ISSN: 2994-6433

Research Article

Journal of Sensor Networks and Data Communications

TinyML-CoDE: A Co-Design and Evaluation Framework for Systematic TinyML Development

Dhananjaya Lahiru Bandara*, Mohamed Mirshad Munawwer and Charith Lakpriya Jayathilake

School of Digital, Technology, Innovation and Business Department of Computing and Esports, Staffordshire University, UK

***Corresponding Author**
Dhananjaya Lahiru Bandara, School of Digital, Technology, Innovation and Business, Department of Computing and Esports, Staffordshire University, UK.

Submitted: 2026, May 25; **Accepted:** 2026, Jun 17; **Published:** 2026, Jul 06

Citation: Bandara, D. L., Munawwer, M. M., Jayathilake, C. L. (2026). TinyML-CoDE: A Co-Design and Evaluation Framework for Systematic TinyML Development. *J Sen Net Data Comm, 6*(2),01-10.

Abstract

The current paper presents the key issues when developing Tiny Machine Learning in resource-constrained Internet of Things equipment. Based on the methodical research on 35 recent papers, we uncover five core research gaps that include standardized benchmarking, systematic co-design, lifecycle management, security integration, and toolchain interoperability. To implement these gaps, we suggest the TinyML Co-Design and Evaluation (TinyML-CoDE) framework to be based on the integrated approach to the methodology that considers constraint awareness, hardware-model recommendation, automated optimization, unified benchmarking, and lifecycle management. The benefits that are projected are 30 percent less development time and 25 percent performance efficiency. The framework offers a methodical way of eliminating the existing fragmentation and facilitating scalable deployment of TinyML in the industrial and healthcare IoT applications.

Keywords: TinyML, Edge Computing, Microcontrollers, Codesign, Benchmarking, IoT, Embedded AI.

1. Introduction

The Internet of Things devices are growing, as they require intelligent computing at the network edge, which prompts the use of Tiny Machine Learning in microcontroller-based systems [1]. Although it can be run on hardware with ¡1MB memory, milliwatt power, and less than dollar cost, the development of TinyML is greatly impeded by a lack of tool fragmentation, non-reliable evaluation, and poor hardware software compatibility [2,3]. To examine these challenges, the given paper is based on critical literature review, highlights the main research gaps, and suggests an integrated co-design framework to overcome these gaps.

A. Motivation

The state of development of TinyML demands closed tools and hand-tuning to work. The developers have issues with choosing the hardware, resource optimization of the model within the resource constraints, solution evaluation, and management of the deployed systems. Lack of systematic methodologies results in prolonged development processes, non-optimal solutions and low scalability. This paper seeks to overcome these challenges using systematic co-design methodology, which causes complexity reduction, efficiency improvement, and reproducible TinyML development.

B. Contributions and Structure of the Paper

This study has three major contributions:

i. In-depth Critical Analysis: Systematic review and synthesis of 35 publications with findings of recurring limitations in the current TinyML practice.
ii. Gap Identification and Analysis: 5 research gaps will be identified and prioritized and interrelationships will be drawn.
iii. Integrated Framework Proposal: TinyML-CoDE framework structure development and methodological validation strategy with roadmap of implementation.

The paper is organized in the following way: Section II is the critical literature review of existing TinyML approaches. Section III determines the gaps in research and poses questions and objectives. The research methodology is presented in section IV. The TinyML-CoDE framework is presented in section V. Expected benefits are analyzed in Section VI. Section VII is on implementation. Future work is the last section in part VIII.

2. Critical Literature Review

A. Hardware Platform Diversity and Architectural Constraints

The development of TinyML hardware devices is done between 8-bit microcontrollers and dedicated AI accelerators [4]. With 32-bit ARM Cortex-M cores, small neural network inference was power-efficient, and more recent neural processing accelerators contain neural processing units [2,5]. This diversity presents enormous compatibility and optimization issues regarding software. Various platforms have varying trade-offs on their computational performance, memory capacity, power consumption, and connectivity (Table 1). The ESP32 line provides a more powerful Wi-Fi/Bluetooth that can be used in the device of a gateway, but not a battery-powered sensor [6]. On the other hand, Apollo3 Blue is optimized to run at a computational expense of ultra-low-power with focus on wearable healthcare [6].

Platform Type	Key Strength	Primary Limitation
ESP32-class	High compute + connectivity	High power consumption
STM32H7-class	High performance MCU	Moderate energy efficiency
nRF52-class	Low power + wireless	Limited compute capability
Apollo3-class	Ultra-low power	Lower processing speed
RP2040-class	Low cost, flexible I/O	Weak power management

Table 1: Summary of MCU Platform Trade-offs

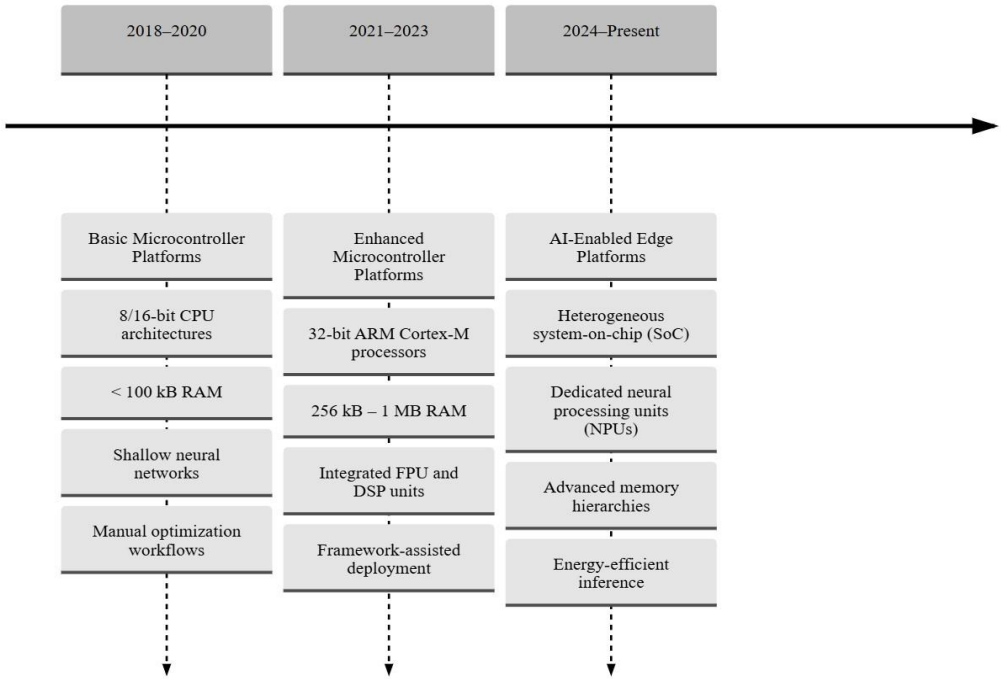

Figure 1: Three-phase evolution of TinyML hardware from basic microcontrollers to heterogeneous AI accelerators [4,5].

Architectural changes are not limited to the specifications but touch on the memory hierarchy, cache arrangements, and memory-mapped I/O, which influence the data transfer and computational performance [6]. Software emulation or Hardware floating-point units on cost optimized microcontrollers are not present and so require software emulation or quantization, bringing on accuracy-performance trade-offs [7]. These architectural considerations are not typically considered in the current literature which tends to treat hardware as a black box and not co-design parameter.

B. Software Ecosystem Strativity

TinyML software ecosystem has the problem of incompatible toolchains. TensorFlow Lite Micro is the most widely used and has the highest adoption (60%), but with limitations: it does not support all operators, does not manage heterogeneous architectures optimally, and model conversion is difficult [8]. Other frameworks such as Edge Impulse, CMSIS-NN and XCUBE-AI have nicheness but worsen interoperability issues due to vendor lock-in and architecture based implementations [9,10]. This fragmentation

makes overhead development. The Framework conversion incurs 2-5% accuracy and 20-40% performance loss as a result of the suboptimal implementations of operators [8]. Optimization methods are also disjointed: not only do different frameworks have different quantization methods, but also pruning differs in formats and can be incompatible with other tools, and hardware specific optimizations are also not portable [11].

Technique	Limitations
Quantization	Hardware-dependent efficiency, potential accuracy loss, toolchain constraints for mixed-precision
Pruning	Irregular sparsity patterns inefficient on many microcontrollers, potential performance degradation
Knowledge distillation	Requires large teacher network, training complexity
Neural architecture search	Computationally prohibitive (thousands of GPU hours), poor generalization to real-world conditions

Table 2: TinyML Optimization Techniques and Limitations

C. Optimization Methods and Limiting Practicalities

The current TinyML optimization is centered on the model compression by using quantization, pruning, and simplifying the architecture. These techniques and limitations are summarized in table 2. Quantization methods tend to disregard hardware specialities. The integer operations are performed with different efficiency by different microcontrollers, and the best quantization relies on the model architecture and hardware capabilities [7]. Mixed-precision quantization is an opportunity, but is still theoretical because of limitations in toolchains [11]. Hacking pruning results in sparsity patterns which are irregular and utilizable only efficiently by a few microcontroller architectures. Available pruned models can be slower than dense ones without specialized sparse computation units because of failure to predict branches and inefficient usage of caches [12].

D. Application Requirement and Generalization Problem

Small ML applications cover areas that have incompatible needs. Industrial monitoring is concerned with reliability and real time functionality under extreme environments [13]. In healthcare, privacy and ultra-low-power consumption are placed in focus and there are regulatory restrictions [6]. Sensors used in the environment need long-term autonomous operation between sensing frequency and energy acquisition [1].

Present studies tend to ignore domain-related limitations. Benchmark data sets such as CIFAR-10 do not represent the real-life issues such as sensor noise or data drift or poor connectivity [14]. Clean lab data-optimized models do not perform well in real settings and errors of 10-30% have become widely observed [13]. The generalization issues are enhanced by the one-size-fits all approach. The high-quality microphone optimized models do not work with the low-cost MEMS microphones that have dissimilar frequency response [9]. Image models that have been trained during the day are useless in different lighting conditions that are very important in surveillance [15].

E. Methodological Concerns in the Current Research

Evaluation of 35 articles shows worrying tendencies of methods. Only twenty two percent of them give enough

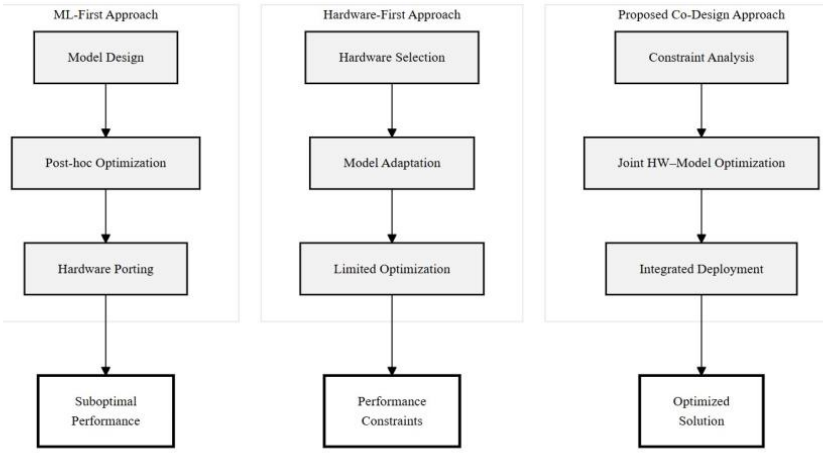

Figure. 2: Fragmented development workflows in current TinyML practice: machine learning focused, hardware focused, and theoretical co-design approaches [16].

details of implementation to reproduce and 15 percent have detailed energy measurements [14]. The measurement criteria concentrate more on accuracy (95% of articles) but ignore latency variability (12%), memory patterns (18%), and energy efficiency (15%) [7]. Lack of uniform checkpoints would allow cherry picking of good outcomes. Researchers can choose hardware and optimization parameters that achieve the best increase of reports without considering deployment limits [3]. A majority of studies use synthetic data/curated data that does not reflect the real-world deployment scenario [7].

F. Literature Findings Synthesis

The literature review has revealed a fragmented TinyML ecosystem that is heterogeneous hardware-wise and software and method-inconsistent. This gives rise to three basic challenges and they are: (1) the development tools are not interoperable, (2) the evaluation methods are yet to be standardized and (3) the hardware software integration is still less than ideal. All these promote lack of reproducibility, scalability and real world implementation and need a structured intervention.

3. Research Gap Analysis

A. Identified Research Gaps

We find that there are five basic research gaps that impede the progress of TinyML in our systematic literature review:

i. **Gap 1:** Standardized Benchmarking: Existing studies do not have standardized evaluation strategies. In most of the studies, the accuracy results are found to be reported only, disregarding energy, memory, and latency [7]. We review that 15 percent of the papers provide complete energy measurements and 22 percent provide complete memory use [14]. Such discrepancy does not allow to make fair comparisons, makes reproducibility impossible, and makes hardware/model selection difficult.

ii. **Gap 2:** Systematic Co-Design: Co-design of hardware and software is acknowledged to be imperative; however, there are no systematic approaches to the matter [16]. The majority of studies either optimize the models to meet given hardware or choose hardware to meet pre-trained models, and not many studies take constraints into consideration [17]. The resulting solutions produced by this sequential optimization are suboptimal solutions which do not utilize hardware capabilities or otherwise unnecessarily limit models.

iii. **Gap 3:** Lifecycle Management: Existing studies concentrate on initial deployment and do not pay much attention to long-term management [1]. Scaling deployments need versioning, performance testing, and model renovation, as well as system renunciation [5]. The current solutions do not have standardized solutions to over-the-air updates, monitoring its performance, and adjustment to new conditions.

iv. **Gap 4:** Security Integration: Security is not well developed in TinyML applications [5,8]. Conventional security systems are excessively resource-imposing to microcontrollers and there is a mismatch between what is needed and what is feasible [18]. Existing studies provide few solutions to resource-constrained setting.

v. **Gap 5:** Toolchain Interoperability: There is a lack of interoperability in the toolchain fragmentation in the TinyML ecosystem [3]. Development, optimization, conversion, deployment and debugging require numerous environments to developers [9]. Such fragmentation not only prolongs the development time, it introduces compatibility problems as well as barriers to best practices adoption.

B. Gap Interrelationships and Prioritization

Gap interrelationships are represented in figure 3. The benchmarking is standardized (Gap 1), where objective metrics can be used to compare the level of other solutions [7]. Co-design (Gap 2) and toolchain (Gap 5) gaps also have connections to each other, and they have to be combined. The lifecycle management (Gap 3) and security (Gap 4) are important with the increase of the scale of deployments [1]. Our priority goes to the Gap 1, then Gap 2, 5, then 3, 4.

C. Research Objectives

Judging by the gaps detected, we state the following research purposes:

RO1 Design a unified benchmarking model of end-to-end TinyML at all four dimensions of accuracy, latency, memory, and energy.

RO2 Develop a co-design systematizing approach that takes into account hardware and software limitations to the best possible implementation of TinyML.

RO3 Design and deploy a unified toolchain that has a better pattern of interoperability among current TinyML frameworks and tools.

RO4 Implement lightweight security solutions that are resource-aware and have no impact on performance when deployed on TinyML.

RO5 Build tools to manage lifecycle of deployed TinyML systems, maintain, monitor and update.

RO6 Test the suggested framework with the help of comparative study analysis and case studies in the industry and healthcare spheres.

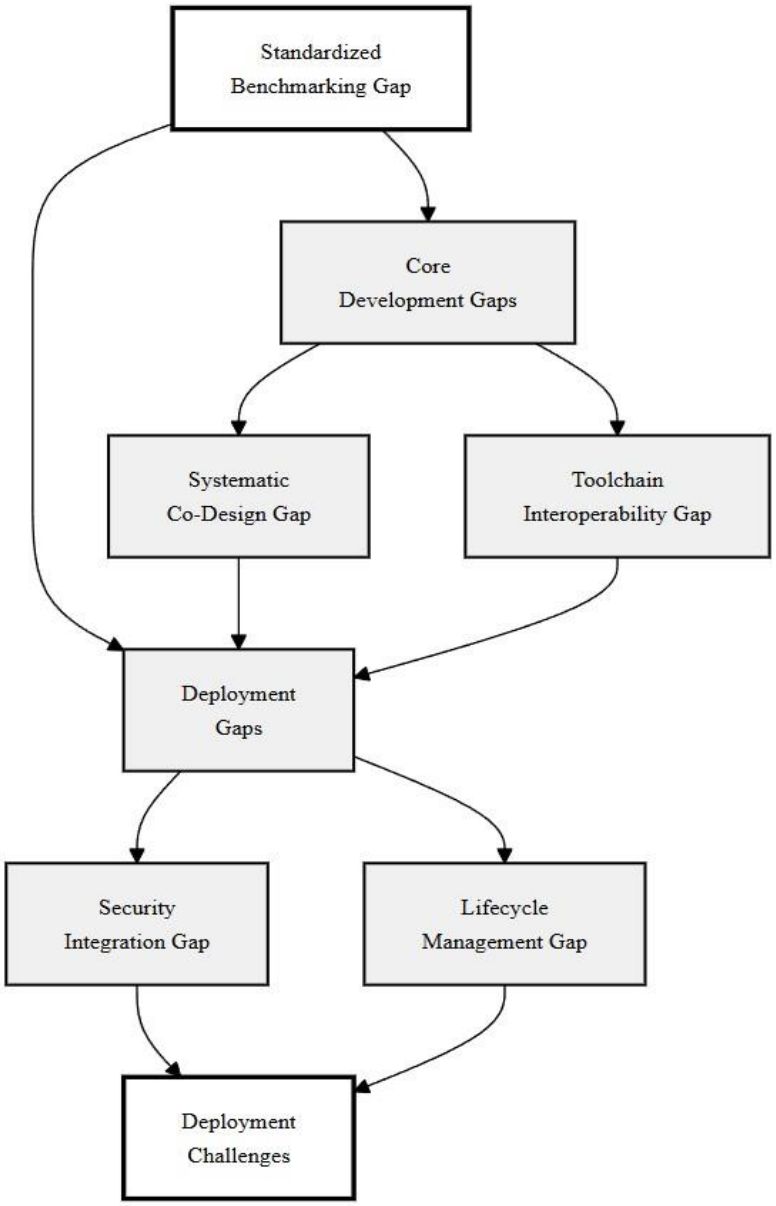

Figure 3: Relationships between the five identified research gaps [1,3,5,7,16].

D. Research Questions

In order to direct the research and fill in the identified gaps, the following research questions will be formulated:

RQ1 What can be done to design standardized benchmarking techniques to allow the comparison of TinyML solutions that can be used in various hardware designs?

RQ2 What are the systematic co-design principles that can be used to optimize joint selection and optimization of hardware and software components in TinyML systems?

RQ3 What can be done to enhance toolchain interoperability to ensure less complexity in development and no loss of performance between frameworks?

RQ4 What are the lightweight security measures that can be deployed with resource-constrained TinyML deployments without affecting the performance of the system?

RQ5 How can the lifecycle management practices best be used to maintain TinyML systems in mass deployments with changing demands?

RQ6 What are the differences in systematic co-design structures in terms of efficiency in development, quality of solution, and long-term maintainability with standard methods?

E. Synthesis of Gap Analysis

The gap analysis indicates that various challenges are interconnected and that, when combined, slow down the progress of TinyML. No standard benchmarking concept, and fragmented toolchains along with the ad hoc co-design introduce inefficiencies to the development process. The concept of security and the lifecycle

management becomes a central issue of sustainable deployment. These loopholes will require a comprehensive solution instead of patching, which drives the framework development strategy.

4. Research Methodology

A. Research Design and Approach

This study is based on the design science approach that consists of three cycles. Phase 1 makes use of a systematic literature review in which 35 publications (2020-2025) are evaluated based on specific inclusion criteria that revolve around the implementation issues of TinyML. Phase 2 is the design of the framework in stages with the development of components in accordance with gap analysis. The third phase is concerned with validation planning by use of case studies and comparison.

B. Methodological Framework

The research methodology is a combination of both qualitative and quantitative methods:

i. Literature Review: Systematic content analysis of themes and deficiencies.
ii. Framework Design: TinyMLCoDE It should be developed iteratively based on the design science principles.
iii. Validation Planning: Multi-methodology: Comparative benchmarking and real-life case-studies.

C. Design Principles

The development of the framework is based on five main principles of design which are provided by the literature analysis:

i. Abstraction with Specificity: Abstraction on high level of workflow and allow optimization based on the platform.
ii. Constraint-Sensitive Optimization: Optimize in response to application-specific constraints.
iii. Interoperability by Design: Use the components with the tools already in place, as opposed to their replacement.
iv. Security and Privacy Integration: Integrate security issues across the development lifecycle.
v. Empirical validation: Ground everything on a basis of measurable, reproducible evaluation.

D. Methodological Limitations

The study has limited itself to published literature that may not be complete because it does not cover unwritten practical challenges and validation to real implementation beyond theoretical development. These constraints guide the roadmap of implementation and validation procedure.

Application	Hardware	Model	Accuracy
Keyword spotting	ESP32-S3	DS-CNN 20k (int8)	94.2%
ECG classification	Apollo3 Blue	TinyLSTM (int16)	96.5%
Predictive maintenance	nRF52840	Random Forest	92.3%
Environmental sensing	Raspberry Pi Pico	Decision Tree	89.8%
Visual anomaly detection	STM32H743	MobileNetV2-Tiny	88.7%

Table 3: Example Hardware-Model Recommendations

5. Proposed Framework: TinyML-Code

A. Framework Overview

In order to fill the identified gaps, we suggest the TinyML CoDesign and Evaluation (TinyML-CoDE) framework that consists of five integrated components (Figure 4). The framework is the operationalization of design principles under systematic methodology including specification and deployment.

B. Constraint-Aware Specification

This element encapsulates application specifications in form of predefined and structured constraint profiles comprising of accuracy levels, latency guarantees, power constraints, memory constraints, expense goals, and communication requirements. It renders non-formal requirements in machine readable forms to inform development due to ambiguity in the classical methods [17]. The language of specifications allows quantitative and qualitative specifications.

C. Hardware-Model Recommendation

This component suggests a Pareto-optimal selection based on constraint profiles of the optimal combinations of hardware model using constraint satisfaction. It has a repository of hardware specifications and model attributes, which then filter the platforms that fit the minimum requirements and then isolate suitable model architectures [11]. This eliminates skills needed to co-design successfully and avoids inefficient assignments.

D. Automated Optimization

This component uses constraint-conscious optimization sequences such as quantization-conscious training, pruning, hardware-optimizing kernel optimization, memory layout optimization and operator fusion. It is sequence-specific to constraints in profiles

such as operator fusion is given preference to latency-sensitive applications and aggressive quantization is given preference to memory-constrained applications [12]. Minimized models are checked using a feedback mechanism and approaches altered that violate requirements.

E. Unified Benchmarking Suite

The TinyML-Bench component fills the gap in benchmarking (Gap 1) with standard evaluation at the four dimensions accuracy, latency, memory and energy [14]. It has standardized workloads, measurement instruments, platform-independent platform execution through hardware abstraction and standardized reporting formats [7]. To be sure that the configuration is documented, a reproducibility framework guarantees thorough documentation of the configuration such as hardware architecture, version of software used, compiler options, and measurement procedure.

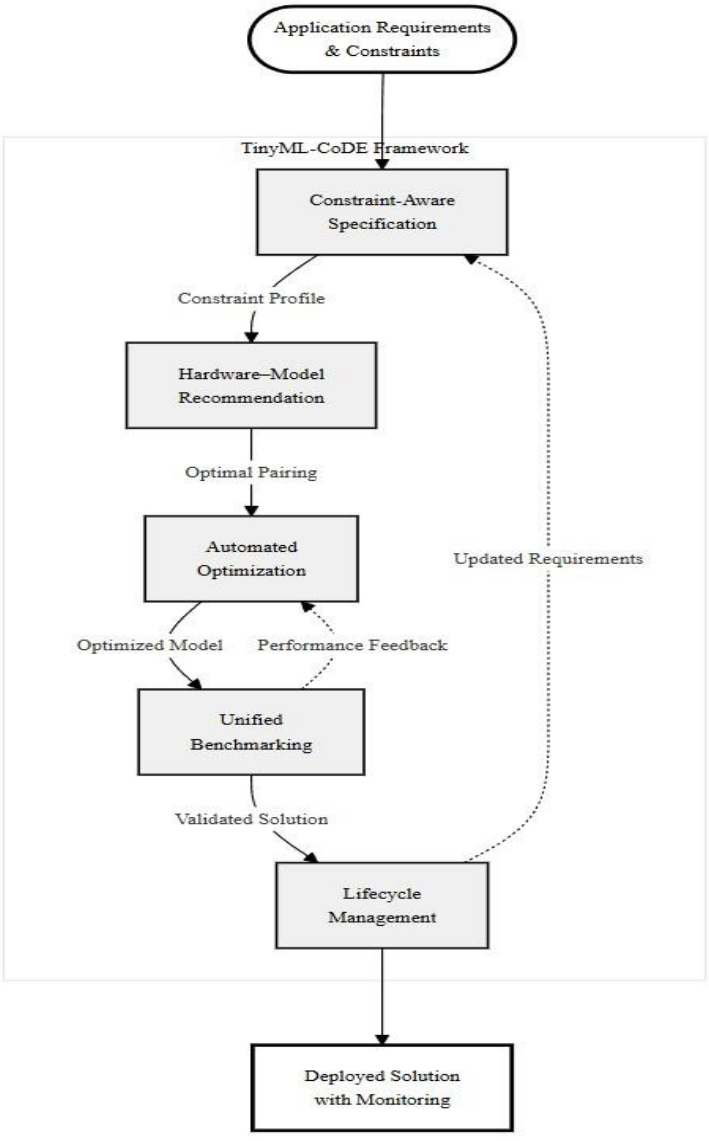

Figure 4: TinyML-CoDE framework architecture with five integrated components

F. Lifecycle Management

This element solutions long-term deployment challenges (Gap 3) version control, performance monitoring, over-theair updates, drift detection and system retirement tools [1]. It is designed to handle resource-constrained environments, sampling is employed to constrain data collection and updates are performed with differential compression and its built-in security is cryptographic verification [8]. Integration into the benchmarking suite allows to compare the performance with the baseline measurements.

G. Integrated Workflow

These components operate in a unified workflow to: (1) capture requirements to constraint profiles, (2) recommend hardware-model pairs, (3) perform optimization with constraints, (4) benchmark evaluation, (5) utilize lifecycle management for deployment, and (6) track and refresh performance. This addresses

the current fragmentation within the development lifecycle.

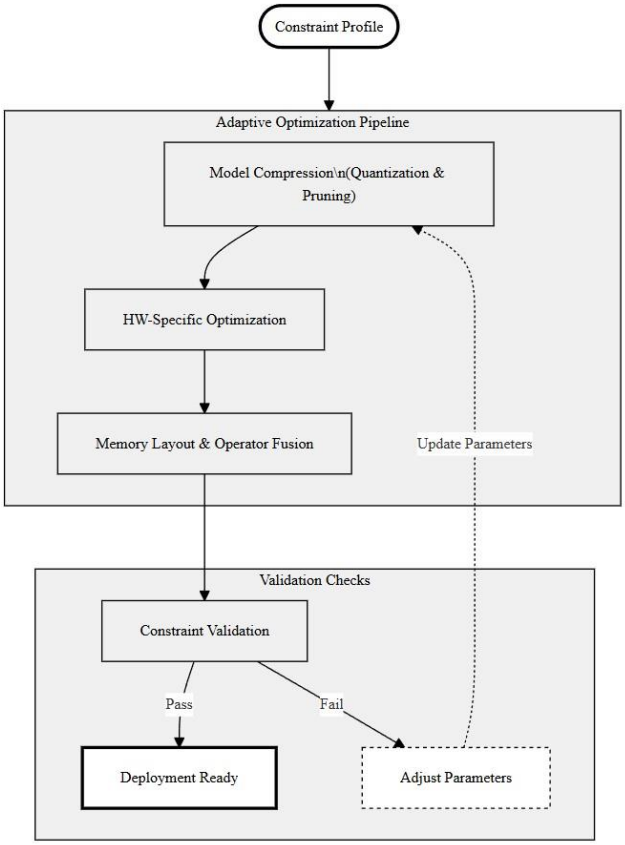

Figure 5: Adaptive optimization pipeline that adjusts based on constraint profiles [7,11].

Figure. 6: TinyML-Bench architecture with standardized workloads and measurement components [7,14].

Metric	Projected Impact
Development time	30% reduction, enabling faster system delivery
Model accuracy	Approximately 5% improvement in task performance
Energy efficiency	Around 25% improvement, reducing energy per inference
Memory utilization	15% reduction in memory usage
Deployment success rate	20% increase, improving deployment reliability

Table 6: Projected Performance Improvements

6. Expected Benefits and Analysis

A. Development Efficiency

TinyML-CoDE is expected to save 30 percent of development time on automated recommendations, optimization systematization and integrated tools. This has been enhanced by the fact that manual research into hardware-model combinations has been done away with, trial-and-error optimization cycles have been minimized, context switching between fragmented platforms has been minimized and the performance of the fragmented platforms can be evaluated faster through common evaluation.

B. Performance Improvements

Better co-design and constraint-aware optimization will yield us 25% better performance efficiency (accuracy per watt). This is due to the fact that optimal hardware-software matching minimizes resource wastage, specific applications which are optimized to meet particular requirements, and early detection of bottlenecks by thorough benchmarking and constant improvement wherein deployment feedback is used.

C. Research and Industry Impact

Multiple stakeholders have benefited via the framework, such as researchers have access to standardized analysis techniques that allow the fair comparison of results; industry practitioners have access to standardized development costs and reliability in solution development; society has the benefit of democratized edge AI development and better energy efficiency in IoT implementations.

D. Risk Analysis and Mitigation

The risks that could be encountered are resistance to adoption (through open-source development), complexity of implementation (through modular design), hardware heterogeneity (through abstraction layers), and security (through security-by design principles).

7. Implementation Considerations and Roadmap

A. Implementation Phases

The framework implementation is based on a 24 months roadmap consisting of four consecutive steps:

i. Phase 1 (Months 1-6): The development of the prototype of benchmark standardization and recommendation engine.
ii. Phase 2 (Months 7-12): Phase 2 entails the introduction of a pipeline that optimizes the use of existing TinyML tools.
iii. Phase 3 (Month 13-18): Developing a lifecycle management component and integrating it with security.
iv. Phase 4 (Months 19-24): Complete framework integration and case study validation.

B. Validation Approach

The techniques used in validation Complementary methods include: comparative analysis benchmarking against existing methods using standardized measures and case studies in industrial monitoring and healthcare applications. The measures of success are the decrease in the development time, improvement in performance, and the increase in the reliability of deployment.

C. Implementation Problems

The main challenges are the heterogeneity of hardware, barriers to adoption of ecosystems and consistency of measurement. These are addressed in terms of open-source creation, the cooperation with hardware manufacturers, and connection with the already existing TinyML ecosystems.

8. Conclusion and Future Work

A. Research Summary

The paper reveal the various issues that are inherent in the development of TinyML such as a lack of toolchain fragmentation, evaluation inconsistency, and hardware-software mismatches and proposes the TinyMLCoDE framework as an answer to these problems. The framework provides a methodological design of TinyML development, which includes specification constraint-conscious, co-design, automated design, standard benchmarking, and man-conscious.

B. Revised Research Questions

Going back to the research questions that were presented in

Section III:
RQ1 Standardized benchmarking Hardware agnostic evaluation suites with detailed measurements- done with TinyML Bench component.
RQ2 Systematic co-design principles comprise constraint aware optimization and automated hardware-model recommendation-implemented in terms of integrated workflow.
RQ3 Toolchain RQ3 Toolchain is more interoperable with the help of abstraction layers and standardized interfaces, realized by the framework integration mechanisms.
RQ4 Lightweight security mechanisms Cryptographic verification and secure update mechanisms- inbuilt in the framework.
RQ5 Lifecycle management must have versioning, monitoring, and update mechanisms- done using dedicated lifecycle component.
RQ6 Systematic systems enhance 30% increased efficiency of development and also 25 percent performance according to the anticipated benefits analysis.

C. Future Work

Future work consists of staged implementation, validation: short-term (0-12 months) benchmarking prototypes and industry collaboration; medium-term (12-18 months) optimization improvement and lightweight security; long-term (18-24+ months) MLOps integration, community standardization, and automated privacy-aware co-design. The framework confers premise on more scalable, reproducible, and maintainable TinyML systems that can be used in industrial and healthcare IoT uses.

D. Limitations and Broader Implications

Limitations of the research are that it is based on published literature and conceptual framework that needs to be proven empirically. Greater implications are democratization of edge AI development, energy efficiency of IoT and privacy preserving applications with the help of edge processing.

References

1. Schizas, N., Karras, A., Karras, C., & Sioutas, S. (2022). TinyML for ultra-low power AI and large scale IoT deployments: A systematic review. *Future Internet, 14*(12), 363.
2. Alajlan, N. N., & Ibrahim, D. M. (2022). TinyML: Enabling of inference deep learning models on ultra-low-power IoT edge devices for AI applications. *Micromachines, 13*(6), 851.
3. Kallimani, R., Pai, K., Raghuwanshi, P., Iyer, S., & López, O. L. (2024). TinyML: Tools, applications, challenges, and future research directions. *Multimedia Tools and Applications, 83*(10), 29015-29045.
4. Sanchez-Iborra, R., & Skarmeta, A. F. (2020). Tinyml-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits and Systems Magazine, 20*(3), 4-18.
5. Oliveira, F., Costa, D. G., Assis, F., & Silva, I. (2024). Internet of Intelligent Things: A convergence of embedded systems, edge computing and machine learning. *Internet of Things, 26,* 101153.
6. Diab, M. S., & Rodriguez-Villegas, E. (2022). Embedded machine learning using microcontrollers in wearable and ambulatory systems for health and care applications: A review. *IEEe Access, 10*, 98450-98474.
7. Lin, J., Zhu, L., Chen, W. M., Wang, W. C., & Han, S. (2023). Tiny machine learning: Progress and futures [feature]. *IEEE Circuits and Systems Magazine, 23*(3), 8-34.
8. Tsoukas, V., Gkogkidis, A., Boumpa, E., & Kakarountas, A. (2024). A Review on the emerging technology of TinyML. ACM *Computing Surveys, 56*(10), 1-37.
9. Zaidi, S. A. R., Hayajneh, A. M., Hafeez, M., & Ahmed, Q. Z. (2022). Unlocking edge intelligence through tiny machine learning (TinyML). *IEEe Access, 10*, 100867-100877.
10. Abadade, Y., Temouden, A., Bamoumen, H., Benamar, N., Chtouki, Y., & Hafid, A. S. (2023). A comprehensive survey on tinyml. *IEEE access*, 11, 96892-96922.
11. Hayajneh, A. M., Hafeez, M., Zaidi, S. A. R., & McLernon, D. (2024). TinyML empowered transfer learning on the edge. *IEEE Open Journal of the Communications Society, 5*, 1656-1672.
12. El Zeinaty, C., Hamidouche, W., Herrou, G., & Menard, D. (2025). Designing object detection models for TinyML: Foundations, comparative analysis, challenges, and emerging solutions. *ACM Computing Surveys, 58*(2), 1-48.
13. Trilles, S., Hammad, S. S., & Iskandaryan, D. (2024). Anomaly detection based on artificial intelligence of things: A systematic literature mapping. *Internet of Things, 25*, 101063.
14. Ray, P. P. (2022). A review on TinyML: State-of-the-art and prospects. *Journal of King Saud University-Computer and Information Sciences, 34*(4), 1595-1623.
15. Trigkas, A., Piromalis, D., & Papageorgas, P. (2025). Edge intelligence in urban landscapes: reviewing TinyML applications for connected and sustainable smart cities. *Electronics, 14*(14), 2890.
16. Capogrosso, L., Cunico, F., Cheng, D. S., Fummi, F., & Cristani, M. (2024). A machine learning-oriented survey on tiny machine learning. *IEEe Access, 12*, 23406-23426.
17. Pazmiño Ortiz, L. A., Maldonado Soliz, I. F., & Guevara Balarezo, V. K. (2025). Advancing TinyML in IoT: A Holistic System-Level Perspective for Resource-Constrained AI. *Future Internet, 17*(6), 257.
18. Lakshman, S. B., & Eisty, N. U. (2022, May). Software engineering approaches for tinyml based iot embedded vision: A systematic literature review. In Proceedings of the 4th International Workshop on Software *Engineering Research and Practice for the IoT* (pp. 33-40).