Impact of Machine Learning in Automation - A Review

Rohan Sahni
Student, Foundation Level, IIT Madras, India

Abstract- **In the last decade, the implementation of machine learning models on compact, low-power devices like smartphones, Arduino, smartwatches, and IoT devices has grown exponentially. This paper reviews the evolution of machine learning in edge devices over this period[1]. It focuses on how these devices have become pivotal in the field by independently implementing machine learning and deep learning models on their locally collected data, thus reducing their dependency on cloud services. The paper will explore the concept of edge devices, their emerging significance, and the transformative impact they hold in the realm of automation and real-time data processing[2][3].**

Index Terms— Machine Learning, Edge Devices, TinyML, Arduino, Edge Computing

I. INTRODUCTION

Imagine a world where each device around you possesses its own intelligence. Beyond performing complex calculations or entertaining us, these machines become interactive agents that assist in decision-making processes. This paper examines the shift from cloud-dependent devices, such as Alexa's cloud-based voice service system, towards independent, onboard machine learning for real-time outcomes. The past decade has witnessed a surge in IoT and embedded devices, offering enhanced functionalities to simplify complex challenges[4]. Big hardware companies are now integrating machine learning into products like smartwatches and smartphones, enabling computing at the data source – a practice known as Edge Computing[5]. This approach decentralizes computing intelligence, echoing the transition from mainframes to powerful, independent PCs. Edge computing enhances response times by reducing network congestion and lowering costs. This paper will also discuss the classification of machine learning algorithms into supervised and unsupervised learning, detailing their application in edge devices for optimized performance[4][5].

II. HISTORY OF MACHINE LEARNING IN DEVICES

A. Traditional Approach

Machine learning's roots can be traced back to the 1950s, marked by key events that established computers' ability to learn. Alan Turing's Turing Test in 1950 set the stage for this evolution, followed by Arthur Lee Samuel's development of a learning program for checkers in 1952. This section will explore the traditional approach to machine learning, involving cloud-based data processing, and the limitations of such an approach. Despite the ubiquity of AI in modern technology, its real power remains largely with major corporations like Google and Amazon. The section will address the necessity for machine learning in edge devices, citing the importance of reliability, low latency, privacy, and power efficiency in applications like autonomous F-16s or Azure Edge Zones.

A straightforward approach for beginners is to utilize pre-existing machine learning models offered by various companies. These services cater to specific needs like speech recognition, text analysis, or image classification. While direct access to these proprietary models isn't possible, they can be utilized via an Application Programming Interface (API). Service providers such as Clarifai, Google Cloud Vision,

Amazon Rekognition, Polly, Lex, Microsoft Azure Cognitive Services, IBM Watson, Amazon AWS, and Oracle Cloud offer machine learning as a service. These services are becoming increasingly common.

Applications requiring these machine learning capabilities can simply send an HTTPS request to the relevant web service with necessary data, like a photo captured by a device's camera. The service provider then processes this data and returns prediction results within seconds. Integration is facilitated through API permissions and often supported by a Software Development Kit (SDK), simplifying the process for developers.Service providers continuously update and refine their models with new data, enhancing the accuracy and efficiency of the models. Users of these services benefit from these improvements without the need for personal servers, GPUs, or expertise in model training. This approach is particularly user-friendly, even for those without programming experience.[6]

However, this method has limitations. It does not support local inference on devices; all predictions require network requests to the server. This dependency on network connectivity means that any network disruption can render the app non-functional. Additionally, while the cost model might seem economical (e.g., $1 for every 1000 requests), it restricts users from training personalized models or using unique data types beyond standard images, videos, and audio.

B. Limitation of the traditional approach

While Artificial Intelligence (AI) has become a ubiquitous presence in modern technology, ranging from smartphone applications to home assistants, its accessibility to the general public remains limited. Major corporations like Google, Amazon, and Facebook dominate AI utilization, leveraging their resources to run AI models that typically require GPUs—expensive hardware not readily available to the average consumer. Most AI applications rely on cloud services for processing, using virtual GPUs. This reliance places a significant demand on internet bandwidth, a challenge that becomes more pronounced with the increasing number of IoT devices. The advent of 5G technology promises to alleviate some of these bandwidth constraints. However, the existing TinyML libraries, often larger in size than their name implies, still pose challenges for edge devices with limited storage capacity, indicating a gap between AI's potential and its practical deployment in everyday technology.

C. The need of Machine Learning in EDGE Devices :

The integration of Machine Learning (ML) into edge devices is increasingly essential, shifting away from reliance on cloud services towards localized processing. This approach, central to modern embedded systems, situates computing power at the network's edge, close to where data is collected. This is crucial for a wide array of applications, from automated industrial machinery and robots to household devices like self-guided vacuums and agricultural tractors, where processing must occur on-site[7].The rationale for embedding AI in edge devices can be distilled into several key factors:

A. Reliability: Dependence on continuous internet connectivity isn't always feasible, especially with the proliferation of IoT devices in domestic settings. While 5G networks promise improved bandwidth, the limitations of current 4G networks can impede the performance of an increasing number of smart devices.
B. Low Latency: Many applications require swift responses that can't afford delays, necessitating real-time decision-making. Technologies like Microsoft Azure's Edge Zones exemplify this, allowing for data processing near users to meet the demands for low latency and high throughput in various applications.
C. Privacy: Transmitting sensitive data externally poses privacy concerns. Devices handling private information, such as baby monitors, mobile phones, and CCTV cameras, require secure, local data processing. Innovations like Pico Voice demonstrate how AI can function on slower chips using minimal data, enhancing privacy.
D. Power and Speed Efficiency: Transmitting data over distances consumes significant power and may be hampered by low connectivity, impacting data transmission speed and volume. This is particularly relevant in fields like logistics, where real-time vehicle-to-vehicle communication and efficient data processing are vital for effective operations.

III. TINY ML LIBRARIES

This part of the paper discusses TinyML, a collaboration between embedded ultra-low power systems and machine learning communities, opening avenues for on-device machine learning applications. It examines the challenges and solutions in adapting machine learning models for small, less powerful IoT devices. This includes using languages like MicroPython for programming and the utility of different TinyML libraries provided by tech giants like Google, Facebook, and Amazon. The paper will explore various libraries and services in the market, their platforms, and their roles in enhancing edge AI capabilities[8].

TinyML represents the synergistic integration of embedded ultra-low power systems and machine learning communities, which have historically functioned in separate spheres. This collaboration has unlocked a multitude of innovative opportunities for machine learning applications directly on devices. Essentially, a machine learning model equips a software system with the capability to receive input and generate predictions. Edge IoT devices often demand quick and dependable performance, necessitating the processing of machine learning predictions on the device itself. However, due to the limited power and capacity of these devices, it's impractical to directly transfer a model developed for a high-powered desktop to an IoT device.

Typically, custom coding in C or C++ is employed for these devices, but this method struggles to scale with more complex machine learning models. This is where MicroPython comes into play, allowing Python commands to be executed on a microcontroller. Instead of the traditional process of coding in C or C++, compiling, and uploading to an edge device, MicroPython enables direct coding in Python, offering a more straightforward and efficient workflow. This is particularly advantageous given Python's reputation for being a resource-intensive and slower language, yet MicroPython optimizes it for use in resource-constrained environments.[7]

When coding with MicroPython, it's crucial to consider hardware limitations, such as the number of cores and memory size. For instance, 64KB of memory might be insufficient for certain applications. Microcontrollers (MCUs) typically have strict resource allocation, but MicroPython incorporates various mechanisms to address these challenges effectively.

MicroPython's compatibility with Python and its associated tools on MCUs democratizes programming for electronics, making it accessible not only to experienced Python programmers but also to beginners. Python's user-friendly nature and growing popularity, often being the first choice for newcomers over languages like JavaScript or C++, makes it an attractive option for those venturing into programming for embedded systems.

A. Different Types of TinyML Libraries:

Presently, a variety of machine learning libraries for edge devices are being developed by leading tech companies like Amazon, Facebook, and Google, all vying for dominance in the Edge AI market. The following is a compilation of the key libraries and services that have recently garnered attention.

S.NO	Libraries and Services	Company	Description	Supported Platforms
1	TensorFlow Lite	Google	It is a framework that translates your models into more mobile freindly versions.	Arduino Nano 33 BLE Sense, SparkFun Edge, STM32F746 Discovery Ki
2	Caffe2Go	Facebook	Facebook has demonstrated this model as several confrences to show how openGL could speed up image recognition and special effects on your phone.	NVIDIA Volta, ARM Mobile GPUs, iOS, Android, Raspberry Pi
3	CoreML	Apple	Apple ML library coreML is used for deploying ML models on Apple devicesIt allows you to design and develop machine learning models for Apple OS apps, and then encapsulate them into the app bundle..	iOS, iPadOS, watchOS, macOS, and tvOS.
4	ML Kit	Google	ML kit is a hosted service by Google for deploying your TensorFlow Lite model on your iOS and Android, which is a benefit over Apple's local solution. ML Kit offers a few different APIs for popular use cases like Image Recognition and Natural Language Processing,	iOS and Android

5	ELL	Microsoft	Microsoft launched The Embedded Learning Library (ELL) that allows users to design and deploy intelligent machine-learned models onto resource constrained platforms and small single-board computers, like Raspberry Pi, Arduino, and micro:bit. The deployed models run locally, without requiring a network connection and without relying on servers in the cloud	Raspberry Pi, Arduino, and micro:bit.
6	EML	Microsoft	These algorithms can train models for classical supervised learning problems with memory requirements that are orders of magnitude lower than other modern ML algorithms. The trained models can be loaded onto edge devices such as Arduino Due, and used to make fast and accurate predictions completely offline.	Raspberry Pi, Arduino, and micro:bit.
7	XNOR AI 2 GO	Apple	Xnor's ai's technology has done a great job by replacing AI models' complex mathematical operations with simpler, rougher, less precise binary equivalent that makes data-driven	Raspberry Pi, Linux, Ambarella, Toradex, etc.

			machine learning, deep learning, and other AI models to to be executed on low-power devices	
8	AWS IOT and FREE RTOS	Amazon	Amazon Web Services' recently released the open-source AutoGluon toolkit. It is an ML pipeline automation tool that includes a "neural architecture search." feature . High performance ML models can be automatically generated from as few as three lines of Python code.	ATECC608A Zero Touch Provisioning Kit for AWS IoT, Cypress CYW943907AEVAL1F Development Kit, Cypress CYW954907AEVAL1F Development Kit, Espressif ESP32-DevKitC, Espressif ESP-WROVER-KIT, Infineon XMC4800 IoT Connectivity Kit, Marvell MW320 AWS IoT Starter Kit, Marvell MW322 AWS IoT Starter Kit, MediaTek MT7697Hx Development Kit, Microchip Curiosity and many more.
9	DeepLite	DeepLite	DeepLite a Montreal-based AI startup Deeplite can automatically infer on a range of edge-device hardware platforms. It does this without requiring manual inputs or guidance from scarce.	ARM, RISC-V, x86 Architecture
10	SWIM EDX	SWIM.AI	Swim is the first general purpose platform for building real-time apps using streaming data and optimized to run	All existing edge devices

			across distributed compute environments.Swim gives you a live window into your apps.	
11.	AutoGluon	Amazon	Simplifies high-performance ML model creation, integrating	Tensorflow and Caffe2
12.	NVIDIA EGX	NVIDIA	Supports robust edge computing needs with compatibility	Jetson, NVIDIA T4 Servers

- **TensorFlow Lite (Google)**: This is an advanced version of TensorFlow, designed to adapt models for better mobile compatibility. It focuses on reduced latency, smaller model size, and swift execution, although being in the early stages of development, it may deliver varying performance levels.

- **Caffe2Go (Facebook)**: An iteration of the Caffe2 deep learning framework, Caffe2Go is Facebook's venture into mobile-friendly models. Demonstrated at various conferences, it highlights how OpenGL can enhance image recognition and special effects on mobile devices, though it remains proprietary and not open-source.

- **Core ML (Apple)**: Apple's Core ML is an ML library tailored for integrating machine learning models into Apple device applications. It facilitates the design and development of ML models for Apple's operating systems and supports model conversion from well-known frameworks like TensorFlow Lite and Caffe2. Core ML received significant enhancements at Apple's 2018 WWDC.

- **ML Kit (Google)**: Provided by Google, ML Kit is a service enabling the deployment of TensorFlow Lite models on both iOS and Android, offering an advantage over Apple's localized solutions. It provides various APIs for tasks like Image Recognition and Natural Language Processing and is integrated with Google’s Firebase platform.

- **Embedded Learning Library (Microsoft)**: Microsoft's ELL allows for the creation and deployment of intelligent machine-learning models on resource-limited platforms and small computers like Raspberry Pi, Arduino, and micro:bit. These models operate locally, eliminating the need for cloud dependency. ELL, still in its nascent stage, showcases Microsoft Research's developments in embedded AI and machine learning.

- **Edge Machine Learning (EML) by Microsoft**: This library encompasses algorithms like Bonsai, ProtoNN, EMI-RNN, Shallow RNN, FastRNN, and FastGRNN, facilitating the training of models for classic supervised learning with significantly reduced memory needs. These models can be efficiently executed on edge devices such as Arduino Due, offering rapid and precise offline predictions.

- **Xnor.ai**: This technology revolutionizes AI models by simplifying complex mathematical operations into more straightforward binary equivalents, enabling execution on low-power devices. This breakthrough supports applications like facial recognition, natural language processing, and augmented reality.

- **Amazon Web Services (AWS)**: AWS has introduced the AutoGluon toolkit, an open-source ML pipeline automation tool featuring neural architecture search. It allows for the swift generation of high-performance ML models using minimal Python coding, optimized for specific environments through reinforcement learning algorithms and capable of integrating with existing AI DevOps pipelines.

- **Deeplite**: This Montreal-based AI startup focuses on optimizing neural network weights and biases for high performance, facilitating automatic inference across various edge-device hardware platforms without manual intervention.

- **Swim.ai**: Utilizing TinyML on streaming data, Swim.ai aims to enhance passenger safety and reduce traffic congestion and emissions. It stands as a pioneering platform for building real-time applications using streaming data, optimized for distributed computing environments. It offers a live view into applications, with native UI/UX tools and plugins for real-time KPIs, maps, and other visualizations.

.

B. Integrating TensorFlow Models with Arduino for Machine Learning Applications:

Numerous developers have successfully implemented machine learning on Arduino by importing pre-trained models as header files. These models are used in various applications, such as creating Arduino-controlled robots capable of making decisions based on real-time sensor data. Well-known examples include applications for micro-speech, magic wand, and person detection. These examples can be readily imported from the Arduino IDE's library manager and used directly.

For those looking to train models with custom data, the process begins by selecting the appropriate port and board in the Arduino IDE. Sensor data can be read using a serial plotter and then exported in CSV format to a local device. The next step involves setting up a Python environment to upload and prepare this data. Once the data is ready, the machine learning model can be built and trained. After training, the model is converted to TensorFlow Lite format and encoded into an Arduino header file. This header file can then be used as a pre-trained model in Arduino projects. The training process involves feeding a set of known input-output pairs to the neural network (NN), with the expectation that the NN can later predict outputs for previously unseen inputs. Training typically occurs on a more powerful workstation or in the cloud, not on the target device, with only the prediction function executed on the Microcontroller Unit (MCU)[8].

To facilitate coding, developers can import functions from libraries like NumPy. For instance, creating a NeuralNetwork class in Python allows for the training and evaluation of a simple neural network. With three binary inputs yielding eight possible combinations, a training set might include four of these combinations, training the NN with 50% of the possible values. An array of these four inputs and their corresponding outputs is created, and the NeuralNetwork class is initialized to view the random initial weights.

Further research in TinyML libraries might focus on two approaches: using hardware that supports binarization to develop algorithms from scratch or refining software and code. This includes pruning unimportant network connections, quantizing the network for efficient operation, enforcing weight sharing,

and applying Huffman encoding for data compression. These techniques are crucial in optimizing machine learning models for resource-constrained environments like those found in Arduino and similar platforms[12].

C. *Current Use of TinyMl and TensorFlow Lite*

The advent of Machine Learning (ML) in Edge Devices heralds significant benefits, including enhanced data security, privacy protection, and rapid, real-time processing capabilities. This technological shift is ushering in an era of innovative intelligent devices, ranging from smart watches and secure versions of digital assistants like Alexa, to advanced smart cameras, personalized home systems, sophisticated vacuum cleaners, and mobiles equipped with 4G and individual AI features.

In the realm of commercial applications, this approach is particularly evident in voice assistants. Utilizing wake words like “Alexa” or “Hey Siri” activates these devices. Once activated, voice recordings are processed, often involving cloud-based steps for parsing, interpretation, and response generation. This functionality is even achievable on devices with limited computational power, such as Arduino or Raspberry Pi, which have minimal memory and low-power processors.

For iOS and Android developers, platforms like Fritz AI provide a comprehensive machine learning solution. Their SDK includes pre-trained ML models, and their Studio allows for the creation and deployment of custom-trained models. Notably, the “Hey Siri” command relies on three distinct deep neural networks (DNNs), with two operating directly on the device. The primary DNN functions as a speech recognizer, operating on an always-on motion coprocessor.

Google's Pixel 2 leverages Qualcomm Snapdragon’s “neural” units to enhance its machine learning operations, enabling features like HDR+ and optical image stabilization in photography, and an always-on Shazam-like feature for audio fingerprinting. Similarly, Apple's iPhone X uses its AI capabilities for Face ID training and inference. Amazon's DeepLens, a deep learning video camera, exemplifies the potential of local compute power with its 100 GFLOPS capability. While currently limited, such devices are anticipated to surpass smartphones in AI application scope.

Examples of these advancements in the field include experiments with Arduino Uno under the Bonsai project, Raspberry Pi 3 in the DeepThings project, and Intel’s Edison board in the DeepIoT initiative. These projects cover a wide array of functionalities, including image classification, object detection, pose estimation, speech recognition, gesture recognition, and smart response systems. This diverse range of applications highlights the growing potential and impact of ML in edge devices.[9]

D. *Techniques and Applications*

Arduino microcontrollers, along with similar devices, offer a simpler and more straightforward experience compared to complex Central Processing Units (CPUs). Their design ensures that no additional processes interrupt their operation, making them an ideal choice for integrating AI algorithms and data. This simplicity is key to their widespread adoption in various applications[9].

In the realm of AI, a notable research project at Stanford's AMPLab is focused on developing methods to compress neural networks. The goal is to enable these networks to function effectively on less powerful processors, utilizing minimal memory, storage, and bandwidth. This is particularly important at the device level where resource constraints are a significant concern. The technique involves streamlining neural networks by pruning redundant or non-critical connections, recalibrating the importance of remaining connections, and applying more efficient encoding methods to the model. Such optimization aims to retain the accuracy of pattern recognition while reducing the computational load.

Complementing this effort, the project named Succinct is dedicated to enhancing the efficiency of data compression, specifically for data stored locally on resource-limited mobile devices and IoT endpoints. This initiative allows deep neural networks and other AI models to directly interact with sensor data stored in flat files. By operating on compressed and cached local data, these models can swiftly execute search queries, perform calculations, and undertake other tasks without the need for extensive processing power or data transmission. These advancements in data compression and model efficiency are crucial for expanding the capabilities of AI in resource-constrained environments.

E. Embedded Systems and Machine Learning Integration

Let's encapsulate the current state and challenges in integrating machine learning with various embedded systems, highlighting the need for specialized hardware and software adaptations to fully leverage AI capabilities in these platforms[10][11][12][13][14][15].

1. Linux on AMD64/Intel IA64 Embedded Systems:
For embedded platforms running full-function Linux, such as AMD64 or Intel IA64, the barriers to implementing machine learning models are minimal. Concerns primarily revolve around installation, performance optimization, and UI adaptation for unique interfaces. High computational tasks like image recognition necessitate hardware acceleration, often through a GPU, and the installation of specialized tools like cuDNN and CUDA. Integration with Nvidia GPUs simplifies the process, requiring minimal code alterations. Limited acceleration demands can be met with devices like Intel's Movidius Neural Compute Stick, and more extensive needs may lead to the adoption of FPGAs or Intel's Arria hardware.

2. ARM Systems and Linux Distribution:
ARM SoC-based systems usually come with comprehensive Linux distributions, such as Raspbian for Raspberry Pi or Ubuntu for Nvidia Jetson. Customization is possible with tools like Gentoo and Debian. Despite some challenges, most Python libraries and frameworks are compatible with ARM platforms. However, machine vision libraries often require building from source. Incorporating an ARM Mali GPU can significantly enhance power efficiency and inference speed, supported to some extent by OpenCL.

3. Custom Linux and Non-Linux for Specialized Hardware:
In cases involving less common CPU architectures, custom Linux distributions are created using tools like Yocto or Buildroot. For specialized applications unable to run Linux, such as those in regulated or safety-critical environments, systems like Wind River's VxWorks are employed. Recent updates have broadened Python support in these systems.

4. FPGAs and ASICs in Machine Learning:
In highly specialized or high-volume environments, FPGAs and ASICs are prominent. These components often require substantial investment in design verification and licensing. Machine learning models, primarily mathematical constructs, can be converted to formats like ONNX for compilation into device-specific codes.

5. Model Compilation and Development Platforms:
The latest in model compilation includes Apache TVM, which optimizes embedded model generation for various frameworks. Development and training of machine learning models are often conducted on major platforms like Google, Amazon, Microsoft, IBM, or through MLaaS platforms. Tools like Anaconda/Jupyter and frameworks such as Keras and TensorFlow facilitate this process.

6. Machine Learning in Microcontrollers:
The feasibility of running machine learning on microcontrollers, such as for language processing tasks without cloud dependency, raises intriguing possibilities. Post-implementation considerations include local processing capabilities, which can significantly enhance interactive functionalities in devices like robots.

IV. CHALLENGES

This section examines the challenges of implementing machine learning in edge devices, focusing on the computational and power constraints of chip-level AI operations. It highlights the importance of industry-standard TinyML benchmarks in evaluating the performance of machine learning inference chips and the difficulty of creating a one-size-fits-all benchmark for TinyML performance. The section also delves into the challenges faced by the edge AI industry in keeping up with the rapidly evolving landscape of AI applications and the diverse architectures of edge devices[16][17][18][19].

A significant challenge in chip-level AI operations is the necessity to perform many calculations, such as those for training and inferencing, sequentially rather than in parallel. This serial processing is not only time-consuming but also places a heavy computational load on devices, leading to rapid battery depletion. The common practice of offloading data processing to cloud-based AI services introduces latency issues, making this solution impractical for AI applications that require high-performance, such as interactive gaming, at the edge.

The effectiveness of TinyML initiatives is ultimately determined by their performance. As the edge AI market evolves, the need for standardized TinyML benchmarks becomes critical to validate claims of speed, resource efficiency, and cost-effectiveness. The Embedded Microprocessor Benchmark Consortium (EEMBC) has introduced MLMark, a benchmarking tool for machine learning inference chips, to facilitate comparisons between different embedded edge device architectures. MLMark tests with popular models like ResNet-50, MobileNet, and SSDMobileNet, and includes support from major hardware providers like Intel, Nvidia, and Arm. The results from these benchmarks, including MLPerf, are becoming increasingly significant for positioning TinyML solutions in the market, highlighting the capabilities of edge AI.

However, developing a universal benchmark for TinyML performance is challenging due to the diverse range of device architectures and applications. Benchmarks must cater to a wide array of devices, from drones and autonomous vehicles to smartphones and computer-vision systems. Keeping up with the rapidly evolving assortment of AI applications, which now extend to innovative areas like real-time human-pose estimation in browsers, further complicates the creation of standardized benchmarks. Moreover, the variety of training and inferencing workflows, whether on the edge, at the gateway, or in data centers, adds another layer of complexity, making it unlikely for a single benchmarking suite to cover all scenarios effectively.[16]

Despite these challenges, the shift from cloud-based to edge computing is a significant step in addressing resource constraints in machine learning models. However, many models still require substantial computing power and memory, exceeding the capacity of smaller microprocessors and wearable devices prevalent in the market today. This underscores the ongoing need for advancements in TinyML technologies and benchmarking methodologies to fully realize the potential of edge AI.

Edge devices in healthcare: In the realm of healthcare, edge devices empowered by Machine Learning (ML) have significantly advanced the capabilities of Intensive Care Units (ICUs). These devices excel in real-time data processing and decision-making, which are critical for maintaining essential physiological parameters like blood glucose levels and blood pressure within safe ranges. Edge devices are increasingly being used to monitor and analyze more complex health indicators, such as neurological activity and cardiac rhythms, thanks to advancements in hardware and ML methodologies[20][21][22].

In environments where connectivity is limited or nonexistent, such as remote oil, gas, and mining sites, EdgeAI demonstrates immense utility. In these areas, employees work far from urban centers, and edge devices like robots

equipped with sensors can gather extensive data to predict variables like pressure changes in pumps or deviations in operational parameters. This technology is also pivotal in manufacturing sectors for predictive maintenance of machinery, helping to reduce costs and prolong the lifespan of equipment. The introduction of Amazon Web Services' AutoGluon toolkit, an open-source ML pipeline automation tool featuring a "neural architecture search," is a testament to these advancements.

Complementing these developments, a project named Succinct focuses on enhancing the efficiency of data compression for locally stored data on resource-limited mobile and IoT devices. This project enables deep neural networks and other AI models to process sensor data stored in flat files, facilitating immediate execution of tasks such as search queries and computations on this compressed, cached data.

Therefore, it's evident that the development of consensus practices, standards, and tools for TinyML is a significant and complex task. These efforts are crucial in harnessing the full potential of edge computing and ML in various industries, particularly in settings where traditional connectivity and data processing methods are impractical.

- **Variability in Data**: Real-world data is inherently noisy and variable, meaning the same target might appear differently under various conditions. This variability is not only in the target itself but also in the background environment. For instance, vibration sensors in industrial settings might capture extraneous vibrations from adjacent machinery. Recognizing and distinguishing these background variations is as crucial as identifying the target variations. Therefore, it's essential to gather a diverse range of examples and counterexamples across multiple scenarios.

- **Real-time Detection in Firmware**: Implementing detection capabilities directly within a device's firmware to provide immediate feedback or trigger time-critical actions in machinery poses additional complexity. Ensuring "real-time" detection and response within the device's firmware requires sophisticated programming and integration.

- **Physical, Power, and Economic Constraints**: The ideal scenario of unlimited computing power doesn't align with the practical limitations of real-world devices. Products must be designed within specific parameters, including size, weight, power consumption, and cost, making it challenging to integrate advanced computing capabilities.

- **Resource Constraints in Data-Generating Devices**: The recent surge in devices capable of generating large amounts of data, such as those in IoT networks, presents a unique challenge. While these devices are potential goldmines for machine learning applications, their restricted computing power often limits their ability to process complex machine learning algorithms effectively.

Addressing these challenges requires innovative solutions that balance the capabilities of machine learning algorithms with the practical constraints of edge devices. The evolution of edge AI and machine learning continues to navigate these complexities, striving for efficient, real-time processing within the physical and economic realities of device production.[19]

V. RESEARCH DIRECTION

The Arduino Uno board is equipped with an 8-bit ATmega328P microcontroller, operating at a speed of 16 MHz. It includes 2 KB of SRAM and 32 KB of read-only flash memory. In comparison, the BBC Micro:Bit features a 32-bit ARM Cortex M0 microcontroller, also running at 16 MHz, but with 16 KB of SRAM and 256 KB of read-only flash memory.

To implement TinyML libraries on such embedded devices, consider the Arduino Due microcontroller as an example. The process involves setting up a Python environment on the Arduino. TensorFlow can be installed using the 'pip' command, followed by uploading training data. An example dataset could include countries and their

respective capitals. This data is then prepared, parsed, and transformed into a suitable format for training a fully connected neural network. The data is divided into three parts: 60% for training, 20% for validation, and 20% for testing. The test data is used to run the model and generate prediction plots.

After training, the model is converted to the TensorFlow Lite (TFLite) format. The TFLite model is then encoded into an Arduino header file, involving the creation of a constant byte array that contains the TFLite model. This step also requires importing the 'os' library. The final stage involves running the training process that has been established. This approach allows for the integration of machine learning capabilities into Arduino and similar microcontrollers, leveraging TinyML libraries for efficient processing in resource-constrained environments.

VI. CONCLUSION

The paper concludes by emphasizing the need for minimal cloud support in edge computing to ensure cost-effectiveness and accessibility. It underscores the importance of running neural network models on edge devices with low energy costs, opening up possibilities for new applications. The conclusion reiterates the potential of TinyML and TensorFlow Lite in revolutionizing the way we interact with and utilize machine learning in everyday devices[23][24].

Minimizing reliance on cloud support and the associated costs of installing local Edge servers is a priority, particularly when considering cost-effectiveness. A practical approach is to focus on machine learning (ML) in devices that are more accessible to the general public, such as Arduino, Vigil, VideoEdge, and DeepSense. These devices have the capability to process noisy sensor data effectively, especially notable in applications where a neural network model can operate under 1 mW of energy[25][26][27]. This low energy requirement opens up a plethora of new possibilities for these devices, making them highly efficient for various applications.

The TensorFlow Lite software framework, known for its stable API, is an excellent example of this approach. There's a growing need for standards in mobile inferencing, and ARM has already started addressing this need with its Compute Library, which includes OpenCL and Neon. However, the competition among chipmakers to develop optimal solutions for deep learning is intense and ongoing. As humans, our innate ability to create tools is evolving to new heights, with the development of advanced technologies. The challenge lies in staying true to the fundamental purpose of tools - to enable us to perform tasks more efficiently and effectively, aligning with our needs and aspirations[28][29][30][31][32][33].

REFERENCES

[1]A. Kusupati, "FastGRNN: A Fast, Accurate, Stable and Tiny Kilobyte Sized Gated Recurrent Neural Network", 32nd Conference on [1]NeuralInformation Processing Systems (NeurIPS 2018), 2018. [Accessed 20 May 2020]

[2]Al-Garadi, M., Mohamed, A., Al-Ali, A., Du, X. and Guizani, M., 2018. A Survey of Machine and Deep Learning Methods for Internet of Things (IoT) Security.

[3]A. Kumar, S. Goyal and M. Verma, "Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things", *Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.* [Accessed 25 May 2020]. (Abstract)

[4]"Why Machine Learning on The Edge?", Medium, 2020. [Online]. *Available: https://towardsdatascience.com/why-machine-learning-on-the-edge-92fac32105e6.* [Accessed: 25- May-2020].(INTRODUCTION)

[5]B. O'Donnell, "Machine Learning for Edge Devices", *Western Digital Corporate Blog, 2020. [Online]. Available: https://blog.westerndigital.com/machine-learning-edge-devices/. [Accessed: 25- May- 2020]*.(INTRODUCTION)

[6]M. Hollemans, "Machine learning on mobile: on the device or in the cloud?", *Machinethink.net, 2020. [Online]. Available: http://machinethink.net/blog/machine-learning-device-or-cloud/.* [Accessed: 25- May- 2020].(HISTORY)

[7] Chen, Jiasi & Ran, Xukan. (2019). Deep Learning With Edge Computing: *A Review. Proceedings of the IEEE. PP. 1-20. 10.1109/JPROC.2019.2921977.* (History)

[8]Smartindustry.com, 2020. [Online]. Available: https://www.smartindustry.com/articles/four-keys-to-machine-learning-on-the-edge/. [Accessed: 25- May- 2020].(IMPLEMENTATION OF ML LIBRARY).

[9]*"Engineering Tiny Machine Learning for the Edge - InformationWeek"*, InformationWeek, *2020.* [Online]. Available: https://www.informationweek.com/big-data/ai-machine-learning/engineering-tiny-machine-learning-for-the-edge/a/d-id/1336972. [Accessed: 25- May- 2020].(SUCCESS AND FAILURE OF ML)

[10]F. TX Zhuo and H. Collins, "Why TinyML is a giant opportunity", *VentureBeat, 2020. [Online]. Available: https://venturebeat.com/2020/01/11/why-tinyml-is-a-giant-opportunity/. [Accessed: 25- May- 2020]*.(SUCCESS AND FAILURE OF ML)

[11]B. Lorica, *"Compressed representations in the age of big data"*, O'Reilly Media, 2020. [Online]. Available: https://www.oreilly.com/content/compressed-representations-in-the-age-of-big-data/?imm_mid=0df7d4&cmp=em-data-na-na-newsltr_20160127. [Accessed: 25- May- 2020].((SUCCESS AND FAILURE OF ML)

[12]Murshed, M. G. Sarwar & Murphy, Christopher & Hou, Daqing & Khan, Nazar & Ananthanarayanan, Ganesh & Hussain, Faraz. (2019). *Machine Learning at the Network Edge: A Survey.* (Implementation of TinyML)

[13] C. Gupta et al., "ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices", *Proceedings of the 34 th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017., 2017. [Accessed 3 June 2020]*.

[14] Murshed, M. G. S., & others. (2019). Machine Learning at the Network Edge: A Survey. [Online]. Available: https://www.researchgate.net/publication/334866807_Machine_Learning_at_the_Network_Edge_A_Survey

[15] InformationWeek. (2020). Engineering Tiny Machine Learning for the Edge. [Online]. Available: https://www.informationweek.com/big-data/ai-machine-learning/engineering-tiny-machine-learning-for-the-edge/a/d-id/1336972

[16] VentureBeat. (2020). Why TinyML is a giant opportunity. [Online]. Available: https://venturebeat.com/2020/01/11/why-tinyml-is-a-giant-opportunity/

[17] O'Reilly. (2016). Compressed Representations in the Age of Big Data. [Online]. Available: https://www.oreilly.com/content/compressed-representations-in-the-age-of-big-data/

[18] Fritz AI. (2019). From AI-Enabled to AI-First: The Changing Mobile Landscape. [Online]. Available: https://heartbeat.fritz.ai/from-ai-enabled-to-ai-first-the-changing-mobile-landscape-308fb0e5adfc

[19] arXiv. (2019). [PDF] 1907.08349. [Online]. Available: https://arxiv.org/pdf/1907.08349

[20] Tryolabs. (2020). Machine Learning on Edge Devices: Benchmark Report. [Online]. Available: https://tryolabs.com/blog/machine-learning-on-edge-devices-benchmark-report/

[21] Stupid Projects. (2020). Machine Learning on Embedded - Part 1. [Online]. Available: https://www.stupid-projects.com/machine-learning-on-embedded-part-1/

[22] arXiv. (2019). [PDF] 1909.00560. [Online]. Available: https://arxiv.org/pdf/1909.00560

[23] Imagimob. (2020). The Past, Present, and Future of Edge Machine Learning. [Online]. Available: https://www.imagimob.com/blog/the-past-present-and-future-of-edge-machine-learning

[24] The Robot Report. (2020). Why and How to Run Machine Learning Algorithms on Edge Devices. [Online]. Available: https://www.therobotreport.com/why-and-how-to-run-machine-learning-algorithms-on-edge-devices/

[25] TensorFlow. (2020). TensorFlow Lite. [Online]. Available: https://www.tensorflow.org/lite

[26] Microsoft Research. (2020). Resource-Efficient ML for the Edge and Endpoint IoT Devices. [Online]. Available: https://www.microsoft.com/en-us/research/project/resource-efficient-ml-for-the-edge-and-endpoint-iot-devices/

[27] GitHub - Microsoft. (2020). EdgeML. [Online]. Available: https://github.com/Microsoft/EdgeML

[28] USENIX. (2018). HotEdge'18. [Online]. Available: https://www.usenix.org/system/files/conference/hotedge18/hotedge18-papers-talagala.pdf

[29] Towards Data Science. (2020). Why Machine Learning on the Edge. [Online]. Available: https://towardsdatascience.com/why-machine-learning-on-the-edge-92fac32105e6

[30] Western Digital. (2020). Machine Learning Edge Devices. [Online]. Available: https://blog.westerndigital.com/machine-learning-edge-devices/

[31] arXiv. (2018). [PDF] 1807.11023. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1807/1807.11023.pdf

[32] PMLR. (2017). Proceedings MLR Press - Kumar et al. [Online]. Available: http://proceedings.mlr.press/v70/kumar17a/kumar17a.pdf

[33] ResearchGate. (2019). Deep Learning with Edge Computing: A Review. [Online]. Available: https://www.researchgate.net/publication/334489669_Deep_Learning_With_Edge_Computing_A_Review