TinyML for Interactive Edge Networks

Sonish Korada*
IIIT Sri City, India

Sai Pavan Lingamallu*
IIIT Sri City, India

Bandaru Sai Hari*
IIIT Sri City, India

Abhishek Hazra
IIIT Sri City, India

Mohan Gurusamy
National University of Singapore

***Abstract*—TinyML, deploying low-complexity Machine Learning (ML) models on microcontroller units, which have constrained the availability of resources, is revolutionizing the technologies for extremely low-end devices. This study investigates the importance, categorization, and uses of TinyML in edge networks, providing an overview of workflow and the importance of different compression techniques. The significance of TinyML is highlighted by its utilization in the healthcare, Industrial Internet of Things (IIoT) and agriculture sectors, displaying its ability to perform on-the-spot data analysis and quick decision-making. Nevertheless, TinyML encounters obstacles to interoperability, memory usage, energy efficiency, and standardization. It is essential to tackle these difficulties to exploit the capabilities of TinyML in edge networks. This will allow for efficient data processing, improved decision-making, and better use of resources at the edge level.**

Introduction

THE proliferation of Internet of Things (IoT) devices over the years has increased excessive data creation, demanding smart and effective processing. Transiting this vast amount of data to the cloud could be problematic because it brings more challenges and vulnerabilities, like data breaches and single points of failure [1]. These issues can be overcome by edge computing, a possible technology that can process data near its source.. Edge devices gather and transmit sensor data as quickly and privately as possible [2]. Nevertheless, providing high-end network services is difficult because of memory and power boundaries at the edge level. This restriction requires the most optimal solutions to tackle complicated edge functionality. Edge devices embedded with Machine Learning (ML) could enhance data processing and decision-making speed at the edge level. Traditional ML models need more computing resources [2], which makes them infeasible to deploy at the edge. TinyML, based on Micro-Controller Units (MCUs) in edge devices, promises an intrinsic shift in which small ML algorithms for resource-constrained components guide into a new embedded intelligence age.

Digital Object Identifier 10.1109/MCE.2023.Doi Number

*Date of publication 00 xxxx 0000; date of current version 00 xxxx 0000, *has equal contributions.*

Limitation of Traditional ML: Why TinyML?

TinyML enables ML algorithms on low-resource devices such as microcontrollers. We use compact models for energy-efficient inference to increase the decision-making rate on advanced IoT devices. TinyML can enable faster inference and reduce cloud or fog reliance, even with restricted resources. It can also evaluate sensor data on battery-powered devices for continuous applications.

- *Resource Requirement*: Traditional ML models may require significant computing resources, particularly memory and processing capacity. Insufficient resources of an edge device can restrict the usage of traditional ML.
- *Energy Efficiency*: Traditional model inference on edge devices can waste energy, especially when transferring data and its updates to the cloud. TinyML saves energy costs by doing local computations for energy-efficient inference rather than sending data to cloud servers [2].
- *Possible Impact on Hardware*: Deploying huge ML models on edge devices may lead to physical effects. Complex ML algorithms might raise processor temperatures and impact device performance. Handling these cases is vital for the device's long-term reliability and sustainability at the edge level.

Features of TinyML

By deploying compact versions of ML models at the edge level, TinyML ensures scalability and simplifies IoT applications by lowering the need for continuous online connectivity. Using fewer cloud services fosters privacy and responsiveness. It allows for real-time on-device decision-making, essential for sensitive applications like surveillance and time-critical applications like predictive maintenance. TinyML also goes over distributed ML [1], which promotes cooperative learning amongst nodes. This allows decentralized model training, also known as federated learning, with low latency and the ability to work with non-uniform online connectivity. It also protects privacy and allows for collaborative learning on different datasets, which could impact the IoT landscape.

Benefits of TinyML in Edge Networks

TinyML offers an affordable remedy for edge networks by allowing localized analysis, which lowers the need for expensive cloud reliance. This makes it an economically efficient alternative to inference [3] on the cloud. It achieves energy competence by optimizing processes to be performed locally, and performing operations on energy-efficient edge devices helps to preserve power and prolong the battery life, making IoT devices ideal for industries with limited resources.

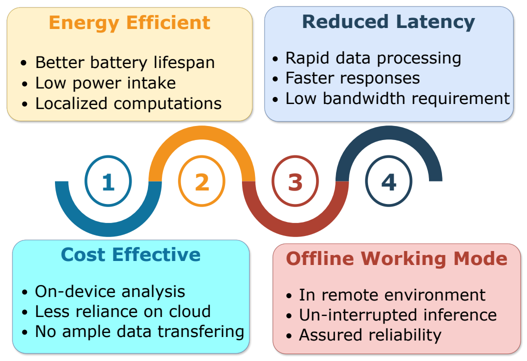
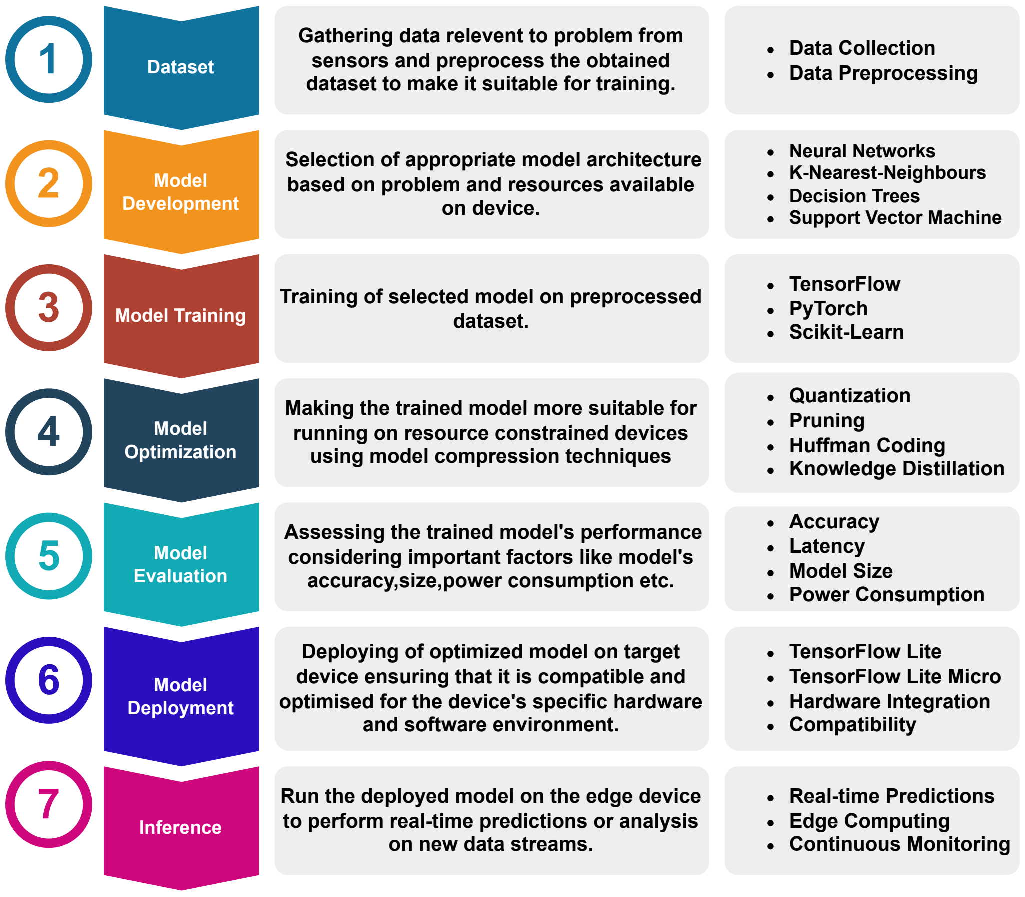


Fig. 1: Major benefits of TinyML at the edge level.

Reducing the duration of data transmission for remote analysis accelerates decision-making [2]. So, it improves the productivity of edge networks, especially in tasks such as autonomous vehicles and industrial automation. These help to function in areas with restricted or unreliable network connectivity by minimizing the reliance on external connections. As mentioned in Fig. 1, TinyML removes the need for a continuous internet connection. This feature guarantees continuous operation and processing, even without connectivity for remote or disconnected operations [4]. Keeping inferences at the local level and focusing on only transmitting a few relevant insights reduces network congestion and optimizes bandwidth utilization. TinyML improves the robustness of edge networks by prioritizing inference tasks over other operations.

Working of TinyML at Edge

Fig. 2 illustrates the comprehensive workflow of TinyML in multiple steps, from data collection and preprocessing to model deployment and inference, describing key processes and important aspects involved in each step. Model optimization plays an important role in the creation of a TinyML model. Upon completion of model training, optimization techniques such as Quantization, Pruning, Huffman Coding, and Knowledge Distillation are employed carefully, considering the model's accuracy, size, and power consumption. We apply these techniques to make the model more suitable for running on targeted edge devices. Pruning involves training a neural network and selecting pivotal synapses by pointing out weights above a specific fixed



1	Dataset	Gathering data relevent to problem from sensors and preprocess the obtained dataset to make it suitable for training.	• Data Collection • Data Preprocessing
2	Model Development	Selection of appropriate model architecture based on problem and resources available on device.	• Neural Networks • K-Nearest-Neighbours • Decision Trees • Support Vector Machine
3	Model Training	Training of selected model on preprocessed dataset.	• TensorFlow • PyTorch • Scikit-Learn
4	Model Optimization	Making the trained model more suitable for running on resource constrained devices using model compression techniques	• Quantization • Pruning • Huffman Coding • Knowledge Distillation
5	Model Evaluation	Assessing the trained model's performance considering important factors like model's accuracy,size,power consumption etc.	• Accuracy • Latency • Model Size • Power Consumption
6	Model Deployment	Deploying of optimized model on target device ensuring that it is compatible and optimised for the device's specific hardware and software environment.	• TensorFlow Lite • TensorFlow Lite Micro • Hardware Integration • Compatibility
7	Inference	Run the deployed model on the edge device to perform real-time predictions or analysis on new data streams.	• Real-time Predictions • Edge Computing • Continuous Monitoring

Fig. 2: Workflow of TinyML.

threshold or loss function. The model is optimized by trimming out the weights that are below this threshold level [5].

Quantization reduces the connection's weight precision from converting floating-point values (32-bit or 64-bit) to lower fixed-point numbers (8-bit), as fixed-point arithmetic is faster and more efficient than floating-point. Quantization can occur during or after model training. Lowering precision too much could result in a significant drop in accuracy. These methods can minimize network parameter storage memory by 20%–30% [5]. There are other compression techniques, like Low-Rank Factorization [4], which approximates a high-dimensional matrix with a low-dimensional one while preserving information. Huffman Coding utilizes binary codes to encode data values for lossless compression. This method is preferable when transmitting machine parameters like weight biases from edge to cloud because frequent data symbol codes are shorter, thus taking up less space.

In addition to model compression, TinyML also uses knowledge distillation [4], where the knowledge distillation training procedure trains the teacher model (with more parameters) on the original training data and then the student model (with fewer parameters) with its predictions.

TinyML Classification at Edge

This section covers Static and Reformable TinyML categorization methods for edge devices in various deployment scenarios.

Static TinyML

Static TinyML is a fundamental technique used to deploy ML models on edge devices. It mainly follows the "*first-train-then-test*" principle. Once the model is trained and deployed on the device, updateability is unlikely. The deployed model does not support upgrades depending on new data or feedback. During the offline training phase on powerful hardware or in the cloud, the model's parameters are set and consistent during deployment. While not appropriate for dynamic surroundings or adaptive applications, Static TinyML offers a simple and fast solution for picture categorization, keyword detection, and basic sensor data processing on resource-constrained edge devices.

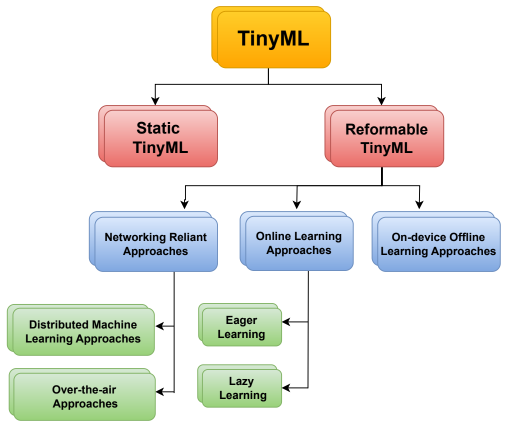

TinyML
Static TinyML
Reformable TinyML
Networking Reliant Approaches
Online Learning Approaches
On-device Offline Learning Approaches
Distributed Machine Learning Approaches
Over-the-air Approaches
Eager Learning
Lazy Learning

Fig. 3: Classification of TinyML.

Reformable TinyML

The Reformable TinyML [1] approach to deliver ML models on restricted hardware is revolutionary and supports ongoing improvement through dynamic updates. Unlike static implementations, reformable models can improve performance locally or over the air. IoT environments with fluctuating data distributions benefit from their dynamic nature. Reformable TinyML updates keep edge computation efficient and reduce model drift. The decision between Reformable and Static TinyML depends upon the application's requirements, availability of resources, and the intended trade-off between model intricacy and adaptability. Based on different techniques used to update the model, it can be further classified as follows:

1) Online Learning Approaches:
   a) *Eager Learning Techniques:* These algorithms require greater computational resources and longer training time but provide faster inference rates. Enthusiastic learners freeze the model weights and gradually adjust the biases to adapt to fresh data. The authors in [6] have introduced Tiny Transfer Learning (TinyTL) that freezes weights and gradually learns biases, minimising memory consumption during training.
   b) *Lazy Learning Techniques:* Certain algorithms save training data until testing data becomes accessible and provide predictions based on stored samples. In [7], the authors implemented Online Learning on Raspberry Pi 3B+ and STM32F7, employing a non-trainable DL-based Feature Extractor and kNN classifier for binary tasks. kNN yielded slightly lower accuracies than SVM on Pi, with the feature extractor showing longer processing times on STM32F7.
2) Network Reliant Approaches:
   a) *Distributed TinyML:* These methods enable collaborative model training on decentralised data sources, using federated or gossip learning mechanisms for privacy and edge computing. A distributed TinyML model can be trained locally on edge devices and aggregated on a central server for global updates. Globe2Train framework by Sudharsan *et al.* [8] lets geographically distributed IoT devices train a single model. Globe2Train is driven by GPU's scarcity compared to IoT devices. G2T-Cloud and G2T-Device operate together to promote collaborative training in the proposed system.
   b) *Over-the-air (OTA) Approaches:* Enable wireless model updates for remote distribution and reformatting without physical access to the device. OTA updates are crucial for the model's performance and security over the long term. In [9], Badawy *et al.* introduced FOTA to flash ATMEL AVR MCUs wirelessly. This is done with Wifi and LoRa, where FOTA allows dynamic TinyML updates.

TinyML Applications

TinyML is transforming various industries, including healthcare, agriculture, industrial IoT, UAVs, and autonomous cars, by offering powerful capabilities in a compact form. Its versatility and efficiency drive innovation across sectors heavily reliant on IoT and embedded technology. Given its rapid expansion, TinyML is revolutionizing numerous fields through its ability to handle data in real time and its decentralised approach. Below are several use case scenarios illustrating the application of TinyML in various domains.

TinyML and Healthcare

TinyML has the potential to transform healthcare by improving monitoring and personal health products.

Wearable devices have various biological sensors that constantly monitor important bodily processes such as heart rate, blood oxygen levels, and activity. These gadgets provide immediate and confidential information about the user's health. TinyML allows for extensive real-time data analysis using compact, pre-trained inference models, thus avoiding continuous streaming. In addition, smart camera sensors powered by TinyML allow for real-time diagnosis, nurse alerts, and personalised treatment. Intelligent microprocessors that utilise TinyML efficiently forecast the probability of accidents, enhancing the monitoring of patients, diagnosis, and therapy. This integration improves healthcare quality by enabling professionals to monitor and treat patients more efficiently and effectively.

TinyML and Industrial IoT

TinyML is transforming the Industrial Internet of Things (IoT) by propelling automation, connectivity, and intelligent sensing progress. Within Industry 4.0's digital transformation context, TinyML enhances these processes by facilitating predictive data analysis on embedded devices. This allows for real-time analysis and overcomes limitations in terms of time restrictions. By incorporating ML into microcontrollers, making intelligent decisions, enhancing production efficiency, monitoring assets, and ensuring quality assurance becomes possible. The scope of TinyML encompasses anomaly detection, which involves the real-time identification of atypical patterns in industrial machinery. Implementing TinyML on edge devices guarantees quick responsiveness, minimizing latency. TinyML improves industrial processes in the era of smart manufacturing by optimising condition monitoring and enabling predictive maintenance.

TinyML and Agriculture

TinyML is significantly changing smart agriculture, revolutionizing crop monitoring, disease detection, and yield optimisation. TinyML can greatly enhance regions like Africa, where embedded systems and AI are underutilized. The Nuru programme by PlantVillage utilises algorithms based on TinyML and can be accessed through mobile phones. This empowers farmers in rural Africa to analyse sensory data in real time and effectively manage crop threats such as cassava. Furthermore, TinyML efficiently monitors coffee beans during the roasting process, improving the beans' overall quality. TinyML enables advanced prediction models to provide farmers with real-time meteorological data, allowing them to make well-informed decisions. The agriculture industry adopts sustainable methods and optimises resource utilisation by incorporating TinyML into IoT devices, thereby addressing the needs of a growing population.

Open Research Challenges

It is necessary to explore the obstacles and prospective analysis directives of TinyML at the edge level to unlock this technology's potential fully. Identifying this wide range of opportunities and their interconnection with other fields is crucial, with some major ones listed below.

Interoperability

As the heterogeneity of the devices at the edge is increasing daily, things like outdated standards and integration compatibility issues hinder interoperability. Even though prevailing partially, difficulties with categorizing, scaling, and emerging innovations still exist. Advanced ML techniques, standards for interoperability testing, and integration with new and updated technologies are needed to make the next phase easy.

Finer Memory Utilization

MCUs typically have an undersized memory space where they should be able to do other things besides inference. Memory needs to be used optimally for extensive and quick computation. These days, memory space bounds cause a lowering speed of response and slow scaling [2]. As mentioned earlier, we can use model compression techniques, but there should be a decent trade-off between compressed model parameters and the original ones regarding storage. Finding a balance between the computational burden and memory requests is a major problem. Research more about integrating hardware and software in memory systems, implementing novel storage designs, compacting data, and efficiently organising data. Estimated computation makes calculations less precise to save memory space.

Optimal Trade-Off for Energy Usage

As environmental concerns exist, it is crucial to find an appropriate balance between performance and functionality within these energy limits. Upcoming updates in TinyML should focus mainly on low-power circuits, energy-efficient designs, and smart task-load management to obtain the best performance while saving energy. Using scheduling and forecasting methods to reduce energy footprints can help achieve the most optimal utilization. TinyML applications at the edge level could be more trusted using techniques like energy harvesting, thermal control and approximation computation.

Resource Constraints

Some concerns for edge devices are optimal storage, high performance, and less processing power. Already existing systems may give up precision or functionality to gain the most from their resources. Future research should shorten and quantify models to save these resources. Throughput and energy economy could be improved by both hardware accelerators and domain-specific designs. Particularly, there is a need for Resource-aware algorithms, approximations, and adaptive methods such as task offloading strategies to cloud and fog environments.

Conclusion

Nowadays, TinyML can assist us in resolving the issues we're experiencing with low-end devices at the edge level. This technology has created an impact and is advancing swiftly, but it is still in its initial implementation phases and needs more development. There should be a standard trade-off between what we can expect and what it produces at the edge. To exploit TinyML's benefits even more, procedures and frameworks must be standardized. This magazine discussed how a traditional ML model can be turned into a TinyML model and the various types of TinyML suited to different circumstances. It also exchanges views about how TinyML is being used in innovative solutions and what complications or research needs we have alongside potential solutions.

REFERENCES

1. V. Rajapakse, I. Karunanayake, and N. Ahmed, "Intelligence at the extreme edge: A survey on reformable tinyml," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–30, 2023.
2. N. Schizas, A. Karras, C. Karras, and S. Sioutas, "Tinyml for ultra-low power ai and large scale iot deployments: A systematic review," *Future Internet*, vol. 14, no. 12, p. 363, 2022.
3. S. A. R. Zaidi, A. M. Hayajneh, M. Hafeez, and Q. Z. Ahmed, "Unlocking edge intelligence through tiny machine learning (tinyml)," *IEEE Access*, vol. 10, pp. 100 867–100 877, 2022.
4. Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, and A. S. Hafid, "A comprehensive survey on tinyml," *IEEE Access*, 2023.
5. S. Soro, "Tinyml for ubiquitous edge ai," *arXiv preprint arXiv:2102.01255*, 2021.
6. H. Cai, C. Gan, L. Zhu, and S. Han, "Tinytl: Reduce memory, not parameters for efficient on-device learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 11 285–11 297.
7. S. Disabato and M. Roveri, "Incremental on-device tiny machine learning," 11 2020, pp. 7–13.
8. B. Sudharsan, J. G. Breslin, and M. Intizar Ali, "Globe2train: A framework for distributed ml model training using iot devices across the globe," in *2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI)*, 2021, pp. 107–114.
9. W. Badawy, A. Ahmed, S. Sharf, R. A. Elhamied, M. Mekky, and M. A. Elhamied, "On flashing over the air "fota" for iot appliances – an atmel prototype," in *2020 IEEE 10th International Conference on Consumer Electronics (ICCE-Berlin)*, 2020, pp. 1–5.

Sonish Korada is with the Indian Institute of Information Technology, SriCity, India. Contact him at `sonishkorada@gmail.com`

Sai Pavan Lingamallu is with the Indian Institute of Information Technology, SriCity, India. Contact him at `saipavanlingamallu@gmail.com`

Bandaru Sai Hari is with the Indian Institute of Information Technology, SriCity, India. Contact him at `Saiharibandaru2002@gmail.com`

Abhishek Hazra is with Indian Institute of Information Technology SriCity, India. Contact him at `abhishek.hazra1@gmail.com.`

Mohan Gurusamy is a Professor with National University of Singapore. Contact him at `gmohan@nus.edu.sg`.