2 Adaptive Sampling in RNNs	42
2.6.3 Adaptive Sampling in Sensor Networks	43
2.6.4 Early Time Series Classification	43
2.7 Conclusion	43

3	PROTECTING ADAPTIVE SAMPLING FROM INFORMATION LEAKAGE ON LOW-POWER SENSORS	45
3.1	Introduction	45
3.2	Background	49
3.2.1	Low-Power Sensors and System Model	49
3.2.2	Subsampling and Adaptive Behavior	50
3.3	Privacy Threats and Adaptive Sampling	51
3.3.1	Threat Model	52
3.3.2	Privacy and Adaptive Sampling	54
3.3.3	Example Systems	55
3.4	Adaptive Group Encoding	56
3.4.1	Notation	58
3.4.2	Measurement Pruning	59
3.4.3	Exponent-Aware Group Formation	60
3.4.4	Data Quantization	61
3.4.5	Discussion	62
3.5	Evaluation	63
3.5.1	Setup	64
3.5.2	Reconstruction Error	66
3.5.3	Information Leakage: Theoretical	68
3.5.4	Information Leakage: Practical	70
3.5.5	Evaluation on Skip RNNs	72
3.5.6	Variants of Adaptive Group Encoding	73
3.5.7	Performance on a Microcontroller	74
3.5.8	Overhead Analysis	75
3.6	Related Work	76
3.7	Discussion and Conclusion	78
4	PREDICTION PRIVACY IN DISTRIBUTED MULTI-EXIT NEURAL NETWORKS: VULNERABILITIES AND SOLUTIONS	81
4.1	Introduction	81
4.2	Background and Motivation	85
4.2.1	Multi-Exit Neural Networks (MeNNs)	86
4.2.2	Early Exit Policies	86
4.2.3	Example of Information Leakage	88
4.2.4	Goals of Private Exit Policies	89
4.3	Threat Model	90
4.3.1	Target System and Attack Goal	90
4.3.2	Adversary Capabilities	91
4.3.3	Example Settings	94
4.4	Per-Class Exiting (PCE)	95
4.4.1	Policy Design	95
4.4.2	Adversarial Data Orderings	96

4.5	Confidence-Guided Randomness (CGR)	98
4.5.1	Confidence-Biased Randomization	98
4.5.2	Adapting the Probability Bias Magnitude	99
4.5.3	Short-Term Exit Quotas	100
4.5.4	Theoretical Benefits	101
4.6	Evaluation	103
4.6.1	Experimental Setup	104
4.6.2	White Box Attack	107
4.6.3	Theoretical Information Leakage	109
4.6.4	Inference Accuracy	110
4.6.5	Alternate Dataset Orders	112
4.6.6	Distribution Shifts	114
4.6.7	Black Box Attack	115
4.6.8	White Box Attack on Low-Power MCUs	117
4.6.9	Energy Consumption	118
4.6.10	Beyond Two Exits	120
4.6.11	Data Order Parameters	121
4.6.12	CGR Parameters	121
4.7	Related Work	123
4.8	Conclusion	126
5	ACOUSTIC KEYSTROKE LEAKAGE ON SMART TELEVISIONS	127
5.1	Introduction	127
5.2	Background	132
5.2.1	Smart TVs and Virtual Keyboards	132
5.2.2	Notation	135
5.2.3	Credit Card Details	136
5.2.4	Example of Acoustic Keystroke Leakage	137
5.3	Threat Model	138
5.4	Acoustic Attack Framework	140
5.4.1	Audio Extraction	141
5.4.2	String Recovery	145
5.4.3	User Timing Patterns	150
5.5	Attack Results in Emulation	153
5.5.1	Setup	153
5.5.2	Credit Card Recovery	155
5.5.3	Password Recovery	156
5.5.4	Impact of the Keyboard Layout	158
5.6	Attack Results on Users	159
5.6.1	Setup	159
5.6.2	Credit Card Recovery	160
5.6.3	Password Recovery	162
5.6.4	Web Search Recovery	164

5.6.5	Impact of Keystroke Timing	165
5.7	Discussion	166
5.8	Related Work	169
5.8.1	Keystroke Side-Channel Attacks	169
5.8.2	Security of Smart Devices	170
5.8.3	Attacks on Passwords and Payment Details	170
5.9	Conclusion	171
6	CONCLUSION	173
6.1	Data Compression in Distributed Neural Networks	174
6.2	Audio and Infrared Signals for Smart TVs	175
	BIBLIOGRAPHY	176

LIST OF FIGURES

2.1	An RNN on a sequence with three elements. S_{θ} represents the RNN cell and g_{ψ} is the readout layer.	12
2.2	A Budget RNN with sequence length $T = 6$, $L = 2$ levels and stride length $K = 1$. 15	15
2.3	A Budget RNN with sequence length $T = 6$, $L = 2$ levels and stride length $K = 2$. 18	18
2.4	The accuracy for two distinct Budget RNNs on the task of classifying Whale sounds [Cox et al., 2006].	28
2.5	System accuracy on the Pen Digits dataset using the Bluetooth energy profile. .	34
2.6	Geometric mean of normalized budget to obtain accuracy equal to the Budget RNN. Values above one indicate better performance by the Budget RNN.	35
3.1	An example of subsampling two accelerometer signals on the task of human activity recognition.	52
3.2	Diagram of the sensor system and threat model.	54
3.3	AGE works in between sampling and encryption, and it requires no changes to either step.	56
3.4	Overview of the data flow through AGE.	57
3.5	MAE for each budget on the Activity dataset.	67
3.6	Median attacker accuracy across all energy budgets. The error bars denote the first and third quartiles, and the points show the maximum accuracy.	71
3.7	Confusion matrices for the attack model under a single budget. The Linear policy yields 100% accuracy on detecting seizures. AGE forces all predictions into one event.	71
4.1	A distributed Multi-exit Neural Network (MeNN) [Teerapittayanon et al., 2016] with two total exits.	82
4.2	The threat model against distributed MeNNs. An exit decision of 0 means stopping on the sensor.	91
4.3	An example of the black box attacker.	93
4.4	CGR’s decision process at the k^{th} exit point.	99
4.5	Maximum attack accuracy across 21 exit rates (lower is better).	108
4.6	Inference accuracy (%) on the Activity dataset. Error bars show the standard deviation over five trials.	111
4.7	Maximum attack accuracy on Nearest-Neighbor dataset orders (lower is better). 113	113
4.8	Early exit rates per prediction under the Max Prob policy for the target MeNN trained on MNIST [LeCun et al., 1998] and the substitute trained on spoken digits [Jackson, 2022].	117
4.9	Inference accuracy and attack accuracy against distributed MeNNs executing on a low-power MCU.	118
4.10	Maximum attack accuracy for MeNNs with two, three, and four exits (lower is better).	120
4.11	Average inference accuracy (left) and maximum attack accuracy (right) on the Activity dataset for window sizes (B) and noise rates (ϵ) under the Same-Label order.	122

4.12	Mean inference accuracy (left) and worst-case attack accuracy (right) for different maximum bias (α) values aggregated across $\beta \in \{1.5, 2.0, 2.5\}$ and $\gamma = \{0.5, 0.7, 0.9, 0.99\}$	123
4.13	Mean inference accuracy (left) and worst-case attack accuracy (right) for different increase factors (β) aggregated across $\alpha \in \{0.4, 0.5, \dots, 0.9\}$ and $\gamma = \{0.5, 0.7, 0.9, 0.99\}$	124
4.14	Mean inference accuracy (left) and worst-case attack accuracy (right) for different decrease factors (γ) aggregated across $\alpha \in \{0.4, 0.5, \dots, 0.9\}$ and $\beta = \{1.5, 2.0, 2.5\}$	124
5.1	Remote controllers for AppleTVs (left) and Samsung Smart TVs (right) annotated with the direction pad.	129
5.2	The default AppleTV keyboard for passwords. The background shows the key's sound when pressed. The only exception is Done , which makes the sound when scrolling onto the key; selecting Done makes no sound.	132
5.3	The default keyboard on the Samsung Smart TV. The background denotes the key's sound when pressed.	132
5.4	Example of dynamic key suggestions on the Samsung Smart TV upon selecting the character d	135
5.5	Example audio from the Samsung TV when typing the string test	138
5.6	The keystroke acoustic attack framework.	141
5.7	Optimal and suboptimal paths between w and t	151
5.8	Accuracy on CCNs (left) and full credit card details (right) in emulation on the Samsung Smart TV.	155
5.9	Password recovery in emulation. The data labels show the Top- K accuracy for $K \in \{1, 10, 100, 250\}$	156
5.10	The ABC keyboard layout. The background denotes the key's sound when pressed.	158
5.11	Accuracy for credit card numbers (left) and full credit card details (right) on the Samsung Smart TV.	161
5.12	Password recovery accuracy for human users. The data labels show the Top- K accuracy for $K \in \{1, 10, 100, 250\}$	162
5.13	Accuracy for human users typing web searches on the Samsung Smart TV with dynamic suggestions.	164
5.14	Comparison between timing-based and exhaustive identification of suboptimal paths.	165

LIST OF TABLES

2.1	Evaluation dataset characteristics.	31
2.2	Geometric mean accuracy across all budgets. RNN refers to the standard RNN.	33
2.3	Geometric mean accuracy across all datasets and budgets in settings with a nonzero energy bias.	36
2.4	Geometric mean accuracy of Budget RNN variants. The adaptive results correspond to the full Budget RNN system.	37
2.5	Wall-to-wall time and training iterations required to fit the neural networks on each dataset. The Phased and Skip RNN results account for training ten distinct RNNs. The Budget RNN values include the halting threshold optimization. . . .	38
2.6	Skip and Budget RNN results on the embedded device using the Pen Digits dataset.	39
2.7	Energy (mJ) required to handle a single sensor measurement. The standard and Budget RNN costs include the energy required for processing.	41
3.1	Average (standard deviation) message size of adaptive policies [Chatterjea and Havinga, 2008, Silva et al., 2017, Campos et al., 2017] when conditioned on the event.	54
3.2	A selection of relevant notation.	58
3.3	Properties of the evaluation datasets.	65
3.4	Arithmetic mean reconstruction error (MAE) across all budgets. The asterisk denotes the lowest error overall, and the boldface marks the lowest error for policies without information leakage. The final row is the median percent error higher than Uniform sampling (lower is better).	66
3.5	Arithmetic mean weighted reconstruction error across all budgets. The asterisk marks the lowest overall error, and the boldface shows the lowest error amongst policies with no leakage.	69
3.6	Median / maximum empirical normalized mutual information between message size and event label. Padded and AGE have the same median and maximum values.	70
3.7	Average MAE, maximum NMI, and maximum attack accuracy when sampling using Skip RNNs.	72
3.8	Median percent error greater than AGE across all budgets and tasks. Higher values indicate higher error.	73
3.9	Average energy per sequence (mJ) over 75 sequences when on a TI MSP430 MCU under three energy budgets.	74
3.10	MAE over 75 sequences on a TI MSP430 MCU under three energy budgets. . .	74
4.1	Dataset properties.	104
4.2	Maximum empirical normalized mutual information (NMI) (all values $\times 10^{-2}$) between exit decisions and MeNN predictions across 21 target rates (lower is better). The final row shows the average (std dev) difference compared to Random.	110
4.3	Average (std dev) inference accuracy across 21 target exit rates for each policy and task (higher is better). The standard deviation shows the variation in the average accuracy across five independent trials.	111

4.4	Average (std dev) inference accuracy across 21 exit rates for Nearest-Neighbor blocks (higher is better).	113
4.5	Mean inference accuracy (Infr. Acc.) and maximum attack accuracy (Att. Acc.) on the Activity task for Uncorrelated and Nearest-Neighbor orders.	114
4.6	Worst-case attack accuracy for exit policies on an MeNN trained on MNIST and tested on either a shifted distribution (EMNIST Digits) or the same distribution (MNIST).	115
4.7	Worst-case attack accuracy (%) averaged (std dev) across trained MeNNs. In each group, the top row is the white box setting. The remaining rows use the black box method.	118
4.8	Average (std dev) energy consumption (mJ) on a TI MSP430 MCU [Instruments, 2021].	119
5.1	Emulated top- K password accuracy under the PhpBB prior for different keyboard layouts.	157

ACKNOWLEDGMENTS

I want to thank my advisor, Hank Hoffmann, for his contributions to every piece of work of this thesis. I would also like to acknowledge my thesis committee members, Nick Feamster and Sanjay Krishnan, who have both provided valuable insights for past, present, and ongoing projects. I would like to thank my friends and family for their unwavering love and support. Nobody ever accomplishes anything alone, and I am incredibly fortunate to have such an amazing support system.

This work was supported by the National Science Foundation (grants CCF-2028427, CNS-1956180, CCF-1837120, CNS-1764039, CCF-2119184, CNS-1952050, CCF-1823032, CCF-1822949, and CISE-ANR-2124393), the Army Research Office (grant W911NF1920321), and a Department of Energy Early Career Award (grant DESC0014195 0003).

ABSTRACT

Battery-powered embedded devices operate under energy constraints, and devices seek methods to manage their energy consumption. Adaptive algorithms are an emerging method to perform this management. Adaptive systems meet device constraints by leveraging data-dependent behavior to optimize the tradeoff between system quality (i.e., error) and energy. This data-dependent behavior comes from using the collected measurements to determine when to conserve resources. We demonstrate this design through a new adaptive system that performs recurrent neural network inference on embedded devices under energy constraints.

This thesis argues that our analysis of embedded systems must go beyond this two-dimensional tradeoff space and explicitly consider data privacy as a third dimension because data-dependent behavior can leak crucial information about the captured measurements. We display this problem by presenting two new side-channel attacks and defenses. The first attack uses the communication volume of embedded devices employing adaptive sampling to learn about the captured data. The second attack exploits the exit decisions of adaptive, multi-exit neural networks to expose the model’s results. In both settings, we develop defenses that eliminate information leakage and incur negligible overhead while achieving higher system quality (i.e., error) than non-adaptive algorithms. These properties make our security measures suitable under the resource constraints of embedded devices.

These privacy issues extend beyond embedded systems and into the broader class of Internet-of-Things devices. We demonstrate this phenomenon by developing a new attack against Smart Televisions (TVs). Users enter information into Smart TVs through on-screen virtual keyboards, and popular Smart TV platforms, such as Apple’s tvOS and Samsung’s Tizen, make sounds as users type. We find that an attacker can use this audio as a side-channel to extract sensitive keystrokes (e.g., credit card details and common passwords) from Smart TVs. Samsung has acknowledged this vulnerability, and this attack highlights how modern Internet-connected devices can leak sensitive information in unintended ways.

CHAPTER 1

INTRODUCTION

Battery-powered embedded sensors are common components in applications for areas such as agriculture [Vasisht et al., 2017], healthcare [Sotirakis et al., 2023], and smart homes [Ho et al., 2016]. Sensor devices collect measurements from their environment, process these values, and communicate results to a server. For reliable performance, devices must meet energy constraints [Gobieski et al., 2019, Juang et al., 2002], and sensors seek ways to reduce their energy consumption. Further, sensors often capture sensitive data such as personal health information [Hiremath et al., 2014, Sotirakis et al., 2023]. Devices must therefore ensure data privacy.

Adaptive behavior is an emerging method to manage the energy consumption of embedded sensing devices [Dennis et al., 2019, Laskaridis et al., 2020, Teerapittayanon et al., 2017]. Adaptive algorithms control the energy spent collecting, processing, and communicating each measurement. Reducing energy consumption, however, comes at a tradeoff in system quality. Adaptive systems use runtime feedback to optimize this tradeoff between system cost (e.g., energy) and answer quality (e.g., error) [Farrell and Hoffmann, 2016, Hoffmann, 2014, Wan et al., 2020b]. Namely, these systems determine how much energy to spend on a single input using the input’s “difficulty.” The method to measure input difficulty is a design property of the particular adaptive system. With this data-dependent method, adaptive algorithms can conserve energy with minimal impact on the system’s answer quality.

We highlight the benefits of adaptive behavior on low-power devices through a novel inference system called Budget RNNs (Chapter 2). Budget RNNs perform recurrent neural network (RNN) inference under long-term energy budgets only known at runtime [Kannan and Hoffmann, 2021]. This inference system uses a new RNN architecture that hierarchically processes subsequences of measurements. The Budget RNN model produces a prediction for each subsequence, allowing the system to save energy by terminating inference early.

This early stopping results in collecting and processing only a subset of inputs. Budget RNNs determine the subsequence granularity using adaptive behavior. The model produces halting signals based on the current inputs; these signals encode the inputs’ difficulty. The system then sets budget-specific thresholds on these halting signals to determine when to stop inference. The inference system adapts to new budgets and dynamic conditions by altering the thresholds with a runtime controller. Across multiple budgets and tasks, Budget RNNs achieve a mean accuracy of 1.5 points higher than prior state-of-the-art RNNs [Campos et al., 2017, Dennis et al., 2018, Neil et al., 2016] that support subsampling.

This thesis argues that the two-dimensional tradeoff space considered by existing adaptive systems is insufficient. Instead, adaptive systems must explicitly consider data privacy as a third tradeoff axis. This consideration is necessary because adaptive systems make data-dependent decisions, and these decisions are observable through side-channels (e.g., wireless communication patterns). Thus, an attacker observing the adaptive system’s decisions can learn about the underlying measurements. Indeed, Budget RNNs are an example of this phenomenon, as they determine the subsequence granularity based on already-collected inputs. The Budget RNN system’s energy consumption, therefore, provides information about the collected measurements. This phenomenon prevents privacy-conscious sensing applications, such as those in healthcare [Sotirakis et al., 2023], from gaining the benefits of adaptive behavior. It is difficult to develop defenses for these side-channel attacks for two reasons:

1. The defense must not significantly increase the application’s execution cost on low-power sensing devices, as these devices do not have excess resources.
2. The security solution must deliver better answer quality (i.e., error) than non-adaptive baselines, which do not suffer from this information leakage. Otherwise, the sensing device would rather use non-adaptive techniques.

We display how adaptive systems can suffer from privacy issues on embedded sensing devices by creating two new side-channel attacks and defenses. The first system employs

general-purpose adaptive sampling [Chatterjea and Havinga, 2008, Silva et al., 2017] to reduce the energy consumption of collecting and transmitting raw values (Chapter 3). As is standard on low-power devices [Juang et al., 2002], the device sends batches of measurements at regular intervals. We discover that an attacker who observes the adaptive sampling policy’s collection rate can expose the system’s sensed events [Kannan and Hoffmann, 2022]. This collection rate gets leaked through the size of transmitted batches. This privacy issue occurs across multiple tasks and adaptive sampling policies, highlighting how the problem stems from the data-dependent design of adaptive techniques. We close this side-channel through a new defense called Adaptive Group Encoding (AGE). AGE works as a post-processing step to any periodic adaptive sampling policy. The defense uses quantization to encode all collected batches as fixed-sized messages, hiding the true collection rate from the attacker. Unlike message padding [Dyer et al., 2012], AGE incurs no communication overhead by setting the fixed message length to that of the underlying adaptive sampling policy’s *average* batch size. With this design, AGE achieves 11% lower error than non-adaptive policies while eliminating information leakage and incurring negligible energy overhead. These properties allow AGE to provide a practical security solution on embedded sensing devices.

The second system performs distributed inference using Multi-exit Neural Networks (MeNNs) [Teerapittayanon et al., 2017, Laskaridis et al., 2020] (Chapter 4). These inference models use multiple exit points and partitioning to reduce inference costs in sensing systems. Existing MeNNs adaptively determine when to exit early using a single threshold on data-dependent prediction confidence values [Teerapittayanon et al., 2016, Kaya et al., 2019]. Thus, prior methods make exit decisions in a data-dependent manner. We discover that an attacker can observe when the system exits early through the communication patterns of distributed MeNNs [Kannan et al., 2023]. The attacker can then use these exit decisions to expose the model’s predictions at over $1.85\times$ the rate of random guessing. In some cases, this side-channel can leak over 80% of the MeNN’s predictions. We solve this

problem by proposing two new defenses. The first defense, Per-Class Exiting (PCE), replaces the single threshold of prior techniques with multiple confidence thresholds. These thresholds balance exit rates across predicted classes, limiting information leakage. PCE displays better privacy with inference accuracy that is close to previous methods. However, we prove that PCE has no privacy guarantees, and we achieve these guarantees through a second solution, Confidence-Guided Randomness (CGR). CGR augments PCE with randomization to obfuscate temporal patterns in early exit decisions. By leveraging prediction confidence and randomness, CGR has consistently better inference accuracy and statistically equivalent privacy compared to a baseline which exits early uniformly at random. Both PCE and CGR show negligible energy overhead, making these methods suitable for sensing devices.

These discovered side-channels extend beyond embedded systems and into the more general class of Internet-of-Things (IoT) devices. We demonstrate this phenomenon by developing a novel attack against modern Smart Televisions (TVs) (Chapter 5). Smart TVs are internet-connected TVs that support video streaming applications and web browsers. Users enter information into Smart TVs through on-screen virtual keyboards. These keyboards allow users to navigate between keys with directional commands from a remote controller. Given the extensive functionality of Smart TVs, users type sensitive information (e.g., passwords) into these devices, making keystroke privacy necessary. We create and demonstrate a new side-channel attack [Kannan et al., 2024] that exposes keystrokes from the audio of two popular Smart TVs: Apple [Apple, 2023] and Samsung [Samsung, 2023]. This side-channel attack exploits how Smart TVs make different sounds when selecting a key, moving the cursor, and deleting a character. These properties allow an attacker to extract the number of cursor movements between selections from the TV’s audio. Our attack uses this extracted information to identify the likeliest typed strings. Against realistic users, the attack finds up to 33.33% of credit card details and 60.19% of common passwords within 100 guesses. Samsung has acknowledged this vulnerability and placed our work in the Samsung Bug Bounty

Hall of Fame for Smart TVs, Audio, and Displays¹.

This thesis builds on prior attacks that extract private information from traditional computing systems [Brumley and Boneh, 2005, Duong and Rizzo, 2012, Kocher et al., 2020, Lipp et al., 2020, Song et al., 2001] by describing how both embedded and IoT devices can leak information in unintended ways. These vulnerabilities show how system designers must carefully consider the privacy implications of every feature interacting with sensitive data.

1. <https://samsungtvbounty.com/hallOfFame>

CHAPTER 2

BUDGET RNNS

2.1 Introduction

Battery-powered sensors face the challenge of a finite lifetime. When the battery is exhausted, devices need maintenance to continue operation, and this can be costly or impractical for sensors located in hard-to-reach places [Martinez et al., 2004, Naylor and Kie, 2004, Tyler et al., 2013, Zhang et al., 2004]. Thus, these systems are often augmented with recharging capabilities; e.g., wirelessly [Yang and Wang, 2015] or through energy harvesting [Bagree et al., 2010, Jang et al., 2010, Tyler et al., 2013, Zhang et al., 2004]. This general technique induces energy budgets: until recharged, sensors must perform using only previously-stored energy. As recharging systems exhibit variance in both generated power and energy availability, the induced budgets are unknown at design time [Sommer et al., 2018].

To meet varying energy budgets, sensors must alter their runtime energy consumption. Sensor energy consumption typically goes into three tasks: collecting, processing, and communicating data. By collecting data less often—i.e., *sampling* the data—devices reduce the energy consumed by both sensing hardware and subsequent processing [Jain and Chang, 2004, Willett et al., 2004, Zhou and De Roure, 2007]. Reducing the data collection rate, however, comes at a cost in error. By collecting fewer values, the sensor creates a less-complete view of the target environment. With this tradeoff, a logical goal of budgeted computation is to minimize the error (or maximize accuracy) while altering energy consumption to meet the energy constraint.

Understanding how data collection affects error requires understanding sensor computations. Sensors often perform local inference for improved reactivity, privacy, and energy-efficiency [Kumar et al., 2017, Shi and Dustdar, 2016]. Furthermore, this processing is

increasingly focused on deep neural networks (DNNs) [Dennis et al., 2018, Gobieski et al., 2019, Teerapittayanon et al., 2017] due to their high accuracy for image [Krizhevsky et al., 2012] and audio [Graves et al., 2013, Koutnik et al., 2014] tasks. As many sensing tasks operate on data streams, recurrent neural networks (RNNs) [Hochreiter and Schmidhuber, 1997, Werbos, 1990] are common DNN models for in-sensor inference [Dennis et al., 2018, 2019, Kusupati et al., 2019].

RNNs operate on data sequences of arbitrary length, making them well-suited to sampling techniques, and thus, operating under energy budgets. The challenge involves determining how to minimize the RNN’s inference error under an energy constraint. This challenge leads to the concrete problem of selecting the sampling technique that both minimizes the RNN error and conserves enough energy to meet the budget. As we assume the exact energy constraint is not known at design time, the system must have sufficient flexibility to provide high accuracy across a range of energy budgets.

There exist prior RNN solutions—specifically, Skip RNNs [Campos et al., 2017] and Phased RNNs [Neil et al., 2016]—that appear to fit into the framework of budgeted computation. While they differ in specifics, these approaches jointly train both an RNN and a sampling strategy. The joint training means that each RNN operates at a single energy consumption level. To meet an energy budget that is only known at runtime, the inference system must use many RNNs, each trained for a different budget. The need to produce many separate networks with different sampling strategies leads to a high training cost. Furthermore, deploying many neural networks will exhaust the memory capacity of low-power devices. An additional downside is the inefficient use of energy budgets. To meet the budget $B \sim [B_{min}, B_{max}]$, the system may have to select an RNN with energy consumption b that is strictly less than the budget. Thus, the system will have a surplus of $B - b > 0$ joules. The goal of this thesis is to produce budgeted inference designs that are low overhead (in training time and memory footprint) yet use the entire energy budget to maximize accuracy,

rather than waste it as would be done with prior work.

Therefore, we propose a new type of RNN called the Budget RNN. Budget RNNs are designed from the ground up for accurate inference under energy budgets only known at runtime. The model has a *leveled* architecture in which each level processes a subsequence of inputs. Budget RNNs conserve energy by allowing execution to halt after each subsequence. This design enables a single Budget RNN to dynamically alter its energy consumption. As Budget RNNs support non-contiguous subsequences, these features come without compromising sampling flexibility. Budget RNNs use trainable halting signals to provide information about when to stop inference. We design a runtime controller that uses these halting signals to optimize the model accuracy while meeting an energy budget. The controller generalizes to new budgets and adapts the system to changes in the runtime environment. Furthermore, the combination of early-stopping and dynamic sampling can be realized in a relatively small number of Budget RNNs, leading to reduced training time and low memory overhead.

We evaluate the proposed system in both a simulated environment and a hardware implementation. We show that, across seven datasets, the Budget RNN system achieves a mean accuracy that is 1.5 points higher than that of Skip RNNs [Campos et al., 2017] and 3 points higher than that of standard RNNs. Alternatively, Budget RNNs can be used to reduce energy needs (and the associated battery and charging capacity). Specifically, for the same accuracy of baseline approaches, Budget RNNs require 20% smaller energy budgets. With respect to Skip RNNs, these benefits come at no cost in training time.

In summary, this thesis makes the following contributions¹:

- We establish a framework for maximizing inference accuracy under energy budgets. We use this framework to formalize a goal for inference systems that adapt to unseen budgets at runtime.
- We present a novel RNN architecture, called the Budget RNN, for performing inference

1. This work appears as a conference paper in RTAS 2021 [Kannan and Hoffmann, 2021]

under energy budgets that are unknown at design time. Budget RNNs change their energy consumption by using a leveled architecture to process subsamples of input sequences. This design achieves better accuracy under energy budgets when compared to existing RNN solutions.

- We design an optimization procedure to control the subsampling behavior of Budget RNNs. This optimizer accounts for energy constraints and provides results that generalize to unseen budgets.
- We create a runtime controller for Budget RNNs to ensure the model meets any observed energy constraint. This controller uses a dynamic setpoint that adapts based on feedback from the Budget RNN. The control policy further adapts the system to changes in the runtime environment.

2.2 Background

In this section, we motivate the problem of inference under runtime energy budgets using examples from rechargeable sensors (§2.2.1), provide background information on RNNs (§2.2.2), establish a formal framework for budgeted inference (§2.2.3), and identify limitations in prior work (§2.2.4).

2.2.1 Examples of Inference under Energy Budgets

Two examples of target systems are wireless rechargeable sensor networks (WRSNs) [Yang and Wang, 2015] and devices with energy harvesting units [Zhang et al., 2004]. We describe these applications below.

WRSNs use a mobile charging unit to wirelessly recharge sensors [Yang and Wang, 2015]. The charging unit uses a protocol, such as on-demand charging, to determine when to visit devices [He et al., 2014]. In on-demand charging, sensors notify the mobile charging unit

when they are low on power, and the charging unit uses a policy to determine the exact recharge time [He et al., 2014, Lin et al., 2018]. This scenario creates an energy budget: once sensors request a recharge, they have finite remaining energy to use before the charging unit arrives. Furthermore, the time until a recharge is dependent on both the state of the network and the visitation policy. Thus, sensors only know the exact energy budget at runtime.

Energy harvesting systems supplement batteries with renewable energy. For example, sensors for wildlife tracking [Bagree et al., 2010, Naylor and Kie, 2004, Sommer et al., 2018, Zhang et al., 2004], bridge monitoring [Jang et al., 2010], and ecosystem observation [Martinez et al., 2004, Tyler et al., 2013] use solar panels. Energy harvesting has varying performance due to a dependence on the environment; e.g., solar panels generate up to six times less power in overcast rather than sunny weather [Zhang et al., 2004]. This variance creates inherent unreliability in sensor lifetime, which is a problem for operators. For example, ZebraNet operators want sensors to last for at least 72 hours on battery power alone [Zhang et al., 2004]. This design requirement leads to energy budgets; given only the energy stored before losing renewable power, sensors must meet the desired uptime. This goal is challenging because the residual battery charge before a loss of renewable power is unknown at design time. Thus, such sensors must handle energy constraints that are not known until runtime.

2.2.2 Recurrent Neural Networks and Sequential Classification

Sensors collect periodic measurements of their surrounding environment, creating a temporal data stream [Madden and Franklin, 2002]. As an example, consider human activity recognition [Kwapisz et al., 2011]. In this task, sensors measure acceleration values at regular intervals. The corresponding inference model uses a sequence of measurements to predict the activity.

Recurrent Neural Networks (RNNs) are a popular model for inference on sequences. RNNs’ key feature is their maintenance of an internal *memory state* [Hochreiter and Schmidhuber, 1997, Pineda, 1987, Werbos, 1990]. At each step, this memory state represents a summary of the inputs observed thus far. RNNs update the memory state using a trainable transition function called the RNN *cell*.

We formally describe RNNs by considering an ordered sequence $\mathbf{X} = \{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{T-1}\}$. Each vector $\mathbf{x}_t \in \mathbb{R}^n$ represents the input measurement at step $t \in [T]^2$; e.g., for human activity recognition, each \mathbf{x}_t holds 3D acceleration values. At step $t \in [T]$, the RNN updates the memory state $\mathbf{s}_t \in \mathbb{R}^d$ using the transition function (cell) $S_{\vartheta} : \mathbb{R}^n \times \mathbb{R}^d \rightarrow \mathbb{R}^d$,

$$\mathbf{s}_t = S_{\vartheta}(\mathbf{x}_t, \mathbf{s}_{t-1}) \quad \forall t \in [T] \tag{2.1}$$

There exist many RNN cells, from single-layer networks [Pineda, 1987] to more complex designs for learning long-term relationships [Cho et al., 2014, Collins et al., 2016, Hochreiter and Schmidhuber, 1997, Koutnik et al., 2014, Kusupati et al., 2019].

RNNs create predictions using a trainable readout function g_{ψ} , which often takes the form of a multi-layer perceptron [Rumelhart et al., 1994]. The readout layer uses a summary of the input sequence to make the prediction. There are many ways to create this summary; in the simplest case, the readout function uses the final memory state \mathbf{s}_{T-1} , and the prediction is $\hat{\mathbf{y}} = g_{\psi}(\mathbf{s}_{T-1})$. Figure 2.1 shows an example of an RNN.

2.2.3 Inference under an Energy Budget

We now formalize the goal of designing an RNN-based inference system that achieves low error under runtime energy constraints. Let $\mathcal{F} = \{f_{\theta_1}, f_{\theta_2}, \dots, f_{\theta_p}\}$ be a family of (sequential) inference models. We assume each model incurs a different average energy cost

2. We use the notation $[N] = \{0, 1, \dots, N - 1\}$

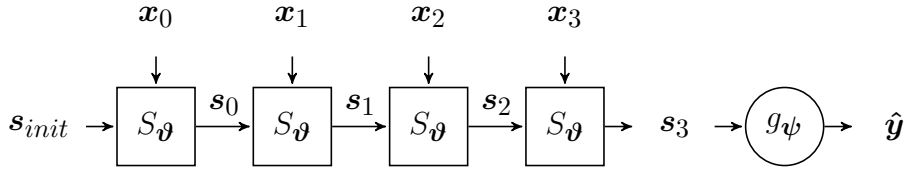


Figure 2.1: An RNN on a sequence with three elements. S_{θ} represents the RNN cell and g_{ψ} is the readout layer.

to process a single sequence. Let $\mathcal{X} = \{\mathbf{X}^{(0)}, \mathbf{X}^{(1)}, \dots, \mathbf{X}^{(M-1)}\}$ be a set of M non-overlapping sequences. Each sequence $\mathbf{X}^{(m)} = [\mathbf{x}_1^{(m)}, \mathbf{x}_2^{(m)}, \dots, \mathbf{x}_T^{(m)}]$ contains T elements where consecutive measurements are sampled Δt seconds apart.

We define an energy constraint as a tuple (B, M) capturing the goal of performing inference on M sequences with energy at most B . We assume that M is a finite integer in $[M_{min}, M_{max}]$ and $B \sim U([B_{min}, B_{max}])$. The ranges describe the set of feasible constraints. Given these constraints, we want a policy π which selects a model in \mathcal{F} at each step $m \in [M]$. The selected model, denoted by $f_m^{\pi} = \pi(\mathcal{F}, \mathbf{X}^{(m)}, B, M)$, performs inference on the m^{th} sequence $\mathbf{X}^{(m)}$. The dependence of π on B and M arises because the system must adapt to any feasible constraint.

We want to construct a policy π and a family of inference models \mathcal{F} to (1) minimize the expected error over all possible budgets and (2) meet all energy constraints. Thus we have the following optimization problem (where y_m is the true label for the m^{th} sequence):

$$\pi, \mathcal{F} = \operatorname{argmin}_{\tilde{\pi}, \tilde{\mathcal{F}}} E_{M,B} \left[\frac{1}{M} \sum_{m=0}^{M-1} \operatorname{Error}(f_m^{\tilde{\pi}}(\mathbf{X}^{(m)}), y_m) \right] \quad (2.2)$$

$$s.t. \quad \sum_{m=0}^{M-1} \operatorname{Energy}(f_m^{\tilde{\pi}}, \mathbf{X}^{(m)}) \leq B \quad \forall M, B \quad (2.3)$$

It is infeasible to solve this problem exactly. Even if we could, we do not know the specific energy budgets at design time.

2.2.4 Adaptive Sampling in Recurrent Neural Networks

A family \mathcal{F} for budgeted inference must contain inference models with varying energy costs. Thus, to build a family \mathcal{F} with RNNs, we need a method to control the energy consumption of RNNs during inference. In sensor environments, RNNs can adjust their energy consumption by varying the number of inputs. For example, rather than using a full sequence $[\mathbf{x}_0, \dots, \mathbf{x}_{T-1}]$ to form a prediction, an RNN can save energy by instead using the subsequence $[\mathbf{x}_{\alpha_0}, \dots, \mathbf{x}_{\alpha_{r-1}}]$ where $r < n$. Skipping inputs saves energy by reducing the frequency of both processing and collection. These savings are significant when compared to the energy required to communicate predictions to a centralized server. Between the tasks of collection, processing, and communication, data processing has the lowest energy cost. For example, both the TI MSP430 FR5994 [Instruments, 2021] and the Atmel SAM L21 [Microchip, 2020] draw under $200\mu\text{A}$ per MHz. In contrast, sensing hardware can consume an order of magnitude more power, and the cost of sensing can even exceed that of radio modules [Alippi et al., 2009b]. As transmission only occurs once per sequence, the relative cost and frequency of collection allow subsampling to yield significant energy savings [Alippi et al., 2009b]. We can thus create a budgeted inference system with a family of models \mathcal{F} composed of RNNs that subsample input sequences.

Two state-of-the-art RNN architectures support subsampling: Skip RNNs [Campos et al., 2017] and Phased RNNs [Neil et al., 2016]. Skip RNNs use a trainable binary gate to skip sequence elements. This gate is jointly trained with the RNN parameters, and it gives the model fine-grained control over the sampling strategy. Phased RNNs use a periodic phase gate to control when the RNN makes updates to its memory state. By unifying the phase gate across all state dimensions, Phased RNNs can use this gate to skip inputs. Thus, both Skip and Phased RNNs can create a family of inference functions (\mathcal{F}) by sub-sampling the sequence with varying granularity.

Both approaches train their sampling strategy at the same time they train their RNN

parameters. This joint training reduces the flexibility to adapt to runtime energy availability and is a bad match for the constraints of low-power sensing systems. In particular, a Skip or Phased RNN cannot change its sampling strategy after training. This property means that an instance of either RNN operates at only one energy consumption level. To meet energy budgets that are unknown at design time, a system based on either model would need many trained neural networks. This strategy requires a longer training time, and memory restrictions cap the number of deployed models. Furthermore, a reasonable selection policy (π) is to always choose the best RNN that meets the energy constraint. As there are a small, discrete number of models (due to memory constraints on embedded devices), this policy inefficiently uses the available energy; for a constraint (B, M) , the best model may use only $b < B$ joules to classify M sequences. Our goal is to design a practical approach that uses the remaining $B - b$ joules to produce more accurate inference results.

2.3 Budget RNN Model Architecture

Budget RNNs are a family of RNN architectures designed to (1) deliver high accuracy under energy constraints and (2) generalize to budgets that are unknown until runtime. There exists a tension between these two guiding principles. On one hand, RNN accuracy depends on the sequence subsampling algorithm. As discussed for Skip and Phased RNNs, however, creating a more flexible subsampling algorithm through joint training limits the ability to generalize to new budgets.

Budget RNNs address this tension using four features. First, Budget RNNs have a *leveled architecture* where each level processes a distinct subsequence (§2.3.1). The Budget RNN produces a prediction after each subsequence. Second, the model avoids recomputation by *merging* RNN memory states across subsequences (§2.3.2). The leveled architecture allows a single Budget RNN to operate at many energy levels. In particular, controlling the number of executed subsequences determines the number of input elements the Budget RNN uses.

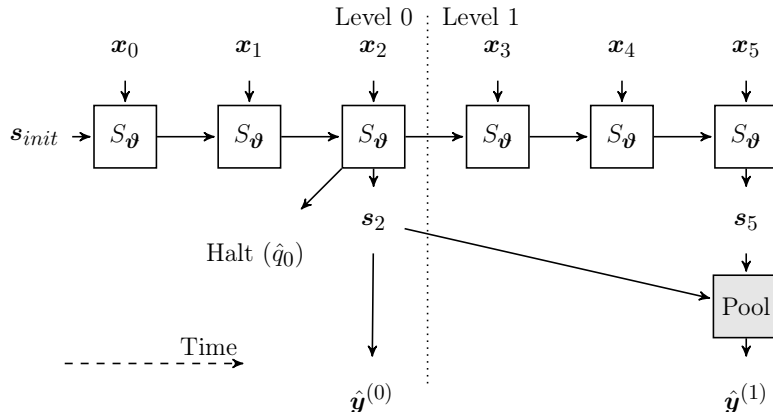


Figure 2.2: A Budget RNN with sequence length $T = 6$, $L = 2$ levels and stride length $K = 1$.

This control is possible due to the third feature of Budget RNNs: *halting signals* (§2.3.3). For each subsequence, Budget RNNs produce a signal which indicates whether the model should halt its execution. By setting thresholds on these signals, an inference policy (π) can dynamically alter the Budget RNN’s sampling granularity. These signals decouple the subsampling strategy from the Budget RNN parameters. We discuss this control strategy in Section 2.4. Finally, Budget RNNs are trained with a novel *loss function* that balances the predictions and halting signals across all subsequences (§2.3.4). Figures 2.2 and 2.3 show two Budget RNN architectures.

2.3.1 Leveled Architecture

Each Budget RNN level applies a shared RNN to a distinct subsequence. Budget RNNs sequentially process these subsequences, producing a prediction from each. These predictions allow Budget RNNs to exit early during inference. Further, the Budget RNN can skip inputs as the data from unprocessed subsequences does not need to be collected. Thus, Budget RNNs can save on both computation and data collection.

We describe this process formally by considering the subsequence $\mathbf{X}^{(\alpha)} = \{\mathbf{x}_{\alpha_0}, \dots, \mathbf{x}_{\alpha_{r-1}}\}$

where r is less than the original sequence length T . Similar to standard RNN update (Equation 2.1), Budget RNNs use the state transition model \mathcal{S}_{θ} and readout layer g_{ψ} to perform on the subsequence $\mathbf{X}^{(\alpha)}$:

$$\mathbf{s}_{\alpha_t} = \mathcal{S}_{\theta}(\mathbf{x}_{\alpha_t}, \mathbf{s}_{\alpha_{t-1}}) \quad \forall t \in [r] \quad (2.4)$$

$$\hat{\mathbf{y}}^{(\alpha)} = g_{\psi}(\mathbf{s}_{\alpha_{r-1}}) \quad (2.5)$$

The transition model \mathcal{S}_{θ} can be any RNN cell [Cho et al., 2014, Collins et al., 2016, Hochreiter and Schmidhuber, 1997]. Both \mathcal{S}_{θ} and g_{ψ} are shared across all subsequences, a key feature for reducing memory overhead.

Budget RNNs apply this recurrent process to each subsequence. Subsequence formation is based on two parameters:

- *Number of Subsequences (L)*: Each subsequence results in a prediction, so this parameter also represents the number of predictions. We also refer to L as the number of levels—the subsequences form *levels* in the model.
- *Stride Length (K)*: This parameter determines the gap between elements in each subsequence. For example, in a sequence of length $T = 6$, the subsequences for a stride length of $K = 2$ are $\{\mathbf{x}_0, \mathbf{x}_2, \mathbf{x}_4\}$ and $\{\mathbf{x}_1, \mathbf{x}_3, \mathbf{x}_5\}$. In general, the ℓ^{th} subsequence has the following form:

$$\mathbf{X}^{(\ell)} = \begin{cases} \{\mathbf{x}_{(\ell T/L)+n} \mid n \in [\frac{T}{L}]\} & \text{if } K = 1 \\ \{\mathbf{x}_{\ell+nK} \mid n \in [\frac{T}{K}]\} & \text{if } K > 1 \end{cases} \quad (2.6)$$

When $K > 1$, we set $L = K$ to avoid the having a single element in multiple subsequences.

Both L and K constitute hyperparameters for Budget RNNs, and we assume both parameters

divide the sequence length T . The stride length has a noticeable impact on model accuracy, and we discuss how to set this parameter in §2.4.4.

2.3.2 Merging Memory States Across Subsequences

To maximize accuracy and avoid energy-wasteful redundant computation, Budget RNNs leverage already-processed sequence elements. When executing on level ℓ , the Budget RNN *merges* the new inputs to improve the prediction from level $\ell - 1$. A key property of this merging is the avoidance of backward dependencies. During inference, a sensor supplies the Budget RNN with measurements collected over time. Further, Budget RNNs conserve energy by skipping subsequences. From these two properties, backward dependencies would require the system to either execute the current step using future data or collect and store elements that may be ignored. The former is impossible, and the latter is energy-inefficient.

When $K = 1$, the Budget RNN looks like a standard RNN with L early-exit points [Dennis et al., 2018]. The model combines information between subsequences using the RNN cell: the final memory state of the $\ell - 1^{th}$ level becomes the initial memory state of the ℓ^{th} level. Figure 2.2 shows an example of this design.

When $K > 1$, the merging is more complex due to the interleaving of subsequences. For example, when $T = 4$ and $K = 2$, the Budget RNN uses subsequences $\{\mathbf{x}_0, \mathbf{x}_2\}$ and $\{\mathbf{x}_1, \mathbf{x}_3\}$. In this case, the final element of the first subsequence, \mathbf{x}_2 , occurs after elements in the second subsequence. Thus, using \mathbf{s}_2 as the initial state of the second subsequence creates a backward dependency. Budget RNNs solve this problem by aligning and merging the memory states from each level. Equations (2.7) and (2.8) below show how the merging layer \mathcal{M} interacts with the standard RNN cell in (2.9). The variables $\mathbf{W}^{(\mathcal{M})}$, $\mathbf{U}^{(\mathcal{M})}$ and

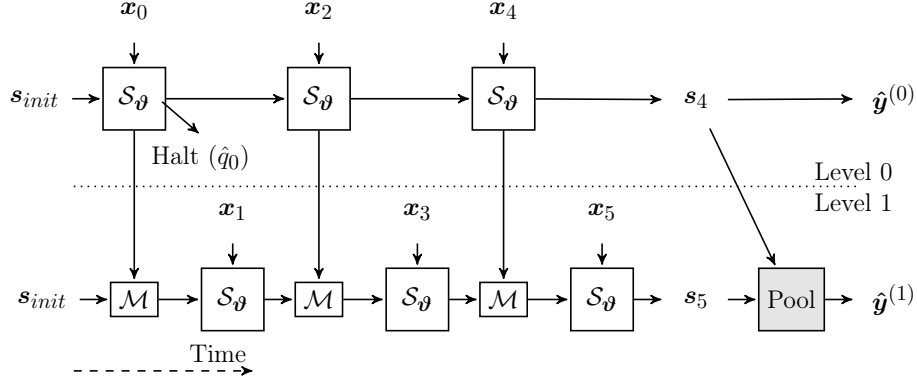


Figure 2.3: A Budget RNN with sequence length $T = 6$, $L = 2$ levels and stride length $K = 2$.

$\mathbf{b}^{(\mathcal{M})}$ are trainable parameters, σ is the sigmoid function, and \odot is the element-wise product.

$$\mathbf{z}_t = \sigma(\mathbf{W}^{(\mathcal{M})} \mathbf{s}_{t-K} + \mathbf{U}^{(\mathcal{M})} \mathbf{s}_{t-1} + \mathbf{b}^{(\mathcal{M})}) \quad (2.7)$$

$$\tilde{\mathbf{s}}_{t-1} = \mathbf{z}_t \odot \mathbf{s}_{t-K} + (1 - \mathbf{z}_t) \odot \mathbf{s}_{t-1} \quad (2.8)$$

$$\mathbf{s}_t = \mathcal{S}_{\theta}(\mathbf{x}_t, \tilde{\mathbf{s}}_{t-1}) \quad (2.9)$$

To clarify this design, let step t belong to subsequence ℓ . This merging layer makes \mathbf{s}_t dependent on all inputs collected thus far. The state \mathbf{s}_{t-K} precedes step t in the ℓ^{th} subsequence and represents a summary of the *current* subsequence up to step t . The state \mathbf{s}_{t-1} belongs to subsequence $\ell - 1$ and represents a summary of the already-collected elements in *previous* subsequences. Using a combination of \mathbf{s}_{t-1} and \mathbf{s}_{t-K} makes \mathbf{s}_t dependent on all elements collected up to step t . Figure 2.3 depicts this design.

Within this leveled architecture, the error should always improve as the model observes more data. In practice, however, this desired behavior does not always hold. We hypothesize that this issue stems from two factors: (1) weight sharing across levels and (2) a loss function based on an unweighted average of the per-level loss (§2.3.4). To mitigate this problem, we

use an additional output pooling layer to combine the predictions across subsequences. The pooling operation allows Budget RNNs to use predictions from previous levels if doing so lowers the error. As shown in the equations below, this layer uses a trainable weighted average to aggregate predictions. The variable $\mathbf{s}^{(\ell)}$ is the final memory state for level ℓ , \mathbf{w}_{pool} , \mathbf{u}_{pool} and b_{pool} are trainable, and g_{ψ} is the readout function.

$$r_k = \mathbf{w}_{pool}^{\top} \mathbf{s}^{(\ell)} + \mathbf{u}_{pool}^{\top} \mathbf{s}^{(k)} + b_{pool} \quad \forall k \leq \ell \quad (2.10)$$

$$\beta_k = \text{Normalize}(r_0, \dots, r_{\ell})_k \quad \forall k \leq \ell \quad (2.11)$$

$$\hat{\mathbf{y}}^{(\ell)} = \sum_{k=0}^{\ell} \beta_k g_{\psi}(\mathbf{s}^{(k)}) \quad (2.12)$$

In our implementation, we use a SparseMax normalization [Martins and Astudillo, 2016] instead of a softmax normalization due to better numerical stability when implemented in fixed-point arithmetic on MCUs.

2.3.3 Halting Signals

Budget RNN’s use early-exit points to alter their energy consumption. For example, to conserve energy, a Budget RNN can stop at an intermediate level ℓ ; the sensor then does not need to capture inputs belonging to subsequences $\ell' > \ell$. To maximize accuracy, the inference policy (π) should control the number of executed levels in a data-dependent manner [Campos et al., 2017]. That is, the system should use inputs to determine when to exit inference. Budget RNNs support the ability to make data-dependent decisions using trainable halting signals [Graves, 2016, Rufwurm et al., 2019, Schmidhuber, 2012]. At each level $\ell \in [L]$, these signals indicate the probability the current level’s prediction is correct. The policy π controls the halting behavior using thresholds on these signals (§2.4).

We describe these halting signals by considering two cases. First, when $K = 1$, the model creates halting signals using the final memory state of each level. Second, when

$K > 1$, the Budget RNN creates halting signals from the first element of each subsequence. This discrepancy is a result of the interleaving of elements across subsequences. For example, during inference, the system must decide whether to collect \mathbf{x}_1 after processing \mathbf{x}_0 . When $K > 1$, \mathbf{x}_0 and \mathbf{x}_1 belong to different subsequences. Consider if the Budget RNN created the halting output after completing level $\ell = 0$. The system would need to collect elements (such as \mathbf{x}_1) that may be ignored if the system decides to halt at $\ell = 0$. Such extraneous data collection wastes energy. To efficiently collect inputs, the system must use \mathbf{x}_0 to decide whether it should halt at level $\ell = 0$. Therefore, when $K > 1$, the Budget RNN must use the first memory state of the ℓ^{th} subsequence to construct the halting signal for level ℓ .

Budget RNNs use a shared, trainable layer to create the halting signals. The equations below show the halting function h_ϕ applied to level ℓ . The term $\mathbf{s}_{\delta(\ell)}$ is the final state of level ℓ when $K = 1$ and the first state of level ℓ when $K > 1$. The variables $\mathbf{W}_{halt,1}$, $\mathbf{w}_{halt,2}$, $\mathbf{b}_{halt,1}$ and $b_{halt,2}$ are trainable, and ϕ is the nonlinear activation function.

$$\mathbf{h}_\ell = \phi(\mathbf{W}_{halt,1}\mathbf{s}_{\delta(\ell)} + \mathbf{b}_{halt,1}) \quad (2.13)$$

$$\hat{q}_\ell = \sigma(\mathbf{w}_{halt,2}^\top \mathbf{h}_\ell + b_{halt,2}) \quad (2.14)$$

Budget RNNs train the halting signal \hat{q}_ℓ to predict whether the model is correct at level ℓ . For classification tasks, the label is $q_\ell = 1[y = \arg \max \hat{\mathbf{y}}^{(\ell)}]$ where $1[\cdot]$ is an indicator function. For regression tasks, the label is $q_\ell = 1[\|\mathbf{y} - \hat{\mathbf{y}}^{(\ell)}\|_2^2 < \epsilon]$ for a threshold $\epsilon > 0$. In either case, the label is a non-differentiable function of the Budget RNN parameters. We address this problem by treating the label as a constant. This treatment is advantageous because it prevents the Budget RNN from crafting predictions to match the halting signals. In particular, this feature avoids behavior where the Budget RNN predicts a sequence wrong *on purpose* to match a halting signal close to zero. Such behavior leads to a poor model.

As a note, there exist neural network systems that implement early-exiting using thresholds on the predicted probabilities [Dennis et al., 2018, Teerapittayanon et al., 2016]. Such

a technique does not work for Budget RNNs as predictions are not always available when halting information is required. For this reason, we favor explicit halting signals.

2.3.4 Loss Function

All neural networks are trained to minimize a loss function. This function typically measures the difference between the predicted output and the true result. The novel architecture of Budget RNNs requires a new loss function to balance the optimization of both predictions and halting signals.

At each level $\ell \in [L]$, Budget RNNs produce two outputs: the prediction $\mathbf{y}^{(\ell)}$ and the halting signal \hat{q}_ℓ . We use a loss function to train the model using the outputs from all levels. The loss function below expresses this goal. The function $\tilde{\mathcal{L}}$ is the per-output loss (e.g. cross-entropy, mean-squared error).

$$\mathcal{L}(\{\hat{\mathbf{y}}^{(\ell)}\}, \{\hat{q}_\ell\}, y, \{q_\ell\}) = \sum_{\ell=0}^{L-1} \left(\tilde{\mathcal{L}}(\hat{\mathbf{y}}^{(\ell)}, y) + \gamma \tilde{\mathcal{L}}(\hat{q}_\ell, q_\ell) \right) \quad (2.15)$$

The variable γ determines the relative emphasis on the halting loss. We slowly increase γ over the first few epochs until it reaches γ_0 . In our experiments, we set $\gamma_0 = 0.01$. This slow increase improves training because the predictions frequently change in the first few epochs.

2.4 Control Policy and Subsampling Algorithm

The Budget RNN output levels compose a family of functions \mathcal{F} for budgeted inference. The inference system needs a selection policy π that minimizes error and adapts to unseen constraints. The Budget RNN policy meets these goals by dynamically selecting the output level in a budget-specific manner. This choice determines the number of collected inputs and

Algorithm 1 Adaptive Inference Algorithm for Budget RNNs

```
1: procedure ADAPTIVEINFERENCE( $f_{\theta}, z, L, K, T$ )
2:    $\ell_{max} \leftarrow L$ 
3:    $(S_{\vartheta}, \mathcal{M}, g_{\psi}, h_{\varphi}) \leftarrow f_{\theta}$ 
4:   for  $t \in [T]$  do
5:      $\ell \leftarrow \text{LevelOf}(t, K, L)$ 
6:     if  $\ell > \ell_{max}$  then
7:       continue
8:      $\mathbf{x}_t \leftarrow \text{CollectInput}()$ 
9:      $\tilde{\mathbf{s}}_{t-1} \leftarrow \mathcal{M}(\mathbf{s}_{t-1}, \mathbf{s}_{t-K})$  ▷ (Eq 2.8)
10:     $\mathbf{s}_t \leftarrow S_{\vartheta}(\mathbf{x}_t, \tilde{\mathbf{s}}_{t-1})$  ▷ (Eq 2.9)
11:     $\hat{q}_{\ell} \leftarrow h_{\varphi}(\mathbf{s}_t)$  ▷ (Eq 2.14)
12:    if  $\text{isHaltState}(t, K, L)$  and  $\hat{q}_{\ell} \geq z_{\ell}$  then
13:       $\ell_{max} \leftarrow \ell$ 
14:      if  $\text{IsLast}(t, K, L)$  and  $\ell = \ell_{max}$  then
15:        return  $\text{Prediction}(\mathbf{s}_t, g_{\psi})$  ▷ (Eq 2.12)
16:    return  $\text{Prediction}(\mathbf{s}_{T-1}, g_{\psi})$  ▷ (Eq 2.12)
```

the performed computation, ensuring efficient use of the available energy.

The Budget RNN policy achieves this dynamic behavior through four features. First, it performs adaptive inference (§2.4.1) by setting thresholds on the Budget RNN’s halting signals. Second, an optimizer (§2.4.2) tunes these thresholds over multiple potential constraints. The policy uses an interpolation method to generalize the thresholds to unseen budgets. Third, a feedback control system mitigates generalization errors while adapting to unforeseen changes in the runtime environment (§2.4.3). Finally, the policy improves accuracy by combining distinct Budget RNNs with different stride lengths (§2.4.4).

2.4.1 Adaptive Inference Algorithm

The Budget RNN inference policy π uses the halting signals to control the number of consumed inputs. In this manner, the policy π uses information from the model to determine the early-exiting behavior. *Thus, the policy adapts the energy consumption based on the input sequence.* These data-dependent decisions allow the system to conserve energy on sequences where the RNN needs fewer inputs to achieve a low error. The policy spends this saved

Algorithm 2 Optimization Algorithm for Halting Thresholds

```
1: procedure FITTHRESHOLDS( $\mathcal{D}, f_{\theta}, L, (B, M)$ )
2:    $\mathbf{z} \sim U([0, 1]^L)$ 
3:   while not converged do
4:      $k \sim \{0, \dots, L - 2\}$  Uniformly at Random
5:      $\mathbf{z}_k \leftarrow \operatorname{argmin}_{\mathbf{z}_k \in [0, 1]} \operatorname{AdjError}(f_{\theta}, \mathcal{D}, \mathbf{z}, B, M)$ 
6:   return  $\mathbf{z}$ 
```

energy to reconcile "harder" sequences.

The policy π implements early-exiting using a budget-specific threshold vector $\mathbf{z}^{(B, M)}$ where (B, M) is the energy constraint. When the halting probability \hat{q}_{ℓ} at level ℓ is greater than the threshold $\mathbf{z}_{\ell}^{(B, M)}$, the inference terminates at level ℓ . Otherwise, it continues to the next level. Algorithm 1 describes this adaptive inference routine. We emphasize an integral feature of this design: when inference halts at level ℓ , the system does *not* collect measurements associated with levels $\ell' > \ell$. Thus, when stopping at lower levels, the Budget RNN exhibits significant energy savings by not engaging the sensing hardware to collect unused measurements.

2.4.2 Optimizing Halting Thresholds

The policy π controls energy consumption using budget-specific thresholds on the Budget RNN halting signals. We create these thresholds using an additional training step. This process uses a coordinate descent technique [Wu et al., 2008] to fit thresholds for a given constraint. Overall, the optimizer creates thresholds to both minimize error and meet the budget.

Consider a Budget RNN f_{θ} and an energy constraint (B, M) . The optimization problem below formalizes the goal of the threshold optimizer. The term $\mathcal{D} = \{\mathbf{X}^{(i)}, y^{(i)}\}_{i=0}^{M-1}$ is the training dataset, and the "Error" function uses predictions from the adaptive inference routine in Algorithm 1. This problem is similar to that of the original problem statement

for budgeted inference (Equation 2.3).

$$\mathbf{z}^* = \operatorname{argmin}_{\mathbf{z} \in [0,1]^L} \sum_{i=0}^{M-1} \operatorname{Error}(f_{\boldsymbol{\theta}}, \mathbf{z}, \mathbf{X}^{(i)}, y^{(i)}) \quad (2.16)$$

$$\text{s.t.} \quad \sum_{i=0}^{M-1} \operatorname{Energy}(f_{\boldsymbol{\theta}}, \mathbf{z}, \mathbf{X}^{(i)}) \leq B \quad (2.17)$$

This optimization problem is difficult to solve for three reasons. First, the runtime energy consumption is often unknown at design time. We thus approximate the energy consumption using profiled values from sensing hardware. Second, the optimization problem may have a discontinuous objective function. For example, in classification tasks, the error function is the negative inference accuracy. To address this challenge, we use a coordinate descent solver. At each step, the optimizer selects a random threshold index and approximately finds its optimal value in $[0, 1]$. Finally, the optimizer must adhere to the energy constraint. We implement this behavior by forming an adjusted error function that penalizes budget violations. The equations below show an adjusted error function for classification accuracy. This function scales the accuracy using the number of sequences that fit under the budget. The variable \hat{b} is the average consumed energy per sequence, and the ‘‘Acc’’ function uses predictions from the adaptive inference routine in Algorithm 1.

$$\hat{a}(f_{\boldsymbol{\theta}}, \mathcal{D}, \mathbf{z}, B, M) = -\operatorname{Acc}(f_{\boldsymbol{\theta}}, \mathbf{z}, \mathcal{D}) \cdot \frac{\min(M, B/\hat{b})}{M} \quad (2.18)$$

Algorithm 2 describes the optimizer. Below we discuss a few implementation details regarding the halting thresholds.

- *Initialization Strategy:* We initialize the threshold for level ℓ using $U([\tilde{q}_{\ell}, 1])$ where \tilde{q}_{ℓ} is the median halting signal. We further set thresholds to zero for levels that violate the budget on a per-step basis; i.e., levels for which the required energy exceeds B/M . This strategy creates thresholds that approximately meet the budget, avoiding cases

where the optimizer makes suboptimal decisions just to meet the constraint early in training.

- *Coordinate Descent Step*: The key step during each iteration involves finding the optimal threshold. We find this value approximately by sweeping over a quantized subset of $[0, 1]$. In our implementation, we search over $\{0, 1\} \cup \{2^{n-k} \mid n \in [k]\}$ for $k = 256$. As a note, the use of fixed-point arithmetic on MCUs requires quantizing the threshold values before deployment. Thus, quantizing the thresholds during training adds no additional downstream error.
- *Convergence Detection*: We detect convergence by assessing how thresholds generalize to unseen validation data. Following standard practice, the optimizer terminates after Q iterations of non-improved validation error [Caruana et al., 2001]. This early-stopping mitigates the risk of overfitting. We set $Q = 25$ in our experiments.
- *Population-based Training*: To mitigate sensitivity to initialization, we use a population-based approach [Jaderberg et al., 2017] to fit many threshold vectors in parallel. Every R iterations, underperforming thresholds are set to the best vector. These copied thresholds are randomly perturbed to explore promising regions of the solution space. We use $R = 10$ in our experiments.
- *Generalizing to Unseen Budgets*: The inference policy must adapt to unseen energy budgets. The policy performs this adaptation by creating new thresholds at runtime. For an unseen budget, the policy finds the two nearest known budgets that bound this new constraint. The policy creates new thresholds by linearly interpolating the thresholds of the bounding budgets. In general, the relationship between thresholds and energy consumption may be nonlinear. We nevertheless find linear interpolation to be a low-overhead heuristic that empirically performs well. To mitigate interpolation errors, we use a runtime controller adjust the selected thresholds (§2.4.3).

Algorithm 3 Budget RNN Controller Policy

```
1: procedure CONTROLPOLICY( $f_{\theta}, (B, M), L, K, T$ )
2:    $Y \leftarrow [ ]$ 
3:    $B_0 \leftarrow B$ 
4:   for  $m \in [M]$  do
5:      $z \leftarrow \text{InterpolateThresholds}(B_m, M)$ 
6:      $\hat{y}_m \leftarrow \text{AdaptiveInference}(f_{\theta}, z)$  ▷ (Alg 1)
7:      $b_{obs} \leftarrow \text{ObserveEnergy}()$ 
8:      $(b_l, b_u) \leftarrow \text{GetSetpoint}(B, M, m)$  ▷ (Eq 2.21)
9:      $e \leftarrow \text{PIDControlError}(b_{obs}, (b_l, b_u))$ 
10:     $B_{m+1} \leftarrow B + e$  ▷  $e > 0$  means  $b_{obs}$  is low
11:     $Y \leftarrow Y \cup [\hat{y}_m]$ 
12:  return  $Y$ 
```

2.4.3 Runtime Controller

The inference policy for Budget RNNs uses budget-specific halting thresholds to minimize error. Alone, the threshold training process has two qualities that hurt the system’s ability to generalize. First, the optimizer approximates energy consumption using profiled values. The thresholds may yield a suboptimal error if the runtime and profiled environments differ. Second, the threshold interpolation method only approximately meets unseen budgets. The inference policy remedies these issues using a PID controller [Rivera et al., 1986]. For each sequence, the controller compares the observed energy consumption with the expected energy based on training. The policy uses the control error to adjust the halting thresholds by changing the target budget used during interpolation. For example, if energy consumption is lower than expected, the controller will use thresholds based on a larger budget. The controller thus maps the runtime environment into the space constructed during training. Algorithm 3 describes this process. In our implementation, we update the budget every $W = 20$ sequences.

The main challenge associated with this controller involves creating the setpoint. This process is nontrivial because the adaptive inference algorithm may not evenly spread the available energy across all M sequences. To account for system variation, the controller

uses a dynamic setpoint based on a confidence bound. This bound is based on the expected energy consumption and the corresponding estimator variance.

We derive this setpoint by considering b_k to be the average energy consumed when the Budget RNN predicts the label $k \in [C]$. For each label, the energy consumption may vary due to the adaptive inference routine (Algorithm 1). We thus assume that b_k is a normally distributed random variable. Consider when the system has processed $m < M$ inputs. We define r_k to be the total number of sequences with label k and $r_k^{(m)}$ to be the number of sequences in class k that occur in the first m steps. Let $X_{0:m}$ and $X_{m:M}$ be random variables representing the energy consumption in the first m and final $M - m$ steps respectively. The controller creates a setpoint based on $E[X_{0:m}]$. Given the total energy budget B , we want to have $B = E[X_{0:m}] + E[X_{m:M}]$. This relationship expresses the goal that the system should use the entire budget. Evaluating $E[X_{m:M}]$ yields the following expression for $E[X_{0:m}]$.

$$E[X_{0:m}] = B - \sum_{k=0}^{C-1} (r_k - r_k^{(m)}) E[b_k] \quad (2.19)$$

We estimate the values for r_k , $r_k^{(m)}$, and $E[b_k]$ using both the training set and the profiled energy values. We update these distributions with values obtained at runtime. Note that we do not have access to the true data labels at runtime. In this setting, the system uses the predicted labels as an approximation. Overall, this design creates an estimator $\hat{E}[X_{0:m}]$ for the desired expectation.

A setpoint based on this expected value alone does not capture the variance associated with Budget RNN execution. We instead use a confidence bound [Auer, 2002] which accounts for the estimator variance. Under the assumption that r_k is deterministic and the b_k variables are independent, we obtain the following equation for the variance of $X_{0:m}$. In practice, we again use an estimator of $\text{Var}[X_{0:m}]$ based on estimates of r_k , $r_k^{(m)}$ and $\text{Var}[b_k]$ from the

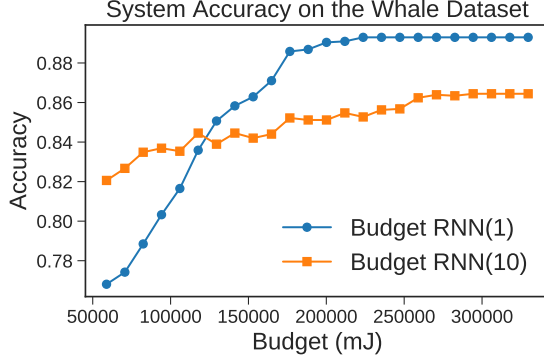


Figure 2.4: The accuracy for two distinct Budget RNNs on the task of classifying Whale sounds [Cox et al., 2006].

training set.

$$\text{Var}[X_{0:m}] = \sum_{k=0}^{C-1} (r_k - r_k^{(m)})^2 \text{Var}[b_k] \quad (2.20)$$

From the Fisher information for Gaussian random variables, the variance of the estimator $\hat{E}[X_{0:m}]$ converges to $\frac{1}{m} \text{Var}[X_{0:m}]$ [Fisher et al., 1920, Ly et al., 2017]. With these estimations, we construct the following setpoint range for the PID controller.

$$\text{Setpoint}(m) = \left(\hat{E}[X_{0:m}] - \sqrt{\frac{1}{m} \hat{\text{Var}}[X_{0:m}]}, \hat{E}[X_{0:m}] + \sqrt{\frac{1}{m} \hat{\text{Var}}[X_{0:m}]} \right) \quad (2.21)$$

When the observed energy is within this confidence bound, the controller error is zero. Otherwise, the PID control error is the difference to the nearest side of the bound.

We highlight one important aspect of this design: the controller does *not* need the budget at design time. At runtime, the controller uses Equations (2.19), (2.20), and (2.21) to dynamically construct a setpoint for an observed budget (B, M) . The system uses this setpoint to adapt the Budget RNN halting thresholds and meet the energy constraint (Algorithm 3).

2.4.4 Budget RNN Selection

The presented inference policy uses thresholds to control a single Budget RNN. On many datasets, we observe that one Budget RNN is not enough to deliver high accuracy across all budgets. This trend occurs due to the impact of the stride length parameter (K). Importantly, when comparing Budget RNNs with the stride length K_1 and K_2 , the model with K_1 may be better for some budgets and worse for others. Figure 2.4 shows such an example. To achieve the best performance across all budgets, the policy should leverage both models.

We use this insight to build the family \mathcal{F} with many distinct Budget RNNs. For a given budget, the policy π first selects the Budget RNN based on validation accuracy. The policy proceeds to use the runtime controller and corresponding thresholds for the chosen Budget RNN. We observe that providing the policy with a few models yields better inference accuracy. Further, this dynamic selection reduces the need to search over stride lengths to find the *single* best model.

Using many distinct RNNs has the downside of increasing the system’s memory footprint. In our experiments, we use only two Budget RNNs to achieve high accuracy over a spectrum of budgets. This modest number meets any reasonable memory constraints imposed by low-power devices.

2.5 Evaluation

We evaluate the Budget RNN’s ability to support accurate inference under energy budgets on embedded sensing systems. We compare to both standard RNNs and current state-of-the-art RNNs that support sub-sampling (Skip RNNs [Campos et al., 2017] and Phased RNNs [Neil et al., 2016]). We compare these systems across many datasets and energy budgets in a simulated environment³. We supplement these results with an evaluation on an embedded

3. All code is available at <https://github.com/tejaskannan/budget-rnn>

device. This evaluation shows the following:

1. Under the same budgets, the Budget RNN system achieves higher accuracy than that of baseline systems. Across all datasets, the Budget RNN system has a mean accuracy of 1.5 points greater than Skip RNNs, 2.7 points greater than Phased RNNs, and 3 points greater than standard RNNs (§2.5.2). Further, Budget RNNs can operate with 20% smaller energy budgets and still achieve accuracy comparable to the baselines (§2.5.3).
2. The control policy allows the Budget RNN system to adapt to new runtime environments. This adaptive behavior enables Budget RNNs to achieve greater improvements in such settings (§2.5.4).
3. The adaptive decisions made by the Budget RNN policy are key to the system’s performance. Removing the adaptive policy or making randomized decisions results in lower accuracy (§2.5.5).
4. Budget RNNs have lower training costs when compared to both Skip and Phased RNNs. These baseline systems take over $2.3\times$ longer to train on average (§2.5.6). Budget RNNs display cheaper training while delivering higher accuracy across all budgets.
5. The higher accuracy of Budget RNNs translates to an embedded system. Across two budgets, Budget RNNs maintain their improvement over Skip RNNs in a realistic hardware environment (§2.5.7).

2.5.1 Experimental Setup

Baseline Systems

We use three RNN variations to create baseline systems: standard RNNs, Phased RNNs [Neil et al., 2016], and Skip RNNs [Campos et al., 2017]. The first baseline uses early-exiting

Table 2.1: Evaluation dataset characteristics.

Dataset	Seq Len	Classes	# Train	# Val	# Test
EMG [Lobov et al., 2018]	50	7	10,330	2,887	4,975
FordA [Bagnall et al., 2012]	20	2	3,060	541	1,320
Pavement [Souza, 2018]	30	3	33,336	5,815	39,939
Pedestrian [Melbourne, 2020a,b]	20	10	2,024	364	4,878
Pen Digits [Alimoglu and Alpaydin, 1996]	8	10	6,033	1,461	3,498
UCI HAR [Anguita et al., 2012]	50	12	18,229	6,800	10,197
Whale [Cox et al., 2006]	30	2	9,351	1,583	1,962

in standard RNNs [Dennis et al., 2018]. This model creates a prediction from each memory state, and we interpret these outputs as models in \mathcal{F}_{rnn} . We train the RNN to minimize the average loss across all predictions. The second baseline uses Phased RNNs [Neil et al., 2016]. Each model in \mathcal{F}_{phased} uses a phase gate with a different open rate. The final baseline uses Skip RNNs [Campos et al., 2017]. We train each Skip RNN in \mathcal{F}_{skip} to meet a different target number of elements. This target is enforced through an L2 loss term. For both the Phased and the Skip RNN systems, we create systems with 10 distinct models⁴. Using 10 models provides these baselines with a good tradeoff between energy and accuracy while maintaining a reasonable memory footprint.

For each baseline, we use a fixed selection policy π_{fixed} . This policy selects the best model whose estimated energy meets a given budget. The policy estimates energy consumption using both profiled energy values and the average number of processed inputs. For all models that meet the budget, π_{fixed} selects the model with the best validation accuracy.

Datasets and Neural Network Training

We evaluate system performance on the seven datasets in Table 2.1. Each dataset constitutes a sensor-like classification task. For all RNNs, we use single-layer UGRNN cells [Collins et al., 2016] with a 20 dimensional state. This design limits the memory footprint of each model.

4. We use 8 models on the Pen Digits dataset due to shorter sequences.

All models use a readout layer with 32 hidden units and a Leaky ReLU [Maas et al., 2013] activation. We fit the RNNs using stochastic gradient descent [Rumelhart et al., 1985] with an Adam optimizer [Kingma and Ba, 2014] and a learning rate of 10^{-4} . We use the same batch size for all models. We train the RNNs in Tensorflow [Abadi et al., 2016a] for at most 250 epochs with early stopping after 25 epochs.

Simulated Environment

We conduct the majority of this evaluation in a simulated environment that bases energy consumption on profiled values from a TI MSP430 FR5994 MCU [Instruments, 2021]. We measure the energy required to capture and process inputs using the EnergyTrace tool [Instruments, 2020]. We experiment with two energy profiles. The first uses a DHT-11 temperature sensor, and the second simulates collection through an HM-10 Bluetooth module. When sampled at 0.5Hz, the 3.3V MCU with the DHT-11 consumes about 5.6mJ per input; the HM-10 system uses about 29.6mJ per input. These values represent varying points on the spectrum of sensor energy consumption. The simulator also incorporates the energy required for data processing. This feature accounts for the higher computational cost of Budget RNNs (§2.5.8). For simplicity, we assume that both Phased and Skip RNNs incur the same processing cost as standard RNNs. This assumption is conservative as Phased and Skip RNNs perform additional computation.

Hardware Environment

We supplement the simulated environment with an experiment on the actual TI MSP430 FR5994 MCU [Instruments, 2021]. We use an HM-10 BLE module to transmit inputs to the device. We power the MCU using a supercapacitor and vary the budget by setting the total capacitance. The Budget RNN controller obtains energy feedback by measuring the voltage across the capacitor. We quantize the neural network weights to 16-bit fixed-point values

Table 2.2: Geometric mean accuracy across all budgets. RNN refers to the standard RNN.

Dataset	Bluetooth Energy Profile				Temperature Energy Profile			
	<i>RNN</i>	<i>Phased</i>	<i>Skip</i>	<i>Budget</i>	<i>RNN</i>	<i>Phased</i>	<i>Skip</i>	<i>Budget</i>
EMG	0.713	0.716	0.713	0.724	0.702	0.704	0.712	0.721
Ford A	0.881	0.859	0.876	0.886	0.867	0.845	0.860	0.868
Pavement	0.715	0.669	0.670	0.716	0.697	0.642	0.658	0.695
Pedestrian	0.607	0.702	0.714	0.720	0.543	0.665	0.649	0.674
Pen Digits	0.832	0.831	0.869	0.881	0.838	0.842	0.871	0.879
UCI HAR	0.862	0.828	0.859	0.865	0.848	0.820	0.856	0.856
Whale	0.864	0.869	0.871	0.879	0.849	0.859	0.865	0.871
All	0.776	0.778	0.791	0.806	0.753	0.761	0.773	0.788

and execute the RNNs using the on-board digital signal processing accelerator.

Budget RNN Parameters

In all experiments, we use a Budget RNN system with two distinct models. These models have stride lengths of 1 and 10 respectively⁵. The policy selects the Budget RNN at runtime as described in §2.4.4. We fit halting thresholds for 11 budgets on the Bluetooth profile and 12 budgets in the temperature setting. The Budget RNN system automatically generalizes to unseen budgets.

2.5.2 Inference Accuracy

For each dataset, we evaluate the inference accuracy on a set of energy budgets in the simulated environment. We set M to the size of each testing set and use energy values B to standardize the ratios B/M across datasets. We use 22 and 26 budgets on the Bluetooth and the temperature profiles respectively. We compare the systems by computing the geometric mean accuracy across all budgets.

On average, the Budget RNN achieves the best accuracy across all datasets and budgets.

5. We use stride lengths of 1 and 4 on the Pen Digits task due to shorter sequences.

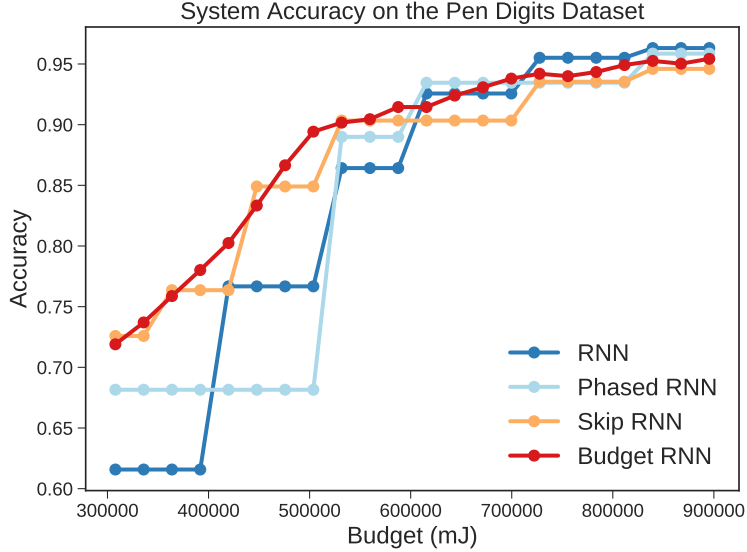


Figure 2.5: System accuracy on the Pen Digits dataset using the Bluetooth energy profile.

Table 2.2 shows these results for both energy profiles. When compared to the standard RNN, Budget RNNs display a mean accuracy gain of over 3 points. Further, Budget RNNs outperform Skip RNNs (1.5 points) and Phased RNNs (2.7 points). These results hold for both energy profiles. The table further shows how no single baseline delivers high accuracy across all datasets. In contrast, the Budget RNN system displays consistency—it almost matches or outperforms the *best* baseline system on all datasets. In a budget-by-budget comparison, the Budget RNN also shows distinct improvements. Using the Bluetooth energy profile, Budget RNNs have higher accuracy than standard RNNs on 77.3% (119 / 154) of budgets. The same comparison against Phased and Skip RNNs shows higher accuracy on 82.5% (127 / 154) and 83.1% (128 / 154) of budgets respectively. A similar trend holds on the temperature profile.

To better understand the observed accuracy differences, we compare the per-budget accuracy on the Pen Digits task. Figure 2.5 displays the obtained accuracy values. We observe how the Budget RNN provides distinct benefits under tight budgets. These gains become smaller as the budget increases. In particular, for large budgets, the Budget RNN shows slightly worse accuracy than the standard and Phased RNNs. Given no assumptions about

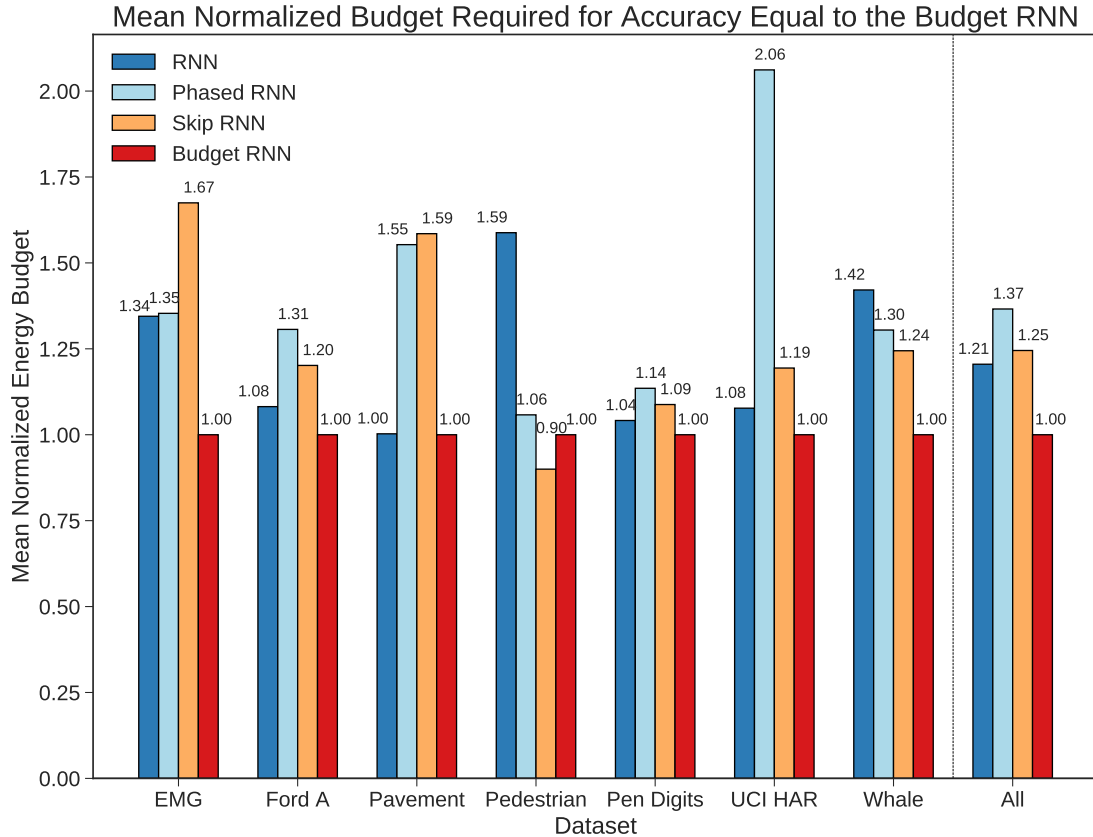


Figure 2.6: Geometric mean of normalized budget to obtain accuracy equal to the Budget RNN. Values above one indicate better performance by the Budget RNN.

the budget, however, Budget RNNs display the best overall result.

Better budget utilization appears to be a reason for the Budget RNN’s improved accuracy. On average, Budget RNNs use 99.2% of the budget, while RNNs (94.7%), Phased RNNs (78.1%), and Skip RNNs (80.4%) display inefficiencies. The Phased and Skip RNNs have lower utilization because the models sometimes show lower accuracy when given more inputs. This utilization pattern occurs on both the Bluetooth and temperature profiles, demonstrating how the advantages of Budget RNNs hold for multiple sensors.

Table 2.3: Geometric mean accuracy across all datasets and budgets in settings with a nonzero energy bias.

Bias	Bluetooth Energy Profile				Temperature Energy Profile			
	<i>RNN</i>	<i>Phased</i>	<i>Skip</i>	<i>Budget</i>	<i>RNN</i>	<i>Phased</i>	<i>Skip</i>	<i>Budget</i>
+20%	0.664	0.724	0.731	0.762	0.603	0.674	0.670	0.702
+10%	0.729	0.764	0.778	0.786	0.681	0.733	0.738	0.750
-10%	0.776	0.778	0.791	0.813	0.753	0.764	0.774	0.797
-20%	0.776	0.778	0.791	0.818	0.753	0.765	0.774	0.802
All	0.735	0.761	0.772	0.794	0.695	0.733	0.738	0.762

2.5.3 Energy Comparison

To place the higher accuracy of Budget RNNs into the context of energy, we estimate the budget needed by Budget RNNs to obtain accuracy equivalent to the baseline systems. For each baseline on budget RNN B , we find the smallest budget B' such that the Budget RNN displays the same accuracy on B' as that of the baseline on B . We use the ratio B/B' to compare the budgets needed to obtain the same accuracy. Figure 2.6 shows this comparison on the Bluetooth energy profile. On average, the Budget RNN system can use over 20% smaller budgets and still deliver comparable accuracy to the baseline systems. This value describes how the accuracy improvement (Table 2.2) manifests itself in terms of energy.

2.5.4 Accuracy on Unseen Energy Profiles

The Budget RNN system constructs halting thresholds using profiled energy values, and the runtime controller decouples the system from the profiled environment. We evaluate this decoupling by measuring how the system performs in a setting with biased noise. Specifically, in this setting, we model the energy consumption to collect and process r samples as $e(r) = \omega_r + \epsilon$ where ω_r is the profiled energy value and $\epsilon \sim \mathcal{N}(\mu, \sigma^2)$. The bias $\mu \neq 0$ is unknown to the systems, and each policy starts under the assumption of an unbiased environment. Positive biases mean that, at each step, the system consumes more energy than expected.

Table 2.4: Geometric mean accuracy of Budget RNN variants. The adaptive results correspond to the full Budget RNN system.

Dataset	Bluetooth Energy Profile			Temperature Energy Profile		
	<i>Fixed</i>	<i>Random</i>	<i>Adaptive</i>	<i>Fixed</i>	<i>Random</i>	<i>Adaptive</i>
EMG	0.722	0.718	0.724	0.720	0.718	0.721
Ford A	0.874	0.856	0.886	0.857	0.844	0.868
Pavement	0.706	0.689	0.716	0.684	0.673	0.695
Pedestrian	0.692	0.624	0.720	0.642	0.596	0.674
Pen Digits	0.843	0.827	0.881	0.843	0.836	0.879
UCI HAR	0.860	0.841	0.865	0.853	0.841	0.856
Whale	0.875	0.852	0.879	0.867	0.849	0.871
All	0.792	0.767	0.806	0.774	0.757	0.788

Negative biases work in the opposite fashion.

We evaluate the systems over four biases that represent about $\pm 10\%$ and $\pm 20\%$ of the profiled energy consumption. Table 2.3 shows the geometric mean accuracy across all budgets and datasets. In this setting, Budget RNNs show even higher accuracy relative to the baselines. Across all biases, Budget RNNs show a mean accuracy that is almost 6 points higher than standard RNNs. This improvement is roughly double what we observe in the unbiased setting. The same comparison against Phased RNNs (2.9 points) and Skip RNNs (2.2 points) also yields a larger improvement for Budget RNNs in the biased environment. These benefits are a direct result of adaptivity. The baseline systems use a fixed policy that does not account for differences in the runtime environment. In contrast, the Budget RNN uses an adaptive controller that automatically handles these differences using runtime feedback.

2.5.5 Evaluation of Budget RNN System Design

The Budget RNN system uses a novel RNN with a runtime controller. We evaluate how these components contribute to the overall accuracy by considering two variants. The first,

Table 2.5: Wall-to-wall time and training iterations required to fit the neural networks on each dataset. The Phased and Skip RNN results account for training ten distinct RNNs. The Budget RNN values include the halting threshold optimization.

Dataset	Wall-to-Wall Time (minutes)				Iterations (thousands)			
	<i>RNN</i>	<i>Phased</i>	<i>Skip</i>	<i>Budget</i>	<i>RNN</i>	<i>Phased</i>	<i>Skip</i>	<i>Budget</i>
EMG	40.3	557.1	464.9	176.9	81.4	803.6	724.8	288.9
FordA	9.4	90.7	92.5	35.5	45.6	293.4	320.9	137.7
Pavement	114.6	553.9	297.4	357.1	260.5	1139.9	664.8	552.6
Pedestrian	13.3	136.5	89.5	52.3	63.3	461.2	311.4	176.5
Pen Digits	24.0	233.1	109.6	92.0	188.8	1355.2	699.1	445.3
UCI HAR	124.0	1143.3	1039.0	346.1	251.7	1591.9	1559.8	512.1
Whale	40.0	393.0	500.1	119.1	112.1	734.0	988.3	272.2

called the fixed system, uses a fixed model selection policy (§2.5.1) in place of the adaptive controller. The second variant uses a randomized selection policy. This policy creates weights based on how often the Budget RNN should stop at each level to fully utilize the budget. At runtime, the policy uses the weights to randomly select the number of levels. In this experiment, we call the full system the “adaptive” Budget RNN.

We evaluate these variants across the same datasets and budgets used in §2.5.2. Table 2.4 shows the geometric mean accuracy for each dataset. The adaptive system outperforms the fixed and random variants on all datasets by a mean of 1.4 points and 3.1 points, respectively. These results yield two takeaways. First, data-dependent decisions are an integral part of the adaptive system’s performance. The randomized variant’s lower accuracy shows how fully utilizing the energy budget is insufficient; selecting when to conserve and when to exploit is key to obtaining accurate results. Second, the fixed variant alone displays an improvement over the standard RNN (compare “Fixed” in Table 2.4 to “RNN” in Table 2.2). This improvement shows the benefits of performing inference on non-contiguous subsequences. Adaptive behavior further exploits this feature to increase accuracy.

Table 2.6: Skip and Budget RNN results on the embedded device using the Pen Digits dataset.

Energy Budget	Skip RNN	Budget RNN
5.6J	0.722 (± 0.040)	0.833 (± 0.038)
7.2J	0.895 (± 0.038)	0.915 (± 0.022)

2.5.6 Training Time

One downside to the Phased and Skip RNN systems is their reliance on many distinct RNNs. We evaluate how this reliance impacts the training time of each system. We measure the training cost in two ways. The first is the wall-to-wall time required to train all models for each system. We obtain these times on a 24 core Intel Xeon Silver 4116 CPU. The second metric is the number of training iterations. This value is the number of gradient descent steps during training. The Budget RNN values also include the halting threshold optimization. As a conservative estimate, we count each threshold optimization step as equivalent to 10 gradient descent steps.

We measure the training cost on all seven datasets. Table 2.5 displays the results. Compared to Skip RNNs, the Budget RNN system takes roughly $2.3\times$ less time and fewer iterations to train. A comparison with the Phased RNN systems yields even greater savings. While reducing the number of models per system would lower this cost, such savings lead to lower accuracy. Only the standard RNN achieves a lower training cost than the Budget RNN. This property follows from the need to train only one neural network. As previously shown, the extra training cost of Budget RNNs leads to better accuracy values, and we believe this extra training cost is worth the accuracy benefits. We note that the Skip RNN system trains faster than the Budget RNN system on the pavement dataset. This result occurs due to early stopping during neural network training and a larger training set used to fit the halting thresholds. On this dataset, the benefits of the lower training time of Skip RNNs are offset by the system’s lower accuracy (Table 2.2).

2.5.7 Performance in the Hardware Environment

We supplement the results from the profiled environment with an evaluation in the embedded setting described in §2.5.1. We compare the accuracy of the Budget RNNs and Skip RNNs over two energy budgets on the Pen Digits dataset. We select Skip RNNs because they form the best-performing baseline on this task. As the device fetches inputs over a Bluetooth link, we use a Budget RNN system optimized on the Bluetooth energy profile. Each experiment uses the inference system to classify $M = 50$ sequences. We compare the resulting accuracy across four independent trials.

As shown in Table 2.6, the Budget RNN displays better accuracy than the Skip RNNs. In the simulated environment, the Budget RNN system achieves an accuracy of 79.1% and 90.3% on the full dataset for the two respective budgets. On these same budgets, the Skip RNN system obtains an accuracy of 75.6% and 89.4%. Thus, in the embedded environment, the gap between Budget RNNs and Skip RNNs is slightly larger than what is observed in simulation. This discrepancy is likely a result of the limited sample size used in the hardware evaluation. Despite this difference, the results from the hardware device confirm the relative accuracy trends we observe in the simulated environment.

2.5.8 Overhead Analysis

Budget RNNs are designed for embedded applications where sensing composes a significant portion of energy consumption. In these settings, the computational overhead of Budget RNNs is not prohibitive. To understand this point further, we use a TI MSP430 FR5994 MCU [Instruments, 2021] and the EnergyTrace Tool [Instruments, 2020] to profile the energy consumption of collecting and processing a single measurement. We compare the energy consumption of standard and Budget RNNs in Table 2.7. To handle a single element, Budget RNNs consume 0.5% (Bluetooth) and 2.7% (temperature) more energy than standard RNNs. This overhead is small enough to overcome and still deliver high accuracy; under the same

Table 2.7: Energy (mJ) required to handle a single sensor measurement. The standard and Budget RNN costs include the energy required for processing.

Sensor	Alone	+RNN	+Budget RNN
Bluetooth	29.63	29.97	30.13
Temperature	5.65	5.99	6.15

constraints, Budget RNNs show better accuracy than the baseline RNNs (§2.5.2). Budget RNNs use more computation to determine a much better sampling strategy. The relatively high cost of sensing makes this additional computation worthwhile.

The computational overhead of Budget RNNs comes from two areas. First, Budget RNNs use additional merging, halting, and pooling layers to control execution and combine intermediate states. Second, the Budget RNN system uses a runtime controller to adapt to energy constraints. On average, profiling shows that standard RNNs consume 0.342mJ to process a single element. Budget RNNs use an average of 0.503mJ. A majority of this overhead comes from the added neural network layers. The runtime controller accounts for less than 3.5% of the CPU cycles associated with Budget RNN execution. As the controller performs all operations in RAM, we expect it to consume a smaller fraction of the overall energy.

2.6 Related Work

2.6.1 *Adapting Neural Network Inference*

Many systems apply adaptation to improve neural network inference. Systems such as Shallow-Deep Networks [Kaya et al., 2019], BranchyNet [Teerapittayanon et al., 2016], ALERT [Wan et al., 2020b,a], and Idk Cascades [Wang et al., 2017] use early-exit points to control inference costs. Numerous systems use variable computation [Cheng et al., 2018, Lee and Nirjon, 2020, Yao et al., 2020] and per-layer adaptivity [Bateni and Liu, 2018, Bateni

et al., 2018] to execute neural networks under explicit constraints. Neural network systems for multi-tenant inference also use adaptive behavior to improve efficiency [Baek et al., 2020, Bateni and Liu, 2020, Casini et al., 2020, Fang et al., 2018]. Other work adapts system accuracy to save energy and latency [Farrell and Hoffmann, 2016, Hoffmann, 2014, 2015].

Other RNN designs adapt model execution. Adaptive Computation Time [Graves et al., 2013], Variable Computation Time [Jernite et al., 2016], and Clockwork RNNs [Koutnik et al., 2014] alter the amount of computation at each RNN timestep. Dilated RNNs [Chang et al., 2017], Shallow RNNs [Dennis et al., 2019], and Sliced RNNs [Yu and Liu, 2018] provide increased parallelism.

Budget RNNs are most similar to prior DNN systems that use variable computation. Budget RNNs vary inference energy costs by changing both computation and data collection. In contrast with the above prior work that only reduces computation, Budget RNNs lower energy costs in sensor environments where acquiring data is expensive.

2.6.2 Adaptive Sampling in RNNs

Multiple RNNs leverage adaptive sampling. Phased LSTMs use a periodic gate to perform state updates [Neil et al., 2016]. LSTM-Jump uses a policy gradient to train an RNN to skip inputs [Yu et al., 2017]. Skip RNNs skip inputs using a binary gate [Campos et al., 2017]. These models jointly optimize sampling behavior and RNN parameters. Unlike these designs, Budget RNNs are multi-capacity models that can change their target sampling level at runtime.

EMI-RNNs classify time-series inputs where the true signal makes up a small part of the sequence [Dennis et al., 2018]. This model performs early classification using thresholds on output probabilities. This design resembles a Budget RNN with a stride length of 1. These models, however, differ in their early-stopping criterion. Furthermore, EMI-RNNs use sliding windows during inference, so they still pay data acquisition costs.

2.6.3 Adaptive Sampling in Sensor Networks

Adaptive sampling is a common approach to conserve energy in sensor networks. Multiple systems use statistical models to control the sampling rate of individual sensors [Alippi et al., 2009a, 2007, Chatterjea and Havinga, 2008, Jain and Chang, 2004, Law et al., 2009, Zhou and De Roure, 2007]. BBQ [Deshpande et al., 2004], ASAP [Gedik et al., 2007], and Backcasting [Willett et al., 2004] selectively activate a subset of nodes based on sensor correlations. Similar to these systems, Budget RNNs also use adaptive sampling to reduce energy while maximizing sensor performance. Unlike this prior work, our system focuses on meeting explicit energy budgets when executing RNNs.

2.6.4 Early Time Series Classification

Systems for early time series classification perform inference on subsequences. ECTS uses a nearest neighbor algorithm on sequence prefixes and halts classification when the prediction becomes reliable [Xing et al., 2012]. Mori et al. use stopping rules to balance accuracy and earliness [Mori et al., 2017]. Similar to Budget RNNs, these systems control inference costs by processing subsequences. Budget RNNs, however, perform inference under energy budgets and do not optimize for earliness.

Rußwurm et al. perform early classification on RNNs by randomly sampling trainable halting probabilities [Rußwurm et al., 2019]. Budget RNNs also use halting signals to control model execution. Unlike this previous work, Budget RNNs use optimized halting thresholds to meet energy budgets without random behavior.

2.7 Conclusion

This thesis develops a novel RNN architecture—the Budget RNN—for in-sensor inference under energy budgets. The Budget RNN uses a leveled design in which each level processes

an input subsequence. By controlling the number of executed levels, the Budget RNN can alter its energy consumption. This control is made possible through trainable halting signals. We design a runtime controller that uses these signals to perform inference under an energy constraint by dynamically adjusting the model’s subsampling behavior. This design maintains sampling flexibility while also decoupling the sampling strategy from the RNN parameters. This loose-coupling allows the system to generalize to unseen budgets better than existing RNN solutions [Campos et al., 2017, Neil et al., 2016], achieving a mean accuracy that is 3 points higher than that of standard RNNs [Dennis et al., 2018]. This improvement allows Budget RNNs to obtain an accuracy that is equivalent to existing RNNs even when operating under 20% smaller budgets.

CHAPTER 3

PROTECTING ADAPTIVE SAMPLING FROM INFORMATION LEAKAGE ON LOW-POWER SENSORS

3.1 Introduction

Battery-powered low-power sensors often lack access to continuous power [Denby and Lucia, 2020, Vasisht et al., 2017, Zhang et al., 2004]. This property makes energy is a critical resource for sensing devices, and operators desire sensors with long lifetimes [Zhang et al., 2004]. Subsampling is one strategy to elongate sensor lifetime [Law et al., 2009]. Rather than capturing and communicating all measurements, individual nodes collect and transmit a subset of values; a statistical model is used to infer the skipped elements [Chatterjea and Havinga, 2008, Deshpande et al., 2004, Gedik et al., 2007]. This design uses the insight that both sensing and communication incur a high energy cost [Alippi et al., 2009b, Denby and Lucia, 2020, Gedik et al., 2007, Gobieski et al., 2019].

Subsampling trades energy for error. When sensors collect fewer elements, the server must infer more values. Adaptive sampling is one strategy that navigates this tradeoff in a near-optimal manner. For a given energy constraint, these algorithms sample frequently in unpredictable environments and compensate by collecting fewer values in predictable settings. For example, consider a smartwatch that performs activity detection with an accelerometer. When the wearer is sitting, the acceleration measurements remain constant; when a person is running, the values change rapidly [Das et al., 2016]. Thus, the policy samples infrequently during sitting events and spends the excess energy during less-predictable running events. The ability to allocate energy based on the observed data allows adaptive algorithms to exhibit low error while meeting energy constraints, making it an attractive option for low-power devices [Gedik et al., 2007, Willett et al., 2004].

As embedded sensing grows in popularity, maintaining data privacy is increasingly im-

portant [Antonakakis et al., 2017, Cherupalli et al., 2017, Trappe et al., 2015]. From this perspective, adaptive sampling faces a significant challenge [Stajano and Anderson, 1999]. By design, adaptive policies display different collection rates in different environments, thus linking their collection pattern to the captured data. This property introduces a possible side-channel: if sensors expose their collection rates within their (encrypted) communication patterns, they may leak information about the captured values. Indeed, previous work shows how attackers can link communication patterns to sensor values for specific IoT devices [Apthorpe et al., 2019, Bezawada et al., 2018, Das et al., 2016, Srinivasan et al., 2008].

Existing work [Apthorpe et al., 2019, Srinivasan et al., 2008] that exploits the communication patterns of adaptive behavior targets IoT devices that vary their transmission times. For example, the FATS attack [Srinivasan et al., 2008] uses the time between messages to uncover activities in a smart home. One suggested defense is to employ *periodic transmissions* [Srinivasan et al., 2008] which eliminate the variance in message times. This defense increases device latency by delaying transmissions and sending values in batches. This latency is manageable as low-power sensors already employ batching to conserve energy [Sung and Han, 2017, Zhang et al., 2004]; batched communication minimizes the time that radio modules spend in active mode [Mathur et al., 2009].

In this work, we demonstrate how periodic transmissions do not protect general-purpose adaptive sampling policies. By "general-purpose," we refer to policies at use widely-applicable statistical models (e.g. linear comparisons [Chatterjea and Havinga, 2008, Silva et al., 2017]) instead of narrow, task-specific knowledge. As adaptive policies vary their collection rates, they create batched messages with a size proportional to the number of collected values. This property holds even when sensors use encryption. Across multiple tasks, we show how three general-purpose adaptive techniques [Campos et al., 2017, Chatterjea and Havinga, 2008, Silva et al., 2017] exhibit a distinct relationship between message sizes and sensed events. Thus, message sizes yield an exploitable side-channel for adaptive policies. In the

worse case, an attacker can infer over 94% of events using encrypted message sizes. This worst-case leakage occurs on tasks involving sensitive data. For example, an attacker can use message sizes to infer the occurrence of an epileptic seizure [Villar et al., 2016] with 100% accuracy (§3.5.4). This issue prevents sensors that require data privacy from realizing the benefits of adaptive sampling.

A solution to close this side-channel is to send fixed-length messages, breaking the link between message size and collection rate. The conventional way to enforce this consistency is to pad messages to the largest possible batch size [Cai et al., 2014b, Dyer et al., 2012]. Padding hides the true data length by adding meaningless bits and increasing the communication volume. This strategy is impractical for low-power sensors due to the high energy cost of communication [Gedik et al., 2007, Gobieski et al., 2019]. Sensors require a defense with a negligible energy cost: if the strategy increases the energy consumption, it counteracts the benefits of adaptive sampling.

To address this problem, we propose a framework called Adaptive Group Encoding (AGE) that protects adaptive sampling under the constraints of low-power sensors. Unlike traffic padding, AGE produces fixed-length messages smaller than the standard adaptive policy’s average message size. AGE is a general defense that works with multiple adaptive policies, sensing tasks, and encryption algorithms. A key challenge to achieving this generality is that AGE cannot make strong assumptions about the policy’s behavior or the sensor’s data values. AGE ensures same-sized messages using a general lossy technique based on fixed-point quantization. The procedure minimizes its encoding error by supplementing standard quantization with three transformations. AGE first prunes measurements to handle cases where the policy exhibits extreme over-sampling. Second, AGE adapts to data ranges with low space overhead by applying run-length encoding (RLE) to data exponents. This step groups values with the same exponent. AGE handles cases where RLE delivers poor compression by merging similar exponent groups. Finally, AGE sets the bit width for each group

and quantizes measurements to this width. The algorithm selects these widths to ensure the result meets the target length. This group-based strategy allows AGE to functionally mimic fractional bit widths, leading to better utilization of the available message space.

Across nine sensing tasks and three adaptive policies, AGE incurs roughly 1% higher error than standard adaptive sampling. This additional error is manageable, as adaptive sampling with AGE obtains over 11% lower error than uniform sampling. AGE’s low error *does not* compromise security. AGE protects adaptive policies from all information leakage through message sizes, and communicating with AGE prevents an attacker from doing any better than predicting the most frequent event. Further, AGE’s low-overhead design outperforms prior solutions based on message padding. Under equivalent energy constraints, AGE obtains lower error than padding on over 94% of budgets. This result stems from budget violations caused by the overhead of padding. Finally, AGE consistently performs better than quantization alone, achieving lower error on over 98% of constraints. In summary, AGE successfully enables adaptive policies to obtain low error without suffering from information leakage or energy budget violations.

We make the following contributions in this work¹.

1. We demonstrate how general-purpose adaptive sampling policies leak information through their collection patterns on multiple distinct tasks. This leakage occurs even though these policies do not use task-specific designs.
2. We present a practical attack against adaptive sampling algorithms for sensors that use energy-efficient batched communication. In the worst case, this attack can use message lengths to infer over 94% of sensed events.
3. We introduce a novel framework called Adaptive Group Encoding (AGE). This system uses lossy encoding to protect any adaptive sampling algorithm from leaking informa-

1. This work appears as a conference paper in ASPLOS 2022 [Kannan and Hoffmann, 2022].

tion through batched message sizes on low-power devices. AGE closes this side-channel with negligible energy overhead².

3.2 Background

This work protects adaptive sampling algorithms on low-power devices from leaking information through their communication patterns. This section provides background on both low-power sensors (§3.2.1) and adaptive sampling methods (§3.2.2).

3.2.1 Low-Power Sensors and System Model

We consider a standard sensing model composed of battery-powered sensors and a centralized server [Gobieski et al., 2019, Vasisht et al., 2017]. Sensor nodes consist of a microcontroller unit (MCU) equipped with sensing hardware. Each device captures measurements and transmits the readings to the server over an encrypted wireless link. We consider sensors that communicate at regular intervals. This design matches a common approach in low-power sensing: batched communication. Batching data into a single communication event saves energy by both maximizing the time spent in sleep mode and amortizing the radio start-up costs over multiple measurements [Mathur et al., 2009, Sung and Han, 2017]. Periodic communication increases the latency for offloading readings, but this overhead is offset by the over $4\times$ lower energy when batching [Mathur et al., 2006]. Indeed, systems such as FarmBeats [Vasisht et al., 2017] and ZebraNet [Zhang et al., 2004] employ sensors with periodic, batched communication. Furthermore, this latency becomes less important when detecting events from sequences of measurements. In this framework, the server requires all sequential elements to perform inference; thus, transmitting batches does not hinder the event detection pipeline. As a concrete example, consider the task of activity recognition with wearable devices [Anguita et al., 2012]. Each battery-powered wearable captures accelerometer and

2. All code is available at <https://github.com/tejaskannan/adaptive-group-encoding>.

gyroscope measurements over time. The sensor transmits measurements to a server, which uses sequences of readings to detect activities such as running or walking.

The finite lifetime of battery-powered sensors complicates the process of recording and communicating measurements. This limited power makes energy conservation a priority, as frequent maintenance is costly and undesirable. Finite-capacity energy sources lead to long-term *energy budgets* where sensors must use their residual battery capacity to meet an operator-defined uptime. For example, ZebraNet sensors are designed to last for at least 72 hours on battery power alone [Zhang et al., 2004].

Sensors consume energy through three tasks: collection, processing, and communication. Of these tasks, collection and communication consume a majority of energy [Alippi et al., 2009b, Deshpande et al., 2004, Gedik et al., 2007]. For example, a commodity HM-10 Bluetooth Low Energy (BLE) radio consumes about 25mJ to connect and send a 40-byte message. In contrast, a TI MSP430 FR5994 MCU requires roughly 0.4mW per clock MHz [Instruments, 2021]; this figure is an order of magnitude less than wireless communication. The high energy cost of radio modules means that embedded applications strive to limit their use of wireless communication. On low-power devices, it is prohibitive to design systems that increase the amount of communication.

3.2.2 *Subsampling and Adaptive Behavior*

Subsampling allows sensors to manage their energy consumption [Alippi et al., 2009a, 2007, Chatterjea and Havinga, 2008, Silva et al., 2017] by collecting and sending a subset of values. The server receives this subset and interpolates the full sequence [Law et al., 2009]. A sampling policy’s energy is proportional to its collection rate. Thus, sampling comes with a tradeoff in error; when sensors capture fewer values, the server must infer more.

Adaptive sampling navigates this tradeoff in a near-optimal manner. These procedures use observed data to determine when to collect the next element. Adaptive policies will

capture more measurements on sequences with high variance and fewer samples during low-variance periods [Law et al., 2009]. For example, Figure 3.1 shows sampling two sequences of 25 acceleration values with a budget corresponding to a 70% average collection rate [Villar et al., 2016]. A random sampler will capture 17 samples per sequence. This rigid strategy is suboptimal, and an adaptive policy [Chatterjea and Havinga, 2008] can better allocate the budget. Across both sequences, the adaptive policy achieves over $2.9\times$ lower error by under-sampling sequence one and over-sampling sequence two. This lower error occurs even though the adaptive sampler captures two fewer values than the random policy.

Under energy constraints, this data-dependent behavior allows adaptive policies to achieve lower error than static sampling. The benefits, however, come at a cost in privacy. For the adaptive policy, collecting either 10 or 22 elements (from the top or bottom of Figure 3.1) leaks whether the event is walking or running, respectively. Thus, an attacker who observes the collection count can infer the sensed event. The random policy does not suffer from this problem because its rate is fixed and independent of the underlying data. From this discussion, we highlight two important aspects of the system model.

1. Long-term energy budgets allow for variance in the sampling policy. The sampler can change its energy per sequence as long as the final energy meets the budget.
2. Sensors use a single message to send a batch of collected measurements. The size of each message is proportional to the number of measurements collected by the policy.

3.3 Privacy Threats and Adaptive Sampling

This section describes the considered threat model (§3.3.1). We then show how adaptive policies leak information within this model (§3.3.2) and present examples of concrete systems (§3.3.3).

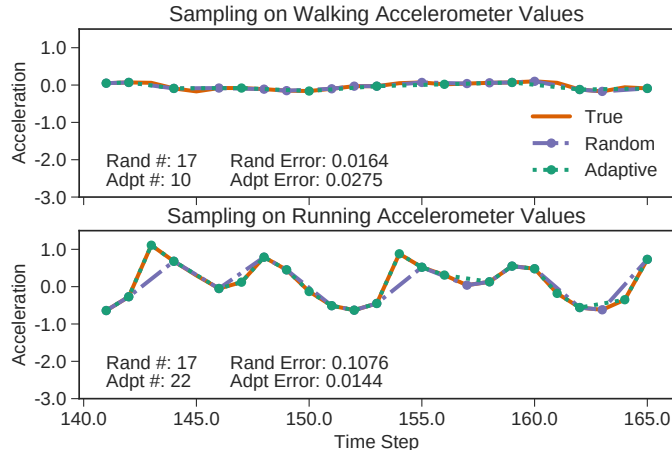


Figure 3.1: An example of subsampling two accelerometer signals on the task of human activity recognition.

3.3.1 Threat Model

Sensing applications often involve private information. For example, wearable medical sensors [Appelboom et al., 2014, Sorber et al., 2012] must collect measurements without exposing a patient’s diagnosis. Thus, sensors must communicate without leaking information about the captured values. Within the presented system model, we want sampling policies that minimize error and meet energy constraints, all without leaking information.

We consider a passive attacker that observes the wireless link between sensor and server using a network sniffing device. The sensor and server encrypt messages, and the attacker cannot learn anything from the ciphertext content. The adversary can instead leverage communication metadata such as message sizes and transmission times. Further, based on prior work, we assume the attacker can identify the sensor for each intercepted message [Apthorpe et al., 2019, Pang et al., 2007, Rasmussen and Capkun, 2007, Siby et al., 2017, Srinivasan et al., 2008]. We highlight that the adversary does not require physical device access. The attacker only needs to intercept the wireless communication between sensor and server. Due to the broadcast nature of wireless communication, this feature allows the adversary to operate anywhere within the range of the sensor’s radio.

The adversary uses this metadata to infer sensed events. As sensors communicate at regular intervals, message times provide no useful information [Srinivasan et al., 2008]. We instead consider attackers who infer events using message sizes (Figure 3.2). Using Kerckhoffs’ Principle, we assume the attacker knows the set of possible events and can link multiple messages to the same (unknown) event. This assumption is realistic as the physical systems being sensed exhibit gradual changes (relative to computing speeds). Further, the attacker knows the sampling policy, as well as any employed defenses. Finally, the attacker has an offline dataset to fit a model that predicts events using batched message lengths.

We highlight two details of this threat model. First, we assume sensors do not use lossless compression as such techniques are known to leak information through message lengths (§3.7). Second, we focus on settings where each batched message corresponds to exactly one event. This setup is ideal for an attacker, as they can attribute any length variance to a single event. Our defense (§3.4) extends to settings where batches contain multiple events.

This threat model presents a realistic attack vector against sensors in hard-to-access places. For example, consider sensors on satellites orbiting the earth [Denby and Lucia, 2020]. This setting deters two types of attacks. First, due to cost, an adversary cannot easily deploy devices to obtain equivalent measurements. Second, remote sensors make it difficult for an adversary to exploit side-channels that require physical device access [Messerges et al., 2002, Sehatbakhsh et al., 2019, Selvaraj et al., 2018]. Instead, the passive observation of message sizes represents a low-cost attack because the adversary can launch the attack remotely. It is thus critical to prevent adaptive sampling from leaking sensor information through its communication patterns. We emphasize that these other attack vectors against sensor networks still exist, and it remains necessary to secure both sensors and adaptive sampling from such techniques.

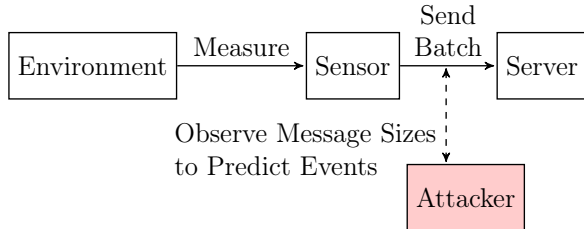


Figure 3.2: Diagram of the sensor system and threat model.

Table 3.1: Average (standard deviation) message size of adaptive policies [Chatterjea and Havinga, 2008, Silva et al., 2017, Campos et al., 2017] when conditioned on the event.

Event	Linear	Deviation	Skip RNN
Seizure	870.12 (± 241.83)	859.00 (± 286.50)	941.76 (± 233.13)
Walking	564.27 (± 67.50)	489.51 (± 42.14)	784.97 (± 165.11)
Running	1127.46 (± 65.85)	1200.92 (± 55.97)	877.41 (± 223.44)
Sawing	1021.80 (± 87.78)	1080.20 (± 98.85)	1235.20 (± 89.86)

3.3.2 Privacy and Adaptive Sampling

Using this threat model, we observe how adaptive sampling leaks information on low-power sensors. We execute three adaptive policies (§3.5.1) on the task of detecting epileptic seizures from a set of activities [Villar et al., 2016]. Each policy shows a distinct message size distribution for each event, and these differences are recognizable (Table 3.1). For example, if a sensor is using the Deviation policy [Silva et al., 2017], an attacker who observes a 450-byte message can infer the subject is walking. This issue occurs for all events and policies; for each policy, the pairwise differences between the conditional distributions are statistically significant under a Welch’s t-test ($\alpha = 0.01$). The scope of this problem indicates the issue is not with the specifics of a single policy. Instead, general-purpose adaptive sampling is *designed* to use context-dependent behavior, and this feature leads to an exploitable side-channel.

3.3.3 Example Systems

The considered system and threat models provide a general attack against adaptive sampling on low-power sensors. We discuss two systems that fit into these models.

Nanosatellites present a cost-effective solution for remote sensing [Frick and Niederstrasser, 2018, Poghosyan and Golkar, 2017, Puig-Suari et al., 2001, Toorian et al., 2008]. These satellites revolve around the earth, capture measurements, and transmit values to ground base stations. This communication is periodic because it occurs when the satellite is within range of a base station. These satellites operate using battery or intermittent power, making energy management a priority. Further, the recent Orbital Edge Computing [Denby and Lucia, 2020] proposal highlights the benefits of employing adaptive behavior by only transmitting relevant measurements. This design makes the batched payload size proportional to the number of "interesting" values. An adversary can passively monitor the long-range downlink and use the payload size to possibly uncover information about the sensed values. We emphasize that it is difficult and costly to launch an attack on satellites that uses physical device contact.

ZebraNet is a sensing system for wildlife monitoring [Zhang et al., 2004]. These sensors collect GPS and acceleration measurements from wild Zebras [Zhang and Martonosi, 2008]. With this information, researchers can track movements and infer activities [Naylor et al., 2009]. ZebraNet sensors transmit measurements every two hours to other nodes within a few kilometers. As sensors are attached to Zebras, device maintenance is costly, making energy management a key priority. Wildlife monitoring requires data privacy when tracking an endangered species. Although Zebras do not fit this category, systems such as TigerCENSE perform such monitoring [Bagree et al., 2010]. When tracking endangered wildlife, it is critical to protect the location of animals from poachers. The sensors should not leak information that can localize the animal to specific areas. In this setting, adversaries will not launch an attack that requires physical device access. If the attacker could access the device, they

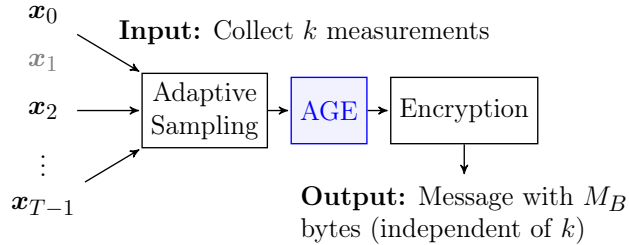


Figure 3.3: AGE works in between sampling and encryption, and it requires no changes to either step.

would also have access to the target animal. An adversary can instead launch an attack by intercepting wireless communication near a centralized base station.

3.4 Adaptive Group Encoding

Adaptive Group Encoding (AGE) is a novel framework to protect adaptive sampling policies from leaking information through the size of batched messages. AGE creates fixed-length messages for all collected batches, breaking the relationship between message size and collection rate. The framework operates as a drop-in step between sampling policy and encryption module, allowing AGE to work with any adaptive sampler, sensing task, and encryption algorithm (Figure 3.3). Achieving this generality prevents AGE from making strong assumptions about the sampling policy and the measurement values. AGE only requires that sensors capture numerical data. Within this setting, AGE must use lossy encoding; there is no lossless algorithm that can guarantee fixed-length messages [Mahoney, 2012]. Fortunately, this design aligns with existing sampling techniques; sampling is already lossy because it drops entire measurements. AGE aims to minimize the additional error required to prevent information leakage.

Central to AGE is fixed-point quantization, a cheap technique that can encode measurements into same-sized messages by properly setting the bit width for each value. Quantization alone, however, faces three issues that lead to high error. First, the sampler may capture

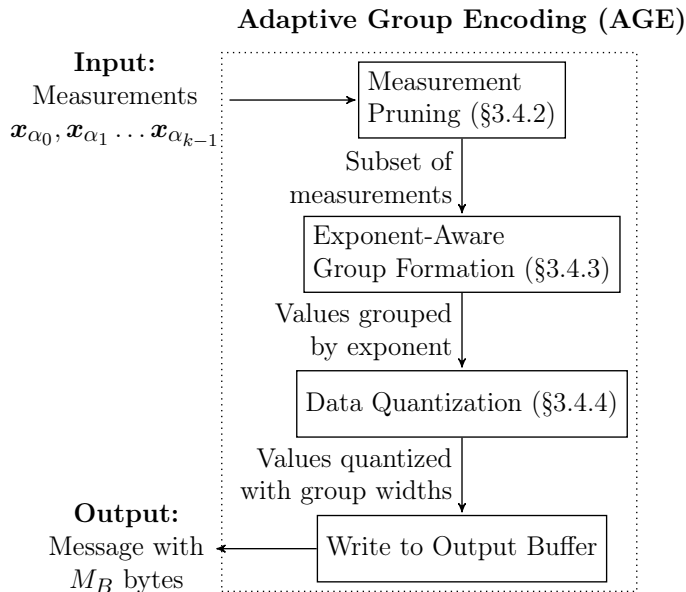


Figure 3.4: Overview of the data flow through AGE.

more values than available bits, forcing this encoder to drop all elements to meet the target size. Second, quantization uses a static number of non-fractional bits, causing unnecessary errors by not adapting to data ranges. Finally, quantization must round the selected width down to the nearest integer, resulting in excessive message padding.

AGE addresses these shortcomings by applying three transformations (Figure 3.4). First, AGE introduces a pruning step that removes measurements from batches with too many values (§3.4.2). Second, AGE adapts to the observed values by supporting dynamic ranges (§3.4.3). The framework applies run-length encoding (RLE) to compress value exponents. This RLE step groups adjacent values with the same exponent. AGE assigns bit widths for each exponent-aware group and quantizes the enclosed values accordingly (§3.4.4). Working with these smaller sets allows AGE to better utilize the available message bytes.

AGE protects adaptive sampling executing on MCUs under energy constraints. We create AGE to have a small energy cost, and AGE offsets any computational overhead by reducing the amount of wireless communication (§3.4.5).

Table 3.2: A selection of relevant notation.

Term	Description
T	The maximum number of elements per batch.
\mathbf{x}_t	The measurement at step t .
d	The number of features in each measurement.
B	The energy budget in joules.
k	The number of elements captured by the sampling policy.
M_B	The target message size for the budget B .
ρ_B	The average collection rate to meet the budget B .
α_t	The original index of the t^{th} collected sample.
w_0	The original number of bits per feature.
n_0	The original number of non-fractional bits per feature.

3.4.1 Notation

We introduce the relevant notation before describing the AGE framework. As MCUs work in fixed-point arithmetic, we consider measurements $\mathbf{x}_t \in \mathbb{R}^d$ encoded as fixed-point values. Each value has w_0 bits, n_0 of which are non-fractional places. The term n_0 logically corresponds to the value’s exponent, as the binary point is in the $(w_0 - n_0)^{\text{th}}$ place. We consider a general adaptive sampler operating under an energy budget of B joules; the policy meets this constraint using an average collection rate ρ_B . AGE knows B and, by extension, ρ_B . The sensor has a sampling period of Δt seconds and sends measurement batches every ΔT seconds (§3.2.1). The sensor sends at most $T = \Delta T / \Delta t$ measurements in every batch. We denote the target message size by M_B ; this term is the number of bytes required to encode $\lfloor \rho_B \cdot T \cdot d \rfloor$ values. The target size M_B is proportional to the budget B through the collection rate ρ_B . For brevity, M_B does not include the fixed amount of metadata required for communication. In practice, AGE handles this overhead by reducing the space available for data. Finally, k is the number of captured measurements in a single batch, and α_t is the index of the t^{th} collected element. Table 3.2 summarizes this notation.

3.4.2 Measurement Pruning

AGE ensures fixed-length messages for any periodic adaptive sampling policy. Hence, AGE cannot make strong assumptions about the policy’s collection patterns. Even under tight energy constraints, the base policy may capture all $k = T$ elements, yet AGE must still create a message with M_B bytes.

Fixed-point quantization alone leads to poor results for scenarios requiring a high compression ratio. For example, consider when $M_B = 35$, $k = 50$ and $d = 6$. In this setting, dedicating one bit per value yields a message with 38 bytes, eclipsing the target $M_B = 35$. For quantization to meet the target, it must drop all values.

AGE achieves better error during extreme over-sampling by taking a more moderate approach. AGE instead removes just enough measurements to ensure that all values get at least w_{min} bits. AGE performs this pruning by first computing the distance scores between consecutive measurements.

$$\text{Dist}(\mathbf{x}_{\alpha_t}) = \|\mathbf{x}_{\alpha_t} - \mathbf{x}_{\alpha_{t+1}}\|_1 + \frac{1}{8} \cdot |\alpha_t - \alpha_{t+1}| \quad (3.1)$$

AGE removes the ℓ (see below) measurements \mathbf{x}_{α_t} which have the smallest Dist . This metric estimates the error caused by dropping \mathbf{x}_{α_t} . By including the time difference, AGE avoids creating long stretches with no collected values. Long gaps often lead to high error because the system cannot detect signal changes in the interim [Law et al., 2009]. The factor of $\frac{1}{8}$ enables scaling using cheap bit shifting.

AGE sets ℓ as the largest positive integer such that $\lceil \frac{1}{8} \cdot w_{min} \cdot (k - \ell) \cdot d \rceil \leq M_B$. If no such $\ell > 0$ exists, AGE skips the pruning step. This setting ensures that all remaining values get represented with at least w_{min} bits. In our experiments, we set $w_{min} = 5$; a smaller minimum leads to higher error during quantization (§3.4.4). We note that incrementally updating the Dist scores yields an algorithm with lower error, but we find the overhead is

not worth the benefits.

3.4.3 Exponent-Aware Group Formation

When the adaptive policy over-samples, AGE must compress values into a message with M_B bytes. AGE performs this compression using quantization (§3.4.4). By itself, fixed-point quantization uses a static number of non-fractional places to limit the error for large values; this strategy causes unnecessary errors by not adapting to data ranges. For example, let $w_0 = 7$ and $n_0 = 5$, and consider quantizing to 3 bits. A static quantizer encodes 1.5 as 0 because the 0 and 4 are the two closest representable values in the 3-bit format. To avoid such errors, it is critical to enable dynamic ranges.

The challenge is that the optimal number of non-fractional bits is value-dependent. In the example above, 1.5 requires $n = 2$ non-fractional places in the 3-bit format while 0.25 needs $n = 1$. Providing each value with its own exponent takes considerable space, where these exponents store the number of non-fractional bits. AGE manages this tension by noting that adjacent sensor values often fall in similar ranges [Saidani et al., 2020]. Thus, AGE can compress exponents with run-length encoding (RLE). This process creates measurement groups consisting of adjacent values with the same exponent.

RLE alone provides no guarantees on its compression ratio. In the worst case, the encoded exponents can exceed M_B bytes. Thus, AGE must cap the maximum space for the exponents using a merging step. This process scores adjacent groups g_1 and g_2 with non-fractional places n_1 and n_2 as defined below.

$$\text{Score}(g_1, g_2) = \text{Count}(g_1) + \text{Count}(g_2) + 2 \cdot |n_1 - n_2| \quad (3.2)$$

AGE greedily merges groups with the lowest initial scores to produce at most G sets (see below). Upon merging, the new group uses $\max(n_1, n_2)$ non-fractional places to minimize

the error of large values. The factor of two balances the impact of exponent values and group counts, and we implement this scaling using cheap bit shifts. AGE skips this merging step when the initial number of groups is less than G . We note that an algorithm that updates scores after each merge yields a better approximation. The benefits of this approach, however, are not worth the overhead on an MCU.

The maximum number of groups G controls the tradeoff between exponent fidelity and space overhead. AGE sets this parameter by first finding the number of bytes m required if each value were encoded using the full w_0 bits. AGE dedicates the remaining $M_B - m$ bytes to exponents. That is, AGE sets G to the greatest number of groups whose metadata fits within $M_B - m$ bytes. We enforce that $G = \max(G, G_0)$ to limit the exponent approximation error when $k > \rho_B T$. By expanding the number of groups when possible, AGE reduces space wasted on padding when the policy under-samples. We use $G_0 = 6$ in our experiments, though we find that AGE’s performance is not sensitive across $G_0 = 4, 6, 8$.

3.4.4 Data Quantization

AGE leverages the exponent-aware groups to quantize values and control the message length. In particular, AGE sets the bit width of the values in each group such that the encoded result is at most M_B bytes. This group-based strategy addresses a limitation of standard fixed-point quantization. The standard encoder will use the width $w = \left\lfloor \frac{8 \cdot M_B}{k \cdot d} \right\rfloor$ to ensure a result of at most M_B bytes. This system must round the width down, leading to wasted space. For example, let $M_B = 220, k = 50$ and $d = 6$. These settings yield a width of $w = 5$, corresponding to 188 data bytes. This result means that over 14% of the message gets wasted padding up to $M_B = 220$ bytes.

AGE uses its exponent-aware groups (§3.4.3) to improve this utilization. The framework gives each group a bit width to use when quantizing its enclosed values. AGE sets these widths using a round-robin process to optimize the number of bytes used under the message

size M_B . This strategy works better than a uniform assignment because the groups enable adjustments in smaller multiples. For instance, using 5 groups of 10 measurements in the previous example, AGE will assign 5 bits to one group and 6 bits to the remaining four. This assignment creates a message with 218 data bytes; only 1% of the message gets wasted on padding. The per-group assignments allow AGE to functionally mimic fractional bit widths.

AGE’s per-group quantization requires encoding the widths into the message. This overhead is manageable because the widths are small and only need a few bits each—e.g., four bits when $w_0 = 16$. Further, the maximum number of groups G places a cap on the total number of widths. We note that AGE accounts for the space required to store exponents, group sizes, and bit widths when selecting the quantization parameters.

In summary, AGE quantizes values in the i^{th} group as fixed-point integers with w_i bits, n_i of which are non-fractional places (§3.4.3). AGE packs the quantized values, as well as the group metadata, into an output buffer. This design allows AGE to optimize its use of the available M_B bytes and preserve the benefits of dynamic ranges.

3.4.5 Discussion

AGE uses a multi-step process to minimize its encoding error. This algorithm is more expensive than a standard procedure which directly packs values into an output buffer (§3.5.8).

We address this overhead by saving energy on wireless communication. AGE produces messages with M_B bytes, and the framework can counteract any encoding overhead by reducing this target size. Intuitively, AGE spends a bit more energy on computation and saves significant energy on communication. In practice, we reduce the target length by about 30 bytes and include an additional 20-byte reduction for every 500-byte multiple in M_B . This conservative estimate allows AGE to display negligible energy overhead in deployment (§3.5.7).

We emphasize the generality of the presented system. In particular, AGE works for

both block and stream ciphers. For block ciphers, AGE uses the target size M_B rounded to the nearest block. In the case of a stream cipher, AGE uses the size M_B as given. The only knowledge AGE requires of the encryption algorithm is the size of associated metadata (e.g. nonces). These quantities allow AGE to determine the number of bytes available for measurements.

As a final point, AGE ensures that the sent message has M_B bytes absent any external faults. This guarantee is similar to those provided by systems with hard real-time constraints [Stankovic, 1988]. For example, if the network drops a packet, an attacker may observe a message of size $\tilde{M} < M_B$. AGE assumes that such faults occur independently of the sensed events. Thus, an attacker cannot use intermittent failures as a source of information leakage.

3.5 Evaluation

AGE protects low-power adaptive policies from leaking information through message sizes. We evaluate AGE by measuring its error, information leakage, and energy consumption. This evaluation demonstrates the following.

1. Under strict energy budgets, adaptive policies with AGE achieve lower error than non-adaptive sampling and defenses using message padding (§3.5.2).
2. General adaptive sampling policies leak information through messages sizes. By standardizing message lengths, AGE protects multiple adaptive policies from this leakage both in theory (§3.5.3) and in practice (§3.5.4).
3. AGE’s generality allows it to protect a variety of adaptive policies, including strategies based on trainable models such as neural networks (§3.5.5).
4. AGE uses multiple transformations to minimize its error. With this design, AGE obtains a lower error than variants using quantization and pruning alone (§3.5.6).

5. On a resource-constrained MCU, AGE retains the low error of adaptive sampling and eliminates all information leakage through message sizes (§3.5.7). These benefits come with negligible energy overhead (§3.5.8).

3.5.1 Setup

We evaluate AGE both in simulation and on a low-power MCU. We use nine sensor datasets (Table 3.3) that include both floating-point and integer measurements. Each sequence constitutes a batch, and the batches range from 98 to 3,138 bytes. We highlight two datasets for their relation to the examples in §3.3.3. The Activity [Anguita et al., 2012] task uses accelerometer values to classify human activities; these measurements are similar to those of ZebraNet [Zhang et al., 2004], albeit on humans instead of wildlife. The Tiselac [Ienco et al., 2017] dataset uses satellite image features to infer land cover types; this task relates to nanosatellites.

We execute each policy under energy constraints. Upon budget violation, the policy uses random values for the remaining sequences. The server receives a subsequence and infers missing elements using linear interpolation. We measure the mean absolute error (MAE) of this reconstruction. We use eight budgets per dataset corresponding to the energy of a Uniform sampler capturing 30%, 40%, . . . , 100% of elements. The simulator tracks energy using traces from a TI MSP430 FR5994 [Instruments, 2021]. We conservatively multiply AGE’s energy cost by $4\times$. The simulator results use a ChaCha20 stream cipher (IETF RFC 7539). For brevity, we omit the block cipher results. AGE provides equivalent protection with block ciphers, and we find AGE has a better relative error in this setting due to the extra bytes available from block padding the target size.

The hardware environment consists of a TI MSP430 FR5994 MCU [Instruments, 2021] with an HM-10 BLE module. The MCU performs sensing by reading measurements from FRAM, and the system processes one sequence every six seconds. We measure the energy

Table 3.3: Properties of the evaluation datasets.

Dataset	# Seq	Seq Len	# Feat	Labels	Bits (Frac)	Range
Activity [Anguita et al., 2012]	11,119	50	6	12	16 (13)	10.6
Characters [Williams et al., 2006]	1,436	100	3	20	16 (13)	7.8
EOG [Fang and Shinozaki, 2018]	362	1,250	1	12	20 (8)	2640.4
Epilepsy [Villar et al., 2016]	138	206	3	4	16 (13)	7.2
MNIST [LeCun et al., 1998]	10,000	784	1	10	9 (0)	255
Password [Bagnall, 2021]	308	1,092	1	5	16 (11)	18.8
Pavement [Souza, 2018]	8,864	120	1	3	16 (10)	68.4
Strawberry [Holland et al., 1998]	370	235	1	2	16 (13)	5.9
Tiselac [Inco et al., 2017]	17,973	23	10	9	16 (0)	3379

consumption of each policy using the EnergyTrace™ tool [Instruments, 2020]. We use an AES-128 block cipher [Dworkin et al., 2001] because the MCU has an AES accelerator.

For each adaptive policy, we compare AGE to two alternative approaches. The first is the standard adaptive policy with no post-processing step. The second baseline uses a message padding algorithm analogous to BuFLO [Dyer et al., 2012]. We use the minimum padding amount by assuming the defense knows the largest batch within the evaluation data.

We experiment with one non-adaptive policy (Uniform) and three adaptive sampling algorithms (Linear, Deviation, and Skip RNN). We also try a Random sampling baseline, but we omit these results because Uniform policy displays better error. We evaluate AGE on all adaptive policies.

Uniform The Uniform sampling algorithm collects k elements from each sequence where k is the maximum amount adhering to the budget B . This policy collects indices $t = r\lceil T/k \rceil$ for $i \in \{0, 1, \dots, k - 1\}$. When k does not divide T , we include additional random indices to ensure the policy collects k elements.

Linear The Linear adaptive policy [Chatterjea and Havinga, 2008] alters its collection rate based on the differences in consecutive measurements. When the absolute difference exceeds a threshold, the policy collects the next element. Otherwise, it increases its collection period by one. We use an offline training step to set a threshold for each budget.

Deviation The Deviation adaptive policy [Silva et al., 2017] uses a weighted moving average

Table 3.4: Arithmetic mean reconstruction error (MAE) across all budgets. The asterisk denotes the lowest error overall, and the boldface marks the lowest error for policies without information leakage. The final row is the median percent error higher than Uniform sampling (lower is better).

Dataset	Unif.	Linear			Deviation		
		Std	Padded	AGE	Std	Padded	AGE
Activity	0.0146	0.0090*	0.0565	0.0095	0.0099	0.0622	0.0104
Characters	0.0046	0.0046	0.0404	0.0046	0.0045*	0.0443	0.0046
EOG	0.1343	0.1251*	31.7824	0.1259	0.1301	37.7492	0.1321
Epilepsy	0.1090	0.0992*	0.2076	0.0997	0.0998	0.2295	0.1005
MNIST	5.0770	4.9231	6.5239	4.9397	4.6694*	7.1768	4.6958
Password	0.0073	0.0024*	0.1002	0.0024	0.0026	0.1063	0.0026
Pavement	0.7594	0.6477	0.7233	0.6886	0.6301*	0.7228	0.6786
Strawberry	0.0059	0.0049	0.0320	0.0050	0.0048*	0.0415	0.0049
Tiselac	2.7539	2.6547*	9.5832	2.6770	2.7762	9.5012	2.7934
Overall (%)	0.00	-15.84*	135.43	-13.41	-15.18	137.74	-11.34

to track the measurement variance. When the average variance exceeds the threshold, the policy doubles its collection rate. Otherwise, the policy halves its rate. We set the per-budget thresholds using an offline process.

Skip RNN The adaptive Skip RNN algorithm [Campos et al., 2017] uses offline data to train a recurrent neural network (RNN) which learns to sample. As Skip RNNs consume more energy than the other policies, we do not use this sampler under energy budgets. We instead use Skip RNNs to display the generality of AGE and its applicability to near-future adaptive sampling. This application shows how AGE can protect adaptive neural networks that support subsampling [Kannan and Hoffmann, 2021, Neil et al., 2016].

3.5.2 Reconstruction Error

Sampling policies aim to minimize error and meet energy constraints. Thus, AGE cannot have a high error cost. We evaluate this cost by measuring the adaptive policies’ error with AGE (Table 3.4). These results have three main takeaways.

First, AGE retains adaptive policies’ dominance over non-adaptive sampling. On all

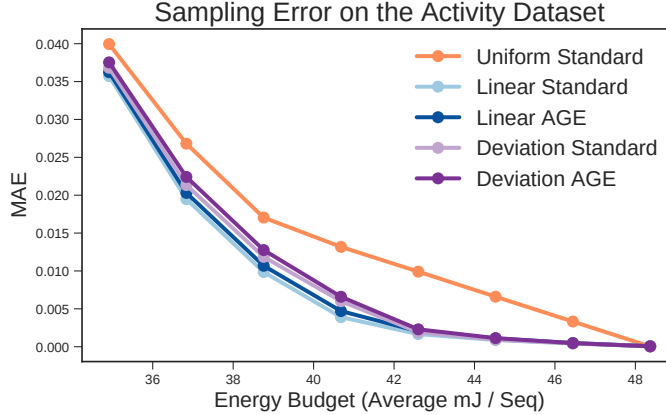


Figure 3.5: MAE for each budget on the Activity dataset.

tasks, the best adaptive policy with AGE has a lower error than Uniform sampling, and AGE shows roughly 13.4% (Linear) and 11.3% (Deviation) lower error overall. Further, AGE has better error than Uniform sampling on 73.6% (Linear) and 70.8% (Deviation) of constraints, and AGE makes adaptive sampling worse than the Uniform policy on only one additional budget. On some tasks, policies with AGE dominate Uniform sampling (Figure 3.5).

Second, unlike message padding, AGE meets the energy constraints of low-power devices. The Padded defense has a high error on all tasks due to budget violations caused by extra communication. In contrast, AGE never exceeds the budget because it decreases the amount of wireless communication to limit its energy overhead (§3.4.5). This design allows AGE to achieve equal or better error than Padded policies on 94.4% of budgets. Amongst the policies which leak no information (§3.5.3), AGE provides the best overall error.

Third, for adaptive policies, AGE does incur a cost in error. In particular, adaptive sampling with AGE displays a median of about 0.92% higher error than their standard counterparts. We expect this additional error based on the lossy approach employed by AGE; however, we believe this very small error is a small price for eliminating information leakage.

AGE requires the highest compression ratio when the underlying policy captures the

most elements. This phenomenon generally occurs on sequences with high variation. To ensure that AGE’s error is not prohibitive in these situations, we supplement the MAE values with a weighted error metric based on sequence deviation. Specifically, we weigh the MAE from each sequence by the standard deviation of its measurement values. Table 3.5 shows the weighted error values averaged across all budgets. AGE continues to consistently outperform Uniform sampling. AGE achieves a lower weighted error on every dataset, leading to a 15% lower median error than the Uniform policy. This aggregate result eclipses the unweighted value (Table 3.4) due to the better performance of adaptive sampling on high-deviation sequences. Furthermore, on all datasets but one, AGE is the best-performing policy that protects against information leakage. On the Pavement task, the Padded variant achieves a lower error than AGE. This result occurs because the sequences after the budget violation exhibit low deviation. Thus, the sequences that require random guessing get less weight, leading to a lower aggregate error value. This result is a product of the arbitrary dataset ordering and is not a fundamental benefit of the Padding strategy. The far superior performance of AGE on the other eight datasets shows the benefits of the proposed approach. In general, the low weighted error shows how AGE outperforms baseline approaches when focusing on scenarios that require the highest compression from the lossy encoding routine.

3.5.3 *Information Leakage: Theoretical*

AGE protects adaptive sampling from leaking information through message lengths. We demonstrate this ability from an information-theoretic perspective (§3.5.4 explores practical implications). For each budget, we estimate the normalized mutual information (NMI) between the event L and the message size M (see below) [Kvålseth, 2017]. This metric represents the reduction in uncertainty about the event after observing the message size [Chatzikokolakis et al., 2010]. The terms $\hat{I}(L, M)$, $\hat{H}(L)$, and $\hat{H}(M)$ are the maximum

Table 3.5: Arithmetic mean weighted reconstruction error across all budgets. The asterisk marks the lowest overall error, and the boldface shows the lowest error amongst policies with no leakage.

Dataset	Unif.	Linear			Deviation		
		<i>Std</i>	<i>Padded</i>	<i>AGE</i>	<i>Std</i>	<i>Padded</i>	<i>AGE</i>
Activity	0.0274	0.0134*	0.0652	0.0144	0.0147	0.0756	0.0159
Characters	0.0049	0.0047	0.0343	0.0048	0.0046*	0.0376	0.0047
EOG	0.1382	0.1205*	40.4317	0.1274	0.1243	46.3721	0.1392
Epilepsy	0.1394	0.1067	0.1993	0.1073	0.1027*	0.2299	0.1039
MNIST	5.1211	5.0021	6.5698	5.0206	4.7047*	7.1624	4.7344
Password	0.0073	0.0024*	0.1002	0.0024	0.0026	0.1063	0.0026
Pavement	0.8919	0.6835	0.7185	0.7451	0.6261*	0.6693	0.7019
Strawberry	0.0059	0.0049	0.0320	0.0050	0.0048*	0.0415	0.0049
Tiselac	4.6389	4.5302*	12.1781	4.5784	4.8623	12.2598	4.9076
Overall (%)	0.00	-16.95	162.52	-15.25	-18.64*	175.91	-16.95

likelihood estimators for the mutual information and (Shannon) entropy.

$$\text{NMI}(L, M) = \frac{2 \cdot \hat{I}(L, M)}{\hat{H}(L) + \hat{H}(M)} \quad (3.3)$$

A policy with no leakage should have zero mutual information, implying that the message size is independent of the event.

The Uniform policy has zero NMI because the policy captures the same number of elements for all events. In contrast, *both* standard adaptive policies display nonzero NMI across all tasks (Table 3.6), indicating that the size of messages from adaptive policies leaks information about event labels. We estimate the significance of this leakage using an approximate permutation test [Good, 2013]. This test randomly shuffles the message lengths and recomputes the NMI. With this design, the null hypothesis is that random variation in the lengths leads to the nonzero NMI instead of any true dependence between sizes and events. We use 15,000 permutations for each test, creating a worst-case 95% confidence interval of $\hat{p} \pm 1.96 \cdot \frac{1}{2\sqrt{15000}}$ where \hat{p} is the estimated p -value [Ojala and Garriga, 2010]. With this methodology, we find that the adaptive policies have an entire 95% p -value confidence

Table 3.6: Median / maximum empirical normalized mutual information between message size and event label. Padded and AGE have the same median and maximum values.

Dataset	Linear			Deviation		
	<i>Std</i>	<i>Padded</i>	<i>AGE</i>	<i>Std</i>	<i>Padded</i>	<i>AGE</i>
Activity	0.34 / 0.35	0.00	0.00	0.33 / 0.40	0.00	0.00
Characters	0.24 / 0.25	0.00	0.00	0.24 / 0.26	0.00	0.00
EOG	0.20 / 0.30	0.00	0.00	0.19 / 0.31	0.00	0.00
Epilepsy	0.41 / 0.44	0.00	0.00	0.39 / 0.47	0.00	0.00
MNIST	0.13 / 0.13	0.00	0.00	0.14 / 0.27	0.00	0.00
Password	0.07 / 0.10	0.00	0.00	0.09 / 0.12	0.00	0.00
Pavement	0.13 / 0.13	0.00	0.00	0.13 / 0.15	0.00	0.00
Strawberry	0.05 / 0.09	0.00	0.00	0.06 / 0.09	0.00	0.00
Tiselac	0.07 / 0.12	0.00	0.00	0.11 / 0.20	0.00	0.00

interval less than 0.01 on 83.3% (Linear) and 81.9% (Deviation) of evaluated budgets. This result shows that, with a high likelihood, the observed NMI occurs due to a link between message lengths and event labels. Thus, the observed NMI corresponds to a meaningful amount of leakage. We emphasize that this leakage occurs for both policies and across all tasks, showing the scope of the privacy issue.

AGE successfully eliminates the information leakage of adaptive sampling. As a result of same-sized messages, adaptive policies with AGE show zero NMI between message length and event label. Thus, AGE ensures that the message size is independent of the event. This result holds despite the base adaptive policy using data-dependent collection rates. Further, this protection applies to both adaptive policies across all tasks, supporting the generality of this defense. We note that Padding provides equivalent security, but it does so at a much higher energy cost (§3.5.7).

3.5.4 Information Leakage: Practical

The NMI results provide a strong indication that AGE protects standard adaptive policies from leaking information through message lengths. We supplement this theoretical analysis by presenting a practical attack. We consider an adversary who uses message lengths of

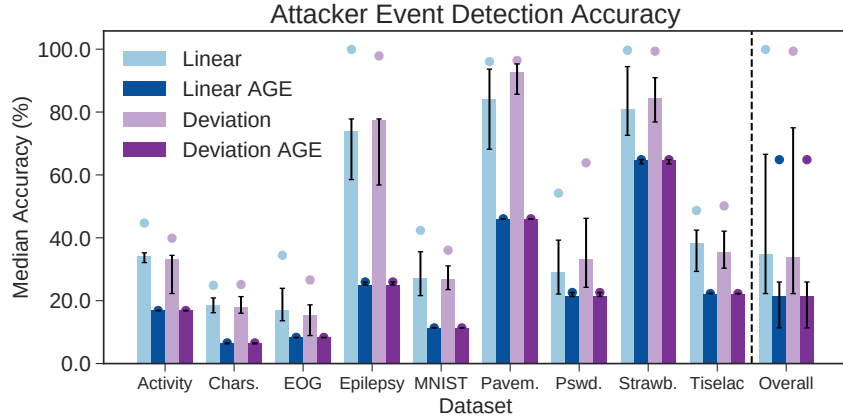


Figure 3.6: Median attacker accuracy across all energy budgets. The error bars denote the first and third quartiles, and the points show the maximum accuracy.

Tr \ Pr	Seizure	Other	Tr \ Pr	Seizure	Other
<i>Seizure</i>	500	0	<i>Seizure</i>	0	500
<i>Other</i>	0	1500	<i>Other</i>	0	1500

(a) Linear
(b) Linear with AGE

Figure 3.7: Confusion matrices for the attack model under a single budget. The Linear policy yields 100% accuracy on detecting seizures. AGE forces all predictions into one event.

ten random sequences of the same event (§3.3.1). The input features include the average, median, standard deviation, and IQR of the message sizes. We use an ensemble of 50 decision trees fit with AdaBoost [Freund et al., 1999, Schapire, 2013] and measure the test accuracy using stratified five-fold cross-validation. The training and testing sets contain 8,000 and 2,000 samples, respectively. Policies with no leakage should have a test accuracy equivalent to the most frequent event. As this attack uses one methodology, the presented approach is a lower bound for what an adversary may uncover.

With this attack, the adversary can infer events from standard adaptive policies (Figure 3.6). The adaptive policies have an overall median attack accuracy that is $1.58\times$ higher than the most frequent event. In the worst case, the policies allow the attacker to infer over 94% of the event labels on the Epilepsy, Pavement, and Strawberry datasets. This result occurs for both the Linear and the Deviation policies, confirming the privacy problem associated

Table 3.7: Average MAE, maximum NMI, and maximum attack accuracy when sampling using Skip RNNs.

Dataset	MAE		NMI		Attack (%)	
	<i>Skip RNN</i>	<i>AGE</i>	<i>Skip RNN</i>	<i>AGE</i>	<i>Skip RNN</i>	<i>AGE</i>
Activity	0.0081	0.0087	0.40	0.00	59.26	16.87
Characters	0.0043	0.0044	0.34	0.00	26.60	6.59
EOG	0.1434	0.1439	0.39	0.00	35.15	8.33
Epilepsy	0.0892	0.0896	0.31	0.00	92.80	25.95
MNIST	0.9192	1.0207	0.13	0.00	44.70	11.35
Password	0.0073	0.0073	0.08	0.00	43.82	22.60
Pavement	0.5845	0.6091	0.20	0.00	96.45	46.05
Strawberry	0.0040	0.0040	0.09	0.00	97.70	64.85
Tiselac	2.0832	2.2810	0.16	0.00	54.73	22.26

with general-purpose adaptive sampling. We emphasize that this worse-case leakage occurs for sensitive events. For example, the attack model against the Linear policy can obtain 100% precision and recall on classifying the occurrence of an epileptic seizure (Figure 3.7a).

AGE fixes this issue for both adaptive policies. Due to fixed-length messages, both policies with AGE display an attack accuracy equivalent to the most frequent label, the best that is practically achievable. We highlight this behavior on the previous scenario for adaptive sampling (Figure 3.7). AGE’s same-sized messages force all predictions into a single event, thereby fully protecting adaptive sampling even in worst-case situations (Figure 3.7b). Overall, AGE eliminates the information leakage problem from adaptive policies, meets energy requirements, and retains a low error (Table 3.4).

3.5.5 Evaluation on Skip RNNs

AGE works as a post-processing step to adaptive sampling, making AGE compatible with any policy. We display this generality by applying AGE to Skip RNNs [Campos et al., 2017]. We evaluate these models when collecting 30%, 40%, . . . , 100% of elements.

As shown in Table 3.7, AGE incurs roughly 1.60% greater error than standard Skip RNNs. The Skip RNN policy, however, leaks information on each task. With a permutation

Table 3.8: Median percent error greater than AGE across all budgets and tasks. Higher values indicate higher error.

Variants	Linear	Deviation
Single	2.664%	2.870%
Unshifted	2.068%	2.548%
Pruned	58.882%	57.278%
AGE	0.000%	0.000%

test, the observed nonzero NMI values are significant on over 72.2% of rates. Furthermore, this leakage translates into a practical attack with a worst-case accuracy of over 95%. This result further supports that general-purpose adaptive sampling algorithms leak information on multiple tasks. AGE successfully closes this side-channel. Skip RNNs with AGE show zero NMI between message size and event, and the attack accuracy gets reduced to the most frequent label. The successful application of AGE to a diverse set of policies shows the framework’s generality and small performance overhead.

3.5.6 Variants of Adaptive Group Encoding

AGE performs encoding by compressing exponents and quantizing values. We evaluate these transformations using three variants. The Single variant uses fixed-point quantization with a single bit-width. The Unshifted variant uses six even-sized groups but fixes the exponent for all values. The Pruned variant removes measurements to control the message size (§3.4.2). All three variants create fixed-length messages, so we focus this comparison on sampling error. Across all tasks, AGE displays lower error than these baselines (Table 3.8). Further, AGE shows an equal or better error than all variants on over 98% of budgets. These results show how all AGE’s features are necessary to achieve low error.

We highlight an important aspect of these results: quantization alone does not retain adaptive sampling’s dominance over Uniform sampling. The Single and Unshifted policies incur high errors on the Tiselac task, showing MAEs of 6.245 (Single) and 8.998 (Unshifted) with the Linear policy. This error exceeds that of Uniform sampling by over $2.2\times$ (Table

Table 3.9: Average energy per sequence (mJ) over 75 sequences when on a TI MSP430 MCU under three energy budgets.

Policy	Activity			Tiselac		
	2.837J	3.285J	3.634J	2.606J	2.929J	3.288J
Uniform	37.83	43.81	48.46	34.75	39.05	43.85
Linear	37.22	42.86	48.18	34.84	38.93	43.75
Padded	45.35	48.21	48.22	37.38	43.87	43.67
AGE	36.37	42.62	47.09	34.46	37.59	43.63
Deviation	37.30	42.68	48.17	35.01	39.18	43.75
Padded	48.36	48.11	48.60	37.28	43.50	43.70
AGE	36.31	42.01	47.00	34.52	37.90	43.61

Table 3.10: MAE over 75 sequences on a TI MSP430 MCU under three energy budgets.

Policy	Activity			Tiselac		
	2.837J	3.285J	3.634J	2.606J	2.929J	3.288J
Uniform	0.0280	0.0103	0.0000	3.4848	1.5383	0.0000
Linear	0.0203	0.0016	0.0000	4.0615	0.9974	0.0000
Padded	0.0720	0.0306	0.0000	15.0244	19.2171	0.0000
AGE	0.0223	0.0023	0.0001	4.0615	1.0005	0.0000
Deviation	0.0231	0.0019	0.0001	4.0046	1.1654	0.0000
Padded	0.0916	0.0309	0.0001	14.9644	17.3346	0.0000
AGE	0.0254	0.0027	0.0002	4.0052	1.1709	0.0000

3.4). In contrast, AGE outperforms Uniform sampling on this task. Unlike encoding using fixed-point quantization alone, AGE allows adaptive policies to dominate Uniform sampling.

3.5.7 Performance on a Microcontroller

In simulation, AGE incurs minimal error, ensures fixed-length messages, and satisfies energy constraints. We validate these results on a TI MSP430 FR5994 [Instruments, 2021] with an HM-10 BLE radio. We run each policy over 75 sequences on the Activity and Tiselac tasks. We set three budgets using Uniform policy’s energy when collecting 40%, 70%, and 100% of elements. We enforce budget violations when the energy per sequence is significantly higher than Uniform sampling. We assess significance using a one-sided Welch’s t-test ($\alpha = 0.05$).

Across all budgets, AGE shows a lower average energy per sequence than both Uniform sampling and standard adaptive policies (Table 3.9). These results confirm the negligible energy overhead associated with AGE. This performance contrasts with the Padded policies; AGE requires a median of 9.70% (Linear) and 10.04% (Deviation) less energy per sequence than Padding. This minimal overhead comes at little cost in error (Table 3.10). On the MCU, AGE shows the same error under these three constraints as we observe in simulation. AGE shows a far higher error than Uniform sampling on only one budget (Tiselac at 2.606J), and this result comes from higher error by the base adaptive policies. The Padded variants show high errors due to budget violations.

Standard adaptive sampling continues to show batch size variance. On the Activity task, adaptive sampling has a median of 0.088 (Linear) and 0.101 (Deviation) NMI between message size and event label. AGE instead shows zero NMI by always sending fixed-length messages. These results show how AGE successfully protects adaptive policies and meets the constraints of low-power sensors.

3.5.8 *Overhead Analysis*

AGE’s multi-step encoding leads to computational overhead. We analyze how this computation translates to energy on a TI MSP430 FR5994 MCU [Instruments, 2021]. When encoding a full sequence from the Activity dataset, AGE consumes roughly 0.154mJ. This figure compares to the 0.016mJ needed for a standard process that writes values directly into an output buffer. To offset this cost, AGE reduces the amount of communication by roughly 30 bytes per batch (§3.4.5). For an HM-10 BLE radio, this reduction saves about 0.9mJ, far eclipsing the energy required for encoding. AGE further protects against any worse-case scaling by reducing the communication by an additional 20 bytes every 250 measurement values. With this design, AGE trades some computational overhead for greatly reduced communication, leading to negligible total energy overhead in practice (§3.5.7)

3.6 Related Work

Security in IoT and Cyber-Physical Systems

Attacks such as the Mirai botnet [Antonakakis et al., 2017] highlight the importance of low-power device security [Stajano and Anderson, 1999, Trappe et al., 2015]. Previous systems examine the security of smart homes [Acar et al., 2018, Mohajeri Moghaddam et al., 2019] and smart cities [Giuliani et al., 2021, Leu et al., 2018]. Additional work fingerprints wireless devices [Bezawada et al., 2018, Bratus et al., 2008, Formby et al., 2016, Gao et al., 2010, Pang et al., 2007, Rasmussen and Capkun, 2007, Siby et al., 2017]. AGE builds on this work by protecting against attackers who can fingerprint sensors.

Multiple attacks extract information through device side-channels such as electromagnetic waves [Kocher et al., 2011, Messerges et al., 2002, Sehatbakhsh et al., 2019, Selvaraj et al., 2018] and cache timing [Takarabt et al., 2019]. AGE also addresses a side-channel of low-power devices, but our work focuses on message lengths. The MoLe attack uses a smartwatch accelerometer to infer typed words [Wang et al., 2015]. This attack highlights the need for systems such as AGE which protect sensor measurements. RCAD protects sensor networks from leaking measurement times [Kamat et al., 2007]. AGE also addresses data leakage on low-power sensors, but it instead focuses on message sizes rather than times.

Both FATS and STP show how attackers can use transmission times and device fingerprints to infer activities in smart homes [Apthorpe et al., 2019, Srinivasan et al., 2008]. Along with AGE, these systems study communication side-channels. These approaches, however, target smart home devices that vary their transmission times. AGE instead protects low-power sensors that batch communication at regular intervals. Further, STP increases network traffic; this strategy is too costly for low-power devices. Finally, Das et al. use the communication volume from specific fitness trackers to infer activities [Das et al., 2016]. Our work builds on this study by displaying how general-purpose adaptive policies leak in-

formation on multiple tasks. Further, we propose a novel framework to protect sensors in an energy-efficient manner.

Website Fingerprinting

Website fingerprinting involves extracting information from encrypted network traffic [Cai et al., 2014a,b, 2012, Lu et al., 2010]. Previous work shows how packet sizes leak information about visited webpages [Bhat et al., 2019, Herrmann et al., 2009, Liberatore and Levine, 2006, Panchenko et al., 2016]. Dyer et al. show how coarse features are sufficient for fingerprinting [Dyer et al., 2012]. Wright et al. shape traffic to standardize website patterns [Wright et al., 2009]. CRIME [Duong and Rizzo, 2012] and BREACH [Gluck et al., 2013] use compressed sizes as a side-channel against TLS. Similar to these systems, we focus on a network traffic side-channel. Our work, however, concerns sensors with energy constraints; any bandwidth overhead leads to an untenable energy cost. We develop a defense that uses the numerical properties of sensor measurements to standardize communication with a negligible energy overhead.

Low-Power Compression

Data compression is a common method to reduce the communication energy on low-power sensing devices. Mamaghanian et al. develop a compression algorithm to extend the lifetime of wearable body sensors [Mamaghanian et al., 2011]. Azar et al. apply lossy compression to IoT sensors and recover values using neural networks [Azar et al., 2019]. Other approaches use delta encoding [Saidani et al., 2020], Chebyshev polynomials [Ukil et al., 2015] and Huffman coding [Marcelloni and Vecchio, 2008]. Similar to these systems, AGE can use lossy encoding to compress sensor messages. In contrast, our system creates fixed-length messages to avoid information leakage. Unlike standard compression, AGE will expand messages *on purpose* to meet the target size.

Adaptive Sampling on Low-Power Devices

Adaptive sampling is a popular method for reducing energy consumption with minimal loss in error. Existing systems use adaptive procedures based on linear or autoregressive features [Alippi et al., 2009a, 2007, Chatterjea and Havinga, 2008, Law et al., 2009, Lou et al., 2020, Silva et al., 2017]. Further systems adapt the data collection pattern using Gaussian Processes [Tan et al., 2018], Reinforcement Learning [Dias et al., 2016, Murad et al., 2020], and Neural Networks [Campos et al., 2017]. We show how three different adaptive policies leak information on multiple tasks. AGE protects these systems with minimal overhead.

3.7 Discussion and Conclusion

Adaptive Group Encoding (AGE) enforces data privacy by creating fixed-length messages using a lossy technique. We discuss the limitations of the proposed framework, as well as how AGE compares to alternative defenses.

Batch Sizes

AGE offsets its slight computational overhead by reducing communication (§3.4.5). This strategy struggles when the target size is small. For example, if M_B is 40 bytes, lowering the target by 30 bytes means AGE loses 75% of the message. When batches are small, padding provides a better defense as its overhead becomes minimal. In our evaluation, AGE uses at least 98 bytes, and we consider AGE superior to padding when the batches have at least 100 bytes.

AGE and Energy Savings

AGE creates fixed-length messages that meet a target size M_B . This target size does not need to be derived from an energy constraint. Instead, AGE can work with any feasible

target message size, and we use this property in the AGE framework to offset the encoding procedure’s computational overhead (§3.4.5). Thus, similar to standard compression techniques, AGE supports the ability to save energy through reduced wireless communication. This feature allows AGE to display energy savings beyond those of the underlying adaptive sampling policy.

Lossless Compression

AGE’s creation of fixed-length messages makes it incompatible with lossless compression; applying compression after AGE will alter the final message size. Compression, however, is known to leak information about plaintext content [Duong and Rizzo, 2012, Gluck et al., 2013, Kelsey, 2002]. Thus, even sensors using compression with non-adaptive sampling will suffer from data privacy issues. We leave solutions to compression’s information leakage for future work.

Alternative Defenses

We discuss two suboptimal alternative defenses. First, on-device inference [Gobieski et al., 2019] eliminates the communication side-channel by never transmitting raw values. However, it is difficult to understand and debug inference results without raw data; sending measurements provides flexibility during analysis. Additionally, real-world systems [Vasisht et al., 2017] process data at centralized servers. These applications should not suffer from data leakage.

A second alternative is to create same-sized messages by buffering excess values. This approach, however, increases the reporting latency, and this overhead worsens when the policy over-samples on consecutive batches. Further, this strategy needs enough memory—a limited resource on low-power sensors—to buffer excess values. Over-sampling for many consecutive batches would force the system to drop samples, producing a high error similar

to the Pruned variant in §3.5.6.

AGE is an efficient approach to protect adaptive sampling from leaking information through batched communication. The framework emphasizes having a low overhead in terms of both error and energy. Along with AGE’s modular design, these properties make the framework compatible with existing sensor applications. We believe AGE represents a valuable step in bringing low-overhead security measures to resource-constrained sensing devices.

CHAPTER 4

PREDICTION PRIVACY IN DISTRIBUTED MULTI-EXIT NEURAL NETWORKS: VULNERABILITIES AND SOLUTIONS

4.1 Introduction

Battery-powered sensor devices must meet energy [Gobieski et al., 2019, Juang et al., 2002] and latency [Kang et al., 2017, Yildirim et al., 2018] constraints to achieve reliable performance. Thus, sensors seek ways to improve their efficiency, and a common technique for doing so is to push data processing onto the sensor device [Denby and Lucia, 2020]. This method is beneficial because local processing allows the device to transmit smaller aggregate results [Gobieski et al., 2019].

Modern sensor processing uses deep neural networks (DNNs) due to their high-quality results [Gobieski et al., 2019, Lin et al., 2020]. However, DNNs have high resource costs, making them challenging to deploy on low-power devices [Lin et al., 2020, Yao et al., 2017]. Prior systems address this challenge by partitioning DNNs between sensor and server [Banitalebi-Dehkordi et al., 2021, Kang et al., 2017, Li et al., 2019]. The sensor device holds a subset of the DNN, and the system performs inference as follows:

1. Process measurements on the sensor with the DNN subset.
2. Transmit the intermediate DNN activations to the server.
3. Complete inference with the remaining DNN layers.

In this process, the sensor always transmits the intermediate state (Step 2), and this step requires expensive wireless communication [Gobieski et al., 2019, Kang et al., 2017]. Sensing systems address this problem by augmenting DNNs with early exits (Figure 4.1). These inference models, called Multi-exit Neural Networks (MeNNs), contain early exit points

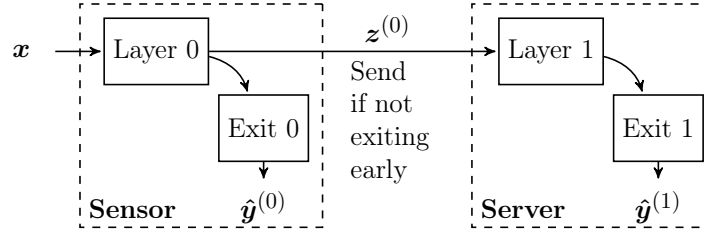


Figure 4.1: A distributed Multi-exit Neural Network (MeNN) [Teerapittayanon et al., 2016] with two total exits.

which create predictions using a subset of the entire DNN [Teerapittayanon et al., 2016]. Distributed MeNNs [Laskaridis et al., 2020, Teerapittayanon et al., 2017] form an initial prediction on the sensor, alleviating the need to communicate with the server.

MeNNs face a key decision when reaching an exit point: whether to terminate inference. This *exit decision*, which represents where the MeNN stops inference, comes with a tradeoff. Exiting earlier leads to lower inference costs by skipping subsequent layers. Early exits, however, generally reduce accuracy [Teerapittayanon et al., 2016, Wan et al., 2020a]. *Data-dependent* adaptive behavior is an emerging method to balance this tradeoff [Hu et al., 2020, Huang et al., 2017, Kaya et al., 2019, Scardapane et al., 2020, Teerapittayanon et al., 2016, Wan et al., 2020a, Wang et al., 2017, Zhou et al., 2020]. These methods determine when to terminate inference by comparing the neural network’s prediction *confidence* (e.g., the maximum classification probability) to a single *threshold* [Wang et al., 2017]. Inference terminates when the confidence exceeds the threshold. This strategy is data-dependent because the prediction confidence is a function of the given input. These data-dependent methods yield high accuracy under cost constraints because not all inputs are equally difficult to classify [Kaya et al., 2019, Wang et al., 2017].

In this work, we study MeNN early exiting from a new perspective: privacy. We demonstrate how previous data-dependent exit policies [Teerapittayanon et al., 2016, Wang et al., 2017] show *asymmetric behavior* on different predicted classes. That is, the MeNN exits early for some classes more frequently than others. This behavior occurs because MeNNs

produce different distributions of prediction confidence for different classes (§4.2.3). Standard data-dependent methods, however, apply a *single* threshold to all prediction confidence values. This single threshold thus causes different average exit behavior across predictions.

This asymmetry means an adversary can learn about a distributed MeNN’s predictions by observing its exit decisions, creating a privacy issue in sensing systems for two reasons:

1. Sensors leak when the distributed MeNN exits early through side-channels derived from encrypted communication patterns (§4.3.2, §4.6.8).
2. Sensors collect values with temporal correlations [Deshpande et al., 2004, Gedik et al., 2007], so an adversary who extracts when the MeNN exits early observes consecutive elements with similar predictions.

With these properties, we discover that an adversary can use communication side-channels to observe a distributed MeNN’s pattern of exit decisions. The adversary can then use this pattern to infer the model’s most frequent prediction over short timescales. For example, on the task of activity recognition [Kwapisz et al., 2011], we show that this side-channel allows a **white box** attacker (§4.3.2) to uncover 52.00% of the MeNN’s predictions (§4.6.8). This leakage extends to ten tasks; on average, we demonstrate how data-dependent policies enable a white box attacker to infer MeNN predictions $1.85\times$ more frequently than random guessing (§4.6.2). Further, we build a **black box** attacker (§4.3.2) which can still infer MeNN predictions at $1.56\times$ the rate of random guessing (§4.6.7). Thus, privacy-conscious sensor systems [Hiremath et al., 2014, Winkler et al., 2008, Sotirakis et al., 2023, Vasisht et al., 2017] cannot gain the benefits of MeNNs.

There are two common approaches to closing side-channels based on asymmetric behavior. The first method standardizes resource use [Brumley and Boneh, 2005, Dyer et al., 2012, Kocher et al., 1999]. For MeNNs, this principle forces all inputs to exit at the same point. This design negates the benefits of MeNNs, resulting in either suboptimal accuracy (§4.6.4) or prohibitive overhead (§4.6.9). The second approach randomizes behavior [Apthorpe et al.,

2019, Kocher et al., 1999]. This method leads to an MeNN policy that exits early uniformly at random. Unfortunately, random exiting imposes a high accuracy cost (§4.6.4). We instead want exit policies with the following properties:

- (P1) Achieve perfect privacy by having no observable relation between exit decisions and MeNN predictions.
- (P2) Exhibit minimal energy overhead compared to previous data-dependent methods.
- (P3) Display greater inference accuracy than Random exiting.
- (P4) Do not require retraining or redesign of existing MeNNs.

This last property is important because neural network training is expensive. Solutions requiring new architectures or retraining are not compatible with existing MeNNs.

We develop two new exit policies to meet (P1)–(P4). Our first approach, *Per-Class Exiting (PCE)*, augments prior data-dependent methods by using different confidence thresholds for each class (§4.4). PCE tunes these thresholds to exhibit symmetric exit rates for better privacy (§4.6.2). The policy retains high inference accuracy because it still makes decisions using prediction confidence (§4.6.4).

Despite improved empirical privacy, we prove that PCE has no privacy guarantees (P1). To achieve these guarantees, we augment PCE with randomization through a new policy called *Confidence-Guided Randomness (CGR)* (§4.5). CGR randomly selects an exit using probabilities biased toward PCE’s confidence-based decisions. Further, CGR optimizes the MeNN’s accuracy while maintaining privacy by adapting the bias magnitude using trends in the MeNN’s predictions. By using both prediction confidence and randomization, CGR has higher accuracy than exiting early uniformly at random (§4.6.4) with statistically equivalent privacy (§4.6.2). CGR also incurs negligible overhead (§4.6.9) and works with already-trained MeNNs, allowing the policy to successfully satisfy (P1)–(P4).

To the best of our knowledge, this is the first work to study the prediction privacy of data-dependent early exiting for MeNNs. Overall, we make the following contributions¹:

1. We show that previous data-dependent MeNN exit policies leak information about model predictions. For two-exit distributed MeNNs, an adversary can infer the model’s predictions under both white box and black box assumptions.
2. We create a policy called Per-Class Exiting (PCE) that uses different thresholds for inputs of each class. This method reduces the leakage of prior data-dependent policies and preserves accuracy to within 0.4 percentage points.
3. We construct a policy, Confidence-Guided Randomness (CGR), which integrates randomization into PCE. CGR has theoretical privacy benefits. Compared to exiting uniformly at random, CGR displays statistically equivalent privacy with higher accuracy on over 90% of target exit rates.

Our work demonstrates the privacy implications of performing data-dependent distributed MeNN inference on sensing systems. With our proposed methods, privacy-conscious applications can safely achieve the performance benefits of MeNNs.

4.2 Background and Motivation

This section provides background on multi-exit neural networks (§4.2.1) before discussing policies for early exiting (§4.2.2). We then provide an example of information leakage (§4.2.3) and state the goal of privacy-preserving exit policies (§4.2.4).

1. The code is available at <https://github.com/tejaskannan/privacy-dnn-early-exit>. This work appears as a conference paper at CCS 2023 [Kannan et al., 2023].

4.2.1 Multi-Exit Neural Networks (MeNNs)

Deep neural networks (DNNs) are statistical inference models with layers of linear and nonlinear transformations [LeCun et al., 2015]. Under supervised learning, DNNs $f_{\boldsymbol{\theta}}$ fit their parameters $\boldsymbol{\theta}$ by minimizing a loss function (e.g., cross-entropy) on a labelled dataset $D = \{(\mathbf{x}^{(t)}, y^{(t)})\}_{t=0}^{N-1}$ [Rumelhart et al., 1986]². We consider DNNs on classification tasks where $\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}) \in \mathbb{R}^L$ has the predicted probabilities for each class in $[L] = \{0, 1, \dots, L-1\}$. The predicted class is $\hat{y} = \operatorname{argmax}_{\ell \in [L]} \hat{y}_{\ell}$.

Standard DNNs process inputs \mathbf{x} with all parameters $\boldsymbol{\theta}$. However, this design is unnecessary to achieve high accuracy, as not all inputs are equally difficult to classify [Kaya et al., 2019, Wang et al., 2017]. DNNs can preserve accuracy with reduced execution costs using early exits; each exit is an intermediate classifier that creates a prediction with a subset of model parameters [Teerapittayanon et al., 2016, Wan et al., 2020a, Wang et al., 2017]. We describe this approach, called a Multi-exit Neural Network (MeNN), as a collection of K classifiers $f_{\boldsymbol{\theta}}^{(K)} = [f_{\boldsymbol{\theta}_0}^{(0)}, f_{\boldsymbol{\theta}_1}^{(1)}, \dots, f_{\boldsymbol{\theta}_{K-1}}^{(K-1)}]$ where $f_{\boldsymbol{\theta}_k}^{(k)}$ is a DNN that takes an input or output of $f_{\boldsymbol{\theta}_{k-1}}^{(k-1)}$, $\mathbf{z}^{(k-1)}$, and creates the predicted probabilities $\hat{\mathbf{y}}^{(k)} \in \mathbb{R}^L$. At time t , the output of $f_{\boldsymbol{\theta}_k}^{(k)}$ is $\hat{\mathbf{y}}^{(k,t)}$.

We consider MeNNs distributed across devices in a sensing system [Laskaridis et al., 2020, Teerapittayanon et al., 2017]. The sensor device contains the initial layers of the MeNN, and the server contains the remaining portion of the model (Figure 4.1). When crossing between devices, the system must transmit the required state (e.g., $\mathbf{z}^{(0)}$ in Figure 4.1) to continue inference.

4.2.2 Early Exit Policies

MeNNs must determine the exit point at which to terminate inference. This decision comes with a tradeoff [Teerapittayanon et al., 2016, Wan et al., 2020a]. Later exits apply more

2. We denote vectors in boldface and scalars in plain text.

Algorithm 4 MeNN inference routine

```
procedure MENNINFERENCE( $\mathbf{x}$ ,  $f_{\theta}^{(K)}$ ,  $\pi$ )  
   $\mathbf{z}^{(-1)} \leftarrow \mathbf{x}$   
  for  $k \in [K - 1]$  do  
     $\hat{\mathbf{y}}^{(k)}, \mathbf{z}^{(k)} \leftarrow f_{\theta_k}^{(k)}(\mathbf{z}^{(k-1)})$   
    if  $\pi(\hat{\mathbf{y}}^{(k)}, k) = 0$  then  
      return  $\hat{\mathbf{y}}^{(k)}$   
   $\hat{\mathbf{y}}^{(K-1)}, \mathbf{z}^{(K-1)} \leftarrow f_{\theta_k}^{(K-1)}(\mathbf{z}^{(K-2)})$   
  return  $\hat{\mathbf{y}}^{(K-1)}$ 
```

parameters and achieve higher accuracy. This benefit comes with higher execution costs, as the system must compute more layers and communicate between devices.

Prior MeNNs manage this tradeoff in a data-dependent manner using prediction confidence [Berestizshevsky and Even, 2019, Teerapittayanon et al., 2016, Wang et al., 2017]. Common confidence functions $h : \mathbb{R}^L \rightarrow \mathbb{R}$ include the maximum value [Wang et al., 2017] and entropy [Teerapittayanon et al., 2016] in the predicted distribution. When inference reaches the k^{th} exit, the system compares the confidence $h(\hat{\mathbf{y}}^{(k)})$ to a threshold $\tau^{(k)}$. If $h(\hat{\mathbf{y}}^{(k)}) \geq \tau^{(k)}$, the model is "confident enough" and stops inference; otherwise, the system continues to the next exit. The thresholds $\tau^{(k)}$ control how the MeNN balances the tradeoff between accuracy and execution cost. Larger thresholds yield more accurate results, and smaller thresholds result in low-cost inference. We emphasize that these existing methods make *data-dependent* decisions because the confidence is a deterministic function of the k^{th} prediction, $\hat{\mathbf{y}}^{(k)}$. Thus, the MeNN's exit decisions contain information about the model's predictions, where the *exit decision* for time t , $k_t \in [K]$, is the exit at which the MeNN stops inference.

We represent early exit methods using a *policy* $\pi : \mathbb{R}^L \times \mathbb{N} \rightarrow \{0, 1\}$, which takes the prediction $\hat{\mathbf{y}}^{(k)} \in \mathbb{R}^L$ and the exit $k \in [K]$. The function outputs 0 to terminate inference and 1 to continue. Algorithm 4 shows this procedure. The equation below is a general data-dependent policy for the confidence h where $[\cdot]_1$ is 1 when the condition holds and 0 otherwise. Concrete policies use a specific implementation for h such as $h_{MaxProb}(\hat{\mathbf{y}}) = \max_{i \in [L]} \hat{y}_i$

[Wang et al., 2017].

$$\pi_{DataDep}(\hat{\mathbf{y}}^{(k)}, k) = [h(\hat{\mathbf{y}}^{(k)}) < \tau^{(k)}]_1 \quad (4.1)$$

4.2.3 Example of Information Leakage

Data-dependent MeNN exit policies create a relationship between their predictions and exit decisions. We find that this relation allows an adversary to learn about the MeNN’s predictions by observing its exit pattern. This ability is useful when the attacker cannot view the model’s classification directly due to a lack of physical device access and encrypted wireless communication (§4.3).

We demonstrate this property on a speech detection task [Warden, 2018]. We use a BranchyNet [Teerapittayanon et al., 2016] MeNN with two total exits and a data-dependent policy (Equation 4.1) with $h_{MaxProb}$ [Wang et al., 2017]. When predicting the word "on," the MeNN exits early 61.61% of the time; when predicting "off," the early exit rate is 33.44%. Thus, early exiting means the MeNN is more likely to have predicted "on" than "off." An adversary observing these exit decisions can infer the presence of either word from this difference, indicating that data-dependent policies expose valuable information about the MeNN’s predictions.

This asymmetric behavior stems from the prediction confidence having different distributions for different classes. In this example, the average confidence is 0.8328 for "on" and 0.7623 for "off." The policy, however, uses the *same* threshold for all inputs (Equation 4.1). Thus, instances of "on" are more likely to exit early through confidence scores above the single threshold, causing asymmetric behavior. This insight leads to a key novelty of our work: the use of multiple thresholds to account for distribution differences (§4.4, §4.5).

As we show, this privacy problem occurs on multiple tasks, MeNNs, and confidence functions (§4.6.2). The breadth of this leakage means the issue goes beyond a specific

dataset, model architecture, or prior data-dependent policy. Further, we emphasize that MeNNs are not trained to exhibit this asymmetric behavior. Nevertheless, we empirically observe this phenomenon on every considered task.

4.2.4 Goals of Private Exit Policies

We design MeNN exit policies $\pi : \mathbb{R}^L \times \mathbb{N} \rightarrow \{0, 1\}$ to meet four criteria on ordered input streams $\tilde{D} = [\mathbf{x}^{(t)}]_{t=0}^{T-1}$ of length T . First, π should not leak information about the MeNN’s predictions; there should be no observable relationship between the policy’s decisions and the MeNN’s classifications (P1). This quality should hold for all possible ordered streams \tilde{D} , as system designers cannot anticipate the exact stream at design time. This consideration encompasses datasets with temporal correlations and shifting input distributions.

Second, the policy must adhere to given exit rates $\{\rho_k\}_{k=0}^{K-1}$ where $\sum_{k=0}^{K-1} \rho_k = 1$. Under π , the MeNN should stop at exit k on $\rho_k \cdot T$ of inputs in \tilde{D} . This criterion is necessary to meet the resource limits of low-power devices (P2).

A purely randomized policy meets these two criteria by stopping at exit k with probability ρ_k . However, this method reduces the MeNN’s inference accuracy (§4.6.4). Thus, the third goal is to build policies with better inference accuracy than random exiting (P3).

Finally, solutions must not require retraining or redesigning the MeNN (P4). This property is necessary because DNN training is expensive, and security solutions should work for existing MeNNs. Policies under the definition π operate only on the MeNN’s predictions. Thus, these policies satisfy this property, as π is not tightly coupled with the MeNN parameters. Indeed, Algorithm 4 shows how π can change without altering the MeNN.

We emphasize that a priori, it is not guaranteed that there exist MeNN exit policies that meet all four properties. *A key contribution of this work is demonstrating that such policies exist.*

4.3 Threat Model

We first state the target system and the adversary’s goal (§4.3.1). We then discuss the attacker’s capabilities (§4.3.2) and present examples of distributed MeNNs requiring prediction privacy (§4.3.3).

4.3.1 Target System and Attack Goal

We consider a sensing system composed of edge devices and a centralized server [Mainwaring et al., 2002]. Each device periodically captures measurements and processes these values using a distributed MeNN [Laskaridis et al., 2020, Teerapittayanon et al., 2017], encrypting all communication. We call this MeNN the *target* model.

The adversary is a passive observer who uses the target MeNN’s exit decisions to expose its predictions. Specifically, the attacker sees blocks of $B > 0$ exit decisions and uses an *attack model* $g_\phi : [K]^B \rightarrow [L]$ to infer the MeNN’s most frequent prediction in this block. Section 4.3.2 discusses how the attacker observes these exit decisions. The attacker uses a training phase to find the parameters ϕ below where $v^{(t)}$ is the exit decision and $\hat{y}^{(k_t, t)}$ is the MeNN’s prediction at time t . $MostFreq(\cdot)$ returns the most common argument value.

$$\hat{y}(t) = MostFreq(\hat{y}^{(k_t, t)}, \hat{y}^{(k_{t+1}, t+1)}, \dots, \hat{y}^{(k_{t+B-1}, t+B-1)}) \quad (4.2)$$

$$\phi = \operatorname{argmax}_{\tilde{\phi}} \sum_{q=0}^{\lfloor \frac{T}{B} \rfloor - 1} [g_{\tilde{\phi}}(v^{(qB)}, v^{(qB+1)}, \dots, v^{(qB+B-1)}) = \hat{y}(qB)]_1 \quad (4.3)$$

We assume the stream \tilde{D} contains temporal correlations. Such correlations occur in sensor settings [Deshpande et al., 2004, Gedik et al., 2007]. On correlated streams, each block contains related inputs with similar labels. Correlated streams present a greater privacy challenge. Intuitively, correlated inputs cause MeNNs to make similar predictions and related exit decisions under data-dependent policies, allowing the adversary to view blocks of nearby

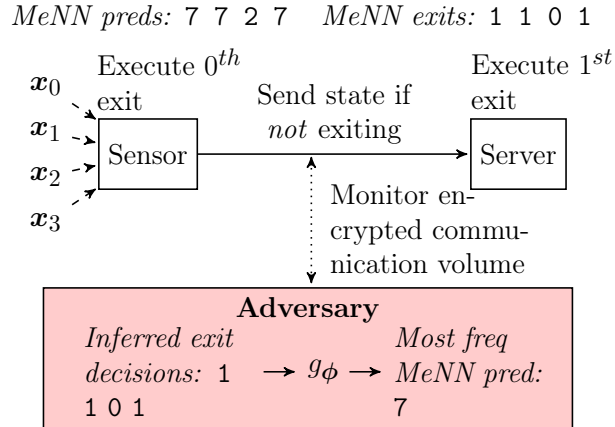


Figure 4.2: The threat model against distributed MeNNs. An exit decision of 0 means stopping on the sensor.

decisions under one class. Streams with independent inputs prevent this temporal linkage. More precisely, an exit decision from one input gives the adversary one of K options to recover a prediction $\ell \in [L]$. For uncorrelated inputs, the attacker must view each decision in isolation; when $K < L$, this recovery is underdetermined. For correlated inputs, the attacker can link adjacent decisions under approximately the same prediction. With this ability, the attacker can use one of $K^B > K$ possible inputs to extract this block’s most frequent prediction $\ell \in [L]$. Thus, the attacker can use more features on correlated streams. We emphasize that the adversary targets the MeNN’s predictions instead of the true labels, as ground truth is unavailable at runtime. An adversary who infers the MeNN’s results learns what the target system knows.

4.3.2 Adversary Capabilities

We assume the adversary targets a sensing system known to use a distributed MeNN. The attacker has no physical device access but can sniff the communication between the sensor and the server [Apthorpe et al., 2019, Dyer et al., 2012]. These assumptions are realistic for wearable sensors [Hiremath et al., 2014, Kwapisz et al., 2011] and devices in remote locations [Denby and Lucia, 2020, Juang et al., 2002, Winkler et al., 2008]. The adversary

cannot directly read the MeNN’s predictions due to encrypted communication. Further, without physical access, the attacker cannot deploy their own sensor to derive equivalent insights. As we design exit policies, we follow Kerckhoffs’s Principle [Shannon, 1949] and allow the adversary to know the policy’s details. We assume the attacker knows the sampling period and the target task’s label space.

The adversary exposes the MeNN’s predictions by inferring the model’s exit decisions using communication side-channels. Below, we describe two examples of how distributed MeNNs leak the decision to exit early through communication patterns. This general pattern holds for all distributed MeNNs we are aware of.

DDNNs Deep distributed neural networks (DDNNs) implement distributed MeNNs across a hierarchy of devices [Teerapittayanon et al., 2017]. The system conserves resources by only transmitting information when continuing inference to the next device. An adversary can learn the exit decision using the presence or absence of wireless traffic.

SPINN SPINN performs distributed MeNN inference under latency constraints [Laskaridis et al., 2020]. When the MeNN partition point occurs after the first early exit, SPINN only communicates when not exiting early. This behavior creates the same side-channel as that of DDNNs. SPINN also supports partitions before the first exit. This setting still leaks information because SPINN requires the sensor to always compute a local prediction by continuing until the first early exit. If this exit signals termination, the sensor sends a second message to the server to stop computation. The attacker can infer an early exit using the presence of this second message.

In both cases, the passive adversary can use communication patterns to discover early exit behavior (Figure 4.2). This side-channel exists even when data is encrypted, as encryption does not obfuscate communication volume. This adversary only knows whether the system

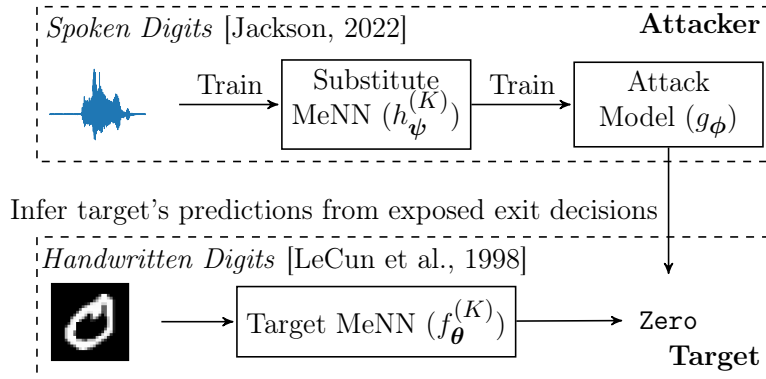


Figure 4.3: An example of the black box attacker.

exits on the sensor or server. Thus, we focus on MeNNs with $K = 2$ exits where the sensor holds the initial exit. Our methods extend to MeNNs with $K > 2$ (§4.6.10).

The adversary uses the attack model g_ϕ to infer the target MeNN’s predictions from the extracted exit decisions. We consider two sets of assumptions for how the attacker fits the parameters ϕ (Equation 4.3) from examples of blocks of exit decisions and MeNN predictions. In both cases, the attacker applies g_ϕ at runtime to the target MeNN executing on an unseen testing dataset.

1. **White Box:** The white box adversary has access to an offline version of the target MeNN and training dataset. The attacker uses exit patterns and predictions from this MeNN on the target task to fit the attack model g_ϕ .
2. **Black Box:** The black box adversary cannot access the target MeNN or training dataset. Instead, this adversary trains a substitute MeNN $h_\psi^{(K)}$ on a related task and fits g_ϕ using exit decisions and predictions from $h_\psi^{(K)}$ (Figure 4.3).

We use the black box adversary to confirm that the white box assumptions are not too strong.

4.3.3 *Example Settings*

This section describes two examples [Banitalebi-Dehkordi et al., 2021, Zeng et al., 2019] of distributed DNNs in applications requiring prediction privacy.

Sensitive facilities use DNNs to perform license plate recognition for authorized access [Banitalebi-Dehkordi et al., 2021]. Huawei’s AutoSplit [Banitalebi-Dehkordi et al., 2021] framework applies distributed DNNs in this context. Prediction privacy is necessary, as the DNN’s predictions indicate which license plates have access. The attacker could use leaked predictions to create fake credentials which pass the authorization check. Further, the server may be hosted off-premise (e.g., in the cloud). Thus, when the adversary cannot access the physical location without credentials, they can still sniff the communication to the remote server. This property means the system cannot leak information about authorization decisions through communication patterns.

Manufacturing plants apply DNNs to detect defective parts, and Boomerang [Zeng et al., 2019] leverages distributed DNNs for this application. Prediction privacy is essential for two reasons. First, the defect rate indicates the manufacturer’s efficiency. This information is valuable to competitors. Second, an adversary launching a supply-chain attack can use exposed DNN predictions to know whether their attack is successful. In both cases, an adversary may be unable to physically access all the validation points without alerting the building’s security. Instead, the attacker can more easily observe the communication patterns to a single server, especially if the server is remote. Further, when validating a supply chain attack, the adversary wants to know if their inserted defect passes the target’s inspection. Thus, rather than finding the true defect rate, it is more valuable to learn what the target system knows.

4.4 Per-Class Exiting (PCE)

Data-dependent MeNN policies using a single threshold can leak information through their exit decisions (§4.2.3). Our first solution, Per-Class Exiting (PCE), replaces the single confidence threshold with separate thresholds for each class (§4.4.1). With this design, PCE stops a fraction ρ_k of inputs for *every class* at exit k , thus preserving resource usage and improving privacy compared to prior work. Unfortunately, PCE has no theoretical privacy guarantees, and we construct adversarial orderings with high prediction leakage (§4.4.2).

4.4.1 Policy Design

We formally describe PCE using an MeNN $f_{\theta}^{(K)}$ with target exit rates $\rho_k \in [0, 1] \forall k \in [K]$. Consider the prediction confidence function $h : \mathbb{R}^L \rightarrow \mathbb{R}$ (§4.2.2). Focusing on the k^{th} exit, PCE uses thresholds $\tau_{\ell}^{(k)}$ for each class $\ell \in [L]$ that satisfy the probability below. The terms $\hat{\mathbf{Y}}^{(k)} \in \mathbb{R}^L$ and Y are random variables for the k^{th} exit’s prediction and the true label, respectively.

$$P(h(\hat{\mathbf{Y}}^{(k)}) \geq \tau_{\ell}^{(k)}, h(\hat{\mathbf{Y}}^{(r)}) < \tau_{\ell}^{(r)} \forall r \in [k] \mid Y = \ell) = \rho_k \quad (4.4)$$

Equation 4.4 states that for each label, inputs should stop at exit k with rate ρ_k . In practice, we fit the thresholds using the empirical confidence distributions on the task’s validation set. PCE performs inference using Algorithm 4 with π_{PCE} below where $\hat{y}_i^{(k)}$ the predicted probability for class $i \in [L]$ at exit k . We omit the time $t \in [T]$, as the policy is stateless.

$$\ell(k) = \arg \max_{i \in [L]} \hat{y}_i^{(k)} \quad (4.5)$$

$$\pi_{PCE}(\hat{\mathbf{y}}^{(k)}, k) = [h(\hat{\mathbf{y}}^{(k)}) < \tau_{\ell(k)}^{(k)}]_1 \quad (4.6)$$

This design augments previous data-dependent exit policies (Equation 4.1) with different thresholds $\tau_\ell^{(k)}$ for each class. We emphasize that the policy selects thresholds using the MeNN’s prediction at each exit. To protect the overall classification, PCE should instead choose thresholds using the final result i.e., use $\tau_\ell^{(k)}$ where $\ell = \arg \max_{i \in [L]} \hat{y}_i$ and $\hat{\mathbf{y}}$ is the MeNN’s final predicted probabilities. At an early exit, however, we do not know the final prediction $\hat{\mathbf{y}}$ when *not* terminating inference. PCE thus uses the current exit’s prediction $\hat{\mathbf{y}}^{(k)}$ as an approximation.

4.4.2 Adversarial Data Orderings

PCE uses a data-dependent approach that fits thresholds such that the *overall* exit rates for each label are ρ_k . This behavior means that PCE delivers good privacy on uncorrelated input orders without introducing randomization, as such streams only require long-term balancing (§4.6.5). However, this long-term balancing makes no guarantees about eliminating short-term patterns.

This insight suggests there exist input orders causing high leakage. We formalize this idea in Proposition 4.4.1 below. The proof describes how to build the adversarial ordering \tilde{D} and attack model g_ϕ (§4.3.1). As confirmation, we follow the proof and build an adversarial ordering with inputs from Fashion MNIST [Xiao et al., 2017]. We attack a two-exit MeNN with PCE ($\rho_0 = 0.9$). As expected, the adversary infers 100% of the MeNN’s predictions. Thus, despite lower empirical leakage than prior methods (§4.6.2), PCE delivers no privacy guarantees on correlated inputs. Instead, PCE better protects MeNNs processing unrelated inputs over time. We emphasize that Proposition 4.4.1 applies to any deterministic policy, not just PCE. We omit the exit index $k \in [K]$ from π below because we consider a two-exit MeNN, which only applies π at exit $k = 0$ (Algorithm 4).

Proposition 4.4.1. *Let $\pi : \mathbb{R}^L \rightarrow \{0, 1\}$ be a deterministic policy on a two-exit MeNN $f_\theta^{(2)}$ for a rational exit rate $\rho_0 = \frac{m}{M} \in (0, 1)$. Let D be the set of possible samples (\mathbf{x}, y) ,*

$D_x = \{\mathbf{x} : \exists \ell \in [L], (\mathbf{x}, \ell) \in D\}$ be the inputs, and $D_\ell = \{\mathbf{x} : (\mathbf{x}, \ell) \in D\}$ be the inputs with label ℓ . Assume $\forall k \in \{0, 1\}$, $\pi^{-1}(k) \cap D_\ell \neq \emptyset$ where $\pi^{-1}(k) = \{\mathbf{x} \in D_x : \pi(f_{\theta_0}^{(0)}(\mathbf{x})) = k\}$. Then, there exists an ordered dataset $\tilde{D} = [(\mathbf{x}^{(t)}, y^{(t)}) \in D]_{t=0}^{T-1}$ for $T = n \cdot LM$, $n > 1$ with the following.

1. The policy π exhibits an early exit rate of ρ_0 on \tilde{D} .
2. There exists a function $g_\phi : \{0, 1\}^{LM} \rightarrow [L]$ such that

$$g_\phi(\pi(\hat{\mathbf{y}}^{(t)}), \dots, \pi(\hat{\mathbf{y}}^{(t+LM-1)})) = \text{MostFreq}(y^{(t)}, \dots, y^{(t+LM-1)})$$

where $t = j \cdot LM$ for any $j \in [n]$.

3. Not all non-overlapping blocks of LM exit decisions (in property 2) have the same most frequent label.

Proof. We design an ordered dataset with $T = n \cdot LM$ elements in blocks of LM . Consider the j^{th} block for $j \in [n]$. We will fill this block with inputs of label $\ell \equiv j \pmod{L}$. Let $\mathbf{x}_\ell^{(0)} \in \pi^{-1}(0) \cap D_\ell$ and $\mathbf{x}_\ell^{(1)} \in \pi^{-1}(1) \cap D_\ell$ be arbitrary. In this block, for $0 \leq t < L$, we set \mathbf{x}_t to $\mathbf{x}_\ell^{(0)}$ when $t = \ell$ and $\mathbf{x}_\ell^{(1)}$ otherwise. On the remaining $LM - L$ inputs, we arbitrarily select $Lm - 1$ positions for early exiting, setting these elements to $\mathbf{x}_\ell^{(0)}$ and the remaining to $\mathbf{x}_\ell^{(1)}$. Note that there are $Lm - 1$ available slots because $M - 1 \geq m$ so $LM - L \geq Lm > Lm - 1$. This construction satisfies the desired properties. First, each block contains exactly Lm early exits, yielding an early exit rate of $(nLm)/(nLM) = m/M = \rho_0$. Second, let $g_\phi : \{0, 1\}^{LM} \rightarrow [K]$ output the position of the first early exit. By construction, the j^{th} block (with inputs of label ℓ) has only one early exit in the first L positions. This exit occurs in position ℓ , so the function g_ϕ correctly maps to the most frequent label. Finally, there are $n > 1$ blocks. As the j^{th} block has inputs of label $\ell \equiv j \pmod{L}$, \tilde{D} will have at least two blocks with different most frequent labels. \square

4.5 Confidence-Guided Randomness (CGR)

PCE uses a modified data-dependent method to balance the long-term exit rates across classes. This method, however, does not address temporal correlations. In fact, Proposition 4.4.1 indicates that temporal dependencies can compromise any deterministic policy. Thus, we must obfuscate temporal patterns through randomization. Unfortunately, exiting early uniformly at random achieves poor inference accuracy (§4.6.4).

We instead present a new policy, Confidence-Guided Randomness (CGR), that is an interpolation between PCE and uniformly randomized exiting, merging the benefits of both approaches. When detecting uncorrelated inputs, CGR behaves like PCE to achieve higher accuracy. On segments with high correlations, CGR applies greater randomization to maintain privacy.

CGR has three features. First, the policy evaluates PCE and uses its decision to create a *confidence-biased* exit probability which determines the exiting behavior (§4.5.1). Second, CGR *adapts* the bias magnitude on highly-correlated streams (§4.5.2). Finally, CGR enforces *exit quotas* over short windows to limit an adversary’s ability to discover data-dependent information (§4.5.3). By employing randomness, CGR achieves theoretical benefits over PCE (§4.5.4).

4.5.1 Confidence-Biased Randomization

CGR uses PCE as an internal data-dependent method (Figure 4.4). When reaching the k^{th} exit, CGR builds an exit probability that is biased toward $\pi_{PCE}(\hat{\mathbf{y}}^{(k,t)}, k)$ at step $t \in [T]$. CGR samples this biased probability to make a random exit decision. Thus, CGR leverages prediction confidence through PCE, enabling higher inference accuracy than pure randomization.

We formalize this design by considering the k^{th} exit of an MeNN $f_{\theta}^{(K)}$ with exit rate $\rho_k \in [0, 1]$. CGR uses a bias $\alpha^{(k,t)} \in [0, 1]$ (discussed in §4.5.2) at step t to make randomized

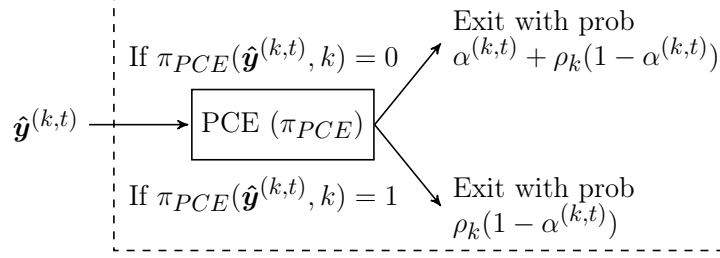


Figure 4.4: CGR’s decision process at the k^{th} exit point.

decisions with the following probabilities. The rates for $\pi_{\text{CGR}}(\hat{\mathbf{y}}^{(k,t)}, k) = 1$ are one minus those below. We omit the exit indices for brevity.

$$P(\pi_{\text{CGR}}(\hat{\mathbf{y}}^{(k,t)}) = 0 | \pi_{\text{PCE}}(\hat{\mathbf{y}}^{(k,t)}) = 0) = \alpha^{(k,t)} + \rho_k(1 - \alpha^{(k,t)}) \quad (4.7a)$$

$$P(\pi_{\text{CGR}}(\hat{\mathbf{y}}^{(k,t)}) = 0 | \pi_{\text{PCE}}(\hat{\mathbf{y}}^{(k,t)}) = 1) = \rho_k(1 - \alpha^{(k,t)}) \quad (4.7b)$$

These equations show how the exit probabilities are biased in the direction of PCE. For example, when $\pi_{\text{PCE}}(\hat{\mathbf{y}}^{(k,t)}, k) = 0$, CGR exits with rate $\alpha^{(k,t)} + \rho_k(1 - \alpha^{(k,t)}) = \rho_k + \alpha^{(k,t)}(1 - \rho_k) \geq \rho_k$ where the inequality holds because $0 \leq \rho_k, \alpha^{(k,t)} \leq 1$. Thus, CGR aligns with the PCE’s decision and exits more frequently than the target rate ρ_k . Note that these probabilities maintain an overall exit rate of ρ_k when $P(\pi_{\text{PCE}}(\hat{\mathbf{Y}}^{(k,t)}, k) = 0) = \rho_k$. This condition holds when the test distribution matches that of the training set.

4.5.2 Adapting the Probability Bias Magnitude

The probability biases control a tradeoff between accuracy and privacy. If $\alpha^{(k,t)} \approx 1$, CGR skews toward PCE, yielding high inference accuracy with possible leakage (§4.4, §4.6.12). If $\alpha^{(k,t)} = 0$, CGR is fully randomized. CGR balances this tradeoff using the insight that PCE provides good privacy on uncorrelated input streams. Thus, CGR exploits periods of low correlation by leveraging PCE to achieve high accuracy. On highly correlated segments, CGR applies more randomness to ensure privacy. To protect the MeNN’s predictions (§4.2.4), CGR measures correlations using the model’s results.

CGR implements this design by adaptively setting the bias $\alpha^{(k,t)}$ with the parameters $\gamma < 1 < \beta$. The policy has a maximum bias of $\alpha \in [0, 1)$. CGR sets $\alpha^{(k,t)}$ at step $t \geq 1$ as follows where $\alpha^{(k,0)} = \alpha$.

$$\alpha^{(k,t)} = \begin{cases} \min(\beta \cdot \alpha^{(k,t-1)}, \alpha) & \text{if } \hat{y}^{(k,t)} \neq \hat{y}^{(k,t-1)} \\ \gamma \cdot \alpha^{(k,t-1)} & \text{if } \hat{y}^{(k,t)} = \hat{y}^{(k,t-1)} \end{cases} \quad (4.8)$$

The parameters β and γ control how the bias changes in response to the MeNN’s predictions. For example, when $\beta \gg 1$ and $\gamma \approx 1$, CGR will quickly increase and slowly decrease the bias. This behavior causes CGR to have a higher *average* bias, thereby aligning more with PCE than pure randomization (§4.6.12). Based on our experiments, $\beta = 2.0$ and $\gamma = 0.9$ provide favorable results, and these settings are robust across multiple datasets and data orders. Overall, CGR uses this procedure to adapt the bias to offset temporal trends in the MeNN’s predictions. We emphasize that CGR still makes randomized decisions even when using a high bias $\alpha^{(k,t)} < 1$.

4.5.3 Short-Term Exit Quotas

CGR may leak predictions if the attacker infers the policy’s biased probabilities, as these probabilities encode PCE’s decisions (Equation 4.7). CGR protects against this leakage by enforcing exit quotas over short windows. The quotas ensure the MeNN stops a set number of times at each exit point, balancing the exit counts and reducing the adversary’s analysis to smaller sample sizes. For each window, the attacker can only extract the bias direction before an exit quota becomes saturated; afterward, the policy never stops at this exit and uses no information from PCE. Exposing small samples benefits privacy because it limits the adversary’s ability to derive meaningful statistical significance on biased exit rates.

CGR implements exit quotas using a window $W \in \mathbb{N}$. The policy enforces that $\omega_k = \lfloor \rho_k \cdot W \rfloor + \eta_k$ inputs stop at exit k where $\eta_k \in \{0, 1\}$ are random such that $\sum_{k=0}^{K-1} \omega_k = W$.

The system no longer exits at output k upon meeting its quota. After each window, the policy resets the quotas and randomly selects a new $W \sim [W_{min}, W_{max}]$ where the bounds are parameters. This randomization limits the adversary’s ability to locate each window.

Randomizing the window size, however, does not fully prevent the attacker from discovering when CGR uses biased probabilities. For example, the adversary can use a run of exits at a single output to infer a window’s end. The theoretical privacy of this attack is not well-established. However, we believe this attack does not present a problem for three reasons. First, the adversary’s recovery of each window is only approximate due to randomization. Second, CGR already protects against periods of high potential leakage by adapting its bias parameter (§4.5.2). Finally, CGR’s empirical information leakage is statistically equivalent to random exiting (§4.6.2, §4.6.5).

4.5.4 Theoretical Benefits

This section demonstrates CGR’s theoretical benefits. The main result is Proposition 4.5.1 which establishes a bound on the probability ratio of finite exit patterns from inputs of different classes. This result improves upon PCE, which can leak an unbounded amount of information over finite time horizons (Proposition 4.4.1). A key property of CGR is that its bounds apply to *any* data stream, including those with temporal correlations and distributional shifts.

Proposition 4.5.1 further provides a guideline on how to set α from a security perspective, as α determines the difference between the upper and lower probability bounds. However, CGR will not be tight with the established bounds because the policy uses biases $\alpha^{(k,t)} < \alpha$ (§4.6.5) due to similar predictions over time (§4.5.2). Smaller biases create narrower probability bounds, allowing CGR to provide better empirical privacy than the proposition guarantees.

Before presenting the proposition, we introduce relevant notation. Let π_r be the CGR

policy without exit quotas and $\delta, \epsilon : \mathbb{R} \rightarrow \mathbb{R}$ be functions such that $\delta(\alpha) = (1 - \alpha - \rho_0(1 - \alpha))/\rho_0$ and $\epsilon(\alpha) = (1 - \rho_0(1 - \alpha))/\rho_0$ where ρ_0 is the exit rate for $K = 2$. We define these functions for notational convenience.

Proposition 4.5.1. *Consider a two-exit MeNN with a target early exit rate ρ_0 . Suppose we observe a sequence of T exit decisions $\pi_r^{(t)} := \pi_r(\hat{\mathbf{y}}^{(0,t)}, 0) = v_t$ for $v_t \in \{0, 1\}$ and $t \in [T]$. Let these T inputs belong to the same class and $n = \sum_{t=0}^{T-1} v_t$. Then, π_r displays the following bounds for any labels $\ell_0, \ell_1 \in [L]$.*

$$\begin{aligned} \left(\frac{\delta(\alpha)}{\epsilon(\alpha)}\right)^n \left(\frac{1 - \rho_0\epsilon(\alpha)}{1 - \rho_0\delta(\alpha)}\right)^{T-n} &\leq \frac{P(\pi_r^{(t)} = v_t \forall t | Y = \ell_0)}{P(\pi_r^{(t)} = v_t \forall t | Y = \ell_1)} \\ \left(\frac{\epsilon(\alpha)}{\delta(\alpha)}\right)^n \left(\frac{1 - \rho_0\delta(\alpha)}{1 - \rho_0\epsilon(\alpha)}\right)^{T-n} &\geq \frac{P(\pi_r^{(t)} = v_t \forall t | Y = \ell_0)}{P(\pi_r^{(t)} = v_t \forall t | Y = \ell_1)} \end{aligned}$$

Proof. The construction of the CGR policy yields the following bounds, resulting from the alignment (or lack thereof) of PCE with the decisions v_t under the largest possible bias α .

$$\begin{aligned} \rho_0\delta(\alpha) \leq P(\pi_r^{(t)} = 1) \leq \rho_0\epsilon(\alpha) \\ 1 - \rho_0\epsilon(\alpha) \leq P(\pi_r^{(t)} = 0) \leq 1 - \rho_0\delta(\alpha) \end{aligned}$$

The maximum probability of observing the sequence $\{v_t\}_{t=0}^{T-1}$ occurs when PCE aligns with each v_t . The minimal probability happens when PCE goes against all observed decisions. This logic holds any $\ell \in [L]$, creating the following.

$$\begin{aligned} (\rho_0\delta(\alpha))^n (1 - \rho_0\epsilon(\alpha))^{T-n} \leq P(\pi_r^{(t)} = v_t \forall t | Y = \ell) \\ (\rho_0\epsilon(\alpha))^n (1 - \rho_0\delta(\alpha))^{T-n} \geq P(\pi_r^{(t)} = v_t \forall t | Y = \ell) \end{aligned}$$

From these inequalities, we obtain the desired result. □

One consequence of Proposition 4.5.1 results from the bounds having the form $(q_0)^n (s_0)^{T-n}$

and $(q_1)^n(s_1)^{T-n}$ where $q_0, s_0 < 1 < q_1, s_1$. As $T \rightarrow \infty$, the bounds go to zero and infinity, respectively. Thus, the biased probabilities can cause unbounded exit rate differences over an *infinite* horizon. CGR prevents worst-case scenarios by avoiding highly biased rates over long windows, confirming the benefits of the adaptive bias procedure and use of exit quotas.

4.6 Evaluation

We evaluate the information leakage and inference accuracy of distributed MeNNs using previous data-dependent policies and our proposed methods. In summary, we find the following:

1. Standard data-dependent policies leak information about MeNN predictions through their exit patterns. This leakage occurs from practical (§4.6.2) and theoretical (§4.6.3) perspectives. PCE uses multiple thresholds to reduce this leakage, and CGR obtains near-perfect privacy.
2. PCE and CGR use prediction confidence to display higher inference accuracy than a fully randomized policy (§4.6.4).
3. PCE has higher leakage on input streams with stronger correlations (§4.6.5). CGR adapts itself to protect against these trends, showing near-random leakage under temporal correlations and distribution shifts (§4.6.6).
4. Single-threshold data-dependent policies still leak valuable information to a *black box* attacker who has no access to the MeNN and training dataset (§4.6.7).
5. On a realistic distributed MeNN setup, prior data-dependent policies leak predictions to an attacker with access to encrypted communication patterns (§4.6.8). Our methods provide protection in this end-to-end setting.
6. PCE and CGR show negligible overhead on a low-power microcontroller (MCU) (§4.6.9). Thus, our policies improve privacy while retaining the efficiency of MeNNs.

Table 4.1: Dataset properties.

Dataset	# Train	# Val	# Test	# Classes
Activity [Anguita et al., 2012]	36,790	13,629	20,441	6
Cifar10 [Krizhevsky et al., 2009]	39,796	10,204	10,000	10
Cifar100 [Krizhevsky et al., 2009]	39,796	10,204	10,000	100
EMNIST [Cohen et al., 2017]	87,800	25,000	18,800	47
Fash. MNIST [Xiao et al., 2017]	47,798	12,202	10,000	10
Food Quality [Holland et al., 1998]	2,945	196	198	2
GTSRB [Stallkamp et al., 2011]	38,580	10,799	9,120	43
MNIST [LeCun et al., 1998]	47,798	12,202	10,000	10
Speech Ccmds [Warden, 2018]	28,532	3,457	4,482	11
WISDM [Kwapisz et al., 2011]	32,005	5,858	21,769	3

7. Prior data-dependent policies leak more information on MeNNs with more exits (§4.6.10).

In contrast, CGR protects MeNNs independent of the number of exit points.

4.6.1 Experimental Setup

Datasets and Neural Network Parameters

We evaluate MeNNs on ten standard tasks (Table 4.1) covering many input types and label spaces. We use BranchyNet [Teerapittayanon et al., 2016] MeNNs, focusing on models with two total exits. On Cifar [Krizhevsky et al., 2009], we use VGG models [Simonyan and Zisserman, 2014] with early exiting after the first pooling layer. We use pre-trained versions of each VGG model; we attach early exits and fine-tune these output layers [Kaya et al., 2019]. We apply dense models on Activity [Anguita et al., 2012], Food Quality [Holland et al., 1998], and WISDM [Kwapisz et al., 2011] and convolutional networks on the remainder. These MeNNs have four hidden layers, with early exiting after the first. We execute at most ten training epochs using Adam (step size of 10^{-3}) [Kingma and Ba, 2014], a batch size of 16, and a dropout [Srivastava et al., 2014] rate of 0.3.

For two-exit MeNNs, we run policies on 21 exit rates $\rho_0 = 0.0, 0.05, \dots, 1.0$. We use a tighter range for MeNNs with more exits (§4.6.10). The datasets are unordered, and we use

two methods to create the temporal correlations present in sensor settings.

1. *Same-Label* builds blocks of size B by selecting $(1 - \varepsilon) \cdot B$ random elements of a single label and $\varepsilon \cdot B$ inputs from arbitrary classes. We set $B = 10$ and $\varepsilon = 0.2$. Section 4.6.11 considers alternate settings.
2. *Nearest-Neighbor* constructs B -sized blocks by choosing a random anchor element and using the anchor’s $B - 1$ nearest neighbors [Bernhardsson, 2023] in order.

We focus on Same-Label orders, as Nearest-Neighbor produces weak correlations on inputs such as colored images.

Exit Policies

We use following baseline exit policies.

1. *Random* selects the MeNN exit uniformly at random using the rates ρ_k . This policy has perfect privacy because it makes *data-independent* decisions.
2. *Entropy* is a data-dependent method (Equation 4.1) with confidence $h_{Entropy}(\hat{\mathbf{y}}) = (-\sum_{i=0}^{L-1} \hat{y}_i \log(\hat{y}_i))^{-1}$ [Teerapittayanon et al., 2016].
3. *MaxProb* is a data-dependent policy (Equation 4.1) with confidence $h_{MaxProb}(\hat{\mathbf{y}}) = \max_{i \in [L]} \hat{y}_i$ [Wang et al., 2017].

We apply PCE and CGR to both confidence functions. We use CGR with a maximum bias of $\alpha = 0.5$, adaptation factors of $\beta = 2.0$ and $\gamma = 0.9$, and a window range of $[5, 20]$ (§4.5). We set these parameters using a grid search (§4.6.12) and fix them for all datasets.

We fit confidence thresholds τ for an exit rate ρ by setting τ to the $(1 - \rho)^{th}$ quantile of the MeNN’s confidence values on the task’s validation set. We execute both Random and CGR over five trials as these policies have stochastic behavior.

Adversary Design

We design the attack model g_ϕ using an AdaBoost ensemble with 100 decision trees. The attack model uses non-overlapping blocks of B adjacent exit decisions to infer the MeNN’s most frequent prediction in this block. This block size matches that of the dataset order. We create two variants with different input features. The first uses each exit point’s frequency, and the second uses the exact exit pattern. We use the frequencies against Same-Label orders and the patterns on Nearest-Neighbor streams (§4.6.1). The exact pattern leads to overfitting on Same-Label orders. We train g_ϕ with two different assumptions (§4.3.2).

1. **White box** attackers use patterns from the *target* MeNN’s on the *original* task’s validation set.
2. **Black box** adversaries use patterns from a *substitute* MeNN on a *related* task’s validation set.

We always evaluate g_ϕ on the target MeNN processing the test fold.

Aggregate Metrics

We evaluate policies on many sets of target exit rates. For inference accuracy, we compute the average result across all targets. For privacy metrics, we compute show the worst-case result by calculating the maximum over the targets, as policies should not leak information for *any* target exit rates (P1). This methodology aligns with prior work in measuring security from a worst-case perspective [Carlini et al., 2022]. We aggregate trials by taking the average trial result for each target exit rate.

Hardware Setup

We conduct experiments in simulation and on a low-power microcontroller (MCU). The simulator runs MeNNs in Tensorflow [Abadi et al., 2016a] and records the predictions and

exit decisions (§4.6.2-§4.6.7, §4.6.10). The adversary observes the exact exit decisions.

We perform an end-to-end side-channel attack on a two-exit distributed MeNN [Teerapittayanon et al., 2017] with the initial exit on a TI MSP430 FR5994 MCU [Instruments, 2021] and the remaining model on a server (§4.6.8). The MCU processes inputs every second and uses Bluetooth Low Energy (BLE) to transmit the intermediate state when not exiting early. This setup follows DDNNs [Teerapittayanon et al., 2017] and SPINN [Laskaridis et al., 2020] on a two-device system. The sensor applies AES-128 encryption [Daemen and Rijmen, 1999]. We capture the encrypted packets using Wireshark [Orebaugh et al., 2006] and provide this log to the attacker. We drop 5% of packets to simulate a lossy link. The attacker only observes traffic when the system does *not* exit early. The adversary finds the number of early exits between transmissions as follows, where R is the sampling period and d_n is the time of the n^{th} message.

$$\text{num_early_exits}(n) = \max(\lfloor (d_n - d_{n-1})/R \rfloor - 1, 0) \quad (4.9)$$

We emphasize that the side-channel attack applies to variants of this system; e.g., if the sensor sends predictions when exiting early, the attacker can infer early exits using differences in message sizes.

4.6.2 White Box Attack

We first measure the practical privacy of MeNN exit policies under white box assumptions (§4.6.3 discusses theoretical leakage). We use the white box methodology in §4.6.1 to infer predictions from two-exit MeNNs with Same-Label orders ($B = 10, \varepsilon = 0.2$).

Figure 4.5 shows the maximum attack accuracy across all target exit rates. The *attack accuracy* is the accuracy of the attack model g_ϕ ; this metric measures the fraction of the adversary’s predictions that match the MeNN’s most common classification in each temporal

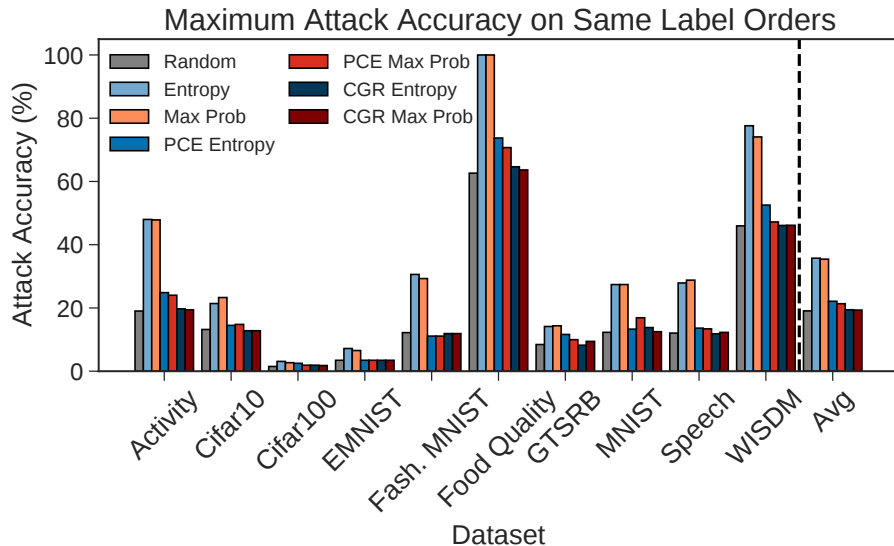


Figure 4.5: Maximum attack accuracy across 21 exit rates (lower is better).

block. These results display how standard data-dependent policies consistently leak information about MeNN predictions. Across all tasks, these policies show mean worst-case attack accuracy that is $1.87\times$ (Entropy) and $1.85\times$ (Max Prob) higher than Random. On the Food Quality task, the attacker infers up to 100% of the MeNN’s most frequent predictions in each block. Further, both methods display worse privacy than Random on all tasks. Thus, this leakage does not result from a single dataset or confidence function; instead, it comes from the data-dependent approach of previous methods.

Both PCE and CGR display better privacy. PCE yields $1.16\times$ (Entropy) and $1.12\times$ (Max Prob) higher worst-case attack accuracy than Random. These values are lower than that of prior data-dependent policies (Figure 4.5), showing the benefit of multiple thresholds. CGR performs even better, displaying $1.02\times$ higher worst-case attack accuracy averaged across all tasks. We further compare these policies to Random using Welch’s t-test. The null hypothesis is the attack accuracy normalized to the most frequent MeNN prediction is no different than that of Random. With this methodology, Random and CGR have an *insignificant* difference at the 0.05 level with p -values of 0.21 (CGR Entropy) and 0.09 (CGR Max Prob). Thus, CGR obtains near-random privacy independent of the confidence function.

Entropy, Max Prob, and PCE show significant differences.

Entropy and Max Prob show low attack accuracy on Cifar100, as this dataset has a large label space (Table 4.1). However, we observe leakage through the attacker’s average rank (AR) of the correct class [Asonov and Agrawal, 2004]. On Cifar100, Entropy and Max Prob have a worst-case AR of 33.60 and 35.77, respectively. These values are far lower than Random (48.20); thus, attackers still learn valuable information on tasks with many labels. Across all datasets, CGR has an AR of $0.99\times$ Random, further demonstrating its near-random privacy.

4.6.3 Theoretical Information Leakage

We supplement the practical attack with an analysis agnostic of the attack model. We quantify privacy using the empirical normalized mutual information (NMI) [Kvålseth, 2017] between the MeNN’s exit decisions and predictions. A high NMI means observing the exit decisions reduces the adversary’s uncertainty about the model’s predictions. A policy with no leakage should exhibit an NMI close to Random. We use the definition $NMI(X, Y) = (2 \cdot I(X, Y)) / (H(X) + H(Y))$ where $I(\cdot)$ is the mutual information and $H(\cdot)$ is the Shannon entropy. We measure the NMI by comparing individual exit decisions (X) and MeNN predictions (Y). This metric does not depend on temporal correlations. We reduce the empirical NMI’s bias with Miller-Madow correction [Paninski, 2003]. We use the same setup as §4.6.2.

Table 4.2 shows the maximum NMI, confirming the trends in §4.6.2. Standard data-dependent policies show high NMI with values up to 0.1725 points higher than Random on average. Further, both policies eclipse Random on all tasks, indicating that this leakage is consistent and independent of the confidence function.

PCE improves privacy, showing an NMI of up to 0.0203 points higher than Random; this rate is over $8.4\times$ lower than previous data-dependent methods (Table 4.2). On average, CGR has a maximum NMI of only 0.0009 points above Random. This figure is over $191\times$ lower

Table 4.2: Maximum empirical normalized mutual information (NMI) (all values $\times 10^{-2}$) between exit decisions and MeNN predictions across 21 target rates (lower is better). The final row shows the average (std dev) difference compared to Random.

Dataset	Rand	Std	Entropy		Std	Max Prob	
			PCE	CGR		PCE	CGR
Activity	0.25	33.99	2.33	0.35	34.22	2.11	0.40
Cifar10	0.25	5.43	0.66	0.31	5.24	0.76	0.29
Cifar100	0.45	4.79	1.21	0.50	4.10	1.25	0.52
EMNIST	0.07	7.19	0.31	0.09	7.29	0.30	0.09
Fash. MNIST	0.04	19.84	0.28	0.04	19.83	0.25	0.04
Food Quality	0.24	49.62	3.73	0.11	49.62	5.73	0.33
GTSRB	1.26	8.92	5.26	1.59	9.23	4.45	1.46
MNIST	0.06	6.30	0.85	0.05	6.29	2.44	0.15
Speech Cnds	0.33	16.15	3.17	0.50	16.29	3.22	0.48
WISDM	0.42	23.61	3.32	0.58	18.77	3.14	0.55
Avg Diff v Rand	0.00 (0.00)	17.25 (14.06)	1.78 (1.38)	0.08 (0.12)	16.75 (14.01)	2.03 (1.58)	0.09 (0.06)

than prior data-dependent policies. These results provide additional evidence that CGR has near-perfect privacy.

4.6.4 Inference Accuracy

The second tradeoff dimension we investigate is inference accuracy, as accuracy represents the MeNN’s answer quality. We use two-exit MeNNs under the setup in §4.6.2.

Table 4.3 shows the average MeNN accuracy across all exit rates, and Figure 4.6 displays the results on the Activity task. We have three takeaways. First, Random has a high accuracy penalty; existing policies achieve an average accuracy of 2.03 (Entropy) and 2.30 (Max Prob) points above Random. Second, PCE retains high MeNN accuracy, showing values within 0.5 points of its standard data-dependent variant on seven of ten datasets. Random achieves this mark only twice. Finally, CGR consistently outperforms Random on all tasks with an overall average accuracy of 0.51 (Entropy) and 0.57 (Max Prob) points higher. For $\rho_0 \in (0, 1)$, CGR eclipses Random on 90% (171 / 190) of target rates under both confidence functions.

We note two additional results. First, an alternate method to eliminate leakage is to use a fixed policy that always exits at the same point. This baseline must use the early

Table 4.3: Average (std dev) inference accuracy across 21 target exit rates for each policy and task (higher is better). The standard deviation shows the variation in the average accuracy across five independent trials.

Dataset	Rand	Std	Entropy		Std	Max Prob	
			PCE	CGR		PCE	CGR
Activity	87.22 (0.02)	89.24	88.46	87.57 (0.01)	89.69	88.85	87.70 (0.02)
Cifar10	80.35 (0.04)	85.18	84.83	81.69 (0.03)	85.50	85.17	81.78 (0.01)
Cifar100	61.68 (0.05)	64.47	64.43	62.40 (0.04)	64.81	64.79	62.52 (0.02)
EMNIST	85.92 (0.02)	87.28	87.35	86.37 (0.01)	87.31	87.37	86.39 (0.01)
Fash. MNIST	90.37 (0.03)	91.59	91.38	90.69 (0.03)	91.66	91.46	90.72 (0.01)
Food Quality	97.01 (0.02)	97.34	97.34	97.09 (0.01)	97.34	97.34	97.09 (0.01)
GTSRB	80.45 (0.04)	84.85	83.21	81.21 (0.04)	85.37	83.74	81.39 (0.05)
MNIST	98.64 (0.01)	99.19	99.19	98.83 (0.01)	99.19	99.19	98.83 (0.01)
Speech Cmds	85.94 (0.04)	88.58	87.76	86.43 (0.02)	88.78	87.92	86.40 (0.01)
WISDM	86.31 (0.02)	86.48	87.81	86.68 (0.01)	87.24	87.93	86.77 (0.02)
Avg Diff v Rand	0.00 (0.00)	2.03 (1.55)	1.79 (1.18)	0.51 (0.34)	2.30 (1.63)	1.99 (1.31)	0.57 (0.38)

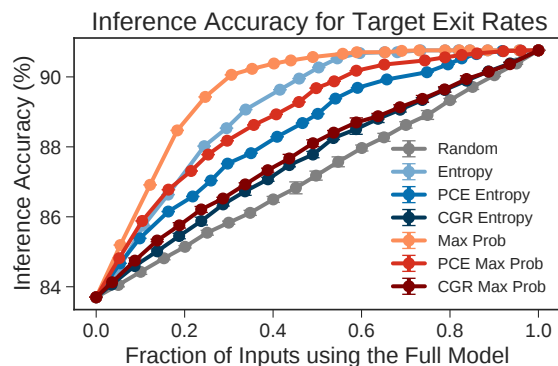


Figure 4.6: Inference accuracy (%) on the Activity dataset. Error bars show the standard deviation over five trials.

exit to meet resource limits (§4.6.9), resulting in low accuracy. From Figure 4.6, the early exit has an accuracy of about 83%; all other policies reach an average accuracy above 87% (Table 4.3). Thus, PCE and CGR show better accuracy under resource limits than a fixed policy. Second, Entropy and Max Prob perform poorly on WISDM. This result comes from suboptimal exit decisions due to MeNN overconfidence. PCE corrects this problem by setting higher thresholds for the overconfident classes, highlighting an alternative benefit of using multiple thresholds.

4.6.5 Alternate Dataset Orders

Nearest Neighbor

We further evaluate privacy using the white box adversary (§4.3.2) on Nearest-Neighbor orders (§4.6.1) with $B = 10$ (Figure 4.7). CGR maintains its near-random privacy, showing $0.99\times$ higher attack accuracy than Random on average. Using the methodology of §4.6.2, CGR’s attack accuracy is *not* significantly greater than Random. In contrast, PCE shows higher leakage on Nearest-Neighbor orders, displaying an average worst-case attack accuracy that is $1.24\times$ (Entropy) and $1.28\times$ (Max Prob) higher than Random. These values exceed the $1.12\times$ (Entropy) and $1.18\times$ (Max Prob) marks from the Same-Label order on these four tasks. This greater leakage comes from the Nearest-Neighbor order’s high correlations. Nevertheless, PCE still displays better privacy than single-threshold techniques.

CGR continues to show improved inference accuracy (Table 4.4), eclipsing Random on 80% (61 / 76) of target exit rates under the Max Prob metric. These results are similar with the Entropy function. However, the gap between CGR and Random is smaller than on Same-Label streams (Table 4.3). This difference results from CGR’s adaptive bias. The Nearest-Neighbor order contains stronger correlations, often having blocks with over 90% of elements in the same class. In turn, CGR acts more randomly. For example, on the Activity task with $\rho_0 = 0.5$, CGR has an average bias of 0.1507 on Nearest-Neighbor and 0.4446 on Same-Label. CGR properly responds to greater correlations by reducing its bias magnitude.

Uncorrelated

We further evaluate the attack on data streams with randomly-ordered inputs. In this setting, the adversary uses each exit decision to infer the MeNN’s individual predictions. Table 4.5 compares the average inference accuracy and maximum attack accuracy on the Activity task for Uncorrelated and Nearest-Neighbor streams. The latter order exhibits the

Table 4.4: Average (std dev) inference accuracy across 21 exit rates for Nearest-Neighbor blocks (higher is better).

Dataset	Rand	Std	MaxProb	
			PCE	CGR
Activity	87.46 (0.02)	89.99	89.17	87.51 (0.01)
EMNIST	87.10 (0.01)	88.04	88.10	87.38 (0.02)
Fash. MNIST	91.11 (0.03)	92.36	92.09	91.34 (0.01)
MNIST	99.44 (0.01)	99.67	99.68	99.48 (0.01)
Avg Diff v Rand	0.00 (0.00)	1.06 (0.57)	0.95 (0.48)	0.17 (0.09)

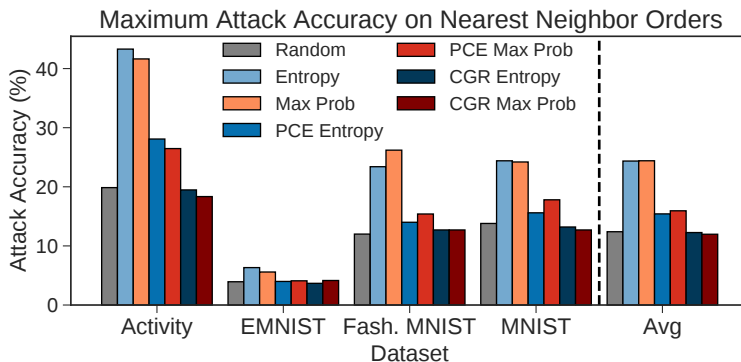


Figure 4.7: Maximum attack accuracy on Nearest-Neighbor dataset orders (lower is better).

strongest temporal relations. The adversary still learns information about MeNN’s results in uncorrelated settings, though the attack efficacy is lower. We hypothesize this result occurs because the adversary has less context on uncorrelated orders (§4.3.1). This comparison further shows the benefits of PCE in isolation. On uncorrelated streams, PCE displays an attack accuracy within 3 points of Random. This figure is over $2\times$ smaller than PCE’s gap to Random on the Nearest-Neighbor order. Thus, for uncorrelated streams, PCE delivers an inference accuracy of 1.7 points above Random for a small cost in privacy. We note that PCE does not achieve an attack accuracy equivalent to either Random or CGR because PCE uses static thresholds fit on a training set, and the testing set contains empirical differences.

Table 4.5: Mean inference accuracy (Infr. Acc.) and maximum attack accuracy (Att. Acc.) on the Activity task for Uncorrelated and Nearest-Neighbor orders.

Policy	Uncorrelated		Nearest-Neighbor	
	<i>Infr. Acc.</i>	<i>Att. Acc.</i>	<i>Infr. Acc.</i>	<i>Att. Acc.</i>
Random	86.86	18.12	87.46	19.86
Max Prob	89.44	36.59	89.99	41.63
PCE Max Prob	88.57	21.14	89.17	26.47
CGR Max Prob	87.36	18.88	87.51	18.35

4.6.6 Distribution Shifts

Sensing systems often face distribution shifts where the data observed at runtime differs from that used during training [Koh et al., 2021]. We evaluate the privacy impact of distribution shifts by constructing an alternate testing set for MNIST [LeCun et al., 1998] using the first 10,000 digits from the Extended MNIST dataset [Cohen et al., 2017]. This alternate dataset has a higher mean ($\mu = 0.172$) and standard deviation ($\sigma = 0.331$) pixel value than that of MNIST ($\mu = 0.131, \sigma = 0.308$) due to differences in image preprocessing. We use the same MeNN trained on MNIST as in §4.6.2 and evaluate the MeNN on this alternate testing set.

Table 4.6 shows the white box attack accuracy with the Same-Label order. Under distributional shifts, PCE has privacy similar to standard data-dependent exiting. This phenomenon occurs because the shifted distribution changes the MeNN’s prediction confidence, breaking the balancing effect of PCE’s multiple thresholds. For example, when $\rho = 0.75$, PCE exits early on 96.88% of the digit 1 and 55.08% of the digit 7 in the shifted test set; on standard MNIST, these exit rates are 73.03% and 72.49%, respectively. Thus, under shifted distributions, PCE shows the same asymmetric exit behavior seen in previous data-dependent methods. In contrast, CGR protects against this issue by leveraging randomness. With the Entropy metric, CGR displays a worst-case attack accuracy less than that of Random. Along with this privacy benefit, CGR shows higher mean inference accuracy on the shifted test set. CGR has an average accuracy of 88.33% (Entropy) and 88.30% (Max Prob), compared to 87.07% for Random. Note that the shifted distribution causes lower MeNN in-

Table 4.6: Worst-case attack accuracy for exit policies on an MeNN trained on MNIST and tested on either a shifted distribution (EMNIST Digits) or the same distribution (MNIST).

Policy	EMNIST Digits	MNIST
Random	13.60	12.50
Entropy	21.80	27.40
Max Prob	20.10	27.40
PCE Entropy	19.30	13.30
PCE Max Prob	21.10	16.90
CGR Entropy	13.50	13.80
CGR Max Prob	14.20	12.50

ference accuracy overall (Table 4.3). This finding aligns with prior work on neural networks facing distributional shifts [Koh et al., 2021].

4.6.7 Black Box Attack

We confirm the white box assumptions are not too strong by considering a weakened attacker with black box access (§4.3.2). This adversary cannot access the target MeNN and only knows the number of MeNN exits K and the target task’s label space (e.g., the digits 0-9 for MNIST). The adversary uses this knowledge to select a related dataset with the same label space. The attacker trains a substitute MeNN $h_{\psi}^{(K)}$ on this related dataset (Figure 4.3). We assume the attacker uses a reasonable MeNN architecture for their selected dataset (e.g., ResNet [He et al., 2016] on Cifar-10). The adversary trains their MeNN by optimizing the average individual classification loss of each exit point [Teerapittayanon et al., 2016]. Finally, following black box adversarial DNN attacks [Papernot et al., 2017], the attacker fits an attack model g_{ϕ} on patterns from the substitute $h_{\psi}^{(K)}$ and applies g_{ϕ} to the target MeNN $f_{\theta}^{(K)}$ on the original dataset. We use the following attack settings.

1. *Cifar10 Blurred*: The attacker has a version of Cifar10 [Krizhevsky et al., 2009] corrupted with a Gaussian blur ($r = 0.5$). This version has different training and validation splits than the original. The attacker generalizes to the standard Cifar10 task.

2. *Pen Digits*: The adversary attacks an MNIST [LeCun et al., 1998] convolutional MeNN with a dense substitute model trained to classify digits from sequences of (x, y) pen coordinates [Alimoglu and Alpaydin, 1996].
3. *Spoken Digits*: The attacker targets an MNIST [LeCun et al., 1998] MeNN with a substitute trained on spoken digit audio [Jackson, 2022].
4. *Speech Noisy*: The adversary uses the Speech [Warden, 2018] dataset perturbed with white noise ($SNR = 50$). The training and validation splits differ from those of the original. The attacker targets an MeNN on the standard Speech task.
5. *WISDM Sim*: The adversary uses the WISDM task’s simulated version to target an MeNN trained on real-world data [Kwapisz et al., 2011], emulating an attacker collecting its own dataset.

Compared to the target MeNN, we use substitutes with different architectures and hyperparameters (e.g., batch sizes). For example, the attacker’s substitute MeNN for *Pen Digits* uses five fully connected layers with sizes $(8, 12, 48, 48, 48)$, early exiting after the second, and Leaky ReLU activations. The target MeNN processes the *MNIST* dataset using four convolutional layers with $(16, 32, 64, 32)$ filters, early exiting after the first, and ReLU activations. On *Cifar-10*, the attacker uses ResNet-18 [He et al., 2016] with early exiting after the second block. The target system uses a VGG architecture. These settings thus consider different neural networks which both achieve good accuracy on their given tasks. We fit each substitute three times.

Table 4.7 shows the maximum attack accuracy. The weakened adversary still achieves the best results against existing data-dependent policies; MeNNs using Max Prob show a mean worst-case attack accuracy of $1.56 \times$ Random. Although this efficacy is lower than white box settings (§4.6.2), the black box attacker still learns valuable information despite having no offline access to the target MeNN. CGR continues to show a worst-case attack

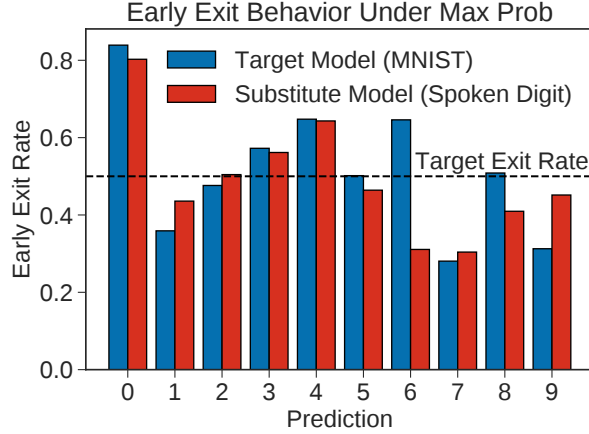


Figure 4.8: Early exit rates per prediction under the Max Prob policy for the target MeNN trained on MNIST [LeCun et al., 1998] and the substitute trained on spoken digits [Jackson, 2022].

accuracy close to Random.

This attack works because the substitute $h_{\psi}^{(K)}$ and target $f_{\theta}^{(K)}$ MeNNs often contain similar exit behavior, even though the target MeNN is unknown to the adversary. Figure 4.8 shows this phenomenon. Predictions for zero and seven have similar exit rates across the two MeNNs despite training the target on images and the substitute on audio. However, these rates are not always consistent. When predicting six, the target exits early more frequently, showing why the black box accuracy does not reach that of white box settings.

4.6.8 White Box Attack on Low-Power MCUs

We launch an end-to-end side-channel attack against distributed MeNNs [Teerapittayanon et al., 2017] executing on a low-power MCU (§4.6.1). We execute each policy for 500 inputs on the Activity [Kwapisz et al., 2011] task with the Same-Label order ($B = 10$), creating 50 temporal windows for the attacker.

Figure 4.9 shows inference and attack accuracy. In all cases, the attacker discovers the correct exit decisions from the packet trace. Using the white box attack model g_{ϕ} , the Max Prob policy exhibits the highest attack accuracy, while CGR reduces this leakage to

Table 4.7: Worst-case attack accuracy (%) averaged (std dev) across trained MeNNs. In each group, the top row is the white box setting. The remaining rows use the black box method.

Train Task	Rand	MaxProb	
		<i>Std</i>	<i>CGR</i>
Cifar10	13.20	23.30	12.80
Cifar10 Blurred	11.93 (0.52)	19.17 (1.51)	12.30 (0.37)
MNIST	12.30	27.40	12.50
Pen Digits	12.83 (0.45)	17.80 (0.72)	12.77 (0.12)
Spoken Digit	12.57 (0.21)	20.17 (1.41)	12.80 (0.16)
Speech Cmds	12.05	28.79	12.28
Speech Cmds Noisy	12.87 (0.42)	24.78 (4.90)	12.80 (0.28)
WISDM	45.96	74.08	46.14
WISDM Sim	45.82 (0.00)	59.30 (2.19)	45.88 (0.09)

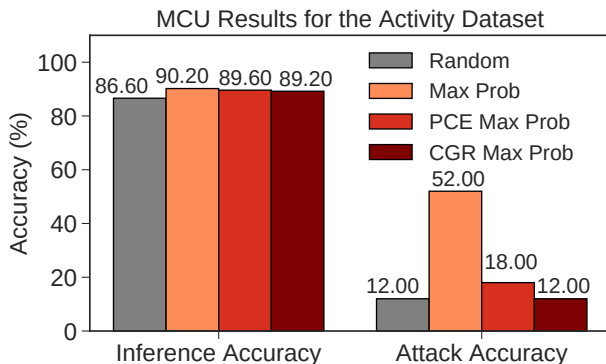


Figure 4.9: Inference accuracy and attack accuracy against distributed MeNNs executing on a low-power MCU.

Random. Further, CGR outperforms Random in inference accuracy on the MCU. These results match those from simulation (§4.6.2), showing how the discovered privacy issue and proposed defenses apply to real hardware.

4.6.9 Energy Consumption

MeNNs reduce the average cost of inference [Teerapittayanon et al., 2016]. We show how CGR and PCE preserve this benefit by measuring their energy on a TI MSP430 [Instruments, 2021, 2020]. We run the distributed MeNN from §4.6.8 over 40 trials, recording the average

Table 4.8: Average (std dev) energy consumption (mJ) on a TI MSP430 MCU [Instruments, 2021].

Policy	Exit Early	Continue
Fixed	0.047 (0.017)	30.813 (5.234)
Random	0.049 (0.018)	31.144 (6.262)
Max Prob	0.059 (0.020)	32.799 (6.114)
PCE Max Prob	0.057 (0.019)	31.249 (6.243)
CGR Max Prob	0.061 (0.020)	32.320 (6.646)

energy to wake the CPU, execute the first exit point, evaluate the policy, and encrypt the result. When continuing inference, we include the energy to transmit the 128-byte state over BLE. The Fixed policy always uses the same exit.

Table 4.8 shows the average energy for each configuration. We highlight two aspects of these results. First, using the full model incurs over two orders of magnitude of overhead. This phenomenon comes from the high energy cost of communication, as early exiting allows the system to keep the BLE module off. This discrepancy shows the prohibitive cost of a Fixed policy that always uses the entire MeNN. Second, PCE and CGR incur some computation overhead compared to Random when exiting early. However, this cost is negligible compared to BLE when using the full MeNN, and for exit rates $\rho_0 < 1$, this BLE cost dominates the energy consumption. Further, under Welch’s t-test, we observe an insignificant energy difference between Random and either PCE or CGR when continuing inference, with p -values of 0.94 (PCE) and 0.39 (CGR). Note that Max Prob has the highest average energy for the full MeNN. This result occurs due to the variance in communication energy; Max Prob also shows an insignificant difference compared to Random when using the full model. Thus, both PCE and CGR incur minimal overhead, allowing them to preserve the efficiency of MeNNs.

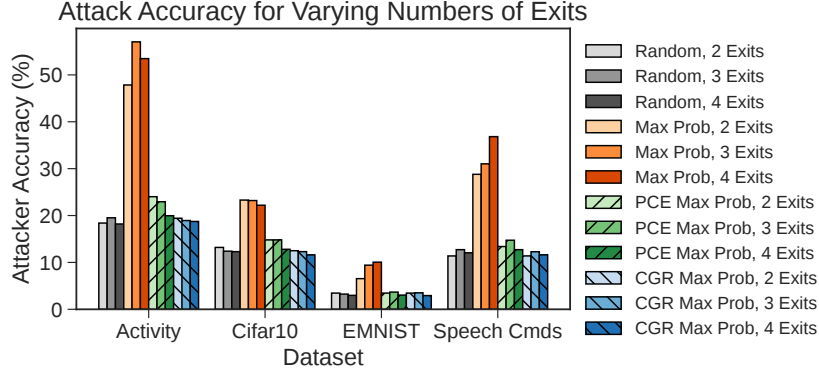


Figure 4.10: Maximum attack accuracy for MeNNs with two, three, and four exits (lower is better).

4.6.10 Beyond Two Exits

Prior sections display the leakage present in two-exit MeNNs. However, MeNNs can have more than two exits [Teerapittayanon et al., 2016, Wan et al., 2020a]. We thus measure how the number of MeNN exit points impacts its privacy in simulation under Same-Label orders ($B = 10$). We emphasize that this analysis does not yield a practical attack under our threat model; the adversary can only observe a binary decision of whether the system exits on the sensor or server (§4.3.2). Instead, we include this analysis to demonstrate (1) the potential for information leakage and (2) the performance of our methods on MeNNs with $K > 2$.

Figure 4.10 displays the maximum white box attack accuracy for MeNNs with $K = 2, 3$, and 4. Max Prob has higher leakage on MeNNs with more exits, showing an average worst-case attack accuracy of $2.20\times$ (two exits), $2.53\times$ (three exits), and $2.79\times$ (four exits) Random. Both PCE and CGR have lower prediction leakage compared to this single-threshold policy. In particular, CGR displays near-random privacy with an average worst-case attack accuracy of $1.01\times$ (two), $1.01\times$ (three), and $0.98\times$ (four) Random. These values have no trend with the number of exits. Furthermore, CGR still shows higher inference accuracy than Random. On the Activity task, CGR has an average accuracy of 88.72 (three exits) and 88.16 (four exits). These values eclipse Random: 88.15 (three) and 87.56 (four). Overall,

previous data-dependent methods exhibit greater leakage on MeNNs with more exits, and both PCE and CGR provide protection in all contexts.

4.6.11 Data Order Parameters

The provided experiments under the Same-Label order use a single block size ($B = 10$) and noise rate ($\varepsilon = 0.2$). We expand this analysis by varying these parameters for $B \in \{10, 20, 30\}$ and $\varepsilon \in \{0.1, 0.2, 0.3\}$. Figure 4.11 shows the inference accuracy and *white box* attack accuracy on the Activity task. We display the difference between each policy’s result and that of Random. CGR consistently has an inference accuracy of about 0.45 points above Random for all configurations. Further, CGR displays a maximum attack accuracy of only 0.51 points higher than Random on average across all settings. This figure is over $60\times$ lower than that of Max Prob. In the worst case, CGR has an attack accuracy of 1.76 points above Random, occurring when $B = 10$ and $\varepsilon = 0.1$. However, CGR does *not* have consistently lower privacy at this noise rate; for $B = 20$ and $\varepsilon = 0.1$, CGR has a worst-case attack accuracy of 0.48 points *lower* than Random. In total, these results align with the previous experiments under a single Same-Label configuration (§4.6.2, 4.6.4). We note that the adversary achieves better results against Max Prob for larger block sizes and lower noise rates, as such characteristics yield stronger temporal correlations.

4.6.12 CGR Parameters

The experiments in §4.6 use the CGR policy with a maximum bias $\alpha = 0.5$, increase factor $\beta = 2.0$, and decrease factor $\gamma = 0.9$. In this section, we evaluate the impact of these parameters on the policy’s inference and attack accuracy. We perform a grid search over $\alpha = \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, $\beta \in \{1.5, 2.0, 2.5\}$, and $\gamma \in \{0.5, 0.7, 0.9, 0.99\}$. For each setting, we compute the mean inference accuracy and maximum white box attack accuracy across all exit rates ρ for the CGR policy with the Max Prob confidence metric. These

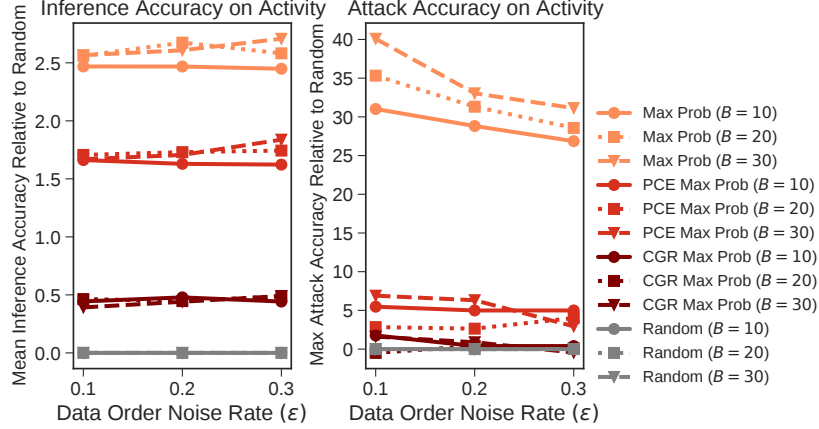


Figure 4.11: Average inference accuracy (left) and maximum attack accuracy (right) on the Activity dataset for window sizes (B) and noise rates (ϵ) under the Same-Label order.

experiments use the Activity dataset [Kwapisz et al., 2011] under Same-Label data order with $B = 10$ and $\epsilon = 0.2$. We execute each configuration over three trials and take the average result for each exit rate. We present the results for each parameter by considering the maximum, average, and minimum values aggregated across all settings for the other two parameters. Figures 4.12, 4.13, and 4.14 display the results from the perspectives of the maximum bias (α), increase factor (β), and decrease factor (γ), respectively.

On average, larger maximum bias α values lead to higher inference and attack accuracy. This trend aligns with CGR’s design (§4.5); larger biases allow CGR to align more with PCE, delivering better inference performance with worse privacy. Note that even with a large bias (e.g., $\alpha = 0.9$), CGR can still display lower inference and attack accuracy when the decrease factor γ is small (e.g., $\gamma = 0.5$).

Both the increase (β) and decrease factors (γ) show positive correlations with the policy’s inference and attack accuracy. When CGR has a small decrease factor γ , the policy is quick to decrease the bias. This phenomenon leads to smaller bias terms on average, causing more randomness, lower inference accuracy, and better privacy. The opposite occurs when γ nears one. In particular, when $\gamma \approx 1$ and $\beta \gg 1$ (e.g., $\gamma = 0.99$, $\beta = 2.5$), CGR shows high inference accuracy (88.14) and attack accuracy (19.73) for a fixed $\alpha = 0.9$. This result

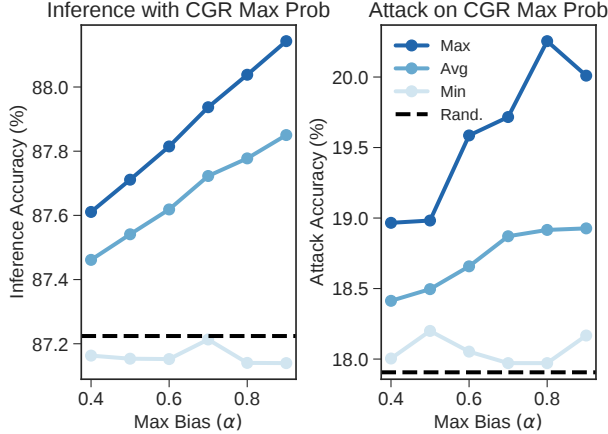


Figure 4.12: Mean inference accuracy (left) and worst-case attack accuracy (right) for different maximum bias (α) values aggregated across $\beta \in \{1.5, 2.0, 2.5\}$ and $\gamma = \{0.5, 0.7, 0.9, 0.99\}$.

matches the theory of CGR’s adaptive bias algorithm (§4.5.2), as these settings result in quickly increasing and slowly decreasing the bias term.

In general, we find that $\alpha = 0.5$, $\beta = 2.0$ and $\gamma = 0.9$ deliver near-random attack privacy with favorable inference accuracy. As demonstrated, these settings are robust across multiple different tasks and confidence metrics (§4.6).

4.7 Related Work

Multi-Exit Neural Networks (MeNNs) Prior work introduces early exits into neural networks [Berestizshevsky and Even, 2019, Huang et al., 2017, Kaya et al., 2019, Lee and Shin, 2018, Teerapittayanon et al., 2016, Veit and Belongie, 2018, Wan et al., 2020a, Wang et al., 2017]. To select an exit point, existing systems use data-dependent exit policies with a single threshold on prediction confidence [Berestizshevsky and Even, 2019, Teerapittayanon et al., 2016, Wang et al., 2017]. Other methods use bandit algorithms [Ju et al., 2021], runtime feedback [Wan et al., 2020b], or decision agreement [Zhou et al., 2020]. We focus on policies using maximum probability and entropy confidence, as they are cheap and well-

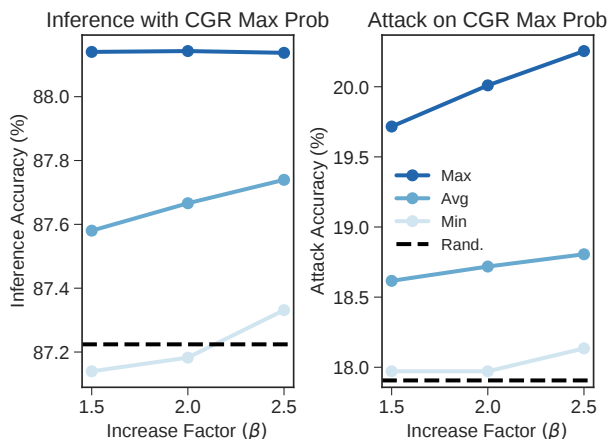


Figure 4.13: Mean inference accuracy (left) and worst-case attack accuracy (right) for different increase factors (β) aggregated across $\alpha \in \{0.4, 0.5, \dots, 0.9\}$ and $\gamma = \{0.5, 0.7, 0.9, 0.99\}$.

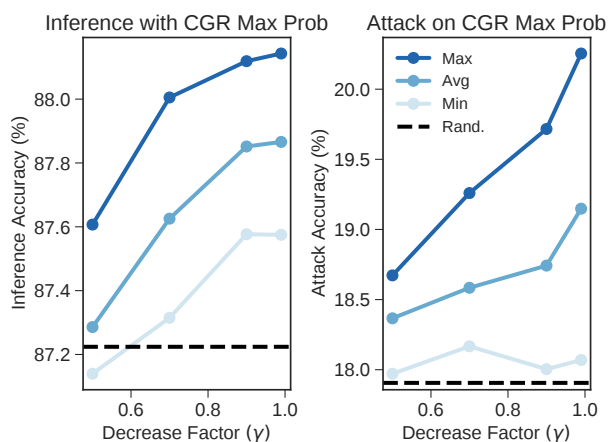


Figure 4.14: Mean inference accuracy (left) and worst-case attack accuracy (right) for different decrease factors (γ) aggregated across $\alpha \in \{0.4, 0.5, \dots, 0.9\}$ and $\beta = \{1.5, 2.0, 2.5\}$.

suiting for low-power sensors. We show how these policies leak information and propose new methods to address this problem.

Distributed Neural Network Inference Existing frameworks partition DNNs across multiple systems to reduce resource costs on edge devices [Banitalebi-Dehkordi et al., 2021, Kang et al., 2017, Li et al., 2019, Zeng et al., 2019]. Both DDNNs [Teerapittayanon et al., 2017] and SPINN [Laskaridis et al., 2020] introduce early exit behavior to improve distributed inference, creating distributed MeNNs. We develop a side-channel attack against the com-

munication patterns of these distributed MeNNs. We defend against this attack through new early exit policies.

Attacks on Neural Networks Common attacks against DNNs force misbehavior through adversarial noise [Biggio et al., 2013, Carlini and Wagner, 2017, Goodfellow et al., 2015, Guesmi et al., 2021, Ilyas et al., 2019, Lecuyer et al., 2019, Madry et al., 2017, Papernot et al., 2017, Shumailov et al., 2021b, Szegedy et al., 2013]. Other work induces adversarial behavior using training set poisoning [Gu et al., 2017], attacker-specified triggers [Pan et al., 2022], or batch orderings [Shumailov et al., 2021a]. Popular countermeasures against these attacks include defensive distillation [Papernot et al., 2016] and adversarial training [Goodfellow et al., 2015, Vaishnavi et al., 2022]. Further, existing proposals observe that MeNNs reduce the impact of adversarial examples [Hu et al., 2020, Kaya et al., 2019]. Previous attacks target MeNNs by crafting adversarial examples to maximize the execution cost [Haque et al., 2020, Hong et al., 2020] and using exit decisions to improve membership inference queries [Li et al., 2022]. Similar to our work, these attacks exploit early-exit behavior in neural networks. However, we evaluate how distributed MeNNs leak predictions through communication patterns.

Neural Networks and Privacy Previous systems address the privacy of DNNs through homomorphic encryption [Gilad-Bachrach et al., 2016, Liu et al., 2017] and secure two-party computation [Mohassel and Zhang, 2017]. Other methods protect DNNs using trusted execution environments [Hashemi et al., 2021, Mo et al., 2020] and differential privacy [Abadi et al., 2016b, Truex et al., 2019, Stevens et al., 2022, Wei et al., 2020]. Our work also examines DNN privacy, but we create a new attack that uses exit patterns to infer MeNN predictions. Prior work leverages power [Wei et al., 2018], electromagnetic [Batina et al., 2019, Yu et al., 2020], and timing/memory [Hua et al., 2018] side-channels to find DNN architectures and parameters. We instead use the communication patterns of distributed MeNNs as a side-

channel to uncover model predictions. We further create efficient solutions for this new privacy concern.

Side-Channel Attacks Many side-channel attacks exploit variable behavior under different inputs or operating conditions [Chen et al., 2010, Das et al., 2016, Mehta et al., 2022]. Previous work uses timing [Brumley and Boneh, 2005] and power discrepancies [Kocher et al., 1999, Messerges and Dabbish, 1999] to extract encryption keys. We also study side-channels against varying behavior, but our work focuses on MeNNs, which is new. Previous work closes side-channels through fixed resource usage or randomized behavior. BuFLO [Dyer et al., 2012] and its extensions [Cai et al., 2014a, Juarez et al., 2016, Nithyanand et al., 2014] standardize traffic patterns to prevent website fingerprinting. Other systems obfuscate compromising communication patterns in sensor networks [Apthorpe et al., 2019, Kamat et al., 2007, Kannan and Hoffmann, 2022]. Our work uses a new randomization technique to retain the accuracy and resource benefits of MeNNs.

4.8 Conclusion

This work creates a side-channel attack that exploits the communication patterns of distributed Multi-exit Neural Networks (MeNNs) with data-dependent early exiting. This side-channel allows an adversary to discover the MeNN’s predictions with over $1.85\times$ the accuracy of random guessing. We address this attack through two new exit policies: Per-Class Exiting (PCE) and Confidence-Guided Randomness (CGR). PCE uses multiple confidence thresholds to reduce information leakage with inference accuracy close to prior methods. CGR augments PCE with randomization to achieve theoretical privacy guarantees and deliver consistently better inference accuracy than exiting early uniformly at random. This attack highlights how modern inference systems must consider the privacy implications of data-dependent behavior.

CHAPTER 5

ACOUSTIC KEYSTROKE LEAKAGE ON SMART TELEVISIONS

5.1 Introduction

Side-channels exist in devices beyond embedded systems. This chapter applies insights from embedded devices to the broader class of Internet-of-Things (IoT) systems. In particular, we show how IoT devices on the market today unintentionally leak information by creating a novel attack against Internet-connected televisions (TVs).

Internet-connected TVs have experienced massive growth over the last decade. These devices, called "Smart TVs", are expected to reach over 266 million units sold globally by 2025 [Markets, 2020]. Unlike their traditional counterparts, Smart TVs allow users to browse the Web, access video streaming applications (e.g., Hulu), and purchase products. This increased functionality brings about new security risks [Aafer et al., 2021, Michéle and Karpow, 2014, Mohajeri Moghaddam et al., 2019, Huang et al., 2023, Majors et al., 2023, Zhang et al., 2022].

Users typically enter information into Smart TVs through on-screen virtual keyboards. Common platforms, such as AppleTVs [Apple, 2023] and Samsung Smart TVs [Samsung, 2023], allow users to navigate these keyboards with a hardware remote controller containing a direction pad (Figure 5.1). Users type by issuing directional commands to sequentially move a cursor between desired keys. Given the wide-ranging capabilities of Smart TVs, users enter sensitive information, such as passwords and credit card details, with these virtual keyboards [Huang et al., 2023]. Thus, Smart TVs must ensure the privacy of user keystrokes. On popular Smart TV platforms, such as Samsung's Tizen [Samsung, 2023], the default keyboard makes sounds as users type.

This work presents a new side-channel attack against Smart TV keyboards that uses the

TV’s audio to discover typed strings. In this attack, audio forms a side-channel because the TV does not intend to convey the user’s exact keystrokes through its acoustic feedback. Developing this attack requires answering four key questions:

- (Q1) How can the attack identify sounds made by the Smart TV that contain valuable information about keystrokes?
- (Q2) How can the attack use these sounds to discover when a user is typing and how they navigate the virtual keyboard?
- (Q3) How can the attack use the extracted movement information to recover user-typed strings?
- (Q4) How can the attack overcome real-world challenges, such as discovering the type of entered information (e.g., passwords, credit cards, etc.) and accounting for user typing behavior?

Our attack addresses the first question (Q1) by exploiting how different Smart TV platforms make distinct sounds. Further, platforms use a small number of sounds, and each sound is consistent because it comes from the platform itself. These properties allow the attack to identify important sounds by pre-recording sounds from common Smart TVs and using Fourier analysis to match these references against the sounds observed at runtime. This method augments standard signal processing techniques to handle distinct features of Smart TVs, such as the conflation of adjacent sounds due to rapid movements on the keyboard.

We answer the second question (Q2) by drawing on the insight that multiple popular Smart TVs make *different* sounds for the following actions: (1) moving the keyboard cursor, (2) selecting a key, and (3) deleting a character. Furthermore, compared to other actions on the TV, the key selection sound is unique to the keyboard. Thus, an adversary with auditory access can learn three critical pieces of information about user keystrokes.

- *When* a user is typing, due to sounds unique to the keyboard.

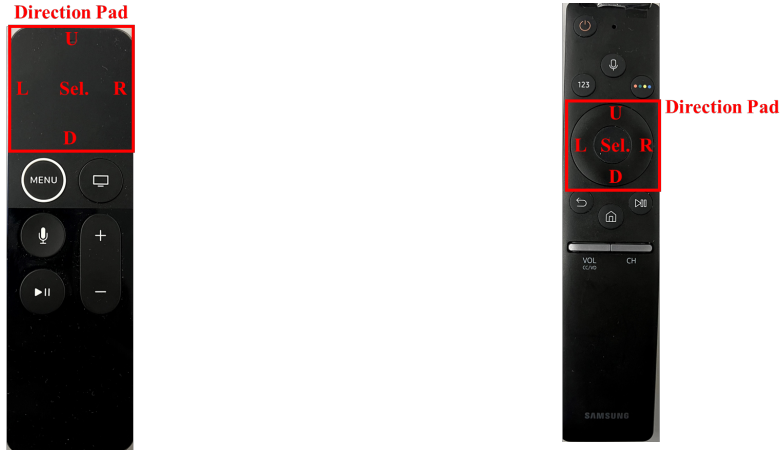


Figure 5.1: Remote controllers for AppleTVs (left) and Samsung Smart TVs (right) annotated with the direction pad.

- The *length* of the entered string, by tracking the number of selections and deletions.
- The *number* of cursor movements between selections.

The TV’s audio, however, does not reveal the *direction* of user movements on the virtual keyboard (Q3). This lack of directionality means that many possible strings may correspond to the audio for a given sequence of keyboard movements. We address this challenge by using a prior dictionary to measure the probability of typing each possible string. This design allows the attack to construct a ranked list of the likeliest results across all possible movement directions. We further leverage user typing patterns and string-length properties to infer the class of entered information, such as passwords or credit cards (Q4). This feature allows the attack to customize the prior without making additional assumptions.

Smart TV keyboards do not require users to take the shortest path between keys, so the extracted movement counts only provide an upper bound on the distance between selections (Q4). Accounting for suboptimal paths that users may traverse increases the search space and complicates keystroke recovery. We tackle this problem by exploiting user behavior through keystroke timing: we observe that users tend to pause when correcting suboptimal paths. By identifying such pauses, the attack only accounts for suboptimal paths when

needed.

We evaluate this attack against two popular Smart TV brands: Apple [Apple, 2023] and Samsung [Samsung, 2023]. These brands account for about 15.4% of the global Smart TV market, with Samsung being the most popular platform [Wire, 2021]. Importantly, the audio profiles of both TVs exactly match the properties enabling our attack. When assuming optimal typing behavior in an emulated environment, our attack recovers 99.95% of credit card numbers (CCNs), 97.65% of full credit card details (CCN, security code, ZIP code, and expiration date), and up to 99.10% of common passwords [Miessler, 2021] within 100 guesses. We extend these results to a realistic setting through a user study¹. For ten subjects typing into real applications on a Samsung TV, the attack recovers 53.33% of CCNs, 33.33% of full credit card details, and up to 60.19% of common passwords [Miessler, 2021] within 100 guesses. We further investigate this attack against users typing passwords on an AppleTV. On this platform, the attack has a top-100 password recovery accuracy of 31.00%; this lower result occurs because users take more suboptimal paths on this device. These top-100 recovery rates mean our attack can verify the extracted information against rate-limited online services [Lu et al., 2018].

Prior work has studied side-channels on user keystrokes from two angles with similar elements. First, multiple attacks use the audio from mechanical keyboards to infer keystrokes [Asonov and Agrawal, 2004, Berger et al., 2006, Compagno et al., 2017, Zhu et al., 2014]. These attacks, however, do not work on Smart TVs due to the differences in keyboard types. Mechanical keyboards allow users to jump between keys instantaneously. In contrast, Smart TVs require users to scroll across the virtual keyboard through directional commands. This difference in dynamics necessitates a new method for string recovery. Second, HomeSpy sniffs unencrypted infrared (IR) signals between Smart TVs and their remote controllers, enabling keystroke recovery [Huang et al., 2023]. Recent Smart TVs, however, have remotes

1. Our university’s institutional review board (IRB) approved this study.

that do *not* use IR [Majors et al., 2023, Zhang et al., 2022], and the HomeSpy attack does not succeed on such devices. Our acoustic attack is agnostic to the TV remote’s communication medium.

We disclosed this issue to Apple and Samsung. Apple responded, "while we do not see any security implications, we have forwarded your report to the appropriate team to investigate as a potential enhancement request to take action." The reply provides no further details. We followed up on the enhancement request to confirm our suggestion to mute the TV when typing sensitive information. We have not heard back. Samsung acknowledged the presented security problem and paid a bounty for finding this vulnerability. They further confirmed they have "multiple teams discussing the issue" to implement a long-term fix.

Overall, we make the following contribution²:

1. We identify a new side-channel attack against Smart TVs that uses audio to learn about user keystrokes.
2. We build an attack framework that uses acoustic information to identify instances of keyboard activity and extract cursor move counts between keyboard selections.
3. We create a recovery module that infers user-typed strings from the information extracted from Smart TV audio.
4. We identify and exploit common human typing behaviors on Smart TVs to improve the attack in practice.

This work identifies a novel side-channel on Smart TVs that leaks sensitive user information. This result displays another example of how Internet-connected, everyday devices must pay more attention to security [Acar et al., 2018, Fernandes et al., 2016, He et al., 2021, Ho et al., 2016].

2. The code for this work is available at <https://github.com/tejaskannan/smart-tv-keyboard-leakage>. This work appears as a conference paper in NDSS 2024 [Kannan et al., 2024]

Sounds □ = Key Select □ = System Select ▣ = Delete

SPACE a b c d e f g h i j k l m n o p q r s t u v w x y z **BACK**

1 2 3 4 5 6 7 8 9 0 · - - @ # \$ % ^ & *

ABC abc # + =

DONE

Figure 5.2: The default AppleTV keyboard for passwords. The background shows the key’s sound when pressed. The only exception is Done, which makes the sound when scrolling onto the key; selecting Done makes no sound.

Sounds □ = Key Select □ = System Select ▣ = Delete

CAPS	1	2	3	4	5	6	7	8	9	0	BACK	CLEAR	
#@!	q	w	e	r	t	y	u	i	o	p	^	*	RET
LANG	a	s	d	f	g	h	j	k	l	~	@	!	DONE
	z	x	c	v	b	n	m	,	·	?	^	-	
	SETT.	SPACE						/	<	∨	>	CANC	

Figure 5.3: The default keyboard on the Samsung Smart TV. The background denotes the key’s sound when pressed.

5.2 Background

This section provides background on Smart TVs and virtual keyboards (§5.2.1) before describing the notation we use in the rest of the paper (§5.2.2). We then discuss details of credit card transactions (§5.2.3) and present an example of acoustic keystroke leakage (§5.2.4).

5.2.1 Smart TVs and Virtual Keyboards

Smart TVs are Internet-connected televisions that support web browsers and third-party applications. Users control Smart TVs with wireless remotes that communicate over Bluetooth or infrared [Majors et al., 2023]. For common platforms [Apple, 2023, Samsung, 2023], these remote controllers contain direction pads that allow users to navigate the TV (Figure 5.1).

Users enter information into Smart TVs using an on-screen virtual keyboard. These keyboards handle sensitive information, such as passwords and credit cards, as users log into

accounts and purchase products (e.g., subscriptions). The keyboard layout depends on the Smart TV operating system (OS) and, if applicable, the current application. We focus on the default keyboards from two Smart TVs: AppleTVs (tvOS) [Apple, 2023] and Samsung Smart TVs (Tizen OS) [Samsung, 2023]. AppleTVs use an alphabetical design (Figure 5.2), and Samsung TVs use a QWERTY layout (Figure 5.3). On these platforms, users type by moving a cursor to the desired characters with the remote's directional pad. For instance, to reach `j` from `q`, users can press the "right" button six times and "down" once. Users may rapidly scroll through system-dependent actions. AppleTV remotes have a directional touchpad, allowing rapid traversal by swiping. On Samsung TVs, users can quickly scroll by holding down a direction button. Finally, Samsung TVs support horizontal wraparound; for example, moving left from `CAPS` places the cursor on `CLEAR`. There is no vertical wraparound. AppleTV supports no wraparound of any kind.

Virtual keyboards have multiple views, each containing a different character set. Users can switch views by selecting a "view-change" key, e.g., `ABC` in Figure 5.2 or `#@!` in Figure 5.3.

We observe a crucial property of both TVs:

(P1) Upon opening the keyboard, the cursor always starts on the same key in the same view.

For AppleTVs, the starting view is the lowercase letters (Figure 5.2) with the cursor on `a`. On Samsung Smart TVs, the cursor starts on `q` within the lowercase QWERTY keyboard (Figure 5.3).

Acoustic Properties

A critical property of popular Smart TVs is that their default keyboards make sounds during user interaction. In particular, both AppleTVs and Samsung Smart TVs have audio profiles with the following properties:

- (P2) The sound of moving the cursor, called `KeyMovement`, differs from that of selecting a key, called `KeySelect`.
- (P3) The `BACK` and `CLEAR` keys make different sounds than that of other keys. We call this sound `Delete`.
- (P4) The `KeySelect` sound is unique to the keyboard. When making selections outside the keyboard (e.g., choosing a video to play), the TV makes a different sound, which we call `SystemSelect`. The keyboard can make the `SystemSelect` sound on special keys (Figure 5.3).

Unlike the acoustic emanations from mechanical keyboards [Asonov and Agrawal, 2004, Zhu et al., 2014], these sounds originate from the TV OS. Thus, the sounds are consistent and user-independent. Further, each platform has its own version of each sound; the `KeySelect` sound on the AppleTV is *not* the same as `KeySelect` on the Samsung Smart TV.

Both TVs have inaudible shortcuts by pressing buttons on the remote. On AppleTVs, users can change the keyboard view. The Samsung TV allows users to capitalize characters. These actions move the cursor deterministically; e.g., if the cursor is on `d`, it moves to `D` after the capitalization shortcut.

Dynamic Keyboards

Samsung keyboards sometimes provide inline suggestions by populating neighboring locations with new characters (Figure 5.4). This design aims to reduce the user's movement on the keyboard. Moving twice in the same direction clears the suggestions. The keyboard always starts with no suggested keys. These suggestions only occur when users enter "predictable" information such as web searches. There are no suggestions when entering passwords. For passwords, the only dynamic behavior occurs when the keyboard suggests `Done` after the user reaches eight characters. AppleTV keyboards have no dynamic behavior.

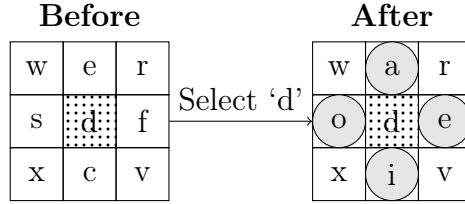


Figure 5.4: Example of dynamic key suggestions on the Samsung Smart TV upon selecting the character **d**.

5.2.2 Notation

Our attack uses move count sequences to recover strings typed on a Smart TV virtual keyboard. A *move count sequence* is an ordered list $S = [M^{(n)} = (k^{(n)}, s^{(n)}, \mathbf{t}^{(n)})]_{n=1}^N$ where $M^{(n)}$ is the n^{th} move. Each move has three items. The first, $k^{(n)}$, is the number of cursor movements to navigate from the $(n - 1)^{\text{th}}$ key to the n^{th} key. Second, $s^{(n)} \in Q = \{\text{KeySelect}, \text{SystemSelect}, \text{Delete}\}$ is the sound made upon selecting the n^{th} key where Q denotes the set of *end* sounds. Finally, $\mathbf{t}^{(n)} \in \mathbb{R}^{k^{(n)}}$ holds the times of the individual movements.³ We infer user behavior from $\mathbf{t}^{(n)}$ (§5.4.3). We denote strings as w where w_ℓ is the ℓ^{th} character and $w_{\ell:r}$ is the substring from position ℓ to r inclusive. We define a *keyboard instance* as a contiguous episode starting with opening the keyboard and ending with the string’s submission (e.g., by clicking **Done**). A move count sequence S can have zero or more keyboard instances. As an example, consider typing the string **test** on the Samsung Smart TV keyboard (Figure 5.3). With timing information omitted, one possible move count sequence S for this string is below. The last move comes from navigating to **Done**.

3. We denote vectors in boldface and scalars in plain text.

$$S = [(k^{(0)} = 4, s^{(0)} = \text{KeySelect}), (k^{(1)} = 2, s^{(1)} = \text{KeySelect}), \\ (k^{(2)} = 2, s^{(2)} = \text{KeySelect}), (k^{(3)} = 4, s^{(3)} = \text{KeySelect}), \\ (k^{(4)} = 9, s^{(4)} = \text{SystemSelect})]$$

There are an infinite number move count sequences for a single string because users can traverse suboptimal paths between keys. Similarly, one move count sequence can correspond to many strings.

We use S as an intermediate representation for the TV's audio. Our attack works by converting audio into a move count sequence S and then finding strings that match S in the keyboard layout.

5.2.3 Credit Card Details

Users type credit card details into Smart TVs when purchasing products (e.g., movies or subscriptions). For example, users must enter their payment information into the Hulu application when creating a new account on Samsung Smart TVs. These purchases are "card-not-present" transactions. Users must provide sufficient payment details to validate the transaction with a card payment network. These details often include the credit card number (CCN), expiration date, security code (CVV), and billing address. The CVV is a 3- or 4-digit sequence of pseudo-random numbers. Popular Smart TV applications, such as Hulu, validate the billing address through a 5-digit ZIP code. Therefore, the 5-tuple of (CCN, expire month, expire year, CVV, ZIP) is sufficient to purchase products, making this information financially valuable to an attacker. We call this 5-tuple the *full credit card details*. We focus on payments in the United States, though the formats of other countries are conceptually similar.

CCNs are generally 15- or 16-digit strings where the first digit identifies the card issuer.

We focus on three popular issuers: American Express (AMEX), Visa, and Mastercard. Apart from the last digit, the remaining numbers identify the issuing institution and user account. The final digit ensures the entire CCN satisfies a checksum under Luhn’s algorithm. This checksum is essential to our attack’s performance when recovering CCNs (§5.5.2, §5.6.2).

To verify stolen credit card information, an attacker must execute a transaction against a card payment service. If the service accepts the transaction, the attacker has valid payment details. Upon rejection, we assume the attacker only knows that *at least* one field is incorrect. Further, we assume all five fields must be correct for a valid transaction. This assumption is conservative, as some services may only validate a subset of fields [Scaife et al., 2018].

5.2.4 Example of Acoustic Keystroke Leakage

We provide an example of acoustic keystroke leakage by typing the string `test` on the Samsung TV (Figure 5.3) and recording the audio. Figure 5.5 shows the audio produced by the TV (§5.6.1 describes the experimental setup). With prior knowledge of the TV’s sounds, we identify the user’s keyboard actions from the raw audio (P2). This identification yields the move count sequence S from §5.2.2. Visually, this example highlights the distinctive nature of Smart TV sounds.

To recover the typed string, an attacker can combine the keyboard layout (Figure 5.3) with S . Assuming the user takes shortest paths, the first character is $w_0 \in \{\mathbf{t}, \mathbf{4}, \mathbf{f}, \mathbf{c}, \hat{}, \mathbf{!}\}$ because the cursor starts on \mathbf{q} (P1). Then, the attacker considers keys at a distance of $k^{(1)} = 2$ from each possible w_0 . Iteratively continuing this process yields possible strings such as `ft6f`, `test`, `tukt`, and so forth. The attack can identify the most likely result using a prior dictionary over all possible strings. For instance, the attacker can correctly recover `test` under an English prior. This general method allows an attacker to recover keystrokes from a Smart TV’s audio.

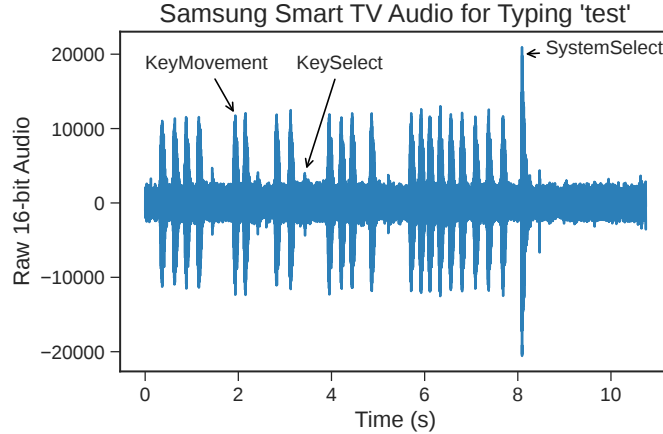


Figure 5.5: Example audio from the Samsung TV when typing the string `test`.

5.3 Threat Model

We consider an attacker with passive audio access to a Smart TV running Samsung’s Tizen OS [Samsung, 2023] or AppleTV’s tvOS [Apple, 2023]. We assume there is only one TV in the target location. An attacker can gain this access to the TV in one of two ways:

1. The adversary can hijack an adjacent device containing a microphone. Such devices are prevalent due to the growth of smart speakers (e.g., Amazon Echo) [Insights, 2022]. It is reasonable to assume that these devices have vulnerabilities exposing their microphone feed; prior attacks have both turned such speakers into wiretaps [Barnes, 2017, Kunze, 2022] and compromised other smart devices [Acar et al., 2018, Antonakakis et al., 2017, Fernandes et al., 2016, Ho et al., 2016, Sami et al., 2020]. Depending on the vulnerability, an adversary can exploit these nearby devices remotely, launching our attack even without physical access to the target TV. These assumptions correspond to the threat model of previous keystroke attacks on Smart TVs [Huang et al., 2023].
2. The attacker can place a malicious microphone near the target TV. This method requires stronger capabilities, as this adversary generally needs physical access to place the microphone. We consider this threat despite this stronger assumption because it

remains reasonable for venues with transient populations, such as hotel rooms, vacation homes, and other short-term rental facilities. This vector may be possible without physical access if the TV is audible in an adjacent space (e.g., through the wall of an apartment) or if the attacker leverages long-range (e.g., laser) microphones [Muscatell, 1984, Sami et al., 2020].

We emphasize what the adversary does *not* assume. The attacker makes no assumptions about the type of information (e.g., passwords, credit card details, etc.) or even that the user is typing at all. The attacker must determine when the user is typing and the class of entered information. When the user is typing, the attacker does not assume the user takes an optimal path between keys. Further, the adversary has no other knowledge about the user. This assumption is conservative; for example, knowing personal information improves password guessing [Wang et al., 2016]. Finally, the attacker must validate inferred keystrokes against online services (e.g., by directly entering a password guess into an online site). The adversary cannot access leaked information on which to validate results offline. This setting mirrors online password guessing and is more challenging due to rate limits and service lockouts [Wang et al., 2016].

We compare this threat to four related attacks: video-based attacks, attacks on voice inputs, attacks on infrared remotes, and attacks on virtual remotes.

Video Similar to shoulder surfing [Wiedenbeck et al., 2006], an adversary with video access to the Smart TV’s screen can read keystrokes. Gaining reliable video access, however, is harder than capturing only audio because the attacker needs an unobstructed view of the TV. Therefore, video access is sensitive to the malicious device’s location. The audio threat is stealthier because it can operate anywhere near the TV within audible range, allowing the recording device to occupy less conspicuous places. Finally, the audio attack is not sensitive to small changes in the location of the TV or microphone.

Voice Inputs Some Smart TVs support voice inputs. This method replaces virtual keyboards by allowing users to speak their desired string into the remote controller. Voice inputs leak information through acoustic signals, as attackers with audio access to the target TV can use speech recognition [Graves et al., 2013] to discover the entered string. Smart TVs, however, do not support voice inputs for all information types. For example, Samsung Smart TVs block voice inputs for password fields; instead, the TV forces users to enter passwords with the virtual keyboard. Thus, an attacker who only targets voice inputs cannot steal all forms of highly sensitive information. Our attack targets the virtual keyboard’s audio and can discover private information such as passwords.

Infrared Remotes HomeSpy steals user keystrokes by sniffing infrared (IR) signals between Smart TVs and remote controllers [Huang et al., 2023]. However, this attack only works for IR remotes. Newer remotes communicate over Bluetooth [Majors et al., 2023, Zhang et al., 2022], and HomeSpy does not succeed on Bluetooth remotes. In contrast, the audio attack is agnostic to the remote control communication protocol.

Virtual Remotes SPOOK exploits vulnerabilities when pairing virtual remotes to steal Smart TV keystrokes [Majors et al., 2023]. Enforcing human attestation during the pairing process stops this attack. This patch, however, does not thwart the considered audio threat.

5.4 Acoustic Attack Framework

This section describes our acoustic attack against Smart TV keyboards. The attack consists of two core modules (Figure 5.6). The first component (§5.4.1) uses the Smart TV’s audio to identify the platform (§5.4.1) and extract move count sequences (§5.4.1). The module then isolates instances of a user typing (§5.4.1). The second module uses each isolated move count sequence to recover the typed string (§5.4.2). This module first infers the type of entered information (e.g., passwords, credit card details, etc.) (§5.4.2). From this inference,

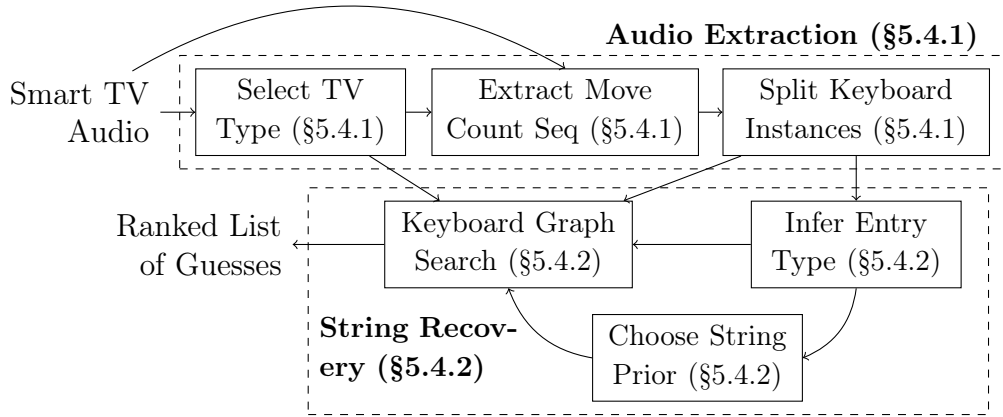


Figure 5.6: The keystroke acoustic attack framework.

the attack customizes a prior dictionary over possible strings (§5.4.2). We combine this prior with the keyboard layout to guess the likely typed strings (§5.4.2). This section concludes by discussing improvements based on timing patterns (§5.4.3).

The attack acts on fixed-length audio recordings. This property is necessary to identify the platform (§5.4.1) and information (§5.4.2) types. The attacker can obtain such recordings by splitting the microphone feed during prolonged silence (e.g., at night). The attack automatically identifies and isolates the instances of users typing within the larger recording. This property implies the adversary cannot launch the attack in real-time. However, this limitation is manageable as it enables the attacker to make minimal assumptions. Further, the value of sensitive information, such as passwords and credit cards, does not significantly diminish over these delays.

5.4.1 Audio Extraction

TV Type Selection

The first step in the attack pipeline uses acoustic properties to identify the Smart TV platform. Different Smart TVs use unique audio profiles, so the TV’s sounds fingerprint the system. On the AppleTV and Samsung Smart TV, the `KeySelect` sounds uniquely identify

keyboard usage (P4). Thus, for a given recording, we match instances of the `KeySelect` sound for each platform (§5.4.1 describes this matching). We classify the TV type as the platform with the most matching `KeySelect` sounds over the recording’s duration. When no matches exist, the platform type is unknown, and we abort the attack. Such cases indicate background noise, an unsupported TV, or no keyboard use.

Extracting Move Count Sequences

Smart TVs make distinctive sounds (P2), (P3), leading to keyboard interactions with audio such as that of Figure 5.5. This section describes a procedure to separate and classify the TV’s relevant sounds (Q1). This process extracts a move count sequence, S , from raw audio (Q2).

From Figure 5.5, the Smart TV’s sounds appear as regions with high amplitude. Thus, the module finds each amplitude peak and identifies the surrounding region as a candidate sound. This process follows prior acoustic attacks against mechanical keyboards [Berger et al., 2006].

The module must then classify each candidate sound as one of `KeySelect`, `SystemSelect`, `KeyMovement`, `Delete`, or `Unknown`. We use a nearest-neighbor classifier due to the consistency of Smart TV sounds over time. We pre-collect recordings for each sound on the target Smart TV platforms. We call these recordings the reference sounds. At runtime, the framework creates the spectrogram of the candidate and reference sounds using a Fourier transform length of 1,024, a segment length of 256, and 32 points overlapping between segments. We then compute the L1 distance between the candidate and reference spectrograms. We select the proper reference sounds using the inferred TV type (§5.4.1). The module classifies the candidate based on the lowest distance. If the best distance exceeds a threshold, the sound is `Unknown` and therefore unrelated to the TV. We empirically determine these thresholds for each reference TV sound to filter out false positives on a set of training examples.

Three challenges arise with this nearest-neighbor approach.

- (a) *Dimension Mismatch*: The candidate and reference spectrograms may have a different number of timesteps. This dimensionality mismatch prevents computing the L1 distance directly. We solve this issue by instead sliding the smaller spectrogram over the larger one. We return the minimum sliding window distance.
- (b) *Background Noise*: The L1 distance is sensitive to background noise. We mitigate this problem by limiting the comparison to the frequency bands which contain each reference sound. Further, we min-max normalize each spectrogram and mask out normalized values below a threshold τ . We empirically set $\tau = 0.7$ based on our training examples. This masking avoids comparing low-amplitude noise in the target frequency bands.
- (c) *Rapid Scrolling*: Users can quickly scroll across keys (§5.2.1). This action generally creates unique sounds for each movement. These movements, however, occur in rapid succession and get clipped as a single candidate. We deduplicate these movements by finding the spectrogram peaks [Wang, 2006] and counting the number of peaks at frequencies tuned for the `KeyMovement` sound. We use the number of peaks to identify the number of individual movements.

After classifying each sound, the framework builds the move count sequence S . The n^{th} move $M^{(n)}$ completes with $s^{(n)}$, the n^{th} instance of an end sound (§5.2.2). Upon observing $s^{(n)}$, we count the number of cursor movements between the $s^{(n-1)}$ and $s^{(n)}$; this count is $k^{(n)}$. Finally, we set the times $\mathbf{t}^{(n)}$ using the amplitude peak times for each detected cursor movement in $M^{(n)}$. These extracted moves $M^{(n)}$ form the move count sequence for the full recording.

This method produces accurate results. For users typing credit card details (§5.6.1), the module identifies the correct number of movements on 98.95% of instances and misses only a

single selection. This algorithmic approach shows an adversary can automate the extraction and launch the attack at scale. We emphasize that our contribution is not in developing new signal processing techniques. Indeed, there exist other methods for audio extraction [Berger et al., 2006, Wang, 2006], and such techniques may exhibit more advantageous properties (e.g., resistance to background noise). Through our procedure, we instead show that it is *possible* to identify Smart TV sounds algorithmically. We find that an attacker can then leverage this information to recover keystrokes (§5.6). Note that our framework supports any accurate audio identification procedure; changing the audio extraction method does not alter the essence of the attack.

Splitting Keyboard Instances

The move count sequence S (§5.4.1) encodes the user’s behavior over an entire recording. Each recording can have zero or more keyboard uses. The attack must isolate each keyboard instance (§5.2.2) to recover the individual user-typed strings. This phase produces disjoint sequences S_1, \dots, S_ℓ such that $\bigcup_{r=1}^{\ell} S_r \subseteq S$ where each S_r holds a single keyboard instance. We perform this splitting using both acoustic and timing information. The details depend on the platform, as discussed below.

Apple The AppleTV keyboard has no dynamic behavior (§5.2.1) and requires users to select `Done` upon completion (Figure 5.2). `Done` is the only key to make the `SystemSelect` sound. We can thus split S using moves with an ending sound of `SystemSelect`. We only retain splits S_r with at least one move ending in a `KeySelect`; otherwise, the user did not interact with the keyboard (P4).

Samsung This platform is more complicated for two reasons. First, `Done` is not the *only* key to make the `SystemSelect` sound (Figure 5.3). Second, the keyboard can dynamically suggest `Done` (§5.2.1), and this suggestion makes the `KeySelect` sound. Therefore, split-

ting the move count sequence using `SystemSelect` sounds is incorrect. Instead, we use timing information based on the insight that applications incur noticeable latency upon submitting a string (e.g., executing a search query). We identify these delays as outliers in the times between adjacent moves. In particular, the cutoff of $\text{avg}(\text{move_diffs}) + 1.5 \cdot \text{stddev}(\text{move_diffs})$ creates the proper splits in almost all cases (§5.6.3) where `move_diffs` is the array below for all $n \in \{1, 2, \dots, \text{len}(S) - 1\}$.

$$\text{move_diffs}[n] = t_1^{(n+1)} - t_{k^{(n)}}^{(n)} \quad (5.1)$$

This method can make mistakes. For example, the attacker will incorrectly split keyboard instances if a user takes a long pause. However, such failure cases rarely occur in practice (§5.6.3). We omit splits without any moves ending in `KeySelect`.

Credit cards are an exception to this time-based approach. Credit card information is entirely numeric, so `Done` is the only required key that makes the `SystemSelect` sound (Figure 5.3). Further, when selecting numbers, the keyboard never dynamically suggests `Done`. Thus, we can split credit card sequences using the `SystemSelect` sound. The challenge, however, is the attack has no prior knowledge that a user is entering credit card details. We address this problem by combining both methods. We first split S using `SystemSelect` sounds and identify any credit card entries (§5.4.2 explains this identification). We then remove the credit card sequences and re-split the remainder using timing. This combined approach addresses the dynamic behavior of Samsung Smart TV virtual keyboards.

5.4.2 *String Recovery*

Inferring Entry Types

Users enter various classes of information into Smart TVs; we focus on three types: passwords, credit cards, and English words. The first two classes are highly sensitive, and the

last class encapsulates inputs such as web searches. These information types have unique details, and the attack should customize the string recovery to each class to optimize performance (§5.4.2). The attack infers the information type to make these customizations without introducing additional assumptions.

We detect credit card information on the Samsung TV by counting the string lengths (P3) of consecutive keyboard instances. Forms accepting credit card details collect five pieces of information with distinctive lengths (§5.2.3). We use these lengths to fingerprint credit card details and look for consecutive instances matching the sizes of each field. We assume the CCN comes first, and the year follows the month; otherwise, this matching is order-insensitive. For a given match, we identify each field by its length and position.

If a keyboard instance is not in a credit card field, we must distinguish whether it is a password or an English word. To make this distinction, we leverage how the Samsung TV employs dynamic suggestions when expecting English words (§5.2.1). These suggestions are *not* present for passwords, allowing the attack to identify the information type from this discrepancy. We use a Random Forest classifier [Breiman, 2001] to determine the presence of dynamic keyboard behavior for each instance S_r . The Random Forest provided better results than other considered models, such as gradient-boosted decision trees. The input features are a histogram of movement counts in S_r , which are valuable because suggestions reduce the average distance between selections. We train this model on generated move count sequences for passwords and English words on the Samsung keyboard (Figure 5.3). We mimic suggestions by manually recording the TV’s suggestions after the first selection; after, we use the likeliest characters from an English dictionary [Semenov, 2022]. We bias this classifier toward passwords due to their greater value for an attacker. We classify S_r as an English word if the model’s prediction probability exceeds 0.6. We set this cutoff empirically to reach over 99% recall on passwords in our validation set.

We limit our analysis of AppleTVs to passwords. AppleTVs have no browser, so the

only English inputs are video titles or application-specific searches. These searches are far less valuable than passwords. Further, there is no location to enter credit card details, as all payments occur through the user’s Apple account. Thus, on AppleTV, the framework assumes all keyboard inputs are passwords.

Choosing the Prior String Dictionary

Smart TV audio does not provide the direction of keyboard movements. Thus, a move count sequence corresponds to many possible strings, and the attack must distinguish these possibilities. We make these determinations using a prior dictionary over possible strings. The prior depends on the information type. For example, credit card numbers (CCNs) are numeric, while passwords have larger character sets. The attack customizes the prior using the inferred information type (§5.4.2).

We design priors for each string type. The CCN prior enforces the prefixes of AMEX, Visa, and Mastercard (§5.2.3); the remaining digits appear uniformly at random. The security code (CVV) prior consists of digits occurring at random. For efficiency reasons, we do not materialize the full CCN and CVV priors. The expiration month and year priors enforce valid dates up to 2035. We construct the ZIP prior using the 33,120 United States ZIP codes from 2019 and weigh ZIPs by population. We follow previous work by building an N-gram model [Ma et al., 2014] for passwords from leaked datasets [Miessler, 2021]. The English prior uses word prefixes from the Wikipedia corpus [Semenov, 2022]. We weigh prefixes by frequency and only keep words with at least 100 appearances. This prior has 95,892 words.

These string priors represent options that work well in practice (§5.6). We emphasize that an adversary can change these priors without fundamentally altering the attack framework.

Keyboard Graph Search

The attack recovers strings from move count sequences S_r by performing a variant of Dijkstra’s algorithm on the Smart TV’s keyboard graph (Q3). This keyboard layout is known from the inferred TV type (§5.4.1). Within Dijkstra’s algorithm, we use search states of the form below.

$$\mathbf{z} = (\mathbf{key}, \mathbf{str}, \mathbf{move_num}, \mathbf{keyboard_view}) \quad (5.2)$$

When expanding the state \mathbf{z} , we use the move $M^{(n)}$ (§5.2.2) from S_r where $n = \mathbf{z.move_num}$. We then find the set of neighbors V by retrieving the keys in the current keyboard view at a distance $k^{(n)}$ from $\mathbf{z.key}$ which make the sound $s^{(n)}$. We consider neighbors at the distance $k^{(n)}$ both with and without wraparound, as users often ignore this feature. We create the candidate string for each neighbor $v \in V$ by appending v to $\mathbf{z.str}$ and accounting for the key’s action. This process is often concatenation, though special keys (e.g., **BACK**) have different behavior. We compute the edge weight for this neighbor using the string prior (§5.4.2).

$$\mathbf{weight} = -\log\left(\frac{\text{Count}(\mathbf{candidate_str}, \text{Prior})}{\text{Count}(\mathbf{z.str}, \text{Prior})}\right) \quad (5.3)$$

We give special keys (e.g., **CAPS**) a weight of zero and ignore neighboring keys with zero count. We use Dijkstra’s algorithm with these weights to find the maximal path. With this method, the score for each string w is logically equivalent to the following probability under the prior.

$$p(w) = p(w_1) \cdot \prod_{\ell=2}^{\text{len}(w)} p(w_\ell | w_{1:(\ell-1)}) \quad (5.4)$$

We construct guesses upon exhausting the available moves in S_r , i.e., when `z.move_num = len(Sr)`. Before producing a guess, we validate the string based on the information type. This validation is crucial for credit cards, as each CCN must pass a checksum (§5.2.3). We continue searching until producing a ranked list of L guesses. The search always starts from the keyboard’s fixed start key (P1).

This graph search forms the backbone of the string recovery process. However, to achieve a useful attack, the procedure must also overcome the following four challenges: (1) suboptimal paths, (2) errors in audio extraction, (3) multiple keyboard views, and (4) dynamic suggestions. We discuss each of these challenges below.

Suboptimal Paths As presented, the search assumes that users take an optimal (i.e., shortest) path between keys. This assumption manifests by finding neighboring keys at a distance $k^{(n)}$. Users, however, may not take optimal paths (Figure 5.7), and considering only optimal paths can cause the recovery to miss the true string (Q4). We handle suboptimal movements by expanding the search at move $M^{(n)}$ to find keys within a distance range of $I = [k^{(n)} - d^{(n)}, k^{(n)} + d^{(n)}]$ (§5.4.3 describes how to set $d^{(n)}$). We discount suboptimal paths using the factor $\gamma^{|d-k^{(n)}|}$ where $d \in I$ is the considered number of movements. We multiply this discount factor with the string probability before applying the logarithm in Equation 5.3. This procedure allows the search to consider suboptimal movements without penalizing optimal paths.

Audio Extraction Errors The audio extraction can make mistakes, and these errors generally occur when conflating adjacent movements (§5.4.1). We handle these errors as suboptimal paths by setting the tolerance $d^{(n)}$ to be at least the number of rapid scrolls (§5.4.3). With this tolerance, the search can still consider the correct neighbor even when miscounting the number of movements.

Keyboard Views Virtual keyboards have multiple views, and users can inaudibly change views with the remote controller (§5.2.1). Such changes are not present in S_r . We handle these view changes using an exhaustive search. When adding a state to the search queue, we also add all states reachable through an inaudible view change. For example, the Samsung TV allows users to capitalize keys with the remote. When adding the state $(d, \text{'str'} \parallel d, \text{move_num, lower})$, we also push $(D, \text{'str'} \parallel D, \text{move_num, upper})$ onto the search queue where \parallel is concatenation. This method works because the cursor moves deterministically across view changes (§5.2.1).

Dynamic Suggestions The Samsung TV contains a final complication: dynamic suggestions (§5.2.1). The suggestions are unknown beforehand, making them a quantity the attacker must predict. We perform this prediction using the most common characters in the English dictionary that follow the current state’s string. The keyboard suggests up to four characters (Figure 5.4). Since our predictor does not exactly match the Smart TV, we use the top six guessed characters. We add these keys to the neighbor set V when the number of movements is at most four, as the suggestions are adjacent to the cursor (Figure 5.4). Finally, with dynamic suggestions, we always use a tolerance $d^{(n)} \geq 1$ because users make an additional movement to clear the suggested keys (§5.2.1).

5.4.3 User Timing Patterns

Through a user study (§5.6), we observe that people exhibit consistent timing patterns when typing on Smart TVs. This section highlights two beneficial aspects of keystroke timing. The attack first leverages timing to detect suboptimal paths (§5.4.3). Second, keystroke timing can provide information on movement directions (§5.4.3).

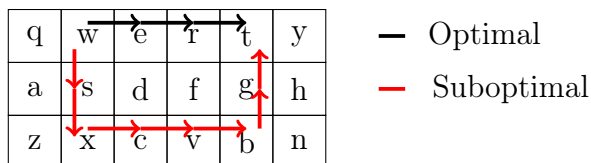


Figure 5.7: Optimal and suboptimal paths between **w** and **t**.

Suboptimal Paths

Virtual keyboards do not enforce that users take an optimal path between keys (Figure 5.7). As described, the string recovery module supports suboptimal paths by expanding the set of considered neighbors using the tolerance terms $d^{(n)}$ (§5.4.2). The attack must determine how to set each $d^{(n)}$.

One possible method is to set $d^{(n)}$ to a fixed $D > 0$ for every move $M^{(n)}$. However, this approach leads to low recovery performance when the information type has no strong prior. For example, CCNs and CVVs place near-uniform priors over all digits (§5.4.2). Under this strategy, the procedure scores strings equally if they use the same number of suboptimal paths. This equality occurs independent of *when* the suboptimal moves happen due to the uniform prior. Therefore, the order of guesses depends on the arbitrary tie-breaking scheme between equal priorities on the search queue.

We create a better method based on typing behavior. Through a user study (§5.6.1), we find that people tend to *pause* when correcting a suboptimal path. This indicator, however, is noisy, and we address this noise using an iterative solution. For each $M^{(n)}$, we compute $u^{(n)}$, the largest time difference between adjacent movements for all $n \in \{1, 2, \dots, \text{len}(S)\}$

$$u^{(n)} = \max_{q \in \{2, \dots, k^{(n)}\}} (t_q^{(n)} - t_{q-1}^{(n)}) \quad (5.5)$$

We sort the moves in decreasing order of their maximal delays $u^{(n)}$. We use this ordering to consider increasingly more moves with suboptimal paths. This process works as follows. We begin the search (§5.4.2) with $d^{(n)} = 0 \forall n$. If this search ends with fewer than L guesses,

we expand it by selecting the move index $n' = \arg \max_n u^{(n)}$. We set $d^{(n')} = D$ and the remaining $d^{(n)}$ to 0. We re-execute the search with these parameters. If this process again produces fewer than L results, we expand the candidate suboptimal paths by selecting the top two highest-delay moves. We continue this expansion until reaching L results. An individual search may produce less than L results because (1) strings can have a zero count in the prior, and (2) results may not pass the information type's validity check (e.g., CCN checksums). Note that we continue to account for rapid scrolls and dynamic suggestions (§5.4.2) on the remaining $d^{(n)}$ at each step. Overall, this timing-based design better identifies where suboptimal paths occur (§5.6.5).

Direction Inference

Users can rapidly scroll across keys using the TV's remote (§5.2.1). Each scroll constitutes movements in a single direction; to change direction, users pause to switch buttons on the remote. Further, neither AppleTV nor Samsung keyboards support vertical wraparound. Therefore, if the user rapidly scrolls across at least four keys, the user moves horizontally with high probability. This cutoff follows from the number of keyboard rows (Figures 5.2, 5.3). Inferring horizontal directions allows the attack to reduce the search space. For example, if we detect four horizontal movements from **q** on the keyboard in Figure 5.3, then the neighbor set is $V_{dir} = \{\mathbf{t}, \hat{\ } \}$. Without directions, the neighbor set is $V = \{\mathbf{t}, \mathbf{4}, \mathbf{f}, \mathbf{c}, \hat{\ }, \mathbf{!}\}$.

The attack infers these directions by identifying rapid scrolls through movement timing. Consider the n^{th} move with times $\mathbf{t}^{(n)}$ (§5.2.2). We compute the time differences between adjacent movements, $\boldsymbol{\alpha}^{(n)}$, and create the cutoff $c^{(n)} = \text{median}(\boldsymbol{\alpha}^{(n)})$. We assign directions by observing windows of four adjacent movements. If all time differences in a window are at most $c^{(n)}$, these movements are considered part of a rapid scroll and given a direction of "horizontal." The remaining movements are in "any" direction. We ignore the inferred directions for suboptimal paths because we do not know *where* the suboptimal movement

occurs. We find that direction inference never harms the recovery (§5.6.5).

5.5 Attack Results in Emulation

We first assess the efficacy of the string recovery module (§5.4.2) under ideal conditions. This recovery is nontrivial because a single move count sequence can refer to many strings (§5.2.2, §5.4.2). This section first describes the emulation setup (§5.5.1) before presenting the results for recovering credit cards (§5.5.2) and passwords (§5.5.3). We conclude this section by evaluating the impact of the keyboard layout on the attack’s ability to recover strings (§5.5.4).

5.5.1 Setup

The emulation environment evaluates string recovery (§5.4.2). We algorithmically generate move count sequences using the Smart TV’s keyboard graph. This sequence uses optimal paths apart from randomly choosing when to use available wraparound. We focus on two types of private information: credit cards and passwords.

Credit Card Recovery

We use credit card details with five fields: credit card number (CCN), expiration date (mm/yy), security code (CVV), and ZIP code (§5.2.3). When entering these details into existing applications, the Samsung Smart TV provides users with the full QWERTY keyboard (Figure 5.3). We randomly generate semantically valid fake Visa, Mastercard, and AMEX CCNs. We select random expiration months and choose arbitrary years between 2023 and 2033. We generate CVVs as three (Visa and Mastercard) or four (AMEX) digit random numbers. Finally, we randomly select valid ZIP codes weighted by population. We create 6,000 entries, with 2,000 for each provider.

The search outputs a list of guesses for each field. The adversary, however, can only validate the full details by getting every field correct (§5.2.3). We compute the rank for the full details by first exhaustively searching over the top 10 CCNs, three CVVs, three ZIPs, two months, and two years in this order. If we find no complete match, we iteratively expand the search using the following cutoffs: (CCN: 30, CVV: 5, ZIP: 5, Month: 2, Year: 2), (CCN: 100, CVV: 12, ZIP: 12, Month: 3, Year: 3). We select this order of fields based on our uncertainty in the corresponding information. For example, CCNs are the most uncertain as they are long, nearly-random numeric strings.

We compute the attack’s results up to 250 CCN guesses and 5,000 guesses for the full details. These cutoffs yield a top- K accuracy which is the rate at which the correct result is in the first K guesses. In this notation, K denotes the guess cutoff i.e., the number of guesses allowed by the adversary. We display $K \in \{1, 5, 10, 50, 100, 250\}$ for CCNs and $K \in \{1, 10, 100, 1000, 2500, 5000\}$ for the full details. In practice, attackers validate the results against rate-limited online services (§5.3). From prior work on password guessing, an attacker can make at least 100 guesses against a single service with a sufficient delay between trials [Lu et al., 2018]. Unlike passwords, payment details are global, and an attacker can circumvent the limits of an individual site by simultaneously testing against different services. Thus, we focus on the top-1000 accuracy for the full details.

Password Recovery

We evaluate the recovery of passwords from the 2014 PhpBB password leak [Miessler, 2021]. To make this set realistic, we follow NIST guidelines [Grassi et al., 2017] and only include passwords with at least eight characters. We purposefully select diverse strings by enforcing that 25% of the passwords have at least one special character, number, uppercase letter, or lowercase letter. We match the credit card benchmark by selecting 6,000 passwords. We experiment with two different 5-gram language models [Ma et al., 2014] using: (1) the same

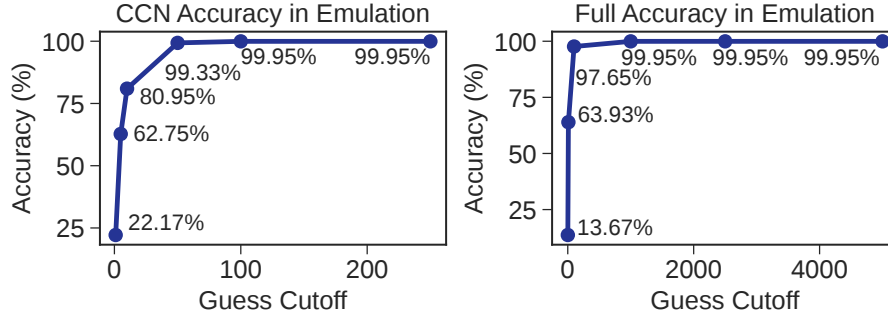


Figure 5.8: Accuracy on CCNs (left) and full credit card details (right) in emulation on the Samsung Smart TV.

PhpBB set and (2) the RockYou password leak [Miessler, 2021].

We measure the performance for the guess cutoffs $K \in \{1, 5, 10, 50, 100, 250\}$ and focus on the top-100 accuracy. This latter cutoff makes our attack applicable to online settings [Wang et al., 2016]; from prior work, attackers can make at least 100 password guesses against a majority of popular websites [Lu et al., 2018].

5.5.2 Credit Card Recovery

We attack credit card details on the Samsung TV. Figure 5.8 shows the Top-K accuracy on the CCN and the full details. The CCN alone can be valuable for attackers, especially when vendors do not validate all payment information [Scaife et al., 2018]. The attack discovers both CCNs and full payment details. For CCNs, the attacker reaches an 80.95% top-10 and 99.95% top-100 accuracy. On the full details, the attacker finds 63.93% (top-10) and 99.95% (top-1000) of the full payment information. This top-1000 accuracy means the attack can use optimal move count sequences to find and validate 99.95% of payment details against rate-limited services (§5.5.1).

We highlight two aspects of these results. First, the attack succeeds on all three providers. On the full details, the top-10 accuracy is 61.55% (Visa), 57.00% (Mastercard), and 73.25% (AMEX). The AMEX results are better because the CCN has only 15 digits. The top-1000

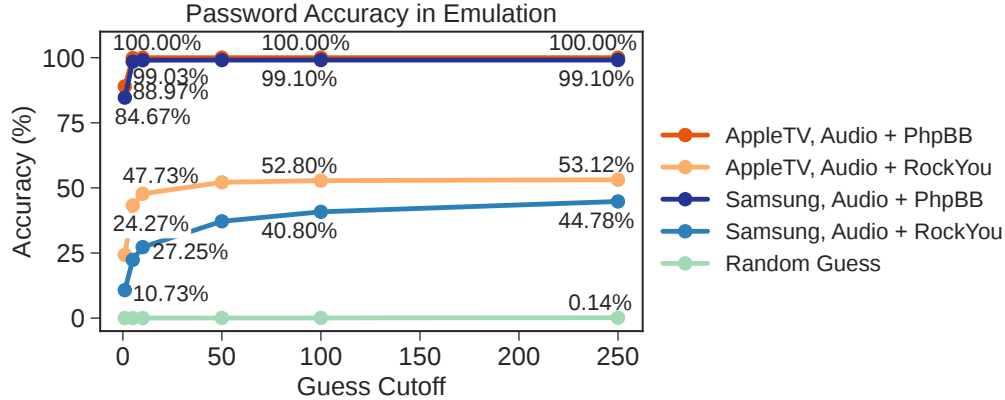


Figure 5.9: Password recovery in emulation. The data labels show the Top- K accuracy for $K \in \{1, 10, 100, 250\}$.

accuracy is above 99.90% for all providers. Second, on average, only 16.26% of the potential CCN guesses satisfy the checksum (§5.2.3). This result highlights the importance of the checksum; without this validation, the CCN ranks would be roughly $6\times$ larger.

5.5.3 Password Recovery

We further recover passwords on both platforms. Figure 5.9 shows the results. The baseline accuracy is the expected result of randomly guessing from the set of 184,388 PhpBB passwords. We discuss four takeaways.

First, with the PhpBB prior, the attacker reaches at least 84.67% top-1 and 99.03% top-10 accuracy on both TVs. These results vastly exceed random guessing, confirming the value of optimal move count sequences. With these results, the adversary can reliably verify the discovered passwords in rate-limited contexts [Lu et al., 2018].

Second, the string prior plays a significant role in the attack’s efficacy. Under the RockYou prior, the top-10 accuracy drops to 47.73% (AppleTV) and 27.25% (Samsung). These results occur due to less string overlap between the PhpBB and RockYou lists. Nevertheless, knowing the move count sequence allows the attack to consistently outperform random guessing by over $330\times$ even when the target password may not be in the string prior.



Table 5.1: Emulated top- K password accuracy under the PhpBB prior for different keyboard layouts.

Layout	Guess Cutoff (Top- K)		
	$K = 1$	$K = 5$	$K = 10$
ABC	85.40%	99.55%	99.93%
AppleTV	88.97%	99.85%	100.00%
Samsung	84.67%	98.48%	99.03%
Random Guess	$5.4 \times 10^{-4}\%$	$2.7 \times 10^{-3}\%$	$5.4 \times 10^{-3}\%$

Third, the attack performs better on the AppleTV for two reasons. First, the AppleTV keyboard (Figure 5.2) limits the number of movement directions compared to the Samsung keyboard (Figure 5.3). Thus, the attack has fewer neighbors to consider for each move. Second, on Samsung instances, we must infer whether the keyboard uses dynamic suggestions (§5.4.2). This inference has a 99.23% recall on passwords. The misclassified cases lead to incorrect guesses, as the attack wrongly uses the English prior on a keyboard with suggestions. Nevertheless, the attack still displays high accuracy on Samsung systems. Section 5.5.4 contains further experiments measuring the keyboard layout’s impact on string recovery.

Finally, the attack finds passwords with diverse character sets. On the AppleTV under the PhpBB prior, the top-1 accuracy is 92.33% (special), 92.80% (numeric), and 76.64% (uppercase). For the Samsung Smart TV, these figures are 88.60% (special), 89.93% (numbers), and 73.68% (uppercase). The uppercase accuracy is lower due to inaudible case changes (§5.2.1); the lowercase version of the password has the same move count sequence and often a higher score.

In total, the attack reliably discovers user keystrokes when given optimal move count sequences.

Sounds  = Key Select  = System Select  = Delete





a	b	c	d	e	f	g	 BACK
h	i	j	k	l	m	n	 123@!
o	p	q	r	s	t	u	
v	w	x	y	z	-	'	
 SPACE		 DONE					

Figure 5.10: The ABC keyboard layout. The background denotes the key’s sound when pressed.

5.5.4 Impact of the Keyboard Layout

We extend our evaluation by considering the efficacy of string recovery on different keyboard layouts. In addition to the default keyboards on AppleTV (Figure 5.2) and Samsung (Figure 5.3) devices, we study a third keyboard shown in Figure 5.10. This design resembles that of the YouTube application on Samsung Smart TVs. This keyboard does not have dynamic suggestions. We emphasize that the YouTube application mutes the keyboard and is not susceptible to the presented attack. Nevertheless, we include this layout to evaluate whether any popular keyboard designs form an effective defense in isolation. In all cases, we assume the keyboard layout remains fixed and is known to the attacker (§5.3).

Table 5.1 shows the top- K accuracy ($K \in \{1, 5, 10\}$) against common passwords under the PhpBB prior. We perform this evaluation in emulation and use the same password list as in §5.5.3. We observe that the keyboard layout has only a small impact on the attack’s accuracy. For all three keyboard designs, the attack yields a top-1 accuracy above 84%; this figure is orders of magnitude higher than random guessing. We note that the AppleTV layout shows the highest recovery accuracy. We hypothesize this trend stems from the linear keyboard design in which users only move in three directions to navigate between characters (Figure 5.2); e.g., when the cursor is on **a**, the user can only move "left", "right," or "down."

In contrast, the other two layouts allow movements in four directions; this design means the attacker’s search must consider larger neighbor sets. Nonetheless, the attacker still exhibits high recovery rates when targeting these two keyboards. These results suggest that the layout of a fixed keyboard does not constitute an effective defense against this attack.

5.6 Attack Results on Users

We evaluate the full attack on human users typing credit card details (§5.6.2), passwords (§5.6.3), and web searches (§5.6.4) into Smart TVs. Finally, we quantify the benefits of keystroke timing (§5.6.5).

5.6.1 Setup

We conduct a user study with ten subjects. Our university’s institutional review board (IRB) approved this study, and subjects were compensated \$25 for an hour. The users were aged 22 through 29 (mean 24.3), with six identifying as male, three as female, and one as non-binary. All subjects had previously used a Smart TV; three owned a Samsung, and one owned an AppleTV. In the study, each subject types three sets of credit card details, ten passwords, and ten web searches. Users type credit card details into the Hulu application on the Samsung TV as a part of the account creation flow. Subjects enter passwords on the Samsung TV as if connecting to a WiFi network. Users type passwords into the AppleTV as if logging into their Apple account. Subjects type web searches using the Samsung TV’s browser. To not bias behavior, we only told subjects we were studying how people type on Smart TVs.

We construct lists of passwords and credit cards using the same method as in emulation (§5.5.1). Half of the passwords contain at least one special character. We select web searches by sampling words with at least five characters from news headlines [Kulkarni, 2020]. We assign the same lists to subjects A/F, B/G, C/H, D/I, and E/J, enabling a comparison of how

different users type the same strings. We use the same passwords on both platforms. Users did *not* enter personal information; we provided the strings to type. For each information type, users iteratively interacted with the relevant workflow, entering new strings each time. We recorded each experiment in its entirety. The attacker has no prior knowledge of the number of entered strings and must identify the individual keyboard instances. We measure the attack’s performance using the same guess cutoffs as in §5.5.1.

We use an A1625 AppleTV with tvOS v16.3.2 [Apple, 2023] and a model UN55MU6300 Samsung Smart TV running Tizen software version T-KTMAKUC-1310.1 [Samsung, 2023]. These TVs have different remote controllers (Figure 5.1). The recording device was a commodity Fifine K699B microphone placed about 5.5 feet from the TV. We set the TV volume to 100. Users did not speak during the experiment. There was some background noise from servers running in the same room, but it did not interfere with the users’ ability to hear the television audio.

We use a discount factor of $\gamma = 0.5$ on the AppleTV and $\gamma = 10^{-2}$ on the Samsung Smart TV. This discrepancy exists because users take more suboptimal paths on the AppleTV (§5.6.3). We further account for this difference in suboptimal paths by setting the suboptimal tolerance to $D = 6$ on the AppleTV and $D = 4$ on the Samsung platform. Beyond these tolerances, the discount factors become so small that they effectively eliminate exploration.

5.6.2 *Credit Card Recovery*

The subjects enter 30 total credit card details on the Samsung Smart TV into the signup flow for Hulu, a popular video streaming application. This application uses the system’s provided keyboard (Figure 5.3). Each user completes the signup flow three times, and the attacker identifies the credit card details within this larger interaction.

The attack successfully identifies that the user enters credit card details in 29 of the 30 total interactions. The one failure occurs because the subject returns and edits a previous

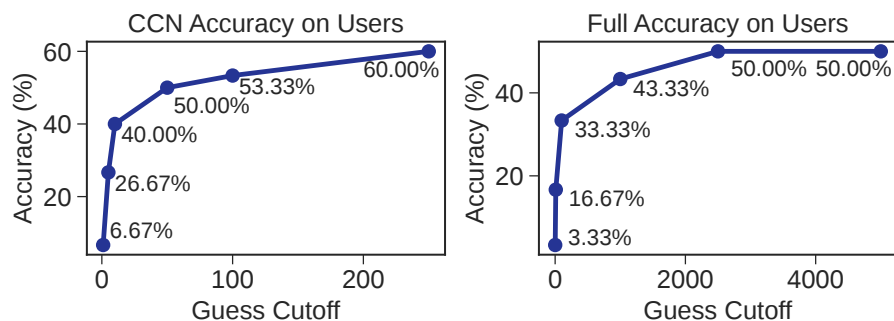


Figure 5.11: Accuracy for credit card numbers (left) and full credit card details (right) on the Samsung Smart TV.

field, breaking the length-based matching (§5.4.2). After identifying the presence of credit cards, the attacker extracts the entered values (§5.4.2). Figure 5.11 shows the accuracy on CCNs and full payment details. The attack reaches a top-100 accuracy of 53.33% (16 / 30) on CCNs and a top-1000 accuracy of 43.33% (13 / 30) on the full details. These values mean the attacker can confirm payment details against rate-limited services 43.33% of the time (§5.5.1). This leakage is significant due to the value of credit card details; the discovered information is sufficient to make online payments on behalf of the target.

These results highlight how the attack works across users and providers. The attack successfully finds at least one CCN in the top 15 guesses for every user. We further recover three pairs of the same details typed by different users. Nevertheless, human behavior impacts the recovery, as we always observe different results across distinct users typing the same CCN. Finally, the attack is successful on the three considered credit card providers, finding 6 / 12 AMEX, 3 / 6 Mastercard, and 4 / 12 Visa details within 1,000 guesses.

Compared to emulation (§5.5.2), the attack performs worse against human users. This trend occurs because users take suboptimal paths. When typing CCNs, users take the optimal path between keys only 89.35% of the time. The framework considers these suboptimal paths with a discounted score (§5.4.2), causing guesses that use suboptimal paths to have a lower rank. Despite this phenomenon, the attack still successfully discovers payment information typed by human users.

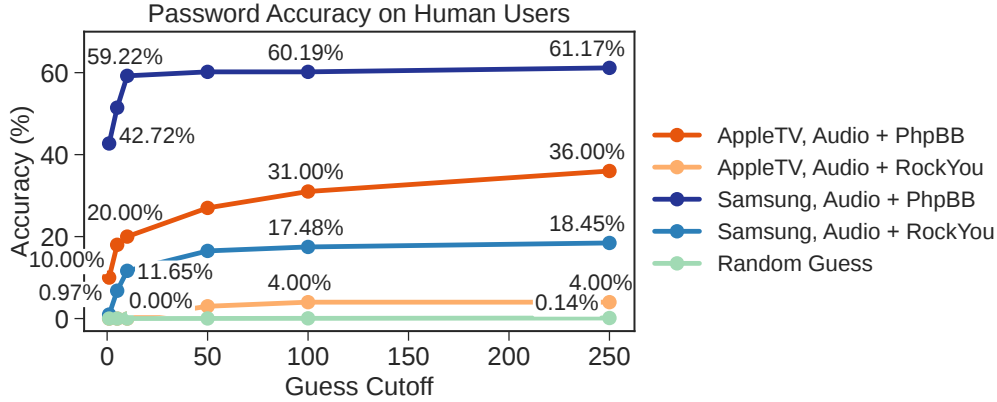


Figure 5.12: Password recovery accuracy for human users. The data labels show the Top- K accuracy for $K \in \{1, 10, 100, 250\}$.

5.6.3 Password Recovery

Users enter passwords from the PhpBB leak [Miessler, 2021] into both TVs. Figure 5.12 displays the attack’s results. With the PhpBB prior, the attack reaches a 33.00% (AppleTV) and 60.19% (Samsung) top-100 accuracy. These results far exceed random guessing from the set of 184,388 PhpBB passwords. Thus, both platforms leak user-typed passwords through audio. The choice of prior is significant; the top-100 accuracy on Samsung drops by over 3.4 \times when the attacker instead uses the RockYou prior. Nevertheless, under the RockYou prior, the attack still finds passwords over 100 \times more often than random guessing on the Samsung TV. Thus, the TV’s audio provides significant information to discover passwords in practical settings.

The attack performs worse on the AppleTV because users take more suboptimal paths on this system. On AppleTV, users traverse the optimal path between keys only about 45.35% of the time; this rate is 85.94% on the Samsung TV. We hypothesize that this discrepancy occurs due to the sensitivity of the AppleTV remote’s navigation touchpad (Figure 5.1). These suboptimal path rates also explain the dropoff between these results and those in emulation (§5.5.3).

The attack handles the challenges of distinct users and diverse passwords. On the Sam-

sung TV, the attack finds at least two passwords in the top 10 guesses for *every* user with the PhpBB prior. Further, when considering unique passwords, the attack has a top-100 accuracy of 42% (21 / 50) for *both* typing users. Thus, the attack is successful across users even after controlling for the typed string. We also observe the ability to recover passwords with various characters, achieving top-100 accuracies of 59.62% (special), 63.79% (numeric), and 43.75% (uppercase) with the PhpBB prior on the Samsung TV. These top-100 figures are 9.62 % (special), 18.97% (numeric), and 12.50% (uppercase) with the RockYou prior.

The attack splits Samsung keyboard instances using timing (§5.4.1). This method correctly identifies 98 / 100 passwords, where the two failures result from long pauses while typing. Further, we infer when a subject types a password (§5.4.2), and this classifier has an accuracy of 99.02% on this set. The only misclassification occurs on the password `naarf666` due to the suffix with repeated characters. This typing pattern matches that of dynamic keyboards, which produce better suggestions toward the string’s end. These failures, however, represent a vast minority of cases, showing how the attack correctly handles instance splitting and keyboard classification.

This performance matches similar keystroke side-channel attacks in other contexts. For example, the MoLe attack uses smartwatch accelerometers to infer keystrokes on mechanical keyboards [Wang et al., 2015]. MoLe involves similar dynamics to Smart TVs, as it recovers strings from keyboard movements. For English words from a known set, the MoLe attack achieves 50% accuracy within 24 guesses. Our attack has a top-5 accuracy of up to 51.46% on the Samsung TV. Note that our setting is more challenging, as the PhpBB set is over $35\times$ larger than the English dictionary used in this prior work.

Overall, these results provide strong evidence that ten users are sufficient to demonstrate this vulnerability. Under normality assumptions, the 95% confidence interval for the attack’s top-100 accuracy on the Samsung TV is $61.48 \pm 1.97 \cdot 26.84/\sqrt{10} = (44.76\%, 78.21\%)$ with

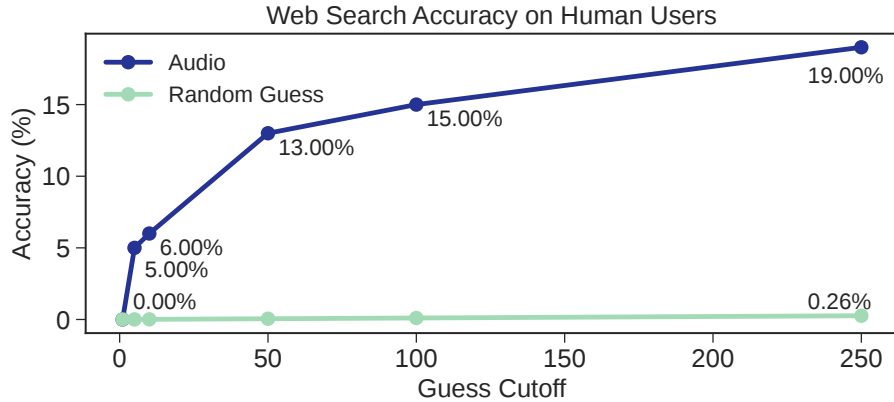


Figure 5.13: Accuracy for human users typing web searches on the Samsung Smart TV with dynamic suggestions.

the PhpBB prior.⁴ For the RockYou prior, the confidence interval is (8.49%, 26.99%). The lower bounds of these intervals exceed random guessing by at least 150 \times . Thus, a larger study would likely not alter the main conclusion: Smart TV audio provides a *useful* signal for discovering user keystrokes.

5.6.4 Web Search Recovery

We apply the attack against web searches on the Samsung Smart TV. This scenario is unique because the attack must overcome dynamic keyboard suggestions (Figure 5.4).

Figure 5.13 displays the attack’s accuracy. The attack finds 15% of words within 100 guesses. This result is lower than that of passwords (§5.6.3) for two reasons. First, the attacker must predict the suggested key values (§5.4.2). Second, the attack infers that the user is typing an English word (§5.4.2), and this classifier is only 42.71% accurate in identifying keyboards with dynamic suggestions. This low accuracy results from the classifier’s designed bias toward passwords, as passwords are more sensitive. Thus, on 57.29% of the entries, the attack cannot recover the target string because it uses a keyboard without suggestions. If

⁴ This mean across users does not exactly match Figure 5.12 because the attack sometimes detects more than ten password entries for a subject.

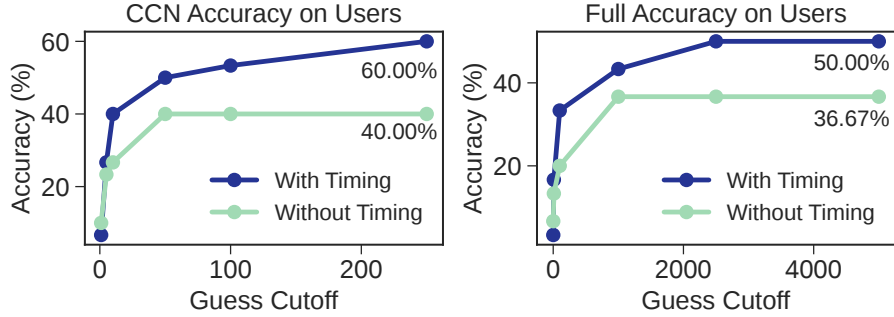


Figure 5.14: Comparison between timing-based and exhaustive identification of suboptimal paths.

we force the pipeline to use keyboards with suggestions, the attack reaches 12.00% (top-10) and 27.00% (top-100) accuracy. These rates nearly double what we observe when inferring the keyboard type. Overall, the attack still vastly outperforms random guessing from the set of 95,892 English words, showing how an attacker can learn about user keystrokes on dynamic keyboards from the TV’s audio.

5.6.5 Impact of Keystroke Timing

The attack leverages two keystroke timing properties to infer user behavior (§5.4.3). This section evaluates the impact of these features.

The string recovery module infers the presence of suboptimal paths using timing (§5.4.3). We evaluate this feature by comparing the recovery of user-typed credit cards with and without timing-based identification. The baseline considers suboptimal paths at every move. As shown in Figure 5.14, keystroke timing improves the attack. The baseline reaches a 40.00% top-100 accuracy on CCNs and a 36.67% top-1000 accuracy on the full details. Under these cutoffs, the timing-based feature shows recovery rates of 53.33% (CCNs) and 43.33% (full details). This improved performance displays the benefits of using timing to identify suboptimal paths.

The attack infers movement directions during rapid scrolls (§5.4.3). We evaluate this

feature by executing password recovery with and without direction inference. We focus on the Samsung TV, as this keyboard supports navigation in all four directions (Figure 5.3). For both priors, direction inference never hurts recovery rates. However, the benefits are modest. With the RockYou prior, only 3 of the 19 recovered passwords have a strictly better rank with direction inference. Nevertheless, we include this feature due these benefits.

We emphasize that the attack uses the same timing metrics for all ten users. The attack’s success across different individuals highlights the generality of these features.

5.7 Discussion

Improvements The results from extracting passwords (§5.6.3) underscore the language prior’s significance. Thus, improving the language model is a promising method to improve this attack. Previous work shows that neural networks can learn effective language models over passwords [Melicher et al., 2016]. We did not pursue these methods due to their computational demands. However, it may be possible to adapt the technique of [Carlini et al., 2019] to make this neural network method more efficient, and we aim to explore this approach in the future.

We assume the attacker does not know personal information about the target user (§5.3). If we relax this assumption, an adversary with user-specific knowledge can improve our audio attack using a customized string prior. Furthermore, with access to personal information, the adversary may already know valuable details such as ZIP codes. These insights suggest that combining user-specific knowledge and Smart TV audio will result in a more effective attack.

Limitations The presented attack only works against AppleTVs and Samsung Smart TVs. The attack does not directly apply to platforms that do not exhibit the acoustic properties in (P2) - (P4). For example, Roku devices do not make a distinct sound when deleting a

character (P3), do not have sounds unique to the keyboard (P4), and mute the keyboard when rapidly scrolling. However, the incompatibility of our attack with these platforms does not guarantee the security of these other systems. Our work only shows these audio properties are *sufficient* for keystroke leakage; it does not prove these properties are *necessary*. In fact, prior work indicates that these properties are not all required. For instance, previous attacks infer when a user is typing on a Smart TV through movement timing [Huang et al., 2023], removing the need for (P4). We emphasize that our attack’s success against Samsung Smart TVs constitutes a significant threat, as Samsung is the single most popular TV platform [Wire, 2021]. Extending this attack to other Smart TVs is a promising source of future work.

Our attack does not apply to every keyboard on the considered platforms. Some applications customize their keyboards and mute the audio, eliminating the acoustic side-channel. However, we show that the default system keyboards are insecure. Thus, applications that use the system defaults, such as the Samsung TV’s browser, are vulnerable. Further, both TVs have areas (e.g., connecting to WiFi) where users enter sensitive information into the default keyboard.

Finally, some applications allow users to enter credentials over the Internet through an external device. This option bypasses our attack by avoiding the TV’s keyboard. However, applications (e.g., Hulu) may not enforce that users take this option, and the alternative involves typing sensitive information (e.g., payment details) into an insecure Smart TV keyboard. Further, there are cases where users *must* type sensitive information into a virtual keyboard. For example, on the Samsung TV used in our evaluation (§5.6.1), users must type their WiFi password with the TV’s default keyboard. There is no option to enter the credentials through an external device. Thus, Smart TVs contain numerous practical settings vulnerable to our acoustic keystroke attack.

Defenses A complete defense is to mute the Smart TV’s audio when typing, especially for sensitive fields. However, this defense can harm the user experience. We describe three possible alternatives.

First, prior work randomizes the layout of virtual keyboards in a structured manner to eliminate knowledge of the keyboard design [Pak et al., 2016]. This approach limits the negative impact on user experience by only adding gaps and swapping rows. While our attack can model this randomization under the framework of suboptimal paths, we can no longer use timing to detect such paths (§5.4.3). We estimate the impact of this defense by randomly adding movements to generated move count sequences for credit card security codes (CVVs). The top-100 recovery rate falls to 30.32% in emulation, compared to the 100% rate achieved under current conditions. These figures suggest this defense reduces, but does not eliminate leakage.

Second, our attack relies on Smart TV keyboards initializing the cursor to the same key (P1). Thus, a possible defense involves randomizing the start position. This defense, however, has severe limitations because some fields (e.g., CCNs) require users to finish on **Done**. We can use this property to search strings *in reverse* starting with **Done**. In emulation, we recover CCNs with a top-100 accuracy of 99.88% even when randomizing the start position.

Finally, Smart TVs could use the same sound for all keyboard actions. This design thwarts our attack, as it breaks (P2) - (P4). Security, however, is not guaranteed. For example, it may be possible to discern keyboard movements and selections using timing, even if both make the same sound. Further study is required to prove the security properties of this technique.

5.8 Related Work

5.8.1 *Keystroke Side-Channel Attacks*

Prior work uses side-channels such as electromagnetic emanations [Jin et al., 2021, Vuagnoux and Pasini, 2009], reflections [Backes et al., 2009], smartphone accelerometers [Marquardt et al., 2011, Owusu et al., 2012], timing [Song et al., 2001], and performance counters [Yang et al., 2022] to infer keystrokes on personal computers and mobile devices.

Three types of prior attacks deserve closer comparison to our work. First, previous methods infer keystrokes using acoustics from mechanical keyboards [Asonov and Agrawal, 2004, Berger et al., 2006, Compagno et al., 2017, Zhu et al., 2014, Zhuang et al., 2009]. Our attack also uses audio, but we target Smart TVs. This setting differs from prior work because Smart TVs make a small number of distinct sounds and use virtual keyboards with a dedicated cursor. Thus, our attack must estimate the cursor’s position, and errors in this tracking cascade to the remainder of the string. Mechanical keyboards have no cursor, and errors in identifying a single keystroke do not propagate. These differences require new string recovery methods.

Second, existing work discovers keystrokes on mechanical keyboards using smartwatch accelerometers [Liu et al., 2015, Meteriz Yıldırım et al., 2022, Maiti et al., 2016, Wang et al., 2015]. Similar to Smart TVs, these attacks track the user’s keyboard position, and tracking errors will cascade. Unlike our work, these attacks use a different side-channel and can only measure the movement of one hand. This limited information leads to lower efficacy than observed in our work (§5.6.3).

Third, and finally, HomeSpy steals user keystrokes by sniffing unencrypted infrared signals between Smart TVs and their remotes [Huang et al., 2023]. However, many Smart TVs support remotes that communicate over Bluetooth [Majors et al., 2023, Zhang et al., 2022], and HomeSpy cannot view such signals. Our attack works with any communication medium

employed by the remote controller.

5.8.2 *Security of Smart Devices*

Previous work highlights the need for increased security on smart devices [He et al., 2021, Jin et al., 2022, Ozmen et al., 2022], compromising smart lights [Ronen and Shamir, 2016], device software platforms [Fernandes et al., 2016, Jia et al., 2021], smart locks [Ho et al., 2016], and smart speakers [Barnes, 2017, Kunze, 2022, Zhang et al., 2017]. Acar et al. [Acar et al., 2018] use DNS rebinding to remotely access devices. Other work uses the communication patterns of smart devices to infer user behavior [Apthorpe et al., 2019, Srinivasan et al., 2008]. Unlike these systems, our attack targets Smart TVs.

Previous studies analyze the security of Smart TVs, finding that platforms contain privacy violations [Mohajeri Moghaddam et al., 2019]. SPOOK gains root access to Smart TVs through a weakness when pairing virtual remotes [Majors et al., 2023]. EvilScreen compromises Smart TVs using issues supporting multiple communication protocols [Zhang et al., 2022]. Other work exploits software vulnerabilities to control Smart TVs [Aafer et al., 2021, Michéle and Karpow, 2014]. Enev et al. [Enev et al., 2011] use energy consumption to infer the TV’s content. Unlike this prior work, our attack leverages the audio from Smart TVs to discover user keystrokes.

5.8.3 *Attacks on Passwords and Payment Details*

Prior work guesses passwords in offline settings with language models [Narayanan and Shmatikov, 2005, Ma et al., 2014, Melicher et al., 2016, Weir et al., 2009] and text transformations [Liu et al., 2019]. Our attack also applies language models to measure password likelihood. However, we use these models to complement Smart TV audio. Wang et al. [Wang et al., 2016] use personal information to guess passwords under rate limits. This work improves our attack, as personalizing the language prior would increase string recovery from

Smart TV audio.

Existing attacks compromise credit cards through skimming [Roland and Langer, 2013, Scaife et al., 2018] and relay attacks [Kfir and Wool, 2005, Roland et al., 2013]. Bond et al. [Bond et al., 2014] attack ATMs by exploiting weak random number generation. Our work also targets credit card details, although our attack steals this information from card-not-present transactions on Smart TVs. Further, our attack complements previous vulnerabilities on physical cards. For example, the credit card’s magnetic stripe does not contain the printed CVV [Scaife et al., 2018]. Our attack steals the CVV from Smart TV forms. Thus, adversaries launching attacks on physical cards can gain valuable information by also exploiting Smart TV virtual keyboards.

5.9 Conclusion

Smart TVs allow users to enter information through on-screen virtual keyboards. Users interact with these keyboards by sequentially scrolling across keys via directional commands on a remote controller. Popular Smart TVs, such as Apple’s tvOS [Apple, 2023] and Samsung’s Tizen [Samsung, 2023], make sounds as users type. This work develops and demonstrates a new side-channel attack against Smart TVs that uses these sounds to expose user keystrokes. Our attack is based on how the AppleTV and the Samsung TV produce different sounds when the user (1) moves between keys, (2) selects a key, and (3) deletes a character. Using these sounds as input signals, an attacker can extract the number of movements between keyboard selections. Our attack combines this extracted information with a prior over possible strings to identify likely keystrokes. Under ideal conditions in an emulated environment, the attack recovers 99.95% of credit card details within 1,000 guesses and up to 99.10% of common passwords within 100 guesses. On ten realistic users, the attack achieves a top-1000 accuracy of 43.33% on credit card details and a top-100 accuracy of up to 60.19% on common passwords when typing on a popular Samsung Smart TV. This attack was acknowledged by

Samsung, and it demonstrates how Smart TVs must consider the privacy implications of all features interacting with sensitive user data.

CHAPTER 6

CONCLUSION

Adaptive systems use data-dependent behavior to optimize the tradeoff between system error and energy consumption on low-power sensor devices. This family of techniques succeeds because it determines its behavior based on the observed inputs. We demonstrate this phenomenon through a novel recurrent neural network (RNN) inference system for sensor devices [Kannan and Hoffmann, 2021]. This system, called Budget RNNs, uses runtime feedback to control the energy required to perform inference on batches of measurements. The use of adaptive behavior allows Budget RNNs to achieve better accuracy under dynamic energy constraints than existing RNN techniques.

This thesis argues how adaptive systems for low-power sensors must go beyond the two-dimensional tradeoff space between system quality (i.e., error) and cost (e.g., energy). In particular, adaptive algorithms must explicitly consider privacy as a third tradeoff dimension. This paradigm shift is required because adaptive algorithms tie their behavior to the collected measurements. Thus, observing the adaptive system’s behavior provides information about the underlying data, and an adversary can make such observations through side-channels such as wireless communication patterns. This leakage allows passive adversaries to launch side-channel attacks against adaptive applications on sensor devices.

We demonstrate this problem with adaptive systems by developing two new side-channel attacks and defenses. First, we show how general adaptive sampling algorithms leak information about sensed events through the device’s communication patterns [Kannan and Hoffmann, 2022]. We close this side-channel with a new framework called Adaptive Group Encoding (AGE). AGE ensures privacy by standardizing communication patterns using lossy encoding. Second, we demonstrate how distributed Multi-exit Neural Networks (MeNNs) with adaptive early exit methods leak information about the model’s predictions through wireless communication patterns [Kannan et al., 2023]. This side-channel attack succeeds

because prior techniques determine when to exit by applying a single threshold to data-dependent prediction confidence values. We eliminate this leakage by employing multiple confidence thresholds and integrating randomness into the exit decision process. In both settings, our defenses exhibit negligible overhead and maintain privacy, all with better system quality (i.e., error) than non-adaptive baselines. With these defenses in place, sensor systems can realize the benefits of adaptive behavior without the data privacy downsides.

We extend this security analysis beyond embedded systems by considering the more general class of Internet-of-Things (IoT) devices. We demonstrate how modern IoT systems contain side-channels that leak critical user information by developing an acoustic keystroke attack against Smart Televisions (TVs). Smart TVs allow users to enter information through on-screen virtual keyboards. Users type by navigating a cursor via directional commands from the TV’s remote controller. Smart TVs make consistent sounds for each keyboard action, and these sounds differ for distinct actions, such as selecting a key, moving between keys, and deleting a character. Using this asymmetry, we develop an attack that uses the TV’s audio to extract these keyboard actions. The attack then uses this extracted information to identify the likeliest strings typed by the user. Against realistic users typing on a Samsung TV, our attack finds 33.33% of credit card details and up to 60.19% of common passwords within 100 guesses. Samsung acknowledged this vulnerability and recognized the work in their Bug Bounty Hall of Fame for Smart TVs, Audio, and Displays.

We conclude by presenting two avenues for future work.

6.1 Data Compression in Distributed Neural Networks

Distributed deep neural networks (DDNNs) reduce inference costs within edge computing systems (Chapter 4). Existing DDNN systems lower their communication overhead by applying lossless data compression to the transmitted intermediate states [Laskaridis et al., 2020]. This compression is effective because the intermediate states are sparse due to the ReLU ac-

tivation function. Lossless compression, however, is data-dependent, and some states will be more compressible than others. Even after encryption, this length discrepancy is noticeable to an attacker intercepting the wireless communication between the sensor device and server. Based on prior work studying the sparsity of intermediate states [Shumailov et al., 2021b], we hypothesize that the sizes of these compressed messages contain information about the DDNN’s predictions. Thus, future work can explore the privacy implications of leveraging data compression within DDNN systems. This type of leakage would represent an additional example of how data-dependent behavior unintentionally exposes sensitive information on resource-constrained devices.

6.2 Audio and Infrared Signals for Smart TVs

Smart TVs support remote controllers that communicate over Infrared (IR) waves. These IR remotes require no setup and transmit signals in plaintext. As displayed by the HomeSpy attack [Huang et al., 2023], attackers can use reflections to sniff IR signals from various places in the target location. With this ability, attackers can extract keystrokes by observing the user’s exact commands to the TV. As presently constructed, however, this attack is unreliable because the adversary can (1) miss transmitted commands and (2) receive commands that the TV misses. We can address these two challenges by combining IR sniffing with audio information. Smart TVs make sounds for user actions on the keyboard (Chapter 5). Thus, by listening to the TV, the attacker can detect *when* they either miss an IR signal or receive an extra command because the TV will audibly register the user’s action. Unlike audio information alone, IR signals carry user directions. Thus, combining both sources of information can create a more potent attack than either individual approach. We further note that the our current acoustic-only attack applies to two Smart TV types: Apple and Samsung. By supplementing audio information with IR sniffing, it may be possible for an attacker to extract keystrokes from other TV platforms (e.g., Roku) as well.

BIBLIOGRAPHY

- Yousra Aafer, Wei You, Yi Sun, Yu Shi, Xiangyu Zhang, and Heng Yin. Android SmartTVs vulnerability discovery via log-guided fuzzing. In *USENIX Security Symposium*, pages 2759–2776. USENIX, 2021.
- Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *Symposium on Operating Systems Design and Implementation*, pages 265–283. USENIX, 2016a.
- Martín Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Computer and Communications Security*, pages 308–318. ACM, 2016b.
- Gunes Acar, Danny Yuxing Huang, Frank Li, Arvind Narayanan, and Nick Feamster. Web-based attacks to discover and control local IoT devices. In *Workshop on IoT Security and Privacy*, pages 29–35, 2018.
- Fevzi Alimoglu and Ethem Alpaydin. Methods of combining multiple classifiers based on different representations for pen-based handwritten digit recognition. In *Turkish Artificial Intelligence and Artificial Neural Networks Symposium*. Citeseer, 1996.
- Cesare Alippi, Giuseppe Anastasi, Cristian Galperti, Francesca Mancini, and Manuel Roveri. Adaptive sampling for energy conservation in wireless sensor networks for snow monitoring applications. In *International Conference on Mobile Ad-Hoc and Sensor Systems*, pages 1–6. IEEE, 2007.
- Cesare Alippi, Giuseppe Anastasi, Mario Di Francesco, and Manuel Roveri. An adaptive sampling algorithm for effective energy management in wireless sensor networks with energy-hungry sensors. *IEEE Transactions on Instrumentation and Measurement*, 59(2):335–344, 2009a.
- Cesare Alippi, Giuseppe Anastasi, Mario Di Francesco, and Manuel Roveri. Energy management in wireless sensor networks with energy-hungry sensors. *IEEE Instrumentation & Measurement Magazine*, 12(2):16–23, 2009b.
- Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge L Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *International Workshop on Ambient Assisted Living*, pages 216–223. Springer, 2012.
- Jason Ansel, Maciej Pacula, Yee Lok Wong, Cy Chan, Marek Olszewski, Una-May O’Reilly, and Saman Amarasinghe. SiblingRivalry: Online autotuning through local competitions. In *International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pages 91–100. ACM, 2012.

- Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *USENIX Security Symposium*, pages 1093–1110. USENIX, 2017.
- Geoff Appelboom, Elvis Camacho, Mickey E Abraham, Samuel S Bruce, Emmanuel LP Dumont, Brad E Zacharia, Randy D’Amico, Justin Slomian, Jean Yves Reginster, Olivier Bruyère, et al. Smart wearable body sensors for patient self-assessment and monitoring. *Archives of Public Health*, 72(1):1–9, 2014.
- Apple. tvOS. <https://developer.apple.com/tvos/>, 2023. Accessed: 03-2023.
- Noah Apthorpe, Danny Yuxing Huang, Dillon Reisman, Arvind Narayanan, and Nick Feamster. Keeping the smart home private with smart(er) iot traffic shaping. *Proceedings on Privacy Enhancing Technologies*, 2019(3):128–148, 2019.
- Chrisil Arackaparambil, Sergey Bratus, Anna Shubina, and David Kotz. On the reliability of wireless fingerprinting using clock skews. In *Conference on Wireless Network Security*, pages 169–174. ACM, 2010.
- Dmitri Asonov and Rakesh Agrawal. Keyboard acoustic emanations. In *IEEE Symposium on Security and Privacy (S&P)*, pages 3–11. IEEE, 2004.
- Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.
- Joseph Azar, Abdallah Makhoul, Mahmoud Barhamgi, and Raphaël Couturier. An energy efficient IoT data compression approach for edge machine learning. *Future Generation Computer Systems*, 96:168–175, 2019.
- Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, pages 2654–2662, 2014.
- Michael Backes, Tongbo Chen, Markus Dürmuth, Hendrik PA Lensch, and Martin Welk. Tempest in a teapot: Compromising reflections revisited. In *Security and Privacy*, pages 315–327. IEEE, 2009.
- Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, Caroline Sporleder, et al. Acoustic side-channel attacks on printers. In *USENIX Security Symposium*. USENIX, 2010.
- Iljoo Baek, Matthew Harding, Akshit Kanda, Kyung Ryeol Choi, Soheil Samii, and Ragu-nathan Raj Rajkumar. CARSS: Client-aware resource sharing and scheduling for heterogeneous applications. In *Real-Time and Embedded Technology and Applications Symposium*, pages 324–335. IEEE, 2020.

- Woongki Baek and Trishul M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *Programming Language Design and Implementation*, pages 198–209. ACM, 2010.
- Anthony Bagnall. Graphical Password Dataset. <https://timeseriesclassification.com/description.php?Dataset=Haptics>, 2021. Accessed: 08-2021.
- Anthony Bagnall, Luke Davis, Jon Hills, and Jason Lines. Transformation based ensembles for time series classification. In *SIAM International Conference on Data Mining*, pages 307–318. SIAM, 2012.
- Ravi Bagree, Vishwas Raj Jain, Aman Kumar, and Prabhat Ranjan. Tigercense: Wireless image sensor network to monitor tiger movement. In *International Workshop on Real-world Wireless Sensor Networks*, pages 13–24. Springer, 2010.
- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473*, 2014.
- Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. Micronets: Neural network architectures for deploying TinyML applications on commodity microcontrollers. *Machine Learning and Systems*, 3:517–532, 2021.
- Amin Banitalebi-Dehkordi, Naveen Vedula, Jian Pei, Fei Xia, Lanjun Wang, and Yong Zhang. Auto-split: A general framework of collaborative edge-cloud AI. In *Conference on Knowledge Discovery and Data Mining*, pages 2543–2553. ACM, 2021.
- Mark Barnes. Alexa, are you listening? <https://labs.withsecure.com/publications/alexa-are-you-listening>, 2017. Accessed: 03-2023.
- Soroush Bateni and Cong Liu. Apnet: Approximation-aware real-time neural network. In *IEEE Real-Time Systems Symposium*, pages 67–79. IEEE, 2018.
- Soroush Bateni and Cong Liu. NeuOS: A Latency-Predictable multi-dimensional optimization framework for DNN-driven autonomous systems. In *USENIX Annual Technical Conference*, pages 371–385, 2020.
- Soroush Bateni, Husheng Zhou, Yuankun Zhu, and Cong Liu. Predjoule: A timing-predictable energy optimization framework for deep neural networks. In *IEEE Real-Time Systems Symposium*, pages 107–118. IEEE, 2018.
- Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel. In *USENIX Security Symposium*, pages 515–532. USENIX, 2019.
- Konstantin Berestizshevsky and Guy Even. Dynamically sacrificing accuracy for reduced computation: Cascaded inference based on softmax confidence. In *International Conference on Artificial Neural Networks*, pages 306–320. Springer, 2019.

- Yigael Berger, Avishai Wool, and Arie Yeredor. Dictionary attacks using keyboard acoustic emanations. In *Conference on Computer and Communications Security*, pages 245–254. ACM, 2006.
- Erik Bernhardsson. Annoy. <https://github.com/spotify/annoy>, 2023.
- Bruhadeshwar Bezawada, Maalvika Bachani, Jordan Peterson, Hossein Shirazi, Indrakshi Ray, and Indrajit Ray. Behavioral fingerprinting of IoT devices. In *Workshop on Attacks and Solutions in Hardware Security*, pages 41–50, 2018.
- Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. Var-CNN: A data-efficient website fingerprinting attack based on deep learning. *Proceedings on Privacy Enhancing Technologies*, 4:292–310, 2019.
- Battista Biggio, Iginio Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 387–402. Springer, 2013.
- Andrey Bogdanov, Thomas Eisenbarth, Christof Paar, and Malte Wienecke. Differential cache-collision timing attacks on AES with applications to embedded CPUs. In *Cryptographers’ Track at the RSA Conference*, pages 235–251. Springer, 2010.
- Mike Bond, Omar Choudary, Steven J Murdoch, Sergei Skorobogatov, and Ross Anderson. Chip and skim: Cloning EMV cards with the pre-play attack. In *Security and Privacy*, pages 49–64. IEEE, 2014.
- Sergey Bratus, Cory Cornelius, David Kotz, and Daniel Peebles. Active behavioral fingerprinting of wireless devices. In *Conference on Wireless Network Security*, pages 56–61. ACM, 2008.
- Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.
- David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.
- Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Computer and Communications Security*, pages 605–616. ACM, 2012.
- Xiang Cai, Rishab Nithyanand, and Rob Johnson. CS-BuFLO: A congestion sensitive website fingerprinting defense. In *Workshop on Privacy in the Electronic Society*, pages 121–130. ACM, 2014a.
- Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Computer and Communications Security*, pages 227–238. ACM, 2014b.

- Víctor Campos, Brendan Jou, Xavier Giró-i Nieto, Jordi Torres, and Shih-Fu Chang. Skip RNN: Learning to skip state updates in recurrent neural networks. *arXiv:1708.06834*, 2017.
- Anthony Canino and Yu David Liu. Proactive and adaptive energy-aware programming with mixed typechecking. In *Programming Language Design and Implementation*, pages 217–232. ACM, 2017.
- Anthony Canino, Yu David Liu, and Hidehiko Masuhara. Stochastic energy optimization for mobile GPS applications. In *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 703–713. ACM, 2018.
- Bogdan Carbunar, Yang Yu, Weidong Shi, Michael Pearce, and Venu Vasudevan. Query privacy in wireless sensor networks. *ACM Transactions on Sensor Networks*, 6(2):1–34, 2010.
- Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy*, pages 39–57. IEEE, 2017.
- Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *USENIX Security Symposium*, pages 267–284. USENIX, 2019.
- Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramèr. Membership inference attacks from first principles. In *Security and Privacy*, pages 1897–1914. IEEE, 2022.
- Rich Caruana, Steve Lawrence, and C Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in Neural Information Processing Systems*, pages 402–408, 2001.
- Daniel Casini, Alessandro Biondi, and Giorgio Buttazzo. Timing isolation and improved scheduling of deep neural networks for real-time systems. *Software: Practice and Experience*, 50(9):1760–1777, 2020.
- Keng-hao Chang, Jeffrey Hightower, and Branislav Kveton. Inferring identity using accelerometers in television remote controls. In *International Conference on Pervasive Computing*, pages 151–167. Springer, 2009.
- Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A Hasegawa-Johnson, and Thomas S Huang. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 77–87, 2017.
- Supriyo Chatterjea and Paul Havinga. An adaptive and autonomous sensor sampling frequency control scheme for energy-efficient data acquisition in wireless sensor networks. In *International Conference on Distributed Computing in Sensor Systems*, pages 60–78. Springer, 2008.

- Konstantinos Chatzikokolakis, Tom Chothia, and Apratim Guha. Statistical measurement of information leakage. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 390–404. Springer, 2010.
- H. Chen, M. Song, J. Song, A. Gavrilovska, and K. Schwan. Hears: A hierarchical energy-aware resource scheduler for virtualized data centers. In *International Conference on Cluster Computing*, pages 508–512. IEEE, 2011.
- Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Security and Privacy*, pages 191–206. IEEE, 2010.
- Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.
- Mu-Hsuan Cheng, Qihui Sun, and Chia-Heng Tu. An adaptive computation framework of distributed deep learning models for internet-of-things applications. In *International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 85–91. IEEE, 2018.
- Hari Cherupalli, Henry Duwe, Weidong Ye, Rakesh Kumar, and John Sartori. Software-based gate-level information flow security for IoT systems. In *IEEE/ACM International Symposium on Microarchitecture*, pages 328–340. IEEE/ACM, 2017.
- Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv:1409.1259*, 2014.
- David Chu, Amol Deshpande, Joseph M Hellerstein, and Wei Hong. Approximate data collection in sensor networks using probabilistic models. In *International Conference on Data Engineering*, pages 48–48. IEEE, 2006.
- Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv:1412.3555*, 2014.
- Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: an extension of MNIST to handwritten letters. *arXiv:1702.05373*, 2017.
- Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and trainability in recurrent neural networks. *arXiv:1611.09913*, 2016.
- Alberto Compagno, Mauro Conti, Daniele Lain, and Gene Tsudik. Don’t Skype & type! acoustic eavesdropping in voice-over-IP. In *Asia Conference on Computer and Communications Security*, pages 703–715. ACM, 2017.

- Tara M Cox, TJ Ragen, AJ Read, E Vos, Robin W Baird, K Balcomb, Jay Barlow, J Caldwell, T Cranford, and L Crum. Understanding the impacts of anthropogenic sound on beaked whales. Technical report, Space and Naval Warfare Systems Center, San Diego, CA, 2006.
- Asma Dachraoui, Alexis Bondu, and Antoine Cornuéjols. Early classification of time series as a non myopic sequential decision making problem. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 433–447. Springer, 2015.
- Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. *National Institute of Standards and Technology*, 1999.
- Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The tangled web of password reuse. In *Network and Distributed Systems Security Symposium*, volume 14, pages 23–26. Internet Society, 2014.
- Aveek K Das, Parth H Pathak, Chen-Nee Chuah, and Prasant Mohapatra. Uncovering privacy leakage in BLE network traffic of wearable fitness trackers. In *International Workshop on Mobile Computing Systems and Applications*, pages 99–104. ACM, 2016.
- Chacko John Deepu, Chun-Huat Heng, and Yong Lian. A hybrid data compression scheme for power reduction in wireless sensors for iot. *IEEE Transactions on Biomedical Circuits and Systems*, 11(2):245–254, 2016.
- Bradley Denby and Brandon Lucia. Orbital edge computing: Nanosatellite constellations as a new class of computer system. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 939–954. ACM, 2020.
- Don Dennis, Chirag Pabbaraju, Harsha Vardhan Simhadri, and Prateek Jain. Multiple instance learning for efficient sequential data classification on resource-constrained devices. In *Advances in Neural Information Processing Systems*, pages 10953–10964, 2018.
- Don Dennis, Durmus Alp Emre Acar, Vikram Mandikal, Vinu Sankar Sadasivan, Venkatesh Saligrama, Harsha Vardhan Simhadri, and Prateek Jain. Shallow RNN: accurate time-series classification on resource constrained devices. In *Advances in Neural Information Processing Systems*, pages 12916–12926, 2019.
- Amol Deshpande, Carlos Guestrin, Samuel R Madden, Joseph M Hellerstein, and Wei Hong. Model-driven data acquisition in sensor networks. In *International Conference on Very Large Databases*, volume 30, pages 588–599, 2004.
- Gabriel Martins Dias, Maddalena Nurchis, and Boris Bellalta. Adapting sampling interval of sensor networks using on-line reinforcement learning. In *World Forum on Internet of Things*, pages 460–465. IEEE, 2016.
- Thai Duong and Julianno Rizzo. The CRIME attack. https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2Gizeu0faLU2H0U, 2012.

- Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray. Advanced encryption standard (aes), 2001-11-26 2001.
- Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail. In *Symposium on Security and Privacy*, pages 332–346. IEEE, 2012.
- Miro Enev, Sidhant Gupta, Tadayoshi Kohno, and Shwetak N Patel. Televisions, video privacy, and powerline electromagnetic interference. In *Computer and Communications Security*, pages 537–550. ACM, 2011.
- Deborah Estrin, David Culler, Kris Pister, and Gaurav Sukhatme. Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1(1):59–69, 2002.
- Biyi Fang, Xiao Zeng, and Mi Zhang. NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *International Conference on Mobile Computing and Networking*, pages 115–127, 2018.
- Fuming Fang and Takahiro Shinozaki. Electrooculography-based continuous eye-writing recognition system for efficient assistive communication systems. *PloS ONE*, 13(2), 2018.
- Anne Farrell and Henry Hoffmann. MEANTIME: Achieving both minimal energy and timeliness with approximate computing. In Ajay Gulati and Hakim Weatherspoon, editors, *USENIX Annual Technical Conference*, pages 421–435. USENIX, 2016.
- Igor Fedorov, Ryan P Adams, Matthew Mattina, and Paul Whatmough. Sparse: Sparse architecture search for CNNs on resource-constrained microcontrollers. *Advances in Neural Information Processing Systems*, 32, 2019.
- Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security analysis of emerging smart home applications. In *2016 IEEE symposium on security and privacy (SP)*, pages 636–654. IEEE, 2016.
- Jody L Ferris. Data privacy and protection in the agriculture industry: Is federal regulation necessary. *Minnesota Journal of Law Science and Technology*, 18:309, 2017.
- Ronald Aylmer Fisher et al. A mathematical examination of the methods of determining the accuracy of an observation by the mean error, and by the mean square error. *Monthly Notices of the Royal Astronomical Society*, 80, 1920.
- Jason Flinn and Mahadev Satyanarayanan. Energy-aware adaptation for mobile applications. In *Symposium on Operating Systems Principles*, page 48–63. ACM, 1999.
- David Formby, Preethi Srinivasan, Andrew M Leonard, Jonathan D Rogers, and Raheem A Beyah. Who’s in control of your control system? Device fingerprinting for cyber-physical systems. In *Network and Distributed System Security Symposium*. Internet Society, 2016.

- Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv:1803.03635*, 2018.
- Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(771-780):1612, 1999.
- Warren Frick and Carlos Niederstrasser. Small launch vehicles-A 2018 state of the industry survey. In *AIAA/USU Conference on Small Satellites*. AIAA/USU, 2018.
- Ixent Galpin, Alvaro A Fernandes, and Norman W Paton. QoS-aware optimization of sensor network queries. *The International Journal on Very Large Databases*, 22(4):495–517, 2013.
- Ke Gao, Cherita Corbett, and Raheem Beyah. A passive approach to wireless device fingerprinting. In *International Conference on Dependable Systems & Networks*, pages 383–392. IEEE, 2010.
- Bugra Gedik, Ling Liu, and S Yu Philip. ASAP: An adaptive sampling approach to data collection in sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 18(12):1766–1783, 2007.
- Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210. PMLR, 2016.
- Giacomo Giuliani, Tommaso Ciussani, Adrian Perrig, and Ankit Singla. ICARUS: Attacking low Earth orbit satellite networks. In *USENIX Annual Technical Conference*, pages 317–331. USENIX, 2021.
- Yoel Gluck, Neal Harris, and Angelo Prado. BREACH: Reviving the CRIME attack. *Unpublished manuscript*, 2013.
- Dennis RE Gnad, Jonas Krautter, and Mehdi B Tahoori. Leaky noise: New side-channel attack vectors in mixed-signal IoT devices. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 305–339, 2019.
- Graham Gobieski, Brandon Lucia, and Nathan Beckmann. Intelligence beyond the edge: Inference on intermittent embedded systems. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 199–213. ACM, 2019.
- Ashvin Goel, David Steere, Calton Pu, and Jonathan Walpole. Swift: A feedback control and dynamic reconfiguration toolkit. Technical report, Oregon Graduate Institute, 1998.
- Phillip Good. *Permutation tests: A practical guide to resampling methods for testing hypotheses*. Springer Science and Business Media, 2013.
- Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

- Paul A. Grassi, James L. Fenton, Elaine M. Newton, Ray A. Perlner, Andrew R. Regenscheid, William E. Burr, Justin P. Richer, Naomi B. Lefkovitz, Jamie M. Danker, Yee-Yin Choong, Kristen K. Greene, and Mary F. Theofanos. Digital identity guidelines. Technical Report 800-63B, National Institute of Standards and Technology, 2017.
- Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv:1603.08983*, 2016.
- Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649. IEEE, 2013.
- Scott Gray, Alec Radford, and Diederik P Kingma. GPU kernels for block-sparse weights. *arXiv:1711.09224*, 3, 2017.
- Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv:1708.06733*, 2017.
- Amira Guesmi, Ihsen Alouani, Khaled N Khasawneh, Mouna Baklouti, Tarek Frikha, Mohamed Abid, and Nael Abu-Ghazaleh. Defensive approximation: Securing CNNs using approximate computing. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 990–1003. ACM, 2021.
- Mohamed Hamdi and Habtamu Abie. Game-based adaptive security in the Internet of Things for eHealth. In *International Conference on Communications*, pages 920–925. IEEE, 2014.
- Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. EIE: Efficient inference engine on compressed deep neural network. *Computer Architecture News*, 44(3):243–254, 2016.
- Mirazul Haque, Anki Chauhan, Cong Liu, and Wei Yang. ILFO: Adversarial attack on adaptive neural networks. In *Conference on Computer Vision and Pattern Recognition*, pages 14264–14273. IEEE/CVF, 2020.
- Hanieh Hashemi, Yongqin Wang, and Murali Annavaram. DarKnight: An accelerated framework for privacy and integrity preserving deep learning using trusted hardware. In *International Symposium on Microarchitecture*, pages 212–224. ACM/IEEE, 2021.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 770–778. IEEE, 2016.
- Liang He, Linghe Kong, Yu Gu, Jianping Pan, and Ting Zhu. Evaluating the on-demand mobile charging in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 14(9):1861–1875, 2014.

- Weijia He, Valerie Zhao, Olivia Morkved, Sabeeka Siddiqui, Earlence Fernandes, Josiah Hester, and Blase Ur. SoK: Context sensing for access control in the adversarial home IoT. In *European Symposium on Security and Privacy*, pages 37–53. IEEE, 2021.
- Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Workshop on Cloud Computing Security*, pages 31–42. ACM, 2009.
- Shivayogi Hiremath, Geng Yang, and Kunal Mankodiya. Wearable Internet of Things: Concept, architectural components and promises for person-centered healthcare. In *International Conference on Wireless Mobile Communication and Healthcare-Transforming Healthcare Through Innovations in Mobile and Wireless Technologies*, pages 304–307. IEEE, 2014.
- Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. Smart locks: Lessons for securing commodity internet of things devices. In *Asia Conference on Computer and Communications Security*, pages 461–472. ACM, 2016.
- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. *Gradient flow in recurrent nets: The difficulty of learning long-term dependencies*, pages 237–243. IEEE, 2001.
- Henry Hoffmann. CoAdapt: Predictable behavior for accuracy-aware applications running on power-aware systems. In *Euromicro Conference on Real-Time Systems*, pages 223–232. IEEE, 2014.
- Henry Hoffmann. JouleGuard: Energy guarantees for approximate applications. In Ethan L. Miller and Steven Hand, editors, *Symposium on Operating Systems Principles*, pages 198–214. ACM, 2015.
- Henry Hoffmann, Jim Holt, George Kurian, Eric Lau, Martina Maggio, Jason E. Miller, Sabrina M. Neuman, Mahmut Sinangil, Yildiz Sinangil, Anant Agarwal, Anantha P. Chandrakasan, and Srinivas Devadas. Self-aware computing in the Angstrom processor. In *Design Automation Conference*, page 259–264. ACM, 2012.
- Henry Hoffmann, Axel Jantsch, and Nikil D. Dutt. Embodied self-aware computing systems. *Proceedings of the IEEE*, 108(7):1027–1046, 2020.
- JK Holland, EK Kemsley, and RH Wilson. Use of fourier transform infrared spectroscopy and partial least squares regression for the detection of adulteration of strawberry purees. *Journal of the Science of Food and Agriculture*, 76(2):263–269, 1998.
- John H Holland. Genetic algorithms. *Scientific American*, 267(1):66–73, 1992.

- Sanghyun Hong, Yiğitcan Kaya, Ionuț-Vlad Modoranu, and Tudor Dumitraș. A panda? No, it's a sloth: Slowdown attacks on adaptive multi-exit neural network inference. *arXiv:2010.02432*, 2020.
- Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.
- Ting-Kuei Hu, Tianlong Chen, Haotao Wang, and Zhangyang Wang. Triple wins: Boosting accuracy, robustness and efficiency together by enabling input-adaptive inference. *arXiv:2002.10025*, 2020.
- Weizhe Hua, Zhiru Zhang, and G Edward Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *Design Automation Conference*, pages 1–6. ACM/ESDA/IEEE, 2018.
- Danny Yuxing Huang, Noah Apthorpe, Frank Li, Gunes Acar, and Nick Feamster. IoT Inspector: Crowdsourcing labeled network traffic from smart home devices at scale. *Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(2):1–21, 2020.
- Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens Van Der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv:1703.09844*, 2017.
- Kong Huang, YuTong Zhou, Ke Zhang, Jiachen Xu, Jiongyi Chen, Di Tang, and Kehuan Zhang. HomeSpy: The Invisible Sniffer of Infrared Remote Control of Smart TVs. In *USENIX Security Symposium*. USENIX, 2023.
- Dino Ienco, Raffaele Gaetano, Claire Dupaquier, and Pierre Maurel. Land cover classification via multitemporal spatial data by deep recurrent neural networks. *IEEE Geoscience and Remote Sensing Letters*, 14(10):1685–1689, 2017.
- Andrey Ignatov. Human activity recognition dataset. <https://github.com/aiff22/HAR>, 2017.
- Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *Advances in Neural Information Processing Systems*, 32, 2019.
- Connor Imes, Steven Hofmeyr, and Henry Hoffmann. Energy-efficient application resource scheduling using machine learning classifiers. In *International Conference on Parallel Processing*, pages 45:1–45:11. ACM, 2018.
- Stripe Inc. Card verification checks. <https://stripe.com/docs/disputes/prevention/verification>, 2023. Accessed: 03-2023.

- Statista Consumer Market Insights. Smart speaker market volume worldwide from 2015 to 2027 . <https://www.statista.com/forecasts/1367982/smart-speaker-market-volume-worldwide>, 2022. Accessed: 03-2023.
- Texas Instruments. TI MSP430 EnergyTrace Technology. https://www.ti.com/lit/ug/slau157as/slau157as.pdf?ts=1628556110549&ref_url=https%253A%252F%252Fwww.ti.com%252Ftool%252FENERGYTRACE, 2020. Accessed: 04-2023.
- Texas Instruments. TI MSP430 FR5994 Datasheet. <https://www.ti.com/lit/ds/symlink/msp430fr5994.pdf>, 2021. Accessed: 05-2023.
- Bashima Islam, Yubo Luo, and Shahriar Nirjon. Zygarde: Time-sensitive on-device deep intelligence on intermittently-powered systems. *arXiv:1905.03854*, 2019.
- Zohar Jackson. Free spoken digit dataset. <https://github.com/Jakobovski/free-spoken-digit-dataset>, 2022. Accessed: 04-2023.
- Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv:1711.09846*, 2017.
- Ankur Jain and Edward Y Chang. Adaptive sampling for sensor networks. In *International Workshop on Data Management for Sensor Networks*, pages 10–16, 2004.
- Shinae Jang, Hongki Jo, Soojin Cho, Kirill Mechtov, Jennifer A Rice, Sung-Han Sim, Hyung-Jo Jung, Chung-Bangm Yun, Billie F Spencer Jr, and Gul Agha. Structural health monitoring of a cable-stayed bridge using smart sensor technology: Deployment and evaluation. *Smart Structures and Systems*, 6(5_6):439–459, 2010.
- Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *arXiv:1702.05659*, 2017.
- Yacine Jernite, Edouard Grave, Armand Joulin, and Tomas Mikolov. Variable computation in recurrent neural networks. *arXiv:1611.06188*, 2016.
- Yan Jia, Bin Yuan, Luyi Xing, Dongfang Zhao, Yifan Zhang, XiaoFeng Wang, Yijing Liu, Kaimin Zheng, Peyton Crnjak, Yuqing Zhang, et al. Who’s in control? On security risks of disjointed IoT device management channels. In *Computer and Communications Security*, pages 1289–1305. ACM, 2021.
- Qing Jin, Linjie Yang, and Zhenyu Liao. Adabits: Neural network quantization with adaptive bit-widths. In *Conference on Computer Vision and Pattern Recognition*, pages 2146–2156. IEEE, 2020.
- Wenqiang Jin, Srinivasan Murali, Huadi Zhu, and Ming Li. Periscope: A keystroke inference attack using human coupled electromagnetic emanations. In *Computer and Communications Security*, pages 700–714. ACM, 2021.

- Xin Jin, Sunil Manandhar, Kaushal Kafle, Zhiqiang Lin, and Adwait Nadkarni. Understanding IoT security from a market-scale perspective. In *Computer and Communications Security*, pages 1615–1629. ACM, 2022.
- Weiyu Ju, Wei Bao, Liming Ge, and Dong Yuan. Dynamic early exit scheduling for deep neural network inference through contextual bandits. In *International Conference on Information and Knowledge Management*, pages 823–832. ACM, 2021.
- Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 96–107. ACM, 2002.
- Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. Toward an efficient website fingerprinting defense. In *European Symposium on Research in Computer Security*, pages 27–46. Springer, 2016.
- Pandurang Kamat, Wenyuan Xu, Wade Trappe, and Yanyong Zhang. Temporal privacy in wireless sensor networks. In *International Conference on Distributed Computing Systems*, pages 23–23. IEEE, 2007.
- Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *Computer Architecture News*, 45(1):615–629, 2017.
- Tejas Kannan and Henry Hoffmann. Budget RNNs: Multi-capacity neural networks to improve in-sensor inference under energy budgets. In *Real-Time and Embedded Technology and Applications Symposium*, pages 143–156. IEEE, 2021.
- Tejas Kannan and Henry Hoffmann. Protecting adaptive sampling from information leakage on low-power sensors. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 240–254. ACM, 2022.
- Tejas Kannan, Nick Feamster, and Henry Hoffmann. Prediction privacy in distributed multi-exit neural networks: Vulnerabilities and solutions. In *Computer and Communications Security*. ACM, 2023.
- Tejas Kannan, Synthia Wang, Max Sunog, Abe Bueno de Mesquita, Nick Feamster, and Henry Hoffmann. Acoustic keystroke leakage on smart televisions. In *Network and Distributed System Security Symposium*. Internet Society, 2024.
- Aman Kansal, Scott Saponas, A.J. Bernheim Brush, Kathryn S. McKinley, Todd Mytkowicz, and Ryder Ziola. The latency, accuracy, and battery (LAB) abstraction: Programmer productivity and energy efficiency for continuous mobile context sensing. In *International Conference on Object Oriented Programming Systems Languages and Applications*, pages 661–676. ACM, 2013.

- Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International Conference on Machine Learning*, pages 3301–3310. PMLR, 2019.
- John Kelsey. Compression and information leakage of plaintext. In *International Workshop on Fast Software Encryption*, pages 263–276. Springer, 2002.
- Ziv Kfir and Avishai Wool. Picking virtual pockets using relay attacks on contactless smartcard. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM 05)*, pages 47–58. IEEE, 2005.
- Minyoung Kim, Mark-Oliver Stehr, Carolyn Talcott, Nikil Dutt, and Nalini Venkatasubramanian. XTune: A formal methodology for cross-layer tuning of mobile embedded systems. *ACM Transactions on Embedded Computing Systems*, 11(4), 2013.
- Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.
- Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual International Cryptology Conference*, pages 388–397. Springer, 1999.
- Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011.
- Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. Spectre attacks: Exploiting speculative execution. *Communications of the ACM*, 63(7):93–101, 2020.
- Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanus Phillips, Irena Gao, et al. Wilds: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning*, pages 5637–5664. PMLR, 2021.
- Tadayoshi Kohno, Andre Broido, and Kimberly C Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005.
- Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork RNN. *arXiv:1402.3511*, 2014.
- Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images, 2009.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.
- Rohit Kulkarni. One week of global news feeds. <https://www.kaggle.com/datasets/therohk/global-news-week>, 2020. Accessed: 04-2023.

- Ashish Kumar, Saurabh Goyal, and Manik Varma. Resource-efficient machine learning in 2 KB RAM for the internet of things. In *International Conference on Machine Learning*, pages 1935–1944, 2017.
- Matthew Kunze. Turning google smart speakers into wiretaps for \$100k. <https://downrightnifty.me/blog/2022/12/26/hacking-google-home.html#the-fixes:~:text=remotely%20control%20it.-,A%20cooler%20attack%20scenario,-Attacker%20wishes%20to>, 2022. Accessed: 03-2023.
- Aditya Kusupati, Manish Singh, Kush Bhatia, Ashish Kumar, Prateek Jain, and Manik Varma. FastGRNN: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. *Advances in Neural Information Processing Systems*, 31, 2018.
- Aditya Kusupati, Manish Singh, Kush Bhatia, Ashish Kumar, Prateek Jain, and Manik Varma. FastGRNN: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. *arXiv:1901.02358*, 2019.
- Tarald O Kvålseth. On normalized mutual information: Measure derivations and properties. *Entropy*, 19(11):631, 2017.
- Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. Activity recognition using cell phone accelerometers. *SigKDD Explorations Newsletter*, 12(2):74–82, 2011.
- Fabian Lanze, Andriy Panchenko, Benjamin Braatz, and Andreas Zinnen. Clock skew based remote device fingerprinting demystified. In *Global Communications Conference*, pages 813–819. IEEE, 2012.
- Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D Lane. SPINN: Synergistic progressive inference of neural networks over device and cloud. In *International Conference on Mobile Computing and Networking*, pages 1–15. ACM, 2020.
- Yee Wei Law, Supriyo Chatterjea, Jiong Jin, Thomas Hanselmann, and Marimuthu Palaniswami. Energy-efficient data acquisition by adaptive sampling for wireless sensor networks. In *International Conference on Wireless Communications and Mobile Computing*, pages 1146–1151, 2009.
- Yann LeCun, Corinna Cortes, and Christopher Burges. The MNIST database of handwritten digits. <http://yann.lecun.com/exdb/mnist/>, 1998.
- Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *Security and Privacy*, pages 656–672. IEEE, 2019.

- Hankook Lee and Jinwoo Shin. Anytime neural prediction via slicing networks vertically. *arXiv:1807.02609*, 2018.
- Seulki Lee and Shahriar Nirjon. SubFlow: A dynamic induced-subgraph strategy toward real-time DNN inference and training. In *Real-Time and Embedded Technology and Applications Symposium*, pages 15–29. IEEE, 2020.
- Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. Occlumency: Privacy-preserving remote deep-learning inference using SGX. In *International Conference on Mobile Computing and Networking*, pages 1–17. ACM, 2019.
- Patrick Leu, Ivan Puddu, Aanjhan Ranganathan, and Srdjan Čapkun. I send, therefore I leak: Information leakage in low-power wide area networks. In *Conference on Security and Privacy in Wireless and Mobile Networks*, pages 23–33. ACM, 2018.
- En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications*, 19(1):447–457, 2019.
- Zheng Li, Yiyong Liu, Xinlei He, Ning Yu, Michael Backes, and Yang Zhang. Auditing membership leakages of multi-exit networks. In *Computer and Communications Security*, pages 1917–1931. ACM, 2022.
- Marc Liberatore and Brian Neil Levine. Inferring the source of encrypted HTTP connections. In *Computer and Communications Security*, pages 255–263. ACM, 2006.
- Edgar Liberis, Łukasz Dudziak, and Nicholas D Lane. μ NAS: Constrained neural architecture search for microcontrollers. In *Workshop on Machine Learning and Systems*, pages 70–79, 2021.
- Chi Lin, Yanhong Zhou, Haipeng Dai, Jing Deng, and Guowei Wu. MPF: Prolonging network lifetime of wireless rechargeable sensor networks by mixing partial charge and full charge. In *International Conference on Sensing, Communication, and Networking*, pages 1–9. IEEE, 2018.
- Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. MCUnet: Tiny deep learning on IoT devices. *Advances in Neural Information Processing Systems*, 33:11711–11722, 2020.
- Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, et al. Meltdown: Reading kernel memory from user space. *Communications of the ACM*, 63(6):46–56, 2020.
- Enze Liu, Amanda Nakanishi, Maximilian Golla, David Cash, and Blase Ur. Reasoning analytically about password-cracking software. In *Security and Privacy*, pages 380–397. IEEE, 2019.

- Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via MiniONN transformations. In *Computer and Communications Security*, pages 619–631. ACM, 2017.
- Xiangyu Liu, Zhe Zhou, Wenrui Diao, Zhou Li, and Kehuan Zhang. When good becomes evil: Keystroke inference with smartwatch. In *Computer and Communications Security*, pages 1273–1285. ACM, 2015.
- Sergey Lobov, Nadia Krilova, Innokentiy Kastalskiy, Victor Kazantsev, and Valeri A Makarov. Latent factors limiting the performance of sEMG-interfaces. *Sensors*, 18(4): 1122, 2018.
- Jeffrey W Lockhart, Gary M Weiss, Jack C Xue, Shaun T Gallagher, Andrew B Grosner, and Tony T Pulickal. Design considerations for the WISDM smart phone-based sensor mining architecture. In *International Workshop on Knowledge Discovery from Sensor Data*, pages 25–33, 2011.
- Ping Lou, Liang Shi, Xiaomei Zhang, Zheng Xiao, and Junwei Yan. A data-driven adaptive sampling method based on edge computing. *Sensors*, 20(8):2174, 2020.
- Bo Lu, Xiaokuan Zhang, Ziman Ling, Yinqian Zhang, and Zhiqiang Lin. A measurement study of authentication rate-limiting mechanisms of modern websites. In *Computer Security Applications Conference*, pages 89–100. ACSA, 2018.
- Liming Lu, Ee-Chien Chang, and Mun Choon Chan. Website fingerprinting and identification using ordered feature sequences. In *European Symposium on Research in Computer Security*, pages 199–214. Springer, 2010.
- Alexander Ly, Maarten Marsman, Josine Verhagen, Raoul PPP Grasman, and Eric-Jan Wagenmakers. A tutorial on Fisher information. *Journal of Mathematical Psychology*, 80: 40–55, 2017.
- Jerry Ma, Weining Yang, Min Luo, and Ninghui Li. A study of probabilistic password models. In *Security and Privacy*, pages 689–704. IEEE, 2014.
- Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, volume 30, page 3, 2013.
- Samuel Madden and Michael J Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *International Conference on Data Engineering*, pages 555–566. IEEE, 2002.
- Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv:1706.06083*, 2017.
- Matt Mahoney. *Data compression explained*, chapter 1,1. mattmahoney.net, 2012.

- Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *International Workshop on Wireless Sensor Networks and Applications*, pages 88–97. ACM, 2002.
- Anindya Maiti, Oscar Armbruster, Murtuza Jadliwala, and Jibo He. Smartwatch-based keystroke inference attacks and context-aware protection mechanisms. In *Asia Conference on Computer and Communications Security*, pages 795–806. ACM, 2016.
- Joshua David Oetting Majors, Edgardo Barsallo Yi, Amiya Maji, Darren Wu, Saurabh Bagchi, and Aravind Machiry. Security properties of virtual remotes and SPOOKing their violations. In *Asia Conference on Computer and Communications Security*, pages 841–854. ACM, 2023.
- Hossein Mamaghanian, Nadia Khaled, David Atienza, and Pierre Vandergheynst. Compressed sensing for real-time energy-efficient ECG compression on wireless body sensor nodes. *IEEE Transactions on Biomedical Engineering*, 58(9):2456–2466, 2011.
- Francesco Marcelloni and Massimo Vecchio. A simple algorithm for data compression in wireless sensor networks. *IEEE Communications Letters*, 12(6):411–413, 2008.
- Research & Markets. Global Smart TV unit sales from 2018 to 2025 [Graph]. <https://www.statista.com/statistics/878372/smart-tv-unit-sales-worldwide/>, 2020. Accessed: 03-2023.
- Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor. (sp)iPhone: Decoding vibrations from nearby keyboards using mobile phone accelerometers. In *Computer and Communications Security*, pages 551–562. ACM, 2011.
- Kirk Martinez and Jane K Hart. Glacier monitoring: Deploying custom hardware in harsh environments. In *Wireless Sensor Networks*, pages 245–258. Springer, 2010.
- Kirk Martinez, Royan Ong, and Jane Hart. Glacsweb: A sensor network for hostile environments. In *Conference on Sensor and Ad-Hoc Communications and Networks*, pages 81–87. IEEE, 2004.
- Andre Martins and Ramon Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International Conference on Machine Learning*, pages 1614–1623, 2016.
- Gaurav Mathur, Peter Desnoyers, Deepak Ganesan, and Prashant Shenoy. Ultra-low power data storage for sensor networks. In *International Conference on Information Processing in Sensor Networks*. IEEE, 2006.
- Gaurav Mathur, Peter Desnoyers, Paul Chukiu, Deepak Ganesan, and Prashant Shenoy. Ultra-low power data storage for sensor networks. *ACM Transactions on Sensor Networks*, 5(4):1–34, 2009.

- Aastha Mehta, Mohamed Alzayat, Roberta De Viti, Björn B. Brandenburg, Peter Druschel, and Deepak Garg. Pacer: Comprehensive Network Side-Channel Mitigation in the Cloud. In *USENIX Security Symposium*, pages 2819–2838. USENIX, 2022.
- Melbourne. Melbourne pedestrian counter. <http://www.pedestrian.melbourne.vic.gov.au/>, 2020a. Accessed: 05-2023.
- Melbourne. Melbourne pedestrian dataset. <http://www.timeseriesclassification.com/description.php?Dataset=MelbournePedestrian>, 2020b.
- William Melicher, Blase Ur, Sean M Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Fast, lean, and accurate: Modeling password guessability using neural networks. In *USENIX Security Symposium*, pages 175–191. USENIX, 2016.
- Thomas S. Messerges and Ezzy A. Dabbish. Investigations of power analysis attacks on smartcards. In *USENIX Workshop on Smartcard Technology*. USENIX, 1999.
- Thomas S Messerges, Ezzat A Dabbish, and Robert H Sloan. Examining smart-card security under the threat of power analysis attacks. *IEEE Transactions on Computers*, 51(5):541–552, 2002.
- Ülkü Meteriz Yöldüran, Necip Fazıl Yöldüran, and David Mohaisen. AcousticType: Smartwatch-enabled cross-device text entry method using keyboard acoustics. In *Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–7. ACM, 2022.
- Benjamin Michéle and Andrew Karpow. Watch and be watched: Compromising all Smart TV generations. In *Consumer Communications and Networking Conference*, pages 351–356. IEEE, 2014.
- Microchip. Atmel sam l21 datasheet. http://ww1.microchip.com/downloads/en/DeviceDoc/SAM_L21_Family_DataSheet_DS60001477C.pdf, 2020. Accessed: 05-2023.
- Daniel Miessler. Leaked password databases. <https://github.com/danielmiessler/SecLists/tree/master/Passwords/Leaked-Databases>, 2021. Accessed: 04-2022.
- Nikita Mishra, Connor Imes, John D Lafferty, and Henry Hoffmann. CALOREE: Learning control for predictable latency and low energy. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 184–198, 2018.
- Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. DarknetTZ: Towards model privacy at the edge using trusted execution environments. In *International Conference on Mobile Systems, Applications, and Services*, pages 161–174. ACM, 2020.

- Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.
- Hooman Mohajeri Moghaddam, Gunes Acar, Ben Burgess, Arunesh Mathur, Danny Yuxing Huang, Nick Feamster, Edward W Felten, Prateek Mittal, and Arvind Narayanan. Watching you watch: The tracking ecosystem of over-the-top TV streaming devices. In *Computer and Communications Security*, pages 131–147. CCS, 2019.
- Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *Security and Privacy*, pages 19–38. IEEE, 2017.
- Usue Mori, Alexander Mendiburu, Sanjoy Dasgupta, and Jose A Lozano. Early classification of time series by simultaneously optimizing the accuracy and earliness. *IEEE Transactions on Neural Networks and Learning Systems*, 29(10):4569–4578, 2017.
- Norman Mu and Justin Gilmer. MNIST-C: A robustness benchmark for computer vision. *arXiv:1906.02337*, 2019.
- Abdulmajid Murad, Frank Alexander Kraemer, Kerstin Bach, and Gavin Taylor. Information-driven adaptive sensing based on deep reinforcement learning. In *International Conference on the Internet of Things*, pages 1–8. ACM, 2020.
- Ralph P Muscatell. Laser microphone. *The Journal of the Acoustical Society of America*, 76(4):1284–1284, 1984.
- Mischa Möstl, Johannes Schlatow, Rolf Ernst, Henry Hoffmann, Arif Merchant, and Alexander Shraer. Self-aware systems for the internet-of-things. In *International Conference on Hardware/Software Codesign and System Synthesis*, pages 1–9, 2016.
- Sharan Narang, Eric Undersander, and Gregory Diamos. Block-sparse recurrent neural networks. *arXiv:1711.02782*, 2017.
- Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *Computer and Communications Security*, pages 364–372. ACM, 2005.
- Leslie M Naylor and John G Kie. Monitoring activity of Rocky Mountain elk using recording accelerometers. *Wildlife Society Bulletin*, 32(4):1108–1113, 2004.
- Leslie M Naylor, Michael J Wisdom, and Robert G Anthony. Behavioral responses of North American Elk to recreational activity. *The Journal of Wildlife Management*, 73(3):328–338, 2009.
- Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased LSTM: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*, pages 3882–3890, 2016.
- Yoav Nir and Adam Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 8439, 2018.

- Rishab Nithyanand, Xiang Cai, and Rob Johnson. Glove: A bespoke website fingerprinting defense. In *Workshop on Privacy in the Electronic Society*, pages 131–134, 2014.
- NVIDIA. NVIDIA Jetson Nano. <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>, 2023. Accessed: 05-2023.
- Markus Ojala and Gemma C Garriga. Permutation tests for studying classifier performance. *Journal of Machine Learning Research*, 11(6), 2010.
- Angela Orebaugh, Gilbert Ramirez, and Jay Beale. *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006.
- Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. ACCessory: Password inference using accelerometers on smartphones. In *Workshop on Mobile Computing Systems and Applications*, pages 1–6. ACM, 2012.
- Muslum Ozgur Ozmen, Xuansong Li, Andrew Chu, Z Berkay Celik, Bardh Hoxha, and Xiangyu Zhang. Discovering IoT physical channel vulnerabilities. In *Computer and Communications Security*, pages 2415–2428. ACM, 2022.
- Colin O’Flynn and Alex Dewar. On-device power analysis across hardware security domains. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 126–153, 2019.
- Wooguil Pak, Youngrok Cha, and Sunki Yeo. High accessible virtual keyboards for preventing key-logging. In *International Conference on Ubiquitous and Future Networks*, pages 205–207. IEEE, 2016.
- Xudong Pan, Mi Zhang, Beina Sheng, Jiaming Zhu, and Min Yang. Hidden trigger backdoor attack on NLP models via linguistic style manipulation. In *USENIX Security Symposium*, pages 3611–3628. USENIX, 2022.
- Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website fingerprinting at internet scale. In *Network and Distributed System Security Symposium*. Internet Society, 2016.
- Jeffrey Pang, Ben Greenstein, Ramakrishna Gummadi, Srinivasan Seshan, and David Wetherall. 802.11 user fingerprinting. In *International Conference on Mobile Computing and Networking*, pages 99–110. ACM, 2007.
- Liam Paninski. Estimation of entropy and mutual information. *Neural Computation*, 15(6): 1191–1253, 2003.
- Nicolas Papernot and Patrick McDaniel. Deep K-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv:1803.04765*, 2018.

- Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy*, pages 582–597. IEEE, 2016.
- Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2017.
- Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.
- Fernando J Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59(19):2229, 1987.
- Armen Poghosyan and Alessandro Golkar. Cubesat evolution: Analyzing cubesat capabilities for conducting science missions. *Progress in Aerospace Sciences*, 88:59–83, 2017.
- Raghavendra Pradyumna Pothukuchi, Sweta Yamini Pothukuchi, Petros Voulgaris, and Josep Torrellas. Yukta: Multilayer resource controllers to maximize efficiency. In *International Symposium on Computer Architecture*, pages 505–518. ACM/IEEE, 2018.
- Jordi Puig-Suari, Clark Turner, and William Ahlgren. Development of the standard CubeSat deployer and a CubeSat class PicoSatellite. In *IEEE Aerospace Conference*, volume 1, pages 1–347. IEEE, 2001.
- Amir M. Rahmani, Bryan Donyanavard, Tiago Mück, Kasra Moazzemi, Axel Jantsch, Onur Mutlu, and Nikil Dutt. SPECTR: Formal supervisory control and coordination for many-core systems resource management. *SIGPLAN Notices*, 53(2):169–183, 2018.
- Kasper Bonne Rasmussen and Srdjan Capkun. Implications of radio fingerprinting on the security of sensor networks. In *International Conference on Security and Privacy in Communications Networks*, pages 331–340. IEEE, 2007.
- Cezar Reinbrecht, Altamiro Susin, Lilian Bossuet, Georg Sigl, and Johanna Sepúlveda. Timing attack on NoC-based systems: Prime+Probe attack and NoC-based protection. *Microprocessors and Microsystems*, 52:556–565, 2017.
- Daniel E Rivera, Manfred Morari, and Sigurd Skogestad. Internal model control: PID controller design. *Industrial & Engineering Chemistry Process Design and Development*, 25(1):252–265, 1986.
- Michael Roland and Josef Langer. Cloning credit cards: A combined pre-play and downgrade attack on EMV contactless. In *USENIX Workshop on Offensive Technologies*. USENIX, 2013.
- Michael Roland, Josef Langer, and Josef Scharinger. Applying relay attacks to Google Wallet. In *International Workshop on Near Field Communication*, pages 1–6. IEEE, 2013.

- Eyal Ronen and Adi Shamir. Extended functionality attacks on IoT devices: The case of smart lights. In *European Symposium on Security and Privacy*, pages 3–12. IEEE, 2016.
- David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, University of California San Diego Institute for Cognitive Science, 1985.
- David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.
- David E Rumelhart, Bernard Widrow, and Michael A Lehr. The basic ideas in neural networks. *Communications of the ACM*, 37(3):87–93, 1994.
- Marc Rußwurm, Sébastien Lefevre, Nicolas Courty, Rémi Emonet, Marco Körner, and Romain Tavenard. End-to-end learning for early classification of time series. *arXiv:1901.10681*, 2019.
- Abdeldjalil Saidani, Xiang Jianwen, and Deloula Mansouri. A Lossless Compression Approach Based on Delta Encoding and T-RLE in WSNs. *Wireless Communications and Mobile Computing*, 2020, 2020.
- Christian Salim, Abdallah Makhoul, Rony Darazi, and Raphaël Couturier. Adaptive sampling algorithms with local emergency detection for energy saving in wireless body sensor networks. In *Network Operations and Management Symposium*, pages 745–749. IEEE/IFIP, 2016.
- Sriram Sami, Yimin Dai, Sean Rui Xiang Tan, Nirupam Roy, and Jun Han. Spying with your robot vacuum cleaner: Eavesdropping via lidar sensors. In *Conference on Embedded Networked Sensor Systems*, pages 354–367, 2020.
- Samsung. Tizen TV. <https://docs.tizen.org/platform/what-is-tizen/profiles/tv/>, 2023. Accessed: 03-2023.
- Iskander Sanchez-Rola, Igor Santos, and Davide Balzarotti. Clock around the clock: Time-based device fingerprinting. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1502–1514, 2018.
- Asanka Sayakkara, Nhien-An Le-Khac, and Mark Scanlon. Leveraging electromagnetic side-channel analysis for the investigation of IoT devices. *Digital Investigation*, 29:S94–S103, 2019.
- Nolen Scaife, Christian Peeters, and Patrick Traynor. Fear the reaper: Characterization and fast detection of card skimmers. In *USENIX Security Symposium*, pages 1–14. USENIX, 2018.
- Simone Scardapane, Michele Scarpiniti, Enzo Baccarelli, and Aurelio Uncini. Why should we add early exits to neural networks? *Cognitive Computation*, 12(5):954–966, 2020.

- Robert E Schapire. Explaining AdaBoost. In *Empirical inference*, pages 37–52. Springer, 2013.
- Jürgen Schmidhuber. Self-delimiting neural networks. *arXiv:1210.0118*, 2012.
- Lars Schor, Philipp Sommer, and Roger Wattenhofer. Towards a zero-configuration wireless sensor network architecture for smart buildings. In *Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pages 31–36. ACM, 2009.
- Nader Sehatbakhsh, Alireza Nazari, Haider Khan, Alenka Zajic, and Milos Prvulovic. Emma: Hardware/software attestation framework for embedded systems using electromagnetic signals. In *International Symposium on Microarchitecture*, pages 983–995. IEEE/ACM, 2019.
- Jayaprakash Selvaraj, Gökçen Yılmaz Dayanıklı, Neelam Prabhu Gaunkar, David Ware, Ryan M Gerdes, and Mani Mina. Electromagnetic induction attacks against embedded systems. In *Asia Conference on Computer and Communications Security*, pages 499–510. ACM, 2018.
- Ilya Semenov. Wikipedia word counts. <https://github.com/IlyaSemenov/wikipedia-word-frequency>, 2022. Accessed: 04-2022.
- Claude E Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 28(4):656–715, 1949.
- Weisong Shi and Schahram Dustdar. The promise of edge computing. *Computer*, 49(5): 78–81, 2016.
- Ilia Shumailov, Zakhar Shumaylov, Dmitry Kazhdan, Yiren Zhao, Nicolas Papernot, Murat A Erdogdu, and Ross J Anderson. Manipulating SGD with data ordering attacks. *Advances in Neural Information Processing Systems*, 34:18021–18032, 2021a.
- Ilia Shumailov, Yiren Zhao, Daniel Bates, Nicolas Papernot, Robert Mullins, and Ross Anderson. Sponge examples: Energy-latency attacks on neural networks. In *European Symposium on Security and Privacy*, pages 212–231. IEEE, 2021b.
- Sandra Siby, Rajib Ranjan Maiti, and Nils Tippenhauer. IoTscanner: Detecting and classifying privacy threats in IoT neighborhoods. *arXiv:1701.05007*, 2017.
- João Marco C Silva, Kalil Araujo Bispo, Paulo Carvalho, and Solange Rito Lima. LiteSense: An adaptive sensing scheme for WSNs. In *International Symposium on Computers and Communications*, pages 1209–1212. IEEE, 2017.
- Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

- David C. Snowdon, Etienne Le Sueur, Stefan M. Petters, and Gernot Heiser. Koala: A platform for OS-level power management. In *European Conference on Computer Systems*, page 289–302. ACM, 2009.
- Philipp Sommer, Branislav Kusy, Philip Valencia, Ross Dungavell, and Raja Jurdak. Delay-tolerant networking for long-term animal tracking. *IEEE Internet Computing*, 22(1):62–72, 2018.
- Dawn Xiaodong Song, David A Wagner, Xuqing Tian, et al. Timing analysis of keystrokes and timing attacks on SSH. In *USENIX Security Symposium*. USENIX, 2001.
- Jacob Sorber, Alexander Kostadinov, Matthew Garber, Matthew Brennan, Mark D. Corner, and Emery D. Berger. Eon: A language and runtime system for perpetual systems. In *International Conference on Embedded Networked Sensor Systems*. ACM, 2007.
- Jacob Sorber, Minh Shin, Ronald Peterson, Cory Cornelius, Shrirang Mare, Aarathi Prasad, Zachary Marois, Emma Smithayer, and David Kotz. An amulet for trustworthy wearable mHealth. In *Workshop on Mobile Computing Systems and Applications*, pages 1–6. ACM, 2012.
- Charalampos Sotirakis, Zi Su, Maksymilian A Brzezicki, Niall Conway, Lionel Tarassenko, James J FitzGerald, and Chrystalina A Antoniadis. Identification of motor progression in Parkinson’s disease using wearable sensors and machine learning. *npj Parkinson’s Disease*, 9(1):142, 2023.
- Vinicius MA Souza. Asphalt pavement classification using smartphone accelerometer and complexity invariant distance. *Engineering Applications of Artificial Intelligence*, 74:198–211, 2018.
- Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv:1507.06149*, 2015.
- Vijay Srinivasan, John Stankovic, and Kamin Whitehouse. Protecting your daily in-home activity information from a wireless snooping attack. In *International Conference on Ubiquitous Computing*, pages 202–211. ACM, 2008.
- Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- Frank Stajano and Ross Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *International Workshop on Security Protocols*, pages 172–182. Springer, 1999.
- Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pages 1453–1460, 2011.

- John A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, 21(10):10–19, 1988.
- Timothy Stevens, Christian Skalka, Christelle Vincent, John Ring, Samuel Clark, and Joseph Near. Efficient differentially private secure aggregation for federated learning via hardness of learning with errors. In *USENIX Security Symposium*, pages 1379–1395. USENIX, 2022.
- Biljana Risteska Stojkoska and Zoran Nikolovski. Data compression for energy efficient IoT solutions. In *Telecommunication Forum*, pages 1–4. IEEE, 2017.
- Jung-Woong Sung and Seung-Jae Han. Data bundling for energy efficient communication of wearable devices. *Computer Networks*, 121:76–88, 2017.
- Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.
- Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv:1312.6199*, 2013.
- Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Conference on Computer Vision and Pattern Recognition*, pages 2818–2826. IEEE, 2016.
- Sofiane Takarabt, Alexander Schaub, Adrien Facon, Sylvain Guilley, Laurent Sauvage, Youssef Souissi, and Yves Mathieu. Cache-timing attacks still threaten IoT devices. In *International Conference on Codes, Cryptology, and Information Security*, pages 13–30. Springer, 2019.
- Yew Teck Tan, Abhinav Kunapareddy, and Marin Kobilarov. Gaussian process adaptive sampling using the cross-entropy method for environmental sensing and monitoring. In *International Conference on Robotics and Automation*, pages 6220–6227. IEEE, 2018.
- Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In *International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016.
- Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *International Conference on Distributed Computing Systems*, pages 328–339. IEEE, 2017.
- Armen Toorian, Ken Diaz, and Simon Lee. The CubeSat approach to space access. In *2008 IEEE Aerospace Conference*, pages 1–14. IEEE, 2008.
- Konstantinos Tovletoglou, Lev Mukhanov, Dimitrios S. Nikolopoulos, and Georgios Karakostas. HaRMony: Heterogeneous-reliability memory and qos-aware energy management on virtualized servers. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, page 575–590. ACM, 2020.

- Wade Trappe, Richard Howard, and Robert S Moore. Low-energy security: Limits and opportunities in the internet of things. *Security and Privacy*, 13(1):14–21, 2015.
- Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning. In *Workshop on Artificial Intelligence and Security*, pages 1–11. ACM, 2019.
- SW Tyler, DM Holland, V Zagorodnov, AA Stern, C Sladek, S Kobs, S White, F Suárez, and J Bryenton. Using distributed temperature sensors to monitor an antarctic ice shelf and sub-ice-shelf cavity. *Journal of Glaciology*, 59(215):583–591, 2013.
- Arijit Ukil, Soma Bandyopadhyay, Aniruddha Sinha, and Arpan Pal. Adaptive sensor data compression in IoT systems: Sensor data analytics based approach. In *International Conference on Acoustics, Speech and Signal Processing*, pages 5515–5519. IEEE, 2015.
- Pratik Vaishnavi, Kevin Eykholt, and Amir Rahmati. Transferring adversarial robustness through robust representation matching. In *USENIX Security Symposium*, pages 2083–2098. USENIX, 2022.
- Matthijs Van Keirsbilck, Alexander Keller, and Xiaodong Yang. Rethinking full connectivity in recurrent neural networks. *arXiv:1905.12340*, 2019.
- Deepak Vasisht, Zerina Kapetanovic, Jongho Won, Xinxin Jin, Ranveer Chandra, Sudipta Sinha, Ashish Kapoor, Madhusudhan Sudarshan, and Sean Stratman. Farmbeats: An iot platform for data-driven agriculture. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 515–529. USENIX, 2017.
- Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *European Conference on Computer Vision*, pages 3–18, 2018.
- Will Vickers and Matthew Middlehurst. Right whale calls dataset. <http://www.timeseriesclassification.com/description.php?Dataset=RightWhaleCalls>, 2013.
- Jose R Villar, Paula Vergara, Manuel Menéndez, Enrique de la Cal, Víctor M González, and Javier Sedano. Generalized models for the classification of abnormal movements in daily life and its applicability to epilepsy convulsion recognition. *International Journal of Neural Systems*, 26(06), 2016.
- Dharma Teja Vooturi, Dheevatsa Mudigere, and Sasikanth Avancha. Hierarchical block sparse neural networks. *arXiv:1808.03420*, 2018.
- Martin Vuagnoux and Sylvain Pasini. Compromising electromagnetic emanations of wired and wireless keyboards. In *USENIX Security Symposium*, volume 1. USENIX, 2009.
- Chengcheng Wan, Henry Hoffmann, Shan Lu, and Michael Maire. Orthogonalized SGD and nested architectures for anytime neural networks. In *International Conference on Machine Learning*, pages 9807–9817. PMLR, 2020a.

- Chengcheng Wan, Muhammad Santriaji, Eri Rogers, Henry Hoffmann, Michael Maire, and Shan Lu. ALERT: Accurate learning for energy and timeliness. In *USENIX Annual Technical Conference*, pages 353–369. USENIX, 2020b.
- Avery Wang. The shazam music recognition service. *Communications of the ACM*, 49(8): 44–48, 2006.
- Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. Targeted online password guessing: An underestimated threat. In *Computer and Communications Security*, pages 1242–1254. ACM, 2016.
- He Wang, Ted Tsung-Te Lai, and Romit Roy Choudhury. MoLe: Motion leaks through smartwatch sensors. In *International Conference on Mobile Computing and Networking*, pages 155–166. ACM, 2015.
- Kai Wang, Zihao Chu, Yanhong Zhou, Kang Wang, Chi Lin, and Mohammad S Obaidat. Partial charging scheduling in wireless rechargeable sensor networks. In *IEEE Global Communications Conference*, pages 1–6. IEEE, 2018a.
- Meiqi Wang, Jianqiao Mo, Jun Lin, Zhongfeng Wang, and Li Du. DynExit: A dynamic early-exit strategy for deep residual networks. In *International Workshop on Signal Processing Systems*, pages 178–183. IEEE, 2019.
- Shu Wang, Chi Li, Henry Hoffmann, Shan Lu, William Sentosa, and Achmad Imam Kistijantoro. Understanding and auto-adjusting performance-sensitive configurations. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 154–168. ACM, 2018b.
- Xin Wang, Yujia Luo, Daniel Crankshaw, Alexey Tumanov, Fisher Yu, and Joseph E Gonzalez. Idk cascades: Fast deep learning by learning not to overthink. *arXiv:1706.00885*, 2017.
- Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv:1804.03209*, 2018.
- Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *Transactions on Information Forensics and Security*, 15:3454–3469, 2020.
- Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang Xu. I know what you see: Power side-channel attack on convolutional neural network accelerators. In *Computer Security Applications Conference*, pages 393–406, 2018.
- Matt Weir, Sudhir Aggarwal, Breno De Medeiros, and Bill Glodek. Password cracking using probabilistic context-free grammars. In *Security and Privacy*, pages 391–405. IEEE, 2009.

- Paul J Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.
- Susan Wiedenbeck, Jim Waters, Leonardo Sobrado, and Jean-Camille Birget. Design and evaluation of a shoulder-surfing resistant graphical password scheme. In *Conference on Advanced Visual Interfaces*, pages 177–184, 2006.
- Rebecca Willett, Aline Martin, and Robert Nowak. Backcasting: Adaptive sampling for sensor networks. In *International Symposium on Information Processing in Sensor Networks*, pages 124–133, 2004.
- Ben H Williams, Marc Toussaint, and Amos J Storkey. Extracting motion primitives from natural handwriting data. In *International Conference on Artificial Neural Networks*, pages 634–643. Springer, 2006.
- Michael Winkler, Klaus-Dieter Tuchs, Kester Hughes, and Graeme Barclay. Theoretical and practical aspects of military wireless sensor networks. *Journal of Telecommunications and Information Technology*, pages 37–45, 2008.
- Business Wire. Smart TV streaming device market share worldwide as of 2020, by platform [Graph]. <https://www.statista.com/statistics/1171132/global-connected-tv-devices-streaming-market-share-by-platform/>, 2021. Accessed: 03-2023.
- Charles V Wright, Scott E Coull, and Fabian Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *Network and Distributed System Security Symposium*, volume 9. Internet Society, 2009.
- Tong Tong Wu, Kenneth Lange, et al. Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics*, 2(1):224–244, 2008.
- Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv:1708.07747*, 2017.
- Zhengzheng Xing, Jian Pei, and S Yu Philip. Early classification on time series. *Knowledge and Information Systems*, 31(1):105–127, 2012.
- Wenzheng Xu, Weifa Liang, Xiaohua Jia, and Zichuan Xu. Maximizing sensor lifetime in a rechargeable sensor network via partial energy charging on sensors. In *International Conference on Sensing, Communication, and Networking*, pages 1–9. IEEE, 2016.
- Boyuan Yang, Ruirong Chen, Kai Huang, Jun Yang, and Wei Gao. Eavesdropping user credentials via GPU side channels on smartphones. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 285–299. ACM, 2022.
- Yuanyuan Yang and Cong Wang. *Wireless rechargeable sensor networks*. Springer, 2015.

- Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek Abdelzaher. Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In *Conference on Embedded Network Sensor Systems*, pages 1–14. ACM, 2017.
- Shuochao Yao, Yifan Hao, Yiran Zhao, Huajie Shao, Dongxin Liu, Shengzhong Liu, Tianshi Wang, Jinyang Li, and Tarek Abdelzaher. Scheduling real-time deep learning services as imprecise computations. In *International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–10. IEEE, 2020.
- Yong Yao, Johannes Gehrke, et al. Query processing in sensor networks. In *Conference on Innovative Data Systems Research*, pages 233–244, 2003.
- Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Patoukas, Koen Schaper, Przemysław Pawelczak, and Josiah Hester. Ink: Reactive kernel for tiny batteryless sensors. In *Conference on Embedded Networked Sensor Systems*, pages 41–53. ACM, 2018.
- Adams Wei Yu, Hongrae Lee, and Quoc V Le. Learning to skim text. *arXiv:1704.06877*, 2017.
- Honggang Yu, Haocheng Ma, Kaichen Yang, Yiqiang Zhao, and Yier Jin. DeepEM: Deep neural networks model recovery through EM side-channel information leakage. In *International Symposium on Hardware Oriented Security and Trust*, pages 209–218. IEEE, 2020.
- Zeping Yu and Gongshen Liu. Sliced recurrent neural networks. *arXiv:1807.02291*, 2018.
- Liekang Zeng, En Li, Zhi Zhou, and Xu Chen. Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the industrial Internet of Things. *IEEE Network*, 33(5):96–103, 2019.
- Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv:1611.03530*, 2016.
- Guoming Zhang, Chen Yan, Xiaoyu Ji, Tianchen Zhang, Taimin Zhang, and Wenyuan Xu. DolphinAttack: Inaudible voice commands. In *Computer and Communications Security*, pages 103–117. ACM, 2017.
- Pei Zhang and Margaret Martonosi. Locale: Collaborative localization estimation for sparse mobile sensor networks. In *International Conference on Information Processing in Sensor Networks*, pages 195–206. IEEE, 2008.
- Pei Zhang, Christopher M Sadler, Stephen A Lyon, and Margaret Martonosi. Hardware design experiences in ZebraNet. In *International Conference on Embedded Networked Sensor Systems*, pages 227–238, 2004.
- Ronghua Zhang, Chenyang Lu, Tarek F. Abdelzaher, and John A. Stankovic. ControlWare: A middleware architecture for feedback control of software performance. In *International Conference on Distributed Computing Systems*, pages 301–310, 2002.

- Yiwei Zhang, Siqi Ma, Tiancheng Chen, Juanru Li, Robert H Deng, and Elisa Bertino. EvilScreen attack: Smart TV hijacking via multi-channel remote control mimicry. *arXiv:2210.03014*, 2022.
- Jing Zhou and David De Roure. Floodnet: Coupling adaptive sampling with energy aware routing in a flood warning system. *Journal of Computer Science and Technology*, 22(1): 121–130, 2007.
- Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33:18330–18341, 2020.
- Tong Zhu, Qiang Ma, Shanfeng Zhang, and Yunhao Liu. Context-free attacks using keyboard acoustic emanations. In *Computer and Communications Security*, pages 453–464. ACM, 2014.
- Li Zhuang, Feng Zhou, and J Doug Tygar. Keyboard acoustic emanations revisited. *ACM Transactions on Information and System Security*, 13(1):1–26, 2009.