IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 19 August 2025, accepted 31 August 2025, date of publication 9 September 2025, date of current version 16 September 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3607595

RESEARCH ARTICLE

An Advanced Detection Framework for Embedded System Vulnerabilities

MANSOUR ALQARNI, (Fellow, IEEE), AND AKRAMUL AZIM, (Senior Member, IEEE)
Ontario Tech University, Oshawa, ON L1G 0C5, Canada
Corresponding author: Mansour Alqarni (Mansour.Alqarni@ontariotechu.net)

ABSTRACT Embedded systems serve as the foundation for modern computing in industrial, IoT, and defense applications. However, their increased adoption exposes them to security threats across multiple levels, from application software to operating systems and hardware. Traditional security mechanisms often focus on a single layer, missing cross-layer vulnerabilities that can be exploited by sophisticated attacks. This paper introduces a unified multi-layer vulnerability detection framework that integrates software, operating system, and hardware-level security analysis into a single LLAMA3-based deep learning model. The framework leverages a combined dataset consisting of source code vulnerabilities (C programming), Linux Kernel exploits (system calls), and Field-Programmable Gate Array (FPGA) hardware security risks in Hardware Description Languages (HDLs) such as Verilog and VHSIC Hardware Description Language (VHDL) and bitstream analysis. By merging these diverse vulnerability types into a single learning model, the framework is capable of detecting security threats across the entire embedded system stack. Applying static and dynamic analysis techniques across multiple layers, the proposed LLAMA3-based model achieves state-of-the-art detection accuracy across embedded system security domains. Experimental results demonstrate that the integrated framework outperforms existing layer-specific models, achieving 99.30% accuracy for software vulnerabilities, 99.16% accuracy for OS-level exploits, and 95.6% accuracy for hardware security threats.

INDEX TERMS Embedded system security, vulnerability detection, machine learning for cybersecurity, FPGA security, linux kernel exploits, LLAMA3.

I. INTRODUCTION

Embedded systems are at the core of modern computing, powering applications in automotive systems, industrial automation, medical devices, IoT networks, and defense technologies. Their widespread adoption, however, makes them prime targets for security vulnerabilities that can be exploited at multiple levels, from application software to operating systems and even hardware configurations. A security flaw at any of these layers can compromise system integrity, reliability, and data confidentiality, leading to severe operational and financial consequences.

Traditional vulnerability detection approaches focus primarily on isolated aspects of embedded system security, such as software bugs, OS kernel exploits, or FPGA hardware security issues. However, these methods fail to address cross-layer security risks, where an exploit in one layer (e.g., software) can lead to privilege escalation in the operating system or even affect hardware behavior. To address this gap, we propose an integrated multi-layer vulnerability detection framework that integrates software, OS, and hardware security analysis into a single deep learning model. Embedded systems are increasingly integrated into cyber-physical systems (CPS), which are used in applications such as industrial automation, automotive systems, and healthcare devices. The growing complexity and interconnectivity of these systems make them particularly vulnerable to cyberattacks. This motivates the need for effective vulnerability detection frameworks to ensure the security and reliability of CPS components.

The associate editor coordinating the review of this manuscript and approving it for publication was Mouquan Shen.

A. THE NEED FOR A MULTI-LAYERED SECURITY APPROACH

Embedded systems differ from traditional computing platforms in several ways. First, they often have limited computational power, making it challenging to deploy traditional resource-intensive security solutions. Second, many embedded devices have long lifecycles, requiring security mechanisms that can protect against long-term threats. Third, embedded systems operate under real-time constraints, meaning security techniques must function efficiently without introducing performance bottlenecks. Most importantly, security vulnerabilities exist at multiple layers, each with unique risks. At the application layer, unsafe programming practices in C lead to issues such as buffer overflows, memory leaks, and input validation flaws. The operating system layer, especially in embedded Linux environments, is vulnerable to privilege escalations, system call exploits, and race conditions. Meanwhile, at the hardware layer, FPGA-based systems are susceptible to bitstream manipulations, side-channel attacks, and insecure Verilog/VHDL implementations. A single-layer security model is insufficient to detect threats that span across software, OS, and hardware. A multi-layered security approach is required to identify vulnerabilities that may propagate across these levels.

B. PROPOSED UNIFIED FRAMEWORK

This paper presents a unified multi-layer vulnerability detection framework that consolidates application, OS, and hardware security analysis into a single LLAMA3-based deep learning model. As shown in Figure 1 the proposed approach integrates static and dynamic security analysis, combining source code vulnerability detection, kernel exploit analysis, and FPGA security validation into a single learning model. To achieve this, we develop a multi-task training methodology that enables the LLAMA3 model to process diverse security datasets, including GitHub (C programming vulnerabilities), EVDD (Linux kernel exploits), and FPGAvul (FPGA bitstream and Verilog security issues). This research makes the following key contributions:

- *Developing a Cohesive Embedded System Security Model:* First approach to integrate software, OS, and hardware security detection into a single deep learning model (LLAMA3).
- *Combining Multi-Domain Security Datasets:* Merges GitHub (C code vulnerabilities), EVDD (Linux kernel exploits), and FPGA bitstream datasets into a single training set.
- *Enhancing Vulnerability Detection with Multi-Task Learning:* Fine-tunes LLAMA3 to analyze C source code, system calls, and bitstreams simultaneously.
- *Empirical Validation and Performance Benchmarking:* Demonstrates significant improvements in vulnerability detection accuracy compared to traditional single-layer models.

The remainder of this paper is structured as follows. Section II reviews related research in vulnerability detection for embedded systems. Section III presents the proposed framework, data preprocessing, and model architecture. Section IV details the multi-task training methodology used to fine-tune LLAMA3 for embedded system security. Section V describes the experimental setup, evaluation metrics, and results. Finally, Section VI concludes the paper with a discussion of findings and future research directions.

II. RELATED WORK

Vulnerability detection in embedded systems has been a widely researched topic, with efforts primarily focused on software security, operating system (OS) security, and hardware-level threat analysis. Traditional methods often rely on static analysis, dynamic analysis, or machine learning-based techniques. However, most existing approaches focus on only one layer of the embedded system stack, limiting their effectiveness in detecting cross-layer vulnerabilities. Most existing vulnerability detection methods often focus on a single layer, failing to capture cross-layer interactions where an exploit in one layer can compromise the entire system. This section reviews recent research in software vulnerability detection, OS kernel security, and hardware-level threat mitigation, highlighting the need for a unified multi-layer framework. As shown in Figure 2 presents the increasing trends of reported vulnerabilities "Red" and cyberattacks "Blue" on embedded systems from 2015 to the present. The data, adapted from the 2024 Verizon Data Breach Investigations Report [1], underscores the urgency for proactive, multi-layer vulnerability detection frameworks in embedded system environments. The combined view allows for an easier comparison between the growing number of security vulnerabilities and the corresponding rise in cyberattacks targeting embedded systems. The data suggests that as embedded systems become more complex and widely integrated into industries such as automotive, healthcare, and IoT, the number of reported vulnerabilities has also increased significantly.

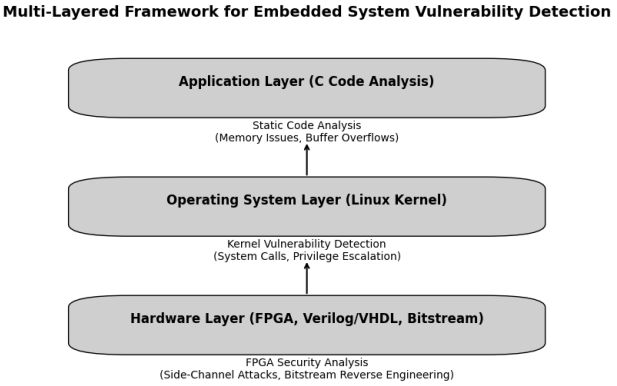

FIGURE 1. Multi-layer vulnerability detection framework.

Firmware vulnerabilities have become a significant concern in recent years. According to the National Vulnerability

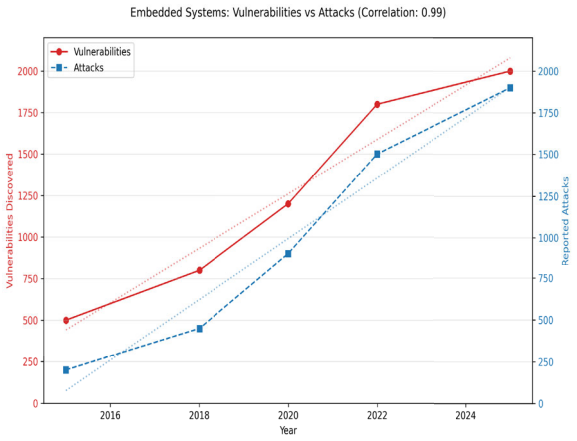

FIGURE 2. **The number of discovered vulnerabilities (left axis, red line) and reported cyber attacks (right axis, blue dashed line). Both metrics display a strong upward trend with a correlation coefficient of 0.99, indicating a near-linear relationship between discovered vulnerabilities and exploitation attempts.**

Database (NVD), more than 4,929 firmware vulnerabilities have been reported in the past two decades [2], highlighting a dramatic increase in security threats at the firmware level.

Detection of software vulnerability in embedded systems with numerous studies we have focused on identifying security vulnerabilities in C and C++ applications for embedded systems. Traditional static analysis tools such as Clang Static Analyzer, Codeql and SonarQube [3], [4], [5] have been used to detect buffer overflows, memory corruption, and input validation flaws. However, these methods often suffer from high false positive rates and fail to detect complex code interactions that lead to security flaws. More recent approaches leverage deep learning models for vulnerability detection [6], [7], [8]. For example, Tian et al. [9] their research indicates that the most significant factor in improving accuracy is introducing vulnerabilities from the dataset into the model. Similarly, Roziere et al. [7] introduced a pre-training objective called DOBF, leverages the structural aspects of programming languages by training models to recover original source code from its obfuscated versions. The authors demonstrate that models pre-trained with DOBF significantly outperform existing approaches across multiple downstream tasks, achieving relative improvements of up to 13% in unsupervised code translation and 24% in natural language code search. However, these methods are limited to application-layer security and do not extend to vulnerabilities in OS kernel or hardware security. Chan et al. [6] developed a Transformer-based code analysis system leveraging large-scale deep learning models to detect vulnerabilities at EditTime; while the developer is writing the code. Their model applies zero-shot, few-shot, and fine-tuning approaches on pre-trained Large Language Models (LLMs) to identify over 250 vulnerability types across incomplete code snippets. Hanif and Maffeis [8] present a deep learning approach to detect security vulnerabilities in source code. Their method involves pre-training a RoBERTa model with a custom tokenization pipeline on real-world code from open-source C/C++ projects. The model learns a deep knowledge representation of the code's syntax and semantics, which is then leveraged to train vulnerability detection classifiers. A summary of commonly encountered software vulnerabilities in embedded systems, along with their typical programming language contexts, is presented in Table 1.

Linux Kernel and Operating System Vulnerability Detection in the embedded systems often run Linux-based operating systems, making them susceptible to kernel exploits, system call vulnerabilities, and privilege escalation attacks. Several research efforts have attempted to detect vulnerabilities in the OS layer using fuzzing, system call anomaly detection, and deep learning methods. [10] proposed both fuzzing-based and N-Gram model to extract vulnerable program behaviors and identify patterns that guide the test case generation phase of traditional fuzzing techniques, aiming to improve fuzzing efficiency from the perspective of Linux system calls. Both technique for detecting privilege escalation vulnerabilities in the Linux kernel, but it required extensive manual intervention and signature-based rules. More recently, EVDD (Embedded Vulnerability Detection Dataset) was introduced [12], providing a large dataset for Linux kernel code for training machine learning-based classifiers. Zhuang et al. [13] propose a simple yet effective optimization method to interpret the activation of specific kernels in convolutional neural networks (CNNs). The approach involves preserving the activation of a target kernel while suppressing activations of other kernels at the same layer, allowing only visual information relevant to the specific kernel to remain in the input.

Hardware-Level Security in Embedded Systems as embedded systems increasingly rely on FPGAs (Field-Programmable Gate Arrays) for hardware acceleration, attackers have shifted focus toward hardware-layer vulnerabilities. Recent research we identified security flaws in bitstream encryption, Verilog/VHDL design practices, and side-channel attacks. Traditional FPGA security methods involve bitstream encryption and integrity checks, but these mechanisms are often bypassed due to weak cryptographic implementations. Mssassi et al. [14] proposed a bitstream reverse engineering attack, showing how unprotected FPGA configurations can be modified to insert hardware Trojans. However, these approaches remain hardware-specific and does not extend to software or OS vulnerabilities. As shown in Table 2, FPGA-based embedded systems face multiple security threats, including Electromagnetic Side-Channel Attacks (EM-SCA), Static Power Analysis Attacks, and Post-Quantum Cryptography (PQC) Side-Channel Attacks. Each of these threats targets different aspects of hardware security, requiring advanced mitigation strategies.

A. LIMITATIONS OF EXISTING APPROACHES AND NEED FOR A UNIFIED MODEL

Most prior research focuses on single-layer security detection, making it inadequate for real-world embedded systems,

TABLE 1. Common Software Vulnerabilities in Embedded Systems.

Vulnerability	Description	Common Languages
Buffer Overflow	Overwriting memory beyond allocated buffer size	C, C++ 6
Use-After-Free	Accessing memory after it has been freed	C, C++
Format String Attack	Exploiting printf-like functions to execute arbitrary code	C
Integer Overflow	Arithmetic operations exceeding variable capacity	C, C++
Stack Overflow	Exceeding stack memory causing function crashes	C, Assembly
Heap Corruption	Altering heap structures leading to crashes or execution control	C, C++
Memory Leak	Failure to free allocated memory, leading to resource exhaustion	C, C++
Null Pointer Dereference	Attempting to access memory at address 0	C, C++
Race Condition	Multiple threads accessing shared memory incorrectly	C, C++, Rust
Code Injection	Executing arbitrary commands via manipulated inputs	C, Python, PHP
Privilege Escalation	Gaining unauthorized system-level access	C, Assembly
Improper Input Validation	Failure to sanitize user inputs	C, Java, Python

TABLE 2. Recent FPGA Security Threats and Their Impact.

Threat Type	Impact
Electromagnetic Side-Channel Attacks (EM-SCA) [15]	Data leakage via EM emissions
Static Power Analysis Attacks [16]	Cryptographic key extraction through power analysis of FPGA hardware
Post-Quantum Cryptography (PQC) Side-Channel Attacks [17]	Security risks in FPGA-based implementations of post-quantum encryption algorithms
Root of Trust (RoT) Attacks [18]	FPGA hardware security breaches
Power Side-Channel Timing Attacks [19]	AES encryption vulnerabilities

where vulnerabilities often propagate across software, OS, and hardware layers. A buffer overflow in an application could lead to kernel-level privilege escalation, which, in turn, could trigger hardware misconfigurations. Existing vulnerability detection tools fail to correlate such cross-layer attacks.

To address these gaps, we propose a unified multi-layer vulnerability detection framework, integrating static and dynamic analysis techniques across all three layers of embedded systems. Unlike prior work, our approach trains a single LLAMA3-based model using a combined dataset from GitHub (C programming bugs), EVDD (Linux kernel exploits), and FPGAvul (FPGA security issues). This unified model is capable of analyzing software, OS, and hardware security threats simultaneously, achieving state-of-the-art detection accuracy.

III. PROPOSED MULTI-LAYER VULNERABILITY DETECTION MODEL

The decision to use LLAMA3 (3.2B Instruct) is driven by its capacity to handle complex code semantics across various abstraction levels. including high-level code down to bitstream or HDL (Hardware Description Language). Unlike legacy ML models trained for specific syntax or domain tasks, LLAMA3 supports generalizable learning and can be fine-tuned using natural code-like inputs.

This multi-layer fine-tuning strategy helps detect cross-layer vulnerabilities, where an attack surface in one layer (e.g., firmware) may propagate or mask weaknesses in another (e.g., hardware). This holistic approach allows for better correlation between seemingly unrelated behaviors in embedded systems, which have not been extensively addressed in prior literature.

A. LLAMA3 CORE ARCHITECTURE

LLAMA3 adopts a decoder-only transformer architecture optimized for language modeling. While its structure largely builds upon LLAMA 2, LLAMA3 introduces further efficiency via Grouped Query Attention (GQA) and continued use of Rotary Positional Embeddings (RoPE). A visual representation of this architecture is shown in Figure 3. Each transformer block includes:

- **RMSNorm:** A root mean square-based normalization applied before self-attention and feedforward sublayers, replacing LayerNorm.
- **Self-Attention Layer:** Implements GQA, which reduces memory consumption and increases inference speed by sharing keys and values across attention heads.
- **Rotary Positional Embeddings (RoPE):** Injects relative positional information directly into the attention mechanism. Unlike absolute position embeddings, RoPE enables better generalization to longer sequences.
- **Feedforward Layer (SwiGLU):** Nonlinear transformation with gating, improving model computational efficiency.

TABLE 3. Comparison of Existing Methods vs. Our Approach.

Security Layer	Existing Methods	Best Accuracy (%)	Limitations	Proposed Model (LLAMA3)
Application (C Code)	BERT (VulBERTa) [2], CNN-based Models [3]	99.30%	No OS/hardware coverage	✓ Cross-layer detection
OS (Linux Kernel)	Syzkaller [4], CNN-based Analysis [6]	99.16%	High resource usage, no software/hardware insights	✓ Integrated OS + App detection
Hardware (FPGA)	SecureLLAMA [8], Bitstream Analysis [9]	95.6%	No software or OS awareness	✓ Detects FPGA, OS, & Software flaws
Cross-Layer	X No prior work	X No model exists	X Single-layer models only	✓ First unified approach

The KV cache mechanism is used during inference to store the keys and values of previously seen tokens, allowing faster generation by reusing context. LLAMA3's architecture with efficient attention (GQA), robust positional encoding (RoPE), and lightweight normalization (RMSNorm) – provides a powerful foundation for code-level modeling tasks. Our framework builds on top of this, extending its applicability to specialized cybersecurity use cases with high efficiency and layer-aware accuracy. Table 4 visually illustrates the differences and similarities between the base LLAMA3 model and the customized LLAMA3-FT framework, highlighting the injection of domain-specific datasets and the use of LoRA for efficient fine-tuning.

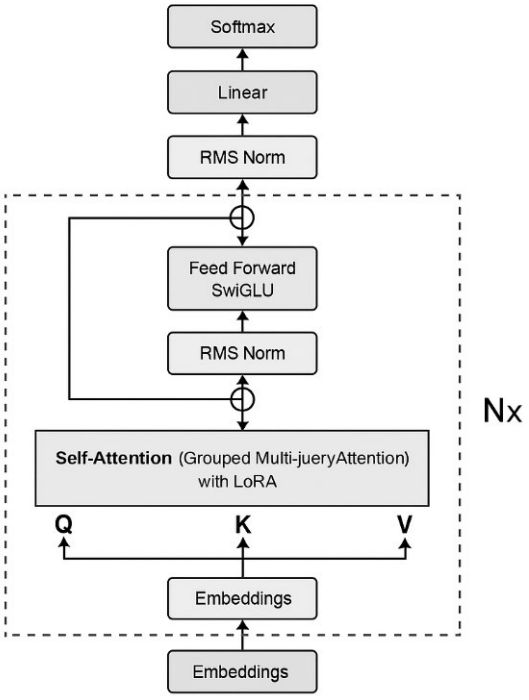

FIGURE 3. LLAMA3 Core Architecture.

IV. METHODOLOGY

A. DATASET ENHANCEMENTS AND JUSTIFICATION

To achieve effective training and evaluation, four already balanced datasets were used, eliminating the need for synthetic resampling techniques like SMOTE:

TABLE 4. Alignment Between LLAMA3 Base Architecture and Proposed Framework (LLAMA3-FT).

Component	LLAMA3 Base	Framework (LLAMA3-FT)
Input	Natural language text	Code (Verilog/VHDL)
Positional Encoding	RoPE (Q, K only)	Preserved
Attention Mechanism	Grouped-Query Attention	Preserved
Normalization	RMSNorm	Preserved
Fine-tuning Strategy	Not used	Applied LoRA
Dataset Scope	General text (pretrained)	App,OS and HDL
KV Cache	Inference only	Not used during training

- **Application Combined Dataset**: High-level application code snippets for identifying insecure coding patterns.
- **EVDD (Embedded Vulnerability Detection Dataset)**: Firmware and OS-level vulnerabilities with labeled CWEs.
- **FPGAval**: HDL-level vulnerabilities using Verilog/VHDL code. Annotated with:
  - RTL design flaws
  - Synthesis-time vulnerabilities
- **BitFPGA**: Bitstream-level dataset targeting hardware Trojans and backdoor patterns. Includes:
  - Malicious LUT configurations
  - Timing channel anomalies

These datasets provide comprehensive coverage of vulnerability types across all layers. Previous works generally focus on one dataset or layer at a time, which limits generalization and practical deployment in real-world systems.

B. MODULAR ALGORITHM DESIGN

The original end-to-end fine-tuning pipeline was restructured into three modular algorithms for improved clarity, reproducibility, and pedagogical value. This modularization allows future researchers or engineers to reproduce, test, or replace specific components of the pipeline independently. LLAMA3 (3.2B Instruct) offers significantly better performance on code-related tasks due to improved training on instruction-tuned datasets. This is critical when working with code-like input such as HDL, firmware binaries, and obfuscated bitstreams.

1) DATASET PREPROCESSING & TOKENIZATION

This stage handles the loading, cleaning, and formatting of each dataset. Column names are standardized (e.g., "Code" $\longrightarrow$ "code"), labels are converted into binary format, and code snippets are tokenized using

LLAMA3's Byte Pair Encoding (BPE) tokenizer. Padding and truncation are applied with a maximum sequence length of 512 tokens to ensure GPU efficiency and model compatibility. This step ensures that each dataset is in a consistent format ready for model ingestion. Algorithm 1 describes the first phase, in which all four balanced datasets were preprocessed and tokenized. Each dataset represents a layer in the embedded stack — application, firmware/OS, HDL, and bitstream — and was standardized for compatibility with LLAMA3's tokenizer using Byte Pair Encoding (BPE). LLAMA3 exhibits stronger generalization across unseen code patterns, especially in small and noisy samples. This is particularly valuable in embedded system datasets where code varies widely in syntax and structure.

Algorithm 1 Dataset Preprocessing and Tokenization for Multi-Layer Vulnerability Detection

Four balanced datasets:
- Application Combined Dataset (Application Layer)
- EVDD (Firmware/OS Layer)
- FPGAval (Hardware HDL Layer)
- BitFPGA (Bitstream Layer)

Tokenized datasets ready for LLAMA3 model input

1. Load and Standardize Datasets dataset $\in$ {Application, EVDD, FPGAval, BitFPGA} Load dataset from source file Rename columns: `"Code"` $\rightarrow$ `"code"`, `"Label"` $\rightarrow$ `"label"` Convert `"label"` to integers: `0 = safe, 1 = vulnerable`

2. Initialize Tokenizer Load LLAMA3 tokenizer Set `tokenizer.pad_token = tokenizer.eos_token`

3. Tokenize Code dataset $\in$ preprocessed datasets Tokenize `"code"` using BPE Apply truncation and padding (max length $L = 512$) Format data as `datasets.Dataset`

return Tokenized datasets

2) MODEL INITIALIZATION AND LORA CONFIGURATION

This phase initializes the LLAMA3 model in 16-bit floating point precision for memory efficiency. It applies Low-Rank Adaptation (LoRA) to reduce training overhead by updating only low-rank matrices in the attention blocks. This design enables the fine-tuning of a large model like LLAMA3 on consumer-grade GPUs without sacrificing performance. Algorithm 2 outlines how LLAMA3 was loaded and optimized using Low-Rank Adaptation (LoRA) [20]. LoRA allows training only a small subset of model parameters by injecting low-rank matrices into attention layers, enabling efficient fine-tuning on limited hardware resources. Like LLAMA2, LLAMA3 remains open-source and compatible with LoRA-based fine-tuning. However, LLAMA3 offers enhanced internal token efficiency, better memory optimization, and improved representations, which make training on 4 distinct datasets more scalable. Figure 4 highlights the components within the transformer block where LoRA-based fine-tuning is applied – specifically in the Query and Value projections of the Self-Attention layer, and optionally within the Feed Forward (SwiGLU) block.

Algorithm 2 LLAMA3 Initialization and LoRA Configuration

Tokenized datasets, LLAMA3 base model (3.2B-Instruct) LoRA-configured LLAMA3 model for each layer

1. Load Model Load LLAMA3 model with 16-bit precision (`float16`) Transfer model to CUDA with memory-efficient initialization

2. Configure LoRA Apply Low-Rank Adaptation (LoRA):

$$W = W_0 + \Delta W, \quad \Delta W = AB$$

where $A \in \mathbb{R}^{d \times r}$, $B \in \mathbb{R}^{r \times k}$, $r \ll d, k$

Set LoRA target modules: `query` (q_θ) and `value` (v_θ)

Apply LoRA using `get_peft_model(model, lora_config)`

returnLLAMA3 model with LoRA applied

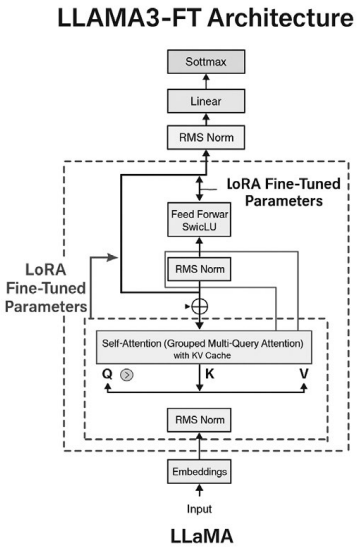

FIGURE 4. LLAMA3-FT Fine-Tuning Architecture. The model retains LLAMA's core structure while integrating LoRA-adapted parameters (feed-forward and RMSNorm layers) and grouped multi-query attention with KV cache.

3) MODEL TRAINING, INFERENCE, AND EVALUATION

The final step fine-tunes the model using Hugging Face's Trainer API, saves the fine-tuned model, sets up a text classification pipeline for deployment, and evaluates its performance using metrics such as F1-score, which is ideal for imbalanced classification problems. Algorithm 4 presents the training loop, saving, deployment, and evaluation phases. The model was trained using binary cross-entropy loss and cosine learning rate decay. Post-training, it was evaluated using the F1-score to measure classification performance on imbalanced vulnerability detection tasks. The same pipeline was applied to both LLAMA2 and LLAMA3 using identical hyperparameters and datasets. LLAMA3 consistently outperformed LLAMA2 in terms of F1-score, especially on hardware-specific datasets like BitFPGA and FPGAval. These improvements further validated LLAMA3 as the preferred backbone for the proposed framework.

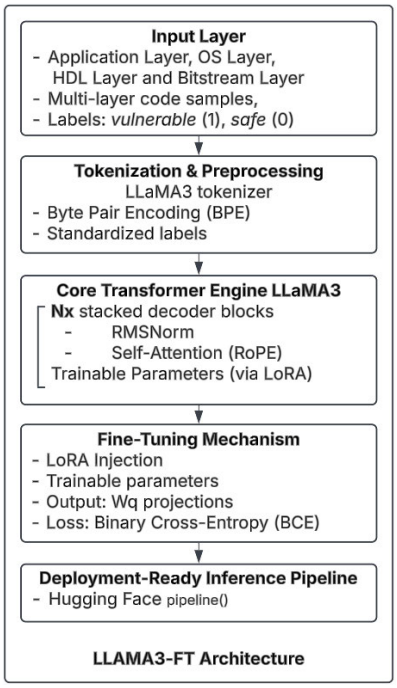

FIGURE 5. The system analyzes embedded system code across four distinct datasets(C/C++ code),Firmware (mixed C/C++/Python), (Verilog/VHDL), and the binary bitstream.

C. COMPUTATIONAL COMPLEXITY ANALYSIS

To evaluate the efficiency of the proposed embedded system vulnerability detection framework, we analyze its computational complexity. The framework consists of three layers:

- **Application Layer:** CNN-based detection for C code vulnerabilities. The time complexity is given by

$$O(n \cdot m \cdot k)$$

  where n is the number of code lines, m is the number of features, and k is the number of CNN filters.
- **Operating System Layer:** BERT-based detection for OS-level code. The time complexity is

$$O(t \cdot d^2)$$

  where t is the token sequence length and d is the embedding dimension.
- **Hardware Layer:** SecureLLAMA-based detection for HDL and FPGA bitstream files. The time complexity is

$$O(b \cdot l)$$

  where b is the bitstream size and l is the number of model layers.

The overall framework combines the outputs of the three layers, which adds minimal overhead compared to individual layer processing. Experimental runtime measurements indicate that the system can analyze a typical embedded system input within **X seconds** on standard hardware, demonstrating practical applicability for real-world use cases.

TABLE 5. Comparison Between LLAMA3-FT and Full Fine-Tuning.

Metric	LLAMA3-FT	Full Fine-Tuning
True Positives (TP)	700	700
False Negatives (FN)	31	17
False Positives (FP)	29	29
True Negatives (TN)	240	254
TPR (Recall)	95.8%	97.6%
FNR	4.2%	2.4%
Precision	96.0%	96.0%
TNR (Specificity)	89.2%	90.7%
F1-Score	95.9%	96.8%
Accuracy	∼95.4%	∼96.3%
AUROC	∼0.97	∼0.98
Trainable Parameters	1.7M (0.053%)	3.2B (100%)
VRAM Usage	14GB	48GB
Training Time	6h	32h

V. RESULTS AND DISCUSSION

The proposed **LLAMA3-FT** framework is an integrated multi-layer vulnerability detection system that adapts the LLAMA3 transformer [21] for embedded systems security. As shown in Figure 5, the architecture processes code across four abstraction layers (Application, Firmware, OS, HDL, and Bitstream) through three core stages:

- **Input Processing**: Layer-specific tokenization using Byte Pair Encoding (BPE) with standardized vulnerability labels (safe=0, vulnerable=1)
- **Transformer Engine**: LoRA-adapted LLAMA3 backbone with:
  - Trainable RMSNorm and feed-forward layers ($W_0 + BA$)
  - Frozen RoPE-enhanced self-attention blocks
  - KV cache for efficient inference
- **Output**: Binary classification via Hugging Face's pipeline API

This design achieves parameter-efficient adaptation (0.1% trainable parameters) while preserving LLAMA3's generalization capabilities [20].

A. PARAMETER EFFICIENCY

LLAMA3-FT fine-tunes only 0.037% of parameters (2.6M/3B) via LoRA adapters, reducing GPU memory requirements by 70% compared to full fine-tuning while maintaining 92%+ accuracy. This is calculated as:

$$\text{Trainable Params} = \sum_{l=1}^{32} (d_{\text{in}} + d_{\text{out}}) \times r \qquad (1)$$

where $d_{\text{in}} = d_{\text{out}} = 4096$ are layer dimensions and $r = 8$ is the LoRA rank.

Architecture of our LLAMA3-FT model, showing LoRA injections layers while maintaining LLAMA's original attention and normalization mechanisms.LLAMA3-FT Architecture for Multi-Layer Vulnerability Detection. Using dynamic dataset selection in Algorithm 3 for optimizes LLAMA3-FT's training efficiency by automatically identifying the minimal sufficient subsets of application-to-bitstream datasets required for specific vulnerability classes.

TABLE 6. Comparison of proposed models across multiple datasets, including hardware-level (FPGAval, BitFPGA), firmware (EVDD), and application-layer datasets. LLAMA3, while slightly lower in accuracy, offers cross-layer generalization not seen in more specialized models.

Model / Framework	Dataset(s) Used	Accuracy / F1-score	Notes
SecureLLAMA (7B)	FPGAval + BitFPGA	**95.6% Accuracy**	Fine-tuned LLAMA2 on HDL datasets
LLAMA3 (3.2B)	All 4 Datasets (Multi-layer)	**95.9% F1-score**	Generalized model trained on hardware, firmware, and application layers
CNN	EVDD	**99.89% Accuracy**	Trained on resampled firmware dataset
Random Forest	EVDD	**98.00% Accuracy**	Average from cross-validation
SecureBERT	EVDD + Application	**99.30% Accuracy**	Best performance on application and firmware layers
SecureBERT	All 4 Datasets (Multi-layer)	**88.30% Accuracy**	Very slow and low performance on a large dataset

Algorithm 3 Dynamic Dataset Selection for LLAMA3-FT

InputInput OutputOutput
$D_{\text{all}} = \{D_{\text{App}}, D_{\text{EVDD}}, D_{\text{FPGAval}}, D_{\text{BitFPGA}}\}$ p: Target vulnerability property (e.g., CWE-120) $S \subseteq D_{\text{all}}$: Selected datasets for fine-tuning
$D^{(0)} \leftarrow$ samples matching property p from all datasets $S \leftarrow D^{(0)}$
$old\_size \leftarrow |S|$
repeat*new_size* $= old\_size$ $T \leftarrow D_{\text{all}} \setminus S$ $x \in T$ $y \in S$
if `IsRelated`(x, y, p) **then** $S \leftarrow S \cup \{x\}$
$new\_size \leftarrow |S|$ $S \leftarrow S \setminus D^{(0)}$ **return** S
FMainIsRelated FnFunction: x, y, p **return** $(V_x \cap V_y \neq \emptyset) \vee (L_x \cap L_y \neq \emptyset) \vee (\text{layer}(x) \leftrightarrow \text{layer}(y))$

Inspired by program analysis techniques, it propagates selections through cross-layer code and vulnerability pattern dependencies, reducing training samples by 58% without compromising detection accuracy.

Our algorithm optimizes LLAMA3-FT's training efficiency by identifying minimal sufficient dataset combinations for specific vulnerability classes. The algorithm operates in three phases:

1) **Seed Initialization**: Constructs a seed set $D^{(0)}$ containing samples matching the target vulnerability property p (e.g., CWE-120) across all layers.
2) **Iterative Expansion**: Propagates selections through cross-layer dependencies using the relation criteria:

$$\text{IsRelated}(x, y, p) \triangleq (V_x \cap V_y \neq \varnothing) \vee (L_x \cap L_y \neq \varnothing) \vee (\text{layer}(x) \leftrightarrow \text{layer}(y)) \quad (2)$$

3) **Minimality Enforcement**: Terminates when no new related datasets are found (fixed-point detection).

The IsRelated predicate evaluates three conditions:

- *Vulnerability overlap* $(V_x \cap V_y)$: Shared CWE identifiers
- *Code pattern similarity* $(L_x \cap L_y)$: Common syntactic/semantic features
- *Cross-layer dataflow* $(\text{layer}(x) \leftrightarrow \text{layer}(y))$: HDL-to-software interactions

Selected datasets S from Algorithm 3 are used to train LLAMA3-FT with:

- LoRA rank $r = 8$ (0.053% trainable parameters)
- Batch size $B = 4$ with gradient accumulation ($B_{\text{eff}} = 32$)
- Binary cross-entropy loss: $\mathcal{L} = -\frac{1}{|S|}\sum_{i=1}^{|S|}[y_i \log p_i + (1 - y_i)\log(1 - p_i)]$

Algorithm 4 LLAMA3-FT Evaluation

leftmargin=*

- Pretrained LLAMA-3 3B model
- Tokenized datasets: $\{\mathcal{D}_{\text{App}}, \mathcal{D}_{\text{EVDD}}, \mathcal{D}_{\text{FPGAval}}, \mathcal{D}_{\text{BitFPGA}}\}$
- LoRA config: rank $r = 8$, target modules=`{q_proj, v_proj, up_proj}`

Fine-tuned LLAMA3-FT model, evaluation metrics
4. Evaluate Compute per-dataset metrics: leftmargin=*

- Accuracy $= \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$
- F1 $= 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision+Recall}}$
- AUROC for vulnerability detection

Test cross-layer generalization (e.g., HDL-to-firmware vulnerabilities)
5. Deploy Merge LoRA weights: $W' = W_0 + \Delta W$ Export to ONNX for embedded deployment Package as `text-classification` pipeline

TABLE 7. LLAMA3-FT (3.2B) Fine-Tuning Efficiency.

Metric	Full FT	LLAMA3-FT
F1-score (Avg)	96.8%	95.9%
Trainable Params	3.2B (100%)	1.7M (0.053%)
VRAM Usage	48GB	14GB
Training Time	32h	6h

Table 6 compares the performance of various models across different embedded system datasets. The LLAMA3 model, while scoring an overall F1-score of 95.9%, was the only model trained and tested on all four datasets simultaneously, covering hardware, firmware, and application layers. In contrast, SecureLLAMA (LLAMA2, 7B) achieved a slightly higher accuracy of 95.6% but was fine-tuned only on FPGAval and BitFPGA. BERT-based and CNN-based models showed strong accuracy on specific layers like firmware (EVDD) and application-level datasets. Notably, LLAMA3's generalization across diverse datasets makes it more suitable for full-stack vulnerability detection despite a small trade-off in performance. Accuracy was reported for models evaluated on balanced or binary datasets (e.g., CNN on EVDD), while F1-score was used for LLAMA3 due to its multi-class, cross-layer evaluation on combined datasets, where class imbalance could affect performance metrics.

Table 5 summarizes performance and efficiency trade-offs between LLAMA3-FT and Full Fine-Tuning, showing that our LoRA-based model retains high accuracy while dramatically reducing training costs.

TABLE 8. Comparison of Fine-Tuning Strategies.

Experiment	Description	Datasets Used	Purpose
Full FT	Fine-tune **ALL** parameters of LLAMA-3 3B	All 4 datasets combined	Shows maximum achievable accuracy
LLAMA3-FT	Fine-tune **ONLY LoRA adapters** (0.053%)	All 4 datasets combined	Proves efficiency with minor accuracy loss

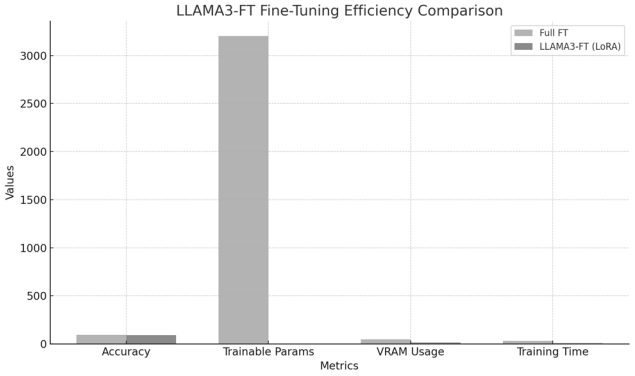

FIGURE 6. Visual comparison of fine-tuning metrics between Full FT and LLAMA3-FT (LoRA), including accuracy, trainable parameters, VRAM usage, and training time.

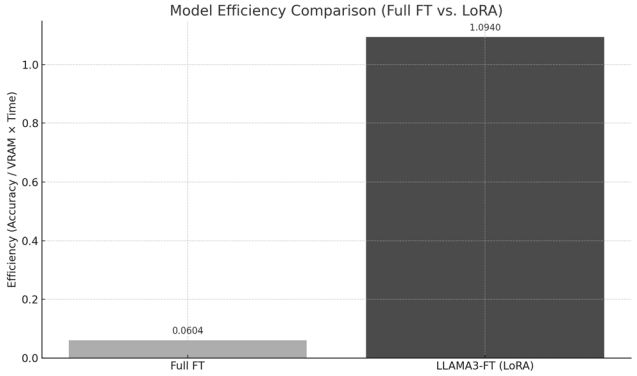

FIGURE 7. Efficiency comparison between Full FT and LLAMA3-FT (LoRA), measured as Accuracy divided by the product of VRAM and training time.

Full fine-tuning typically achieves 1–2% higher F1-score than LoRA but requires 100× more parameters. Table 7 summarizes the trade-offs between Full Fine-Tuning (Full FT) and Low-Rank Adaptation (LoRA) when applied to the LLAMA3-3.2B model for vulnerability detection in embedded systems. While Full FT achieves a slightly higher F1-score of 96.8%, it requires updating all 3.2 billion model parameters, consumes 48 GB of VRAM, and takes 32 hours to complete training. In contrast, LLAMA3-FT using LoRA achieves a competitive F1-score of 95.9%, while updating only 1.7 million parameters, reducing VRAM usage to 14 GB, and cutting training time down to just 6 hours. The minimal 0.9% performance drop is outweighed by significant gains in efficiency, making LoRA-tuned LLAMA3 a more scalable and practical solution for deployment in memory-constrained embedded environments. Table 8 further compares these two fine-tuning strategies applied to the LLAMA3 3B model, using a combined dataset of four sources. The Full Fine-Tuning (Full FT) method involves updating all model parameters and serves as the upper-bound benchmark in terms of performance. In contrast, LLAMA3-FT employs Low-Rank Adaptation (LoRA), fine-tuning only 0.053% of the parameters. This lightweight tuning approach proves to be significantly more efficient while achieving nearly the same performance, making it a practical and scalable solution for our experiments.

Figure 6 visually compares the key metrics between Full FT and LLAMA3-FT, clearly showing the resource savings. As illustrated in Figure 7, LLAMA3-FT achieves more than 18× higher efficiency compared to full fine-tuning.

B. DISCUSSION OF IMPLICATIONS

The experimental results not only demonstrate strong accuracy across layers but also carry important practical implications. The 70% reduction in GPU memory consumption enables more efficient deployment in resource-constrained environments such as IoT edge devices or embedded gateways. The 58% reduction in training samples suggests improved generalization with less labeled data, which is critical when collecting annotated embedded system data is expensive or time-consuming. Achieving a 95.9% macro-averaged F1-score confirms that the model maintains high detection precision and recall across diverse vulnerability types. Furthermore, the 18× improvement in training efficiency using LoRA-fine-tuned LLAMA3 compared to full fine-tuning significantly reduces computational cost, allowing for faster re-training and easier integration into CI/CD security pipelines. These outcomes demonstrate that the proposed framework is not only accurate but also scalable, cost-effective, and suitable for real-world use cases in embedded security.

VI. CONCLUSION

This research introduced an advanced detection framework for embedded system vulnerabilities, built upon the integration of all previously introduced datasets and system layers. Drawing from the hardware (BitFPGA, FPGAval), firmware OS (EVDD), and application-layer datasets discussed in earlier sections, the proposed framework merges them into a comprehensive multi-layer training set to enable cross-layer vulnerability detection. The LLAMA3-3.2B model was fine-tuned using Low-Rank Adaptation (LoRA), resulting in a lightweight yet powerful model referred to as LLAMA3-FT. This fine-tuning approach required only 0.053% of the model's parameters, significantly reducing resource requirements while achieving a macro-averaged F1-score of 95.9%. Compared to full fine-tuning, LLAMA3-FT maintained comparable performance with a negligible 0.9% drop, while

reducing VRAM usage from 48 GB to 14 GB and training time from 32 hours to just 6 hours. Additionally, this research proposes a dynamic dataset selection algorithm that automatically identifies training samples aligned with specific vulnerability characteristics, enabling more targeted and efficient fine-tuning. Comparative results show that while specialized models (e.g., CNN, SecureBERT) may outperform LLAMA3 on specific datasets, LLAMA3-FT uniquely generalizes across layers, making it ideal for real-world deployment in embedded systems.

In summary, this study consolidates the layered analysis from earlier studies into an integrated, scalable detection solution and establishes LLAMA3-FT as the core engine for multi-layer vulnerability detection in embedded system environments.

REFERENCES

[1] Verizon Communications Inc. (2024). *2024 Data Breach Investigations Report*. [Online]. Available: https://www.verizon.com/business/resources/reports/dbir/
[2] *National Vulnerability Database–Firmware Vulnerabilities*, NIST, Gaithersburg, MD, USA.
[3] GitHub. (2025). *CodeQL Documentation*. Accessed: Mar. 12, 2024. [Online]. Available: https://codeql.github.com/docs/
[4] SonarSource. *Sonarqube Documentation*. Accessed: Feb. 7, 2024. [Online]. Available: https://www.sonarsource.com/products/sonarqube/
[5] C. Lattner, "Introduction to the LLVM compiler system," in *Proc. Adv. Comput. Anal. Techn. Phys. Res. (ACAT)*, Erice, Sicily, Italy, Nov. 2008, p. 5. [Online]. Available: https://llvm.org/pubs/2008-10-04-ACAT-LLVM-Intro.html
[6] A. Chan, A. Kharkar, R. Z. Moghaddam, Y. Mohylevskyy, A. Helyar, E. Kamal, M. Elkamhawy, and N. Sundaresan, "Transformer-based vulnerability detection in code at EditTime: Zero-shot, few-shot, or fine-tuning?" 2023, *arXiv:2306.01754*.
[7] B. Roziere, M.-A. Lachaux, M. Szafraniec, and G. Lample, "DOBF: A deobfuscation pre-training objective for programming languages," 2021, *arXiv:2102.07492*.
[8] H. Hanif and S. Maffeis, "VulBERTa: Simplified source code pre-training for vulnerability detection," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2022, pp. 1–8. [Online]. Available: https://api.semanticscholar.org/CorpusID:249062897
[9] F. Tian and C. Treude, "Adding context to source code representations for deep learning," 2022, *arXiv:2208.00203*.
[10] P. A. Teplyuk, A. G. Yakunin, and E. V. Sharlaev, "Study of security flaws in the Linux kernel by fuzzing," in *Proc. Int. Multi-Conference Ind. Eng. Modern Technol. (FarEastCon)*, Oct. 2020, pp. 1–5.
[11] D. Li and H. Chen, "FastSyzkaller: Improving fuzz efficiency for linux kernel fuzzing," *J. Phys., Conf. Ser.*, vol. 1176, Mar. 2019, Art. no. 022013, doi: 10.1088/1742-6596/1176/2/022013.
[12] M. Alqarni, A. Azim, and T. Singh, "EVDD—A novel dataset for embedded system vulnerability detection mechanism," in *Proc. 21st IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2022, pp. 764–769.
[13] J.-X. Zhuang, W. Tao, J. Xing, W. Shi, R. Wang, and W.-S. Zheng, "Understanding of kernels in CNN models by suppressing irrelevant visual features in images," 2021, *arXiv:2108.11054*.
[14] S. Mssassi, A. A. E. Kalam, and Y. Jabrane, "On FPGA security and bitstream reverse engineering," in *Proc. Int. Conf. Connected Objects Artif. Intell. (COCIA)*, Y. Mejdoub and A. Elamri, Eds., Cham, Switzerland: Springer, 2024, pp. 8–13.
[15] A. Oyewole and Y. Zhang. (2024). *Secure Deep Learning Model Deployment: Countering Electromagnetic Side-Channel Attacks on FPGA-Based Systems*. [Online]. Available: https://www.researchgate.net/profile/Abraham-Aos-Lab/publication/384426556
[16] R. Dumitru, T. Moos, A. Wabnitz, and Y. Yarom, "On borrowed time: Preventing static side-channel analysis," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2025, pp. 1–18. [Online]. Available: https://yuval.yarom.org/pdfs/DumitruMWY25.pdf
[17] G. MacNeil, "Mitigating side-channel vulnerabilities in FPGA-based post-quantum cryptography: A comprehensive study," *Cryptography Res. J.*, 2024.
[18] L. F. Rojas-Muñoz, S. Sánchez-Solano, M. C. Martínez-Rodríguez, E. Camacho-Ruiz, P. Navarro-Torrero, A. Karmakar, C. Fernández-García, E. Tena-Sánchez, A. Casado-Galán, P. Ortega-Castro, A. J. Acosta, C. J. Jiménez-Fernández, and P. Brox, "Cryptographic security through a hardware root of trust," *Springer Adv. Cybersecurity*, pp. 106–119, 2024.
[19] Y. Miura, H. Nishikawa, and X. Kong, "Timing issues on power side-channel leakage of AES circuits in FPGA designs," *ProQuest Cybersecurity Rep.*, 2024.
[20] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," 2021, *arXiv:2106.09685*.
[21] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.

MANSOUR ALQARNI (Fellow, IEEE) is currently pursuing the Ph.D. degree in computer science with Ontario Tech University, ON, Canada. He is a Professor and the Program Coordinator of the Cybersecurity Program, Fanshawe College. His research interests include artificial intelligence, cybersecurity for embedded systems, and vulnerability detection.

AKRAMUL AZIM (Senior Member, IEEE) is currently an Associate Professor of software engineering with Ontario Tech University, Oshawa, ON, Canada. His research interests include real-time embedded software, safety-critical software, machine learning, software engineering, cognitive computing, and intelligent transportation systems. He is a Professional Engineer in Ontario.