

DESIGN OF SECURE HARDWARE ARCHITECTURES FOR NEURAL NETWORKS

NIVEDITA SHRIVASTAVA



INDIAN INSTITUTE OF TECHNOLOGY DELHI
JULY 2025

© Nivedita Shrivastava, 2025

DESIGN OF SECURE HARDWARE ARCHITECTURES FOR NEURAL NETWORKS

by

NIVEDITA SHRIVASTAVA

Department of Electrical Engineering

Submitted

in fulfillment of the requirements of the degree of Doctor of
Philosophy

to the



INDIAN INSTITUTE OF TECHNOLOGY DELHI

JULY 2025

Certificate

This is to certify that the thesis titled **DESIGN OF SECURE HARDWARE ARCHITECTURES FOR NEURAL NETWORKS** being submitted by Ms. **NIVEDITA SHRIVASTAVA** for the award of **Doctor of Philosophy** in Electrical Engineering is a record of bona fide work carried out by her under my guidance and supervision at the Department of Electrical Engineering, Indian Institute of Technology Delhi. The work presented in this thesis has not been submitted elsewhere, either in part or full, for the award of any other degree or diploma.

Smruti Ranjan Sarangi
Professor
Department of Electrical Engineering
Indian Institute of Technology Delhi
New Delhi- 110016

For my mother,
Late Mrs. Babita Shrivastava

Acknowledgements

I wish to begin by expressing my deepest gratitude to **Prof. Smruti R. Sarangi** for his guidance, encouragement, and unwavering support throughout my PhD journey. I am beyond words to express my gratitude to him for granting me the invaluable opportunity to pursue a PhD and for guiding me through this transformative journey. I am immensely grateful for his patience and wisdom during moments when I felt utterly lost, and for his ability to steer me back on track with clarity and purpose. His relentless pursuit of excellence and perfection, has been a constant source of motivation for me to push my boundaries, strive for improvement, and excel in both academic and personal endeavors. This thesis would have been inconceivable without his valuable guidance and support.

I would like to express my heartfelt gratitude to my Student Research Committee members: **Prof. Harshan Jagdeesh**, **Prof. Mukul Sarkar**, and **Prof. Huzur Saran**, for their invaluable insights, constructive feedback, and thoughtful guidance throughout my research journey. I am also deeply thankful to **Prof. Sparsh Mittal** for his collaboration on my first paper. His guidance, suggestions, and encouragement provided crucial direction and helped me improve the quality of my work significantly.

I would like to express my sincere gratitude to the administrative staff in the department for their continuous support and assistance. I am especially thankful to Ms. Aditi, who went above and beyond her duties to help me navigate and complete the formalities for various processes.

I would like to extend my thanks to all of my friends at IIT Delhi for the many wonderful discussions we shared, both technical and non-technical. I began my research alongside a senior student Dr. Diksha Moolcandani, and I am deeply thankful to her for being a mentor in my early days, helping me grasp the entire research flow—how to approach problems, conduct in-depth analysis, and structure my work for a paper submission. My gratitude extends to the senior students—Dr. Priyanka Singla, Dr. Shubhankar Singh and Dr. Hameedah Sultan—for their invaluable support. Their guidance and

insights made my initial days at the institute far more manageable. I am also thankful to Akanksha, Rohith, and Shruti for always being available for technical discussions and for fostering a friendly and stress-free environment in the lab. The sense of community made the journey not only productive but also enjoyable. I am also grateful to Ani Sunny for being an incredible teammate. Our long discussions and late-night brainstorming sessions during our recent collaboration were truly memorable and impactful.

I am deeply grateful to my father for his unwavering support, motivation, and unconditional love. His presence has been a constant source of strength throughout this journey. I also extend my heartfelt thanks to my husband and mother-in-law for their incredible support and for being my pillars of strength during this endeavor. Their understanding, patience, and encouragement helped me a lot. I am especially thankful to my mother for her unwavering belief in me and her constant motivation to complete my PhD. This accomplishment would not have been possible without her. I deeply wish she was here to witness this milestone.

I would also like to acknowledge the use of OpenAI's ChatGPT for assisting with language refinement in this thesis.

Nivedita Shrivastava

Abstract

Securing neural processing units (NPU) is critical, as neural network (NN) models embody valuable intellectual property (IP), requiring substantial investments of time, effort, and resources. For example, data collection for a mid-sized AI project via platforms like Amazon Mechanical Turk can exceed \$100K, while training methodologies often involve multi-million dollar investments. Large-scale training on computational clusters can take weeks, further emphasizing their value. This makes ML models prime targets for attacks, where theft can compromise IP, privacy, and competitive advantage.

Securing NPUs necessitates enforcing key security properties: confidentiality (preventing unauthorized access), integrity (detecting and preventing tampering), authentication (ensuring access for genuine users), freshness (preventing replay attacks), and resilience to side-channel attacks. These properties defend against threats such as eavesdropping, tampering, cold boot, replay attacks, and memory and timing-based side-channel leaks.

To ensure the confidentiality of a model, it is essential to encrypt the model weights and biases. Given the large size of modern NN models and increasing demand for fast computation, achieving high-throughput and low-latency encryption becomes critical. AES is among the most widely used and highly secure encryption ciphers. However, its software implementation based on T-tables is susceptible to cache side-channel attacks (CSCA). In our first work, we systematically analyzed more than 100 research articles but we only consider a representative subset for detailed discussion. This selection focuses on techniques with significant impact or novel design, while omitting redundant or obsolete approaches. We show that while address-based countermeasures (CMs) introduce obfuscation, they can still be broken in $n^{O(\log(\log(n)))}$ time for popular ciphers like AES.

However, this time bound translates to billions of encryptions, posing a substantial challenge for practical feasibility. To reduce the required number of attack iterations, first we propose a theoretical framework to systematically classify the state-of-the-art countermeasures. Then, we propose a two-phase attack framework to efficiently retrieve the

secret key even in the presence of these advanced countermeasures in just 51,840 tries. This highlights the need for a secure and fast encryption engine that can withstand current and futuristic attacks.

Sadly, the optimization efforts for designing fast encryption engines have focused solely on architecture, as security remained unquantified. This prevented modifications to the cipher’s core functionality, such as reducing rounds — a tempting shortcut that risks compromising security guarantees. However, recent works have demonstrated that security can now be quantified. Leveraging their insights, we exploit the fact that if the plaintext distribution has sufficient randomness, fewer encryption rounds are required to generate ciphertexts that maintain the same level of security as AES. Building on this foundation, we introduce a novel encryption engine that dynamically generates ephemeral keys for each data block and provides the same security as AES with nearly $5\times$ faster speed.

Next, we observe that large model sizes hinder the fast implementation of basic security measures. Trusted execution environments (TEEs) like Intel SGX and TrustZone offer a protected on-chip execution environment. However, their high overheads make them inefficient for memory-intensive NPUs. To address the broader security challenges, we first conducted an extensive survey of neural processing unit (NPU) architectures to understand their design. The main issue lies in securing the off-chip data. To solve this, we extensively analyzed and characterized the off-chip memory traffic pattern of different NPUs and designed a fast hardware circuit to predict these patterns. Using these predictions, we implemented mechanisms to ensure the integrity and freshness of the off-chip data at the layer level. This helped us to design an ultra-fast and secure NPU that is 20% faster.

In addition to securing model weights, we identified that the model’s parameters itself are attractive targets for adversaries as also they pave the path for mounting other attacks. These parameters are susceptible to leaks through memory- and timing-based side channels. To address this, we classified existing countermeasure into four categories and observed that these countermeasures primarily introduce additive noise to obfuscate the leaked patterns. We highlight the shortcomings of these countermeasures by demonstrating three attacks on them. So, to further enhance security, we introduced an additional layer of defense by incorporating multiplicative noise in the form of compression. Leveraging space-filling curves, our approach securely tiles, compresses, and bins data, effectively mitigating side-channel attacks and expanding the search space to 10^{70} , ensuring robust protection.

In this thesis, we propose a comprehensive and structured approach to tackle the diverse security threats faced by NPUs. By integrating a combination of novel solutions, we aim to develop NPUs that are not only secure but also fast, ensuring robust protection against modern attack vectors.

संक्षेप

तंत्रिका प्रसंस्करण इकाइयों (या एनपीयू) को सुरक्षित करना महत्वपूर्ण है, क्योंकि तंत्रिका नेटवर्क (या एनएन) मॉडल में मूल्यवान बौद्धिक संपदा (या आईपी) शामिल है, जिसमें समय, प्रयास और संसाधनों के पर्याप्त निवेश की आवश्यकता होती है। उदाहरण के लिए, अमेज़न मैकेनिकल टर्क जैसे प्लेटफार्मों के माध्यम से एक मध्यम आकार की कृत्रिम बुद्धिमत्ता परियोजना के लिए डेटा संग्रह १ लाख डॉलर से अधिक हो सकता है, जबकि प्रशिक्षण पद्धतियों में अक्सर बहु-मिलियन डॉलर का निवेश शामिल होता है। कम्प्यूटेशनल समूहों पर बड़े पैमाने पर प्रशिक्षण में हफ्तों लग सकते हैं, जो उनके मूल्य पर और जोर देता है। यह एमएल मॉडल को हमलों के लिए प्रमुख लक्ष्य बनाता है, जहां चोरी आईपी, गोपनीयता और प्रतिस्पर्धी लाभ से समझौता कर सकती है।

एनपीयू को सुरक्षित करने के लिए प्रमुख सुरक्षा गुणों को लागू करने की आवश्यकता होती है: गोपनीयता (अनधिकृत पहुंच को रोकना), अखंडता (छेड़छाड़ का पता लगाना और रोकना), प्रमाणीकरण (वास्तविक उपयोगकर्ताओं के लिए पहुंच सुनिश्चित करना), ताजगी (रिप्ले हमलों को रोकना) और साइड-चैनल हमलों के लिए संघर्ष क्षमता। ये गुण जासूसी, छेड़छाड़, कोल्ड बूट, रिप्ले हमलों, और स्मृति और समय-आधारित साइड-चैनल लीक जैसे खतरों से बचाव करते हैं।

मॉडल की गोपनीयता सुनिश्चित करने के लिए, मॉडल के वेट्स और पूर्वाग्रहों को कूटबद्ध करना आवश्यक है। आधुनिक एनएन मॉडल के बड़े आकार और तेजी से गणना की बढ़ती मांग को देखते हुए, उच्च-श्रुपुट और कम-विलंबता एन्क्रिप्शन प्राप्त करना महत्वपूर्ण हो जाता है। जबकि एईएस, सबसे लोकप्रिय और सुरक्षित साइफर में से एक, व्यापक रूप से अपनाया जाता है, इसका टी-टेबल आधारित सॉफ्टवेयर कार्यान्वयन कैश साइड-चैनल हमलों (या सीएससीए) के लिए असुरक्षित है। हमारे पहले काम में, हमने १०० से अधिक शोध लेखों का व्यवस्थित रूप से विश्लेषण किया और दिखाया कि जबकि पता-आधारित प्रतिरोधात्मक उपाय (या सीएम) अस्पष्टता का परिचय देते हैं, लेकिन उन्हें अभी भी एईएस जैसे लोकप्रिय साइफर के लिए ($n^O(\log(\log(n)))$) समय में तोड़ा जा सकता है।

हालांकि, यह समय सीमा अरबों एन्क्रिप्शन में बदल जाती है, जो व्यावहारिक व्यवहार्यता के लिए एक महत्वपूर्ण चुनौती पेश करती है। हमले के पुनरावृत्तियों की आवश्यक संख्या को कम करने के लिए, पहले हम अत्याधुनिक सीएम को व्यवस्थित रूप से वर्गीकृत करने के लिए एक सैद्धांतिक ढांचे का प्रस्ताव करते हैं। फिर, हम केवल ५१,८४० प्रयासों में इन उन्नत सीएमएस की उपस्थिति में भी गुप्त कुंजी को कुशलता से पुनः प्राप्त करने के लिए दो-चरणीय हमले के ढांचे का प्रस्ताव करते हैं। यह एक सुरक्षित और तेज कूटलेखन इंजन की आवश्यकता पर प्रकाश डालता है जो वर्तमान और भविष्य के हमलों का सामना कर सकता है।

अफसोस की बात है कि तेज एन्क्रिप्शन इंजनों को डिजाइन करने के लिए अनुकूलन प्रयासों ने पूरी तरह से वास्तुकला पर ध्यान केंद्रित किया है, क्योंकि सुरक्षा अगणनीय रही है। इसने साइफर की मुख्य कार्यक्षमता में संशोधनों को रोक दिया, जैसे कि राउंड को कम करना-एक आकर्षक शॉर्टकट जो सुरक्षा गारंटी से समझौता करने का जोखिम उठाता है। हालांकि, हाल के कार्यों ने प्रदर्शित किया है कि सुरक्षा की मात्रा अब निर्धारित की जा सकती है। उनकी अंतर्दृष्टि का लाभ उठाते हुए, हम इस तथ्य का फायदा उठाते हैं कि यदि प्लेनटेक्स्ट वितरण में पर्याप्त यादृच्छिकता है, तो सिफरटेक्स्ट उत्पन्न करने के लिए कम एन्क्रिप्शन राउंड की आवश्यकता होती है जो एईएस के समान स्तर की सुरक्षा बनाए रखते हैं। इसके आधार पर, हम एक नए एन्क्रिप्शन इंजन का प्रस्ताव करते हैं जो गतिशील रूप से प्रत्येक डेटा ब्लॉक के लिए अल्पकालिक कुंजी उत्पन्न करता है और ५ गुना तेज गति के साथ एईएस के समान सुरक्षा प्रदान करता है।

इसके बाद, हम देखते हैं कि बड़े मॉडल आकार बुनियादी सुरक्षा उपायों के तेजी से कार्यान्वयन में बाधा डालते हैं। जबकि इंटेल एसजीएक्स और ट्रस्टजोन जैसे विश्वसनीय निष्पादन वातावरण (या टीईई) एक सुरक्षित ऑन-चिप निष्पादन वातावरण प्रदान कर सकते हैं, उनके पास बहुत अधिक ओवरहेड्स हैं और स्मृति-गहन एनपीयू के लिए उपयुक्त नहीं हैं। व्यापक सुरक्षा चुनौतियों का समाधान करने के लिए, हमने पहले एनपीयू आर्किटेक्चर का एक व्यापक सर्वेक्षण किया ताकि उनके डिजाइन को समझा जा सके। मुख्य मुद्दा ऑफ-चिप डेटा को सुरक्षित करने में निहित है। इसे हल करने के लिए, हमने विभिन्न एनपीयू के ऑफ-चिप मेमोरी ट्रैफिक पैटर्न का व्यापक रूप से विश्लेषण और विशेषता की और इन पैटर्न का पूर्वानुमान करने के लिए एक तेज हार्डवेयर सर्किट तैयार किया। इन पूर्वानुमानों का उपयोग करते हुए, हमने लेयर स्तर पर ऑफ-चिप डेटा की अखंडता और ताजगी सुनिश्चित करने

के लिए तंत्र को लागू किया। इसने हमें एक अल्ट्रा-फास्ट और सुरक्षित एनपीयू डिजाइन करने में मदद की जो २०% तेज है।

मॉडल भार को सुरक्षित करने के अलावा, हमने पहचान की कि मॉडल के पैरामीटर अपने आप में विरोधियों के लिए आकर्षक लक्ष्य हैं और साथ ही वे अन्य हमलों को बढ़ाने का मार्ग भी प्रशस्त करते हैं। ये पैरामीटर स्मृति-और समय-आधारित साइड चैनलों के माध्यम से रिसाव के लिए अतिसंवेदनशील हैं। इसे संबोधित करने के लिए, हमने मौजूदा सीएम्स को चार श्रेणियों में वर्गीकृत किया और देखा कि ये सीएम मुख्य रूप से लीक हुए पैटर्न को अस्पष्ट करने के लिए एडिटिव नॉइज़ पेश करते हैं। हम इन सीएम्स पर तीन हमले करके उनकी कमियों को उजागर करते हैं। इसलिए, सुरक्षा को और बढ़ाने के लिए, हमने संपीड़न के रूप में गुणक नॉइज़ को शामिल करके रक्षा की एक अतिरिक्त परत पेश की। स्पेस-फिलिंग कवर्स का उपयोग करके, हमारा दृष्टिकोण डेटा को सुरक्षित रूप से टाइल करता है, संपीड़ित करता है और बिन करता है, जिससे साइड-चैनल हमलों को प्रभावी रूप से कम किया जाता है और खोज स्थान को 10^6 तक विस्तारित किया जाता है, जिससे मजबूत सुरक्षा सुनिश्चित होती है।

इस थीसिस में, हम एनपीयू द्वारा सामना किए जाने वाले विविध सुरक्षा खतरों से निपटने के लिए एक व्यवस्थित और समग्र दृष्टिकोण का प्रस्ताव करते हैं। नए समाधानों के संयोजन को एकीकृत करके, हमारा लक्ष्य एनपीयू विकसित करना है जो न केवल सुरक्षित हैं बल्कि तेज भी हैं, जो नवीनतम आक्रमण विधियों के खिलाफ ठोस सुरक्षा सुनिश्चित करना।

Contents

Certificate	i
Acknowledgements	v
Abstract	vii
List of Figures	xxiii
List of Tables	xxxix
1 Introduction	1
1.1 Summary	9
2 Hardware Architectures for Generative Adversarial Networks	11
2.1 Introduction	11
2.1.1 Chapter Organization	12
2.2 Background and Motivation	13
2.2.1 Preliminaries	13
2.2.2 Types of Operations	14
2.2.3 Classification	16
2.3 Accelerator Architectures	16

2.3.1	Architectures for Mapping Computations to PEs	17
2.3.2	Interconnect Architectures	18
2.3.3	Architectures for Nearest-Neighbor Upsampling Strategy	20
2.3.4	Architectures for Instance Normalization Layer	22
2.3.5	Architectures for Dilated CONV	25
2.3.6	Architectures for 3D DeCONV	27
2.3.7	Architectures for Training	29
2.4	Optimization Techniques	36
2.4.1	Using Tiling and Unrolling	38
2.4.2	Exploiting Parallelism	41
2.4.3	Using Winograd-based CONV	45
2.4.4	Using Fermat Number Transform Based CONV	49
2.4.5	Using Sparsity-Related Techniques	52
2.4.6	Handling Overlapping Sum Problem	57
2.4.7	Handling Irregular Memory Accesses	63
2.5	Open Research Challenges and Future Roadmap	64
2.6	Conclusion	66
3	Analysis of Countermeasures for Cache Side Channel Attacks	69
3.1	Introduction	69
3.2	Background	71
3.2.1	Multicore Processors and Shared Caches	71
3.2.2	Cache Architecture	72
3.2.3	Cache-based Side-Channel Attacks (CSCAs):	73

3.2.4	Countermeasures (CMs):	75
3.2.5	Mutual Information (MI):	75
3.3	Properties	77
3.4	Examples of Property Violations	80
3.4.1	Property 1: No Functionality Disruption	80
3.4.2	Property 2: A Large Number of Tries	80
3.4.3	Property 3.1: Deterministic Obfuscation	81
3.4.4	Property 3.2: Nondeterministic Obfuscation	82
3.5	Discussion: Impossibility Results and Design of a Universal Countermeasure	82
3.5.1	Timing-based Countermeasure	83
3.5.2	Address-based Countermeasure	84
4	Modeling Cache-based Side-Channel Attacks and Countermeasures	89
4.1	Introduction	89
4.2	Background and Related Work	92
4.2.1	Software Implementation of AES	92
4.2.2	Related Work	93
4.3	Threat Model	94
4.4	Classification of Countermeasures (CMs)	95
4.4.1	Redundancy-Based Techniques	95
4.4.2	Access- Removal-Based Techniques	96
4.4.3	Randomization-Based Techniques	97
4.4.4	Remapping-Based Techniques	98
4.5	Attack Methodology	99

4.5.1	Phase-1: Deriving the VN-to-PN Mapping with the First Key Byte	99
4.5.2	Phase-2: Inferring Unknown Key Bytes	101
4.5.3	In the Presence of Remapping	102
4.6	Results	102
4.7	Conclusion	105
5	DisCrypt: An Ultra-Fast High-Throughput Encryption Engine	107
5.1	Introduction	107
5.1.1	Problem Solved in This Chapter: Practical <i>AES*</i> :	108
5.2	Background	110
5.2.1	Theory of Layouts	110
5.2.2	<i>AES*</i>	111
5.3	Characterization	113
5.3.1	Characterization of the Total Variation Distance (TVD)	113
5.3.2	Sketch of the Solution	114
5.3.3	Characterization of the Random Number Generators	115
5.4	Design of DisCrypt	116
5.4.1	Stage A: Mask Generation Engine	117
5.4.2	Stage B: Encryption Engine	118
5.4.3	<i>AES*</i> Encryption Engine	118
5.5	Evaluation	119
5.5.1	Setup and Configurations	119
5.5.2	Results	120
5.5.3	Discussion on Security	121

5.5.4	Use Cases and Limitations	121
5.6	Related Work	121
5.7	Conclusion	122
6	Securator: A Fast and Secure Neural Processing Unit	123
6.1	Introduction	123
6.2	Background	126
6.2.1	Intel SGX	126
6.2.2	Convolution	127
6.2.3	TNPU and GuardNN	129
6.3	System Design and Threat Model	130
6.4	Characterization of DNN Workloads in a Secure Environment	131
6.4.1	Setup and Benchmarks	131
6.5	Analytical Characterization of Patterns in ML Workloads	133
6.5.1	Convolution Layer	134
6.5.2	Other Kinds of Layers	138
6.6	Design of Securator	141
6.6.1	Overview	141
6.6.2	A Programmable Version Generation Scheme	142
6.6.3	Details of the Encryption Process	143
6.6.4	MAC Generation	143
6.7	Security Analysis	144
6.7.1	Adversary Capabilities	145
6.7.2	Adversary Goals	145

6.7.3	Security Guarantees	145
6.8	Evaluation	147
6.8.1	Setup	147
6.8.2	FPGA Prototype	147
6.8.3	Verilog Synthesis Results	148
6.8.4	Performance Analysis	149
6.8.5	Scalability Analysis: Securator+	150
6.8.6	Limitations	151
6.9	Related Work	151
6.9.1	Outsourcing Computations	152
6.9.2	Optimizing Intel SGX	153
6.9.3	Designing a Custom TEE	153
6.10	Conclusion	154
7	NeuroPlug: Plugging Side-Channel Leaks in NPUs	155
7.1	Introduction	155
7.2	Background	159
7.2.1	Convolution Neural Networks (CNNs)	159
7.2.2	Model Extraction Attacks	160
7.2.3	State-of-the-art Countermeasures (CMs)	161
7.3	System Design and Threat Model	161
7.4	Motivation	163
7.4.1	1 st Generation Statistical Attack (SS)	163
7.4.2	2 nd Generation Kerckhoff Attack (KK)	163

7.4.3	3 rd Generation Residual Side-channel Attack (SI)	164
7.5	Theoretical Analysis	166
7.5.1	The Additive Noise: N	166
7.5.2	The Multiplicative Noise: C	167
7.5.3	Predicted Distribution of X	167
7.6	Characterization of the Entropy	170
7.7	Design of NeuroPlug	173
7.7.1	Overview	173
7.7.2	Deep Look at SFC-based Computation	174
7.7.3	Dealing with Halo Pixels	176
7.7.4	Adding Custom Noise	177
7.7.5	Layer Splitting and Fusion	177
7.7.6	Pooling, ReLU and Skip Connections	177
7.7.7	Overall Dataflow of <i>Neuroplug</i>	178
7.8	Performance Evaluation	179
7.8.1	Setup and Benchmarks	179
7.8.2	Performance Analysis	181
7.8.3	Sensitivity Analysis	182
7.8.4	Hardware Overheads	183
7.8.5	Integration of Securator	184
7.9	Security Evaluation	184
7.9.1	Search Space Sensitivity	184
7.9.2	Case Study 1 – Defense from HuffDuff	186

7.9.3 Case Study 2 – Defense from Reverse Engg.	188
7.10 Related Work	189
7.10.1 DNN Architecture Extraction: CDTV Metrics	189
7.10.2 DNN Architecture Protection	190
7.11 Discussion and Conclusion	191
8 Conclusion and Future Work	193
8.1 Future Work	194
Bibliography	197
List of Publications	223
Biography	225

List of Figures

1.1	Threat model. SCAs refer to the side-channel attacks.	3
1.2	An overview of the thesis. We consider only the inference stage of ML execution in our threat model.	5
2.1	A deep convolutional generative adversarial network, where the generator produces artificial data, and the discriminator learns to distinguish between real and artificially generated data. [1]	14
2.2	Different types of operations: (a) Traditional (direct) CONV with a stride of 2, called strided-CONV (no zero-insertion) (b) Transposed CONV (zero-insertion in the input) (c) Dilated CONV (zero-insertion in kernel)	15
2.3	The accelerator proposed by Yan et al. [2]. The convolution core performs CONV/DeCONV with four convolution engines which consists of (8×8) PE array. The cold buffer stores overlapping results, tiling results, and the input data for performing elementwise addition.	18
2.4	The overall approach of Mao et al. [3]. The design is comprised of five main parts: Program, interface, compiler, memory controller and ReRAM-based PIM.	20
2.5	(a) TrCONV process with padding (b) two upsampling strategies: zero-insertion and nearest-neighbor based [4]	21

2.6	The accelerator proposed by Yu et al. [4] utilizes an <i>ifmap</i> buffer and a kernel weight buffer, both of which employ a ping-pong memory structure. The data processing consists of a post-processing block and a computational engine, where data is fetched and rearranged using the data fetch unit. The instruction control unit sends commands to all blocks.	22
2.7	(a) CONV operation (b) Dataflow for CONV operation (c) DeCONV operations and (d) Dataflow for DeCONV operation [5]	24
2.8	The architecture proposed by Im et al. [6] consists of four PE arrays, a controller along with a “dilation rate” controller and a aggregation core. A PE array has five columns and three rows along with an adder tree and two delay cells. Aggregation core collects the partial sums and accumulates and manages the PE’s computation result.	26
2.9	3D input-oriented mapping [7]. I1 to I3 are neighboring activations of <i>ifmap</i> . They are mapped to neighboring PEs, where they are multiplied with a 3D kernel and produce a 3D result. These results are aggregated over time. The overlapped results from the PEs that process I2 and I3 are sent to the PEs that process I1, and the pointwise summation is done. . . .	28
2.10	Dataflow of a PE proposed by Wang et al. [7]	28
2.11	(a) Zero-skipping output-stationary (ZeSOS) microarchitecture. (b) Zero-skipping weight-stationary (ZeSWS) microarchitecture. [8]	30
2.12	Overall design of Song et al. [8]. ZeSWS computes D_w and G_w . ZeSOS does all the other computations. Both ZeSWS and ZeSOS have 4×4 PEs. The following four types of buffers are used. The first layer’s data are loaded into one of the in/out buffers. The output of ZeSOS is stored in another in/out buffer, which becomes the input buffer for the next layer. D_w and G_w require data from the FWP and error from BWP, and they are stored in the data buffer and error buffer, respectively. The computation of ∇W is done by ZeSWS. When the kernel size exceeds the size of unrolled kernel weights, partial results are generated for ∇W . They are stored in ∇W buffer.	32
2.13	Architecture of distributed on-chip memory in the technique of Hanif et al. [9]	33

2.14 (a) Binary CONV operation [10]. c channels of size $kh \times kw$ from the input batch generates the combined batch with respect to the kernel batch. (b) partial inexact computing, which computes 3 LSBs using an imprecise adder and remaining bits of the 32b ofmap using a precise adder.	34
2.15 The in-memory accelerator proposed by Roohi et al. [10].	35
2.16 (a) The crop module [11] (here, x and y are the thresholds) (b) traditional tiling approach for CONV and DeCONV where all the tiles of a layer are processed sequentially, before moving on to the next layer. (c) Proposed tiling approach for DeCONV [11], where every tile is processed till the last layer to obtain its output tile.	39
2.17 Optimization of DeCONV [12] for a 2×2 feature map. Most of the redundant multiplication can be avoided.	40
2.18 (a) The overall architecture of Hsiao et al. [13]. The two SRAM supports streaming process with external memory access and overlapped computations. (b) functional block diagram. [13]. Blocks are activated according to type of operation.	42
2.19 (a) PE used for DeCONV layer [14] (b) Architecture of DeCONV engine (c) Architecture of DeCONV Unit (DU)	43
2.20 Kernel decomposition in the technique of Shi et al. [15]	44
2.21 Untangling a standard CONV into a set of 1×1 CONV [15]	45
2.22 Block diagram of the accelerator proposed by Chang et al. [16]. In 2D Winograd CONV, $Output = A^T[(GfG^T) \odot (B^TIB)]A$. Here, A, B and G are the transformation matrices and \odot shows the element-wise multiplication. f and I show filter and input, respectively.	46
2.23 Overview of Winograd DeCONV dataflow. [16]. The input tiles and the Winograd filters are reorganized into a $n^2 \times N$ sized matrix and M matrices of size $n^2 \times N$ respectively. The reorganized filters exhibit sparsity at the vector level.	47
2.24 Overall technique of Di et al. [17] (S1: kernel decomposition, S2: Winograd transform, S3: element-wise multiplication, S4: inverse Winograd transform, S5: rearrangement, S6: channel accumulation)	49

2.25	Working of 2D overlap-and-save FNT [18]. 2D FNT transforms ifmap and filters into a finite field. Then, element-wise multiplication is performed, followed by the IFNT.	50
2.26	TrCONV computation based on 1D overlap-and-save FNT [18]. Instead of processing the entire zero-padded fmap, 1D FNT is performed on the non-zero rows.	51
2.27	Stage-pipelined fast FNT kernel for FNT size N [18], capable of performing both 2D and 1D FNT. In 1D FNT, the results are directly taken from the output row, skipping column and transpose FNT. For 2D FNT, the row FNT output is passed through a transpose matrix before applying column FNT.	51
2.28	Overall design of the proposed accelerator for GAN and CNN acceleration (CE = computing element). [18]	52
2.29	Dataflow in the technique of Chen et al. [19]. At cycle 0, the first four activations and weights are fetched and supplied to the “zero-removal unit,” which filters out the ineffectual data. The effectual data is sent to FIFO0 and FIFO1. Cycle 1 follows the same process as Cycle 0, directing the effectual data to FIFO2 and FIFO3. The FIFO with the largest amount of data transfers the data to the “dual-channel accumulation.”	53
2.30	(a) Visualization of the zero-insertion step in TrConv for a 4×4 input, showcasing the transformed input. White squares denote the inserted zero values. (b) Convolution dataflow used for performing TrConv operations [20]. The flow of data after applying (c) output row reorganization and (d) filter row reorganization technique. These optimizations collectively minimize idle (white) operations and enhance resource utilization.	54
2.31	Conversion of DeCONV to CONV and its optimization using (a) a load-balancing approach [21] [Chang et al., TCVST 2018] and (b) a pruning and encoding approach [22] [Chang et al., DATE 2019].	56
2.32	Case 1: Example of $K_d \bmod S_d = 0$. The 4×4 DeCONV kernel and stride 2 is split into four 2×2 subkernels for performing CONVs. ib =input block, zib = zero-inserted input block, sk =subkernel. Performing DeCONV by (a) zero-insertion (b) removing zero-related operations (c) using fine-grain approach proposed by Mao et al. [23].	58

2.33	The architecture for CONV-to-DeCONV system. It comprises of weight buffers, row accumulator, feature buffers, computing elements, controller and selection cells. Selection cells selects ping-pong buffers [23].	59
2.34	Design of computing elements [23], which have m CU groups with n CUs each, and an adder tree array. An input row is shared between a CU group and a CU performs 1D CONV by rows. Adder tree of $m \times 6$ adder sums up the results from the CUs.	60
2.35	(a) Overall architecture of Liu et al. [24]. All processing data and coefficients are stored in off-chip memory. (b) Design of DeCONV accelerator which supports different parameters of the DeCONV layer.	62
2.36	(a) Achieving DeCONV by dividing it into four CONV operations and then, concatenating the results (b) The sequence of operations (c) Storing ifmap1 and ifmap2 in non-contiguous locations to achieve concatenation operation without additional overhead [25]	64
3.1	A timeline of countermeasures and CTAs	71
4.1	VN-to-PN mapping under redundancy-based countermeasure. V_i denotes the i^{th} virtual node and P_i denotes the i^{th} physical node. In (a), the mapping is deterministic and one-to-one. In (b), redundancy is introduced—multiple PNs (cache lines) are mapped to the same VN, creating ambiguity for the attacker.	95
4.2	Effect of removal-based countermeasure on virtual-to-physical node mapping. (a) Original deterministic mapping between virtual nodes V_i and physical nodes P_i . (b) After applying the countermeasure, some mappings are removed or hidden, making some VN-to-PN relationships unobservable to the adversary.	97
4.3	Effect of randomization on virtual-to-physical node mapping. (a) Original deterministic mapping between virtual nodes V_i and physical nodes P_i (b) After applying a randomization-based countermeasure, the mapping becomes non-deterministic, thereby enhancing security by disrupting predictable access patterns.	98

4.4	Remapping-based countermeasure. V_i denotes the i^{th} virtual node and P_i denotes the i^{th} physical node. (a) Without redundancy: the mapping changes randomly over time. (b) With redundancy: additional connections are introduced, which also changes with time, adding ambiguity for the attacker.	99
4.5	Number of tries required in the presence of redundancy (non-Sparse) based countermeasure	103
4.6	Number of tries required in the presence of removal based countermeasure. F represents not enough samples to mount the attack.	103
4.7	Number of tries in the presence of remapping, randomness and redundancy(non-sparse)	104
4.8	Attack time (in seconds) in the presence of remapping, randomness and redundancy (non-sparse)	104
5.1	Variation of the TVD of the ciphertext layout distributions with rounds. Different colored lines represent different initial TVD values of the input layout distributions.	114
5.2	Comparison of the performance of classical pseudo random number generators and chaotic maps	116
5.3	Overall design of <i>Discrypt</i>	117
5.4	<i>XoroShiro</i> engine for generating the <i>tweak</i>	118
6.1	A timeline depicting a pragmatic shift from outsourcing execution from an unsecure compute-rich platform to designing an optimized and secure custom TEE for accelerating ML workloads.	126
6.2	Graphical representation of a tiled convolution operation. The <i>ifmaps</i> , <i>ofmaps</i> , input and output channels are tiled.	128
6.3	An overview of the system design and threat model (note that it is not necessary that the NPU and CPU are in the same package)	130
6.4	Performance comparison	132

6.5	Cache miss rates for the MAC cache and counter cache, respectively. . . .	132
6.6	The high-level design of <i>Securator</i>	142
6.7	Experimental setup	148
6.8	Normalized performance for different workloads	149
6.9	Normalized memory traffic for different workloads	150
6.10	Normalized execution latencies for scalability analysis (normalized to $111 \times 111 \times 3$)	151
7.1	The overall scheme	159
7.2	The system design and the threat model (similar to [26–28]). An adversary snoops the address and the data buses to infer the memory access patterns.	162
7.3	SS attack: Adding dummy writes (784) to obfuscate the true writes (1568) in the scaled version of the first layer of <i>Vgg16</i> ($32 \times 32 \times 3$). Adversaries eliminate the dummy writes as they are not read in the subsequent layer (Liu et al. [29])	163
7.4	KK attack: A noise with a hardwired mean value of 22400 is added to obfuscate the actual volume of the first layer of <i>Vgg16</i> ($224 \times 224 \times 3$) [28, 30]. The adversary will eliminate the noise using an SS attack followed by the <i>Kerckhoff</i> 's (KK) attack	164
7.5	SI attack: Layer divider technique [31] to obfuscate RAW dependencies. The adversary filters out the fake RAW dependencies by checking the unchanged values.	165
7.6	Rank of X_r (layer volume) for different layers of <i>Vgg16</i> . From our point of view the uniform distribution is the best.	169
7.7	Entropy analysis across (a) all 100 inputs and all models; (b) all models with all zeros as the input; (c) all models with all ones as the input; (d) all models with all random values as the input; (e) all models with vertical bars of zeros and ones as the input (f) all 100 inputs in <i>AlexNet</i> ; (g) all 100 inputs in <i>ResNet</i> ; (h) all 100 inputs in <i>Inception</i> net	171
7.8	The high-level design of <i>Neuroplug</i>	173

7.9	SFCs (a) Reading <i>ifmaps</i> ; (b) Reading the filters in a layer; (c) Writing <i>ofmaps</i> ; (d) Reading fused filters across two layers	174
7.10	Halo pixels in a layer	176
7.11	Dataflow of <i>Neuroplug</i>	178
7.12	Normalized performance	181
7.13	Normalized memory traffic	182
7.14	Noise sensitivity analysis	182
7.15	Performance overhead vs search space size (► overheads increase very gradually)	183
7.16	Variation in the smart search space size with and without compression (VGG16). Conclusion: compression is needed.	185
7.17	Accuracy of the models estimated by an attacker	185
7.18	The boundary effect decreases with the layer depth. 64 different 64×64 inputs are created by varying the position of the one in the first row of the input with all zeros.	187
7.19	Security analyses for Case Study 1. <i>MI</i> is b/w the secret parameter and the leaked metric. <i>CC</i> is b/w the leaked traces and the filter size.	188

List of Tables

1.1	Security requirements for a secure Neural Processing Unit (NPU). Integrity - the data cannot be tampered with without getting detected, Freshness - if a part of the data is replaced with an older version, it can be detected, Confidentiality - a malicious user cannot read the data. . . .	2
2.1	List of acronyms used in this chapter	13
2.2	Difference between transposed CONV and dilated CONV	15
2.3	A classification of the research works	16
2.4	A classification of optimization techniques	37
3.1	List of papers violating the properties. The CTA supports the fact that the failure of a CM is due to a property violation.	78
4.1	Summary of the Countermeasures	94
4.2	Simulation Parameters	103
5.1	Comparison between AES-128 and AES*	112
5.2	RNG techniques. (x, y, z) represents random map values while other variables represent control parameters.	116
5.3	Evaluated configurations	120
5.4	Comparative analysis of the encryption schemes. Thr. \rightarrow Throughput . . .	120

6.1	NPU configuration, list of benchmarks, and the terminology used in this chapter	131
6.2	Pattern table for convolution: various possibilities for scheduling the input tiles for input reuse and output reuse. $\alpha_K = \frac{K}{K_T}; \alpha_C = \frac{C}{C_T}; \alpha_{HW} = \frac{H.W}{H_T.W_T}$; RP \rightarrow VN read pattern, WP \rightarrow VN write pattern, – refers to empty or not applicable. Assumption: As per the loop order, we fetch a <i>group of tiles</i> that is read, processed, and written in one go. They have the same VN, which is generated by the aforementioned WP/RP patterns.	134
6.3	Pattern Diagrams: The colored dots, squares, and stars, etc., represent accesses for a group of tiles (fetched and processed at one go). The x -axis represents time; the y -axis represents the VN. Axes not shown in other figures to enhance readability.	135
6.4	Pattern table for convolution: Different methods of scheduling weight tiles for weight reuse.	137
6.5	Pattern table for matrix multiplication ($R = P \times Q$): Various methods for tiling the input. Dimensions of P : $H \times C$, Q : $C \times W$ $\alpha_C = C/C_T; \alpha_H = H/H_T; \alpha_W = W/W_T$	138
6.6	Pattern table for image pre-processing (Style-1)/Pooling ($S_x = T_x(X)$): Possibilities for scheduling the output tiles. (H_T, W_T, C_T) represents the tiling factor. ($C = K$)	139
6.7	Pattern table for image pre-processing (Style-2 : $S = T(R, G, B)$): Methods of scheduling input tiles ($K=1$)	139
6.8	Pattern table for image pre-processing (Style-3 : $S_i = T_i(R, G, B)$)	140
6.9	Simulated designs	147
6.10	Overhead associated with the h/w structures	148
6.11	A comparison of related work	152

7.1	<i>Neuroplug</i> offers comprehensive protection and undergoes thorough validation using various security metrics. ● property is satisfied and ○ means it is not. SrS. is the search space, Acc. is model re-training accuracy or how accurately the layer parameters are estimated. Theo. refers to information-theory based security metrics.	156
7.2	Glossary of the symbols	160
7.3	System configuration	172
7.4	Design principles	175
7.5	Configuration of the NPU	180
7.6	DDR3-1600K DRAM Configuration	180
7.7	Workloads	180
7.8	ASIC area and power utilization for the additional hardware components used in <i>Neuroplug</i>	183
7.9	Security analysis for Case Study 2	188
7.10	Different Attacks on NNs. (C,D,T,V) stands for (Count, Distance, Time, Volume)	189

