UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA

TinyML:
Progresso e applicazioni del Machine Learning su piattaforme Embedded

Relatore: Prof. Daniele Vogrig

Laureando: Nicolò Rasera

ANNO ACCADEMICO 2023 – 2024

Data di laurea 25/09/2024

Indice:

1 – Introduzione

L'apprendimento automatico sta guadagnando sempre più interesse anche nell'ambito dell'elettronica. I motivi principali risiedono nel fatto che le tecniche di *Machine Learning* (ML) richiedono significative quantità di energia e potenza di calcolo per raggiungere i livelli di accuratezza desiderati. Inoltre, la popolarità di questa branca dell'intelligenza artificiale sta portando gli algoritmi di apprendimento automatico anche su dispositivi economici, con vincoli di prestazione e di potenza.

Ciò ha motivato l'emergere del paradigma TinyML (*Tiny Machine Learning*), che mira a mantenere l'accuratezza dei modelli di apprendimento su piccoli dispositivi con risorse limitate, preaddestrando gli algoritmi e ottimizzando la capacità di elaborazione della macchina.

Un gioco di equilibri che potrebbe portare nuovo vigore alla microelettronica e segnare un nuovo capitolo nella storia dell'informazione.

Questo scritto si propone di presentare le definizioni e i concetti fondamentali del TinyML per comprenderne le ragioni della diffusione. Se ne descriveranno le procedure di lavoro, i progressi e i campi applicativi e se ne illustreranno le tendenze future.

1.1 - Machine Learning

Per comprendere il TinyML, cioè il *Machine Learning* sui piccoli (*tiny*) dispositivi [1], è necessario se non essenziale soffermarci sulle sue definizioni più importanti. In questo capitolo verrà spiegato cosa è il ML, la natura dei problemi che può risolvere e i concetti chiave, concludendo con i flussi di lavoro più popolari per approcciare i problemi più comuni.

Gli argomenti che verranno trattati in questo capitolo saranno poi ripresi lungo tutto lo scritto; ci si limita quindi, almeno in questa prima parte, a darne una definizione e una spiegazione sintetica.

1.1.1 - Cosa è il Machine Learning

L'apprendimento automatico (in inglese *machine learning*) è una branca dell'intelligenza artificiale, che si occupa di creare sistemi di apprendimento o di migliorare le prest

azioni di un algoritmo in base all'identificazione di schemi tra i dati, studianti mediante metodi statistici [2], [3].

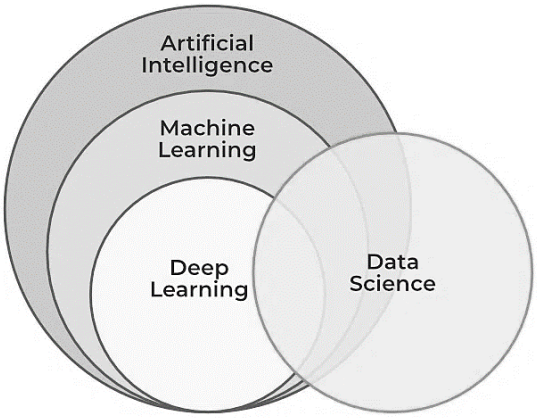

Figura 1

Nell'informatica, l'ML può essere vista come una variante alla programmazione tradizionale. Infatti, nel consueto processo di scrittura di un codice il programmatore progetta un algoritmo che applica predeterminate regole agli ingressi (*input*) ricevuti per generare delle uscite (*output*); tutte le operazioni interne sono organizzate e implementate dal programmatore mediante esplicite linee di codice [1]. Diversamente, nel ML, basterà inserisce una serie di dati all'interno di una particolare tipologia di algoritmo e lasciare che quest'ultimo scopra le regole che legano gli ingressi con le uscite, apprendendo dai dati in maniera autonoma. Questo significa che il programmatore non dovrà scrivere esplicite regole, ma solo il codice che serve alla macchina per costruire in modello ML [1], [3].

Considerando una programmazione tradizionale, facciamo l'esempio di voler prevedere e intervenire quando una macchina industriale sta per guastarsi. In questo caso il programmatore dovrebbe conoscere in quali situazioni i valori dei dati in ingresso misurati rappresentino un problema e scrivere un codice che li controlli di conseguenza.

È anche vero che questo approccio lavora bene per la maggior parte dei problemi più comuni. Se per esempio sappiamo che la macchina avrà problemi a temperature superiori ai 100°C, possiamo studiare ad esempio i dati relativi alla temperatura, per prevedere se raggiugerà e come raggiugerà temperature critiche.

In alcuni casi però è difficile conoscere con esattezza la combinazione di fattori che permettano di predire un particolare stato del sistema. Continuando l'esempio della macchina industriale, ci potrebbero essere differenti combinazioni di temperatura e vibrazioni indice di un problema, rendendo non così ovvia l'interpretazione dei dati.

Proprio in queste situazioni un algoritmo di ML potrebbe essere la soluzione. Il programmatore può creare un algoritmo che faccia predizioni basandosi su una serie di dati complessi, non avendo una completa conoscenza della complessità del sistema su cui sta lavorando.

L'algoritmo di *machine learning* crea un *modello* del sistema, basandosi sui dati forniti, attraverso un processo di *allenamento*. Inserendo quindi dei dati all'interno del modello quest'ultimo computa predizioni, in un processo chiamato *inferenza*[1] [1].

Soffermandoci sull'algoritmo ML, quest'ultimo può riconoscere schemi tra dati, prendere decisioni e adattarsi alle diverse situazioni. Caratteristiche molto utili in diversi contesti; infatti, il ML trova applicazione in campi come quello del *computer vision*, che permette di ricavare informazioni dalle immagini digitali, quindi nel riconoscimento di anomalie [4], nelle classificazioni di oggetti [es] e per utilizzi medici nell'individuazione di tumori [5]. Oppure nell'interpretazione dei dati, per il riconoscimento vocale [6] o nella previsione della tendenza del prezzo di un bene.

1.1.2 - Cosa è il Deep Learning e le ANN (artificial neural network)

Ci sono differenti approcci al ML, uno dei più popolari è il *Deep Learning* (DL), che si basa sull'idea semplificata del funzionamento del cervello umano. Nel DL, una rete di neuroni simulati (*Artificial Neural Network*, ANN) è allenata per modellare le relazioni che intercorrono tra vari ingressi e uscite [7].

Ogni ANN, quindi, contiene dei *nodi* ed è strutturato in cosiddetti *strati*. Ogni neurone è connesso ad ogni altro neurone dello strato successivo; perciò, il suo valore di uscita diventa un valore di ingresso per i nodi dello strato successivo.

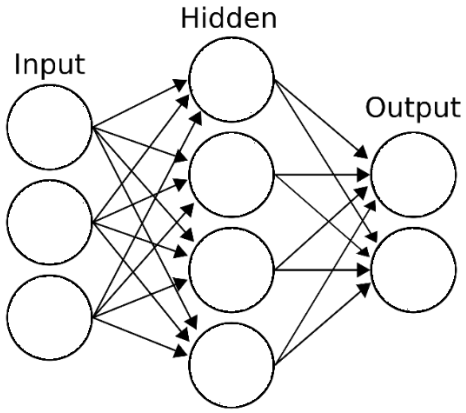

Figura 2 - Artificial neural network

Le connessioni tra i neuroni hanno un *peso*, modificato durante l'addestramento, che influenza il valore dell'ingresso che passa da un neurone all'altro. Questo comportamento segue la seguente formula: $input \times peso$.

[1] Nel Machine Learning, l'inferenza è il processo di generazione di stime del modello per i nuovi dati non usati nella fase di allenamento. Esistono diversi modi per generare stime nella previsione a causa della dipendenza temporale dei dati. Lo scenario più semplice è quando il periodo di inferenza segue immediatamente il periodo di allenamento e vengono generate stime all'orizzonte di previsione [30].

Una volta che il neurone riceve i valori di ingresso da tutti i neuroni dello strato precedente, viene aggiunto un *bias*, un valore costante che viene sommato. Anche quest'ultimo valore viene modificato durante la fase di allenamento della rete.

L'espressione risultante in uscita al neurone, quindi, risulta essere: $output = input \times peso + bias$.

Possiamo vedere i pesi e i bias come delle manopole di una radio (una radio con centinaia se non migliaia di manopole), che aggiustiamo per avere come uscita un suono il meno possibile disturbato.

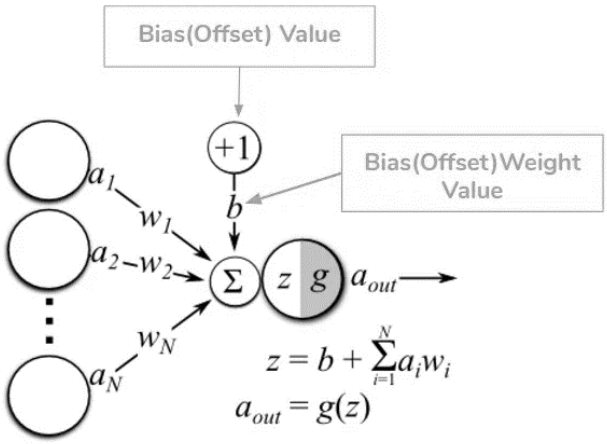

Figura 3 – Pesi e Bias

Sono presenti diverse tipologie di reti neurali, con architetture differenti, utilizzate per approcciare differenti problemi e scopi. Ad esempio, alcune sono molto buone per estrapolare dati da immagini, mentre altre performano per predire il prossimo valore in una sequenza.

Non è obiettivo di questa scritto elencarne tutte le tipologie, tantomeno spiegarne nel dettaglio il funzionamento, ma verranno citate quelle principali e più comuni:

- **FNN** (reti neurali *Feedforward*): sono la tipologia di rete neurale con l'architettura più semplice. Questa architettura viene chiamata così perché è composta da una serie di strati, nella quale la direzione dei dati è unica, dallo strato degli ingressi a quello dell'uscita [8], [9].

- **CNN** (reti neurali convolutive): sono una tipologia di rete neurale *feed-forward* usata per l'elaborazione di dati visuali e dati 2D. Sono quindi in grado di rilevare modelli all'interno di immagini, video e segnali audio [7].

- **RNN** (reti neurali ricorrenti): sono una tipologia di rete neurale il cui grafo è completamente interconnesso, contenente inoltre dei loop, il che gli consente una bidirezionalità del flusso del dato e di mantenere una forma di memoria. Questa struttura ad anello permette alle RNN di catturare le dipendenze all'interno di dati sequenziali, rendendole adatte a compiti come la modellazione del linguaggio, il riconoscimento vocale e l'elaborazione video [7], [9].

1.1.3 - Metodi di apprendimento

Le architetture viste precedentemente devono essere allenate in modo tale che il modello sia in grado di rispondere alle necessità del programmatore. Come già visto in 1.1.1, perché l'architettura possa apprendere le correlazioni tra ingressi e uscite, deve allenarsi con un insieme di dati strutturati in modo tale che ad un relativo valore di input vi sia anche il corrispettivo valore di output desiderato.

Ad oggi i principali metodi di apprendimento nel *Machine Learning* sono quello supervisionato, non supervisionato, semi-supervisionato e rinforzato.

- **Apprendimento supervisionato**: in questo paradigma l'algoritmo impara da dati di allenamento classificati. Cioè, da coppia input-output nel quale ad ogni ingresso dato viene anche suggerita l'uscita corrispondente. Questo processo consente la formazione di modelli robusti, al costo di richiedere un numero elevato di dati etichettati [9].

L'approccio è molto simile a quello che hanno usato i nostri genitori quando da piccoli abbiamo imparato i nomi dei colori. Il papà o la mamma (il programmatore) mostra al bambino

(algoritmo) un pastello di colore arancione (input) e gli dice che il colore di quel pastello si chiama “arancione” (output). Probabilmente il bambino non imparerà subito, con un solo esempio, che quel colore è l’arancione, saranno necessari più e più esempi, con oggetti differenti e in situazioni differenti.

❖ **Apprendimento non supervisionato**: questa tecnica si focalizza su insiemi di dati non classificati, sforzandosi quindi a scoprire schemi, strutture, o relazioni nei dati fornitogli. Basandosi su somiglianze o differenze, questo approccio è utile se vi è la necessità di organizzare un insieme di dati in base a caratteristiche comuni (*clustering*, *association*), o per individuare i valori anomali, cioè discordanti rispetto quanto è usuale, standard o previsto, rendendoli incoerenti con il resto dei dati (*anomaly detection*) [9].

❖ **Apprendimento semi-supervisionato**: in questo caso l’approccio è un ibrido tra l’apprendimento supervisionato e non supervisionato. Avendo a disposizione dati classificati e no, vengono utilizzati quelli classificati per allenare un modello che verrà poi utilizzato per classificare i dati rimanenti. Il dataset composta verrà utilizzato per allenare il modello finale che, teoricamente, dovrebbe performare meglio rispetto a modelli non supervisionati [8], [9].

❖ **Apprendimento per rinforzo**: questa tipologia di allenamento si presta per quei casi specifici nei quali non c’è una singola risposta corretta, ma si desidera un risultato complessivo. In questo processo l’algoritmo gradualmente impara e perfeziona il suo comportamento, rinforzandolo quando quest’ultimo è corretto, finché non acquisisce l’abilità di raggiungere risultati corretti.

Quest’ultimo metodo è considerato il più simile al processo di apprendimento umano, perché, come l’essere umano, impara sperimentando e prendendo decisioni tentativi ed errori, piuttosto che basandosi sui dati preesistenti [8], [9].

Come in una partita a un videogioco, non c’è una sola corretta sequenza di mosse che ci permette di superare il livello, ce ne sono svariate. Il processo che seguiamo è quello di provare più volte, imparando dalle scelte sbagliate e rinforzandoci dalle scelte corrette. Il tutto finché non concludiamo il livello con successo.

La scelta di un algoritmo di machine learning supervisionato o non supervisionato dipende in genere da fattori correlati alla struttura e al volume dei dati e al caso d'uso a cui si desidera applicarlo.

1.1.4 – Stime sui consumi energetici e potenze di calcolo nel Machine Learning

I consumi energetici delle architetture dei computer sono stati ampiamenti studianti per decenni. Mentre solo recentemente sta emergendo la necessità di adottarla come metrica nella valutazione dei modelli di Machine Learning. La maggioranza dei ricercatori, infatti, si focalizza principalmente nell'ottenere elevati livelli di accuratezza senza alcun vincolo computazionale.

La mancanza di valutazioni basate sul consumo energetico per gli algoritmi di ML può essere attribuita alla scarsità di strumenti adeguati a misurare i consumi nei modelli esistenti, data la complessità dell'argomento.

Ciononostante, i modelli stanno richiedendo sempre più memoria, dati, potenza e tempi di calcolo e la necessità di studiare e sviluppare strumenti in grado di valutare e migliorare i consumi energetici e quindi la potenza di calcolo dei modelli, sta diventando sempre più indispensabile [10].

In [10] gli autori confrontano diversi testi accademici e presentano alcune possibili approcci per la stima e il monitoraggio delle prestazioni del modello. Suggerendo per prima cosa una tassonomia dei consumi basata su due livelli: software e hardware.

La prima si interessa ai consumi dell'applicazione o dell'implementazione del software ed esplorare le tecnologie di ottimizzazione per la progettazione di algoritmi più efficienti.

La seconda invece si concentra sui consumi degli specifici componenti hardware, indicando quali sono fortemente correlati alla potenza e alle prestazioni del modello. Suggerendo poi un'analisi della potenza a livello funzionale (FLPA, [11]), soprattutto nel caso in cui si sia interessati a costruire chip specifici per il ML.

Vedremo nel terzo capitolo come anche all'approccio a progetti TinyML si possa incentrare sull'aspetto software oppure hardware, in analogia a quanto suggerito nelle righe precedenti.

Bassi consumi e l'ottimizzazione degli algoritmi sono obiettivi che dovrebbero essere prefissati in ogni progetto e in ogni applicazione, ma come vedremo più avanti, ci sono situazione dove queste considerazioni risultano essenziali.

Infatti, come una rete neurale diventa più grande (aumenta il numero di nodi, neuroni), aumenta la sua complessità. L'allenamento della rete richiederà più tempo, ma soprattutto sarà costoso dal punto di vista computazionale. Ad esempio, quando si implementano algoritmi ML basati su ANN in dispositivi wireless, e quindi con un elevato consumo dovuto alla trasmissione, è

fondamentale ridurre al minimo le risorse di calcolo e la potenza necessaria affinché questi algoritmi funzionino in modo efficiente.

Tradizionalmente la gestione di sistemi ML si basa sulla centralizzazione degli algoritmi in esecuzione su un server centrale e comunicando poi con tutti i nodi che richiedono l'esecuzione dell'algoritmo. L'aumento del numero dei nodi, però, aumenta il carico di elaborazione del server, portando così a ritardi significativi (alta latenza) e inefficienza nel processo decisionale [8].

1.2 – TinyML

Quando parliamo di intelligenza all'interno di sistemi *embedded* (sistemi integrati), intendiamo algoritmi di apprendimento che permettono al dispositivo una (anche se piccola) capacità decisionale basata sui dati che ha acquisito. Sfortunatamente, come è stato anticipato nel 1.1.4 e come analizzeremo più avanti, elaborare modelli di ML all'interno di dispositivi con una limitata capacità computazionale è complicato per vincoli di architettura, latenza ed energia [12].

Prima di analizzarne gli aspetti critici e le possibili soluzioni, è giusto darne una definizione chiara, spiegarne l'evoluzione e citare alcune applicazioni concrete. Questo è quello che verrà fatto in questo capitolo, con la difficoltà di discutere un paradigma recente e con quindi poche definizioni, le quali sono ancora oggetto di discussione all'interno della comunità TinyML [13].

1.2.1 – Cosa è il TinyML

Il TinyML è un paradigma che facilita l'esecuzione del *machine learning* sugli *edge device* con requisiti minimi di memoria e di calcolo; quindi, con un consumo di potenza previsto del sistema nei limiti di qualche milliwatt o meno [14].

Il tutto attraverso architetture, framework, tecniche, strumenti e approcci che siano in grado di eseguire analisi *on-device* per una varietà di modalità di rilevamento (ottico, audio, suono, movimento, chimico, fisico, testuale, cognitivo) a una potenza di mW o inferiore. Rivolgendosi prevalentemente a dispositivi *edge* a batteria con un'implementazione su larga scala, in ambiente IoT mediante sensori wireless[2] [15].

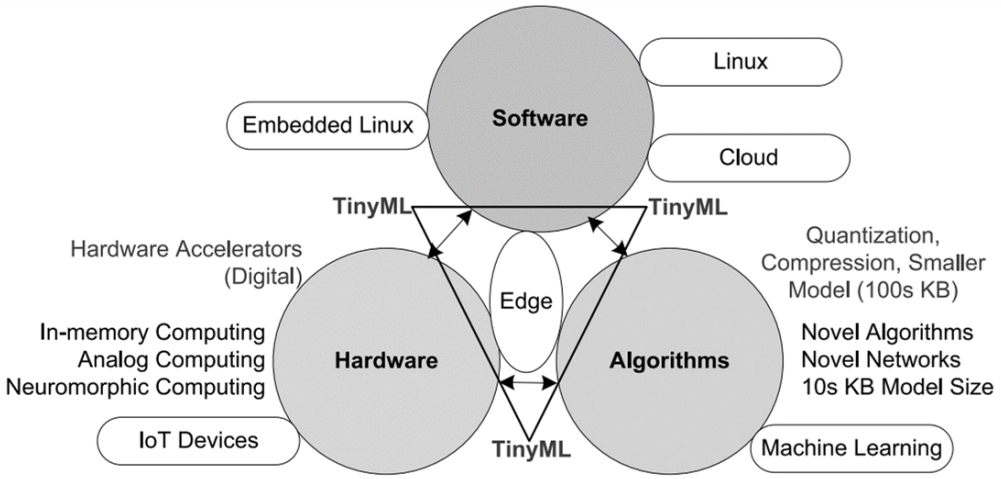

Figura 4 - TinyML secondo [15]

[2] Seguendo la propensione all'industria 4.0 dell'odierna automazione industriale. Avendo, nel concetto di "*Smart Factory*", nuove tecnologie come i dispositivi IoT, il "*Cognitive Computing*", l'"*Industrial internet*" e il "*Big Data Analytics*" insieme all'utilizzo del *Cloud*.

Triangolando i tre punti cardine del TinyML: hardware, software e algoritmo; i vincoli principali che ne ostacolano la crescita sono: (I) Energia: soprattutto nei casi di dispositivi IoT dove la comunicazione wireless è un aspetto non trascurabile per il calcolo dei consumi; (II) Capacità del processore: la maggior parte dei dispositivi "edge" ha una frequenza di clock di 10-1000 MHz, limitante per i modelli ML più complessi; (III) Memoria: i piccoli dispositivi "edge" hanno memorie flash con capacità di circa 1 MB e SRAM intorni ai 1000 KB, caratteristica che impedisce ai modelli di adattarsi al MCU della scheda; (IV) Costi: anche se il costo del singolo dispositivo è basso, una scala cumulativamente più grande[2] può comportare un costo complessivo troppo elevato per l'implementazione massicce [15].

Si accennano poi altri tre motivi, che verranno poi ripresi nel seguito dello scritto, per cui il progresso del TinyML si fa criptico:

1. I modelli di deep learning ottimizzati per dispositivi mobili, come *MobileNets* e *ShuffleNet*, non si adattano ai microcontrollori TinyML a causa delle risorse limitate di questi ultimi. Infatti, i modelli si concentrano sulla riduzione dei parametri e della latenza, ma non sull'uso della memoria.
2. Le limitate risorse di memoria (come 256KB di SRAM) dei dispositivi TinyML sono insufficienti per il training, rendendo necessario sviluppare nuovi algoritmi e schemi di *backpropagation* che riducano l'uso della memoria.
3. Le architetture neurali esistenti non sono adatte all'hardware con risorse limitate dei dispositivi *tiny*. È fondamentale progettare architetture neurali e *framework* di addestramento ottimizzati per funzionare efficientemente in termini di memoria e calcolo su dispositivi TinyML.

1.2.2 - Evoluzione

Spinto dai continui progressi tecnologici e dall'aumento dell'interesse verso il ML, il tinyML ha trovato negli ultimi anni un crescente interesse da parte non solo della comunità accademica mondiale, ma anche dalle industrie e aziende del settore [16].

Il paradigma nasce dall'esigenza di localizzare all'intero dei sistemi embedded i modelli di ML, annullando così la distanza di tra i sensori, dove vengono generati i dati, e i sistemi di elaborazione.

Nell'*Edge computing*, invece, la distribuzione avviene centralizzando il calcolo dei dati in tempo reale, provenienti da diversi dispositivi, su un unico centro di elaborazione posta ai bordi della rete (Karim Arabi, in un Keynote dell'IEEE DAC 2014 [17] e in un intervento al Seminario MTL del MIT nel 2015 [18]).

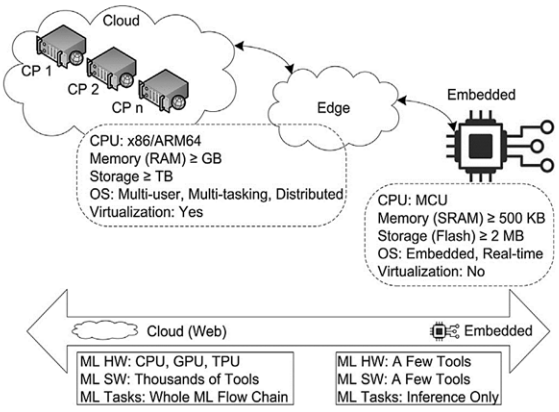

Figura 5 - Edge ML and Cloud ML [15]

Le differenze tra i due archetipi è evidente e porta il tinyML ad avere costi più contenuti e ad essere ideale per specifiche applicazioni descritte a 1.2.3 [15].

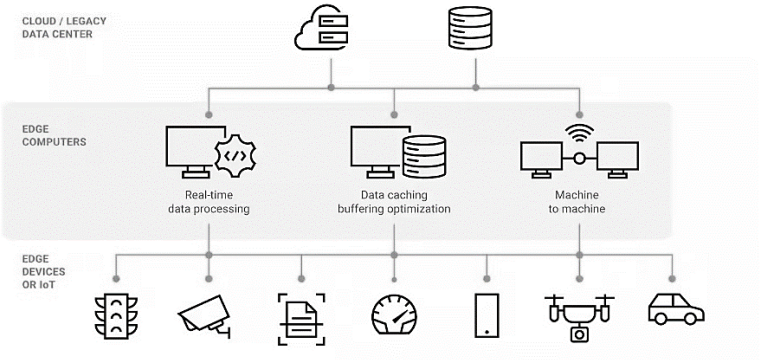

Figura 6 - Edge computing

Per descrivere l'evoluzione e il crescente interesse del tinyML, gli autori del [12] e del [19], analizzano l'evoluzione negli anni del numero di ricerche scientifiche, studi, conferenze pubblicate e altri tipi di documenti presenti nelle principali banche dati. L'analisi, portata avanti

in entrambi i casi con il metodo PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses), mette in evidenza l'aumento, di anno in anno, di pubblicazioni riguardanti il tinyML dal 2019.

Nonostante i numeri delle pubblicazioni siano contenuti, in entrambi gli articoli gli autori concludono che il tinyML è un campo in rapido sviluppo, che propone soluzioni innovative e nuove direzioni di studio.

Vengono di seguito riporti alcuni risultati della ricerca portata avanti dagli autori di [19]:

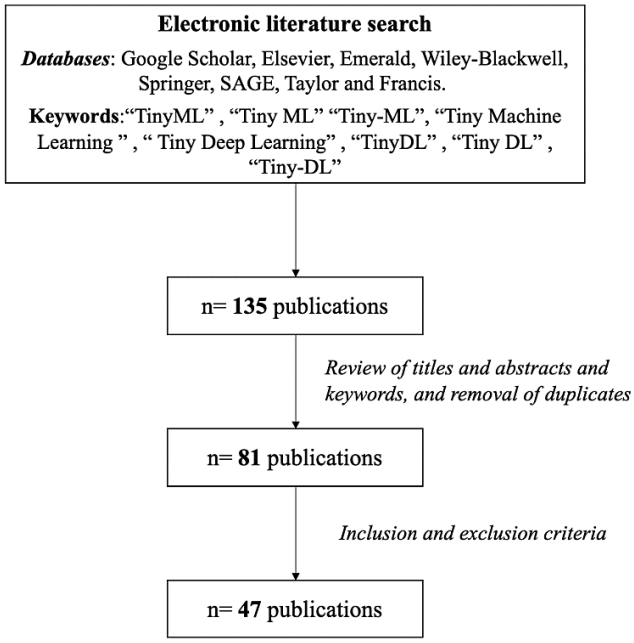

Figura 7 - PRISMA flow diagram

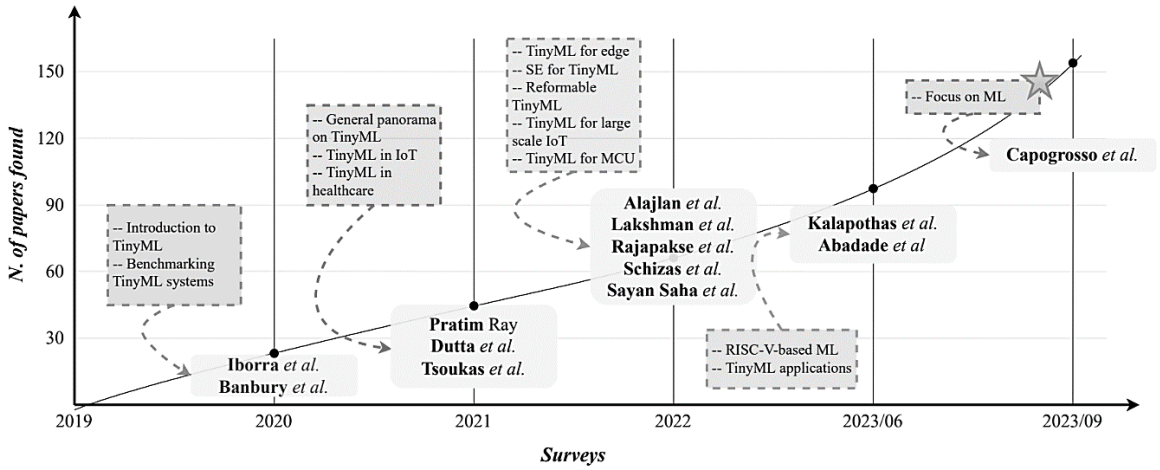

Figura 8 - Numero di pubblicazioni sul TinyML [12]

1.2.3 – Approcci

Nell'apprendimento automatico ci sono due principali sottodomini per l'implementazione dei modelli: *EdgeML* e il *CloudML*.

Mentre il *CloudML* si concentra sul miglioramento della capacità di elaborazione dei server cloud, l'*EdgeML* mira a migliorare l'efficienza energetica, la latenza e la privacy sui dispositivi "edge". Queste due implementazioni si incontrano in aree come gli aggiornamenti OTA[3] e l'apprendimento federato. In quest'ultima tecnica, l'architettura della rete è organizzata in modo tale che la distribuzione dei dati sia decentralizzata sui dispositivi presenti nel campo. Il principio generale è quello di allenare modelli locali su dati locali e scambiare parametri (e.g. pesi e bias per il DNN) tra i dispositivi per creare un modello globale condiviso con tutto il campo.

Com'è stato analizzato precedentemente, negli ultimi anni l'*EdgeML* ha integrato al suo interno dispositivi e applicativi a bassissimo consumo energetico. Nei quali oltre all'inferenza, si sta cercando di abilitare l'addestramento *on-device* su dispostivi IoT, così affinando il modello su stessi piuttosto che trasmettere i dati ai server cloud, migliorando così la privacy e la sicurezza [16].

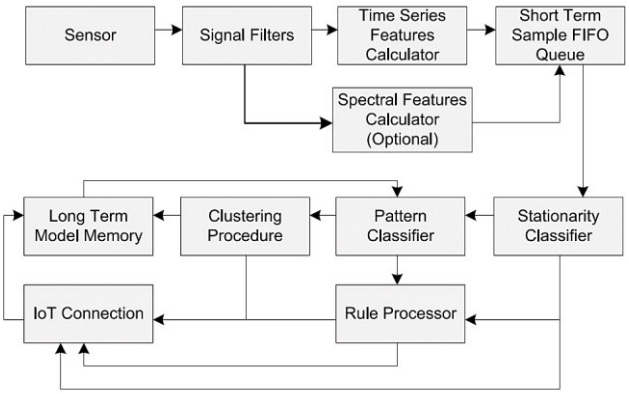

Figura 9 - Edge pipeline per il TinyML [15]

La *pipeline edge standard* per tinyML, illustrata nella Fig.9, ha come prima fase l'acquisizione da parte dei sensori dei dati grezzi. I dati vengono poi filtrati ed elaborati per estrarne le caratteristiche di interesse e conservati in una struttura FIFO. Un classificatore di stazionarietà verifica la stabilità dei dati se sono in formato di serie temporale. Successivamente, vengono inviati a un classificatore di pattern per l'elaborazione basata su predeterminate regole o clustering, a seconda delle esigenze dell'applicazione.

[3]Con l'acronimo Over-the-air (OTA) s'intende una tipologia di scambio dati che consente l'aggiornamento del software di un dispositivo digitale tramite una comunicazione punto-punto e per mezzo di una rete wireless (senza cavo).

1.2.4 – Applicazioni

Le aree per cui il tinyML trova applicazioni, sono molteplici. Spaziano dalla medicina, alla sicurezza e alla sorveglianza, fino al controllo e al monitoraggio industriale [20]. Ma, come detto precedentemente, i campi applicativi sono quelli nei quali vi è la necessità di eliminare la distanza tra l'acquisizione dei dati e la loro elaborazione mediante modelli ML.

Infatti, grazie a quest'ultimi, è possibile applicare a dispositivi *edge* il riconoscimento d'immagine, il riconoscimento vocale e l'elaborazione dati per la classificazione.

Nella cura della persona vi è l'esempio di orologi *smart* capaci di misurare segnali vitali come il battito cardiaco e l'ossigenazione del sangue per fornire in tempo reale un'accurata rappresentazione del proprio stato di salute. Oppure di telecamere atte a monitorare il paziente e notificare comportamenti che richiedano l'assistenza del personale medico.

Telecamere per la sicurezza e la sorveglianza che permettono, mediante l'analisi video, di segnalare la presenza di persone o animali. Oppure perla guida di veicoli, dove le decisioni in tempo reale sono cruciali, evitando ostacoli e seguendo percorsi.

Nello IoT (*Internet of Things*) sono famosi gli esempi di moduli per il riconoscimento della voce e per la traduzione in tempo reale.

In ambito industriale invece viene citato il rilevamento di anomalie *(anomaly detection*) per l'analisi dei pezzi assemblati da una linea produttiva. Oltre alla gestione energetica e alla diagnosi in tempo reale dei macchinari.

Ovviamente le aree di applicazione sopramenzionate sono solo una frazione dello spettro del tinyML e giorno dopo giorno, soprattutto nell'ambito industriale, si generano nuove opportunità per la ricerca e sviluppo di applicazione sempre più performanti.

1.3 – Sistemi embedded e edge device

Sono stati ciati nei capitoli precedenti i sistemi *embedded* e gli *edge device*, ma è importante ora darne una defezione e spiegarne il funzionamento, prima di addentrarsi in argomenti più tecnici del tinyML.

Quest'ultimi, infatti, rappresentano un punto nevralgico della discussione che viene portata avanti con il TinyML; il paradigma, si ricorda, nasce dalla necessità di adattare modelli ML su dispositivi che non riuscirebbero, senza un processo di ottimizzazione iniziale, a supportare algoritmi così complessi. É quindi fondamentale conoscere la macchina e la sua architettura, prima di sviluppare ulteriormente il processo di integrazione con i modelli ML.

1.3.1 – Microprocessore

Un microprocessore (μP) è un circuito integrato (IC) realizzato su un unico *chip*, che contiene tutti i principali elementi funzionali tipici della CPU di un elaboratore: circuiti aritmetici, logici e di controllo.

L'organizzazione usata tipicamente è quella di von Neumann, soprattutto nei processori *general purpose*.

Come μP possiamo trovare i DSP (*Digital Signal Processor*) come integrati ottimizzati per lo svolgimento efficiente di funzioni di elaborazione numerica del segnale, specialmente per quelli svolti in tempo reale (*real-time*).

1.3.2 – Microcontrollori

Un microcontrollore (μC o MCU) è un circuito integrato (IC) digitale che, nella forma più ridotta, contiene in un unico chip un microprocessore (CPU), una memoria di programma di tipo non volatile (ROM), una memoria dati volatile (RAM), una unità di ingresso-uscita (I/O) e almeno un timer.

Un SoC (*system-on-a-chip*) include un microcontrollore, integrandolo con periferiche più avanzate come unità di elaborazione grafica (GPU), moduli Wi-Fi o coprocessori per l'accelerazione hardware.

L'organizzazione usata tipicamente è quella Harvard, che separa la memoria del programma dallo spazio riservato ai dati.

1.3.3 – Sistemi embedded

Con questo termine indentifichiamo generalmente tutti quei sistemi di elaborazione a microcontrollore progettati per un determinato utilizzo (specializzati) e nei quali il μC è incorporato in schede elettroniche ad hoc, progettate per la massima autosufficienza funzionale.

1.3.4 – Edge Device

I dispositivi *edge* sono dispositivi informatici posti vicino ai bordi della rete, solitamente in prossimità ai sensori posti in campo. La loro capacità di elaborazione dati locale, inviati poi a un server centrale, riduce significativamente la latenza e il tempo di risposta. Prima di inviare i dati ai server centrali o ad altri dispositivi, questi dispositivi li aggregano, filtrano e analizzano accuratamente. Variano da sensori semplici a sistemi industriali complessi, inclusi scanner, smartphone, dispositivi medici, strumenti scientifici, veicoli autonomi e macchine automatizzate.

1.3.5 – FPGA

Gli FPGA (*Field Programmable Gate Array*) sono dispositivi logici programmabili, le cui funzioni logiche di elaborazione sono appositamente programmabili e modificabili tramite opportuni linguaggi di descrizione hardware.

Si tratta infatti di dispositivi standard la cui funzionalità da implementare non viene impostata dal produttore, il quale può quindi produrre su larga scala a basso prezzo. La loro genericità li rende adatti a un gran numero di applicazioni in ambito consumer, comunicazioni, automotive eccetera. Essi sono programmati direttamente dall'utente finale, consentendo la diminuzione dei tempi di progettazione, di verifica mediante simulazioni e di prova sul campo dell'applicazione [21].

2 – Progetto TinyML

Discutere il flusso di lavoro e gli strumenti utilizzati allo stato dell'arte per la progettazione e lo sviluppo di un progetto tinyML è un passaggio obbligatorio. Infatti, permette una comprensione più profonda del paradigma e dei problemi e delle difficoltà analizzate nei capitoli precedenti.

Verrà analizzato come ci possano essere differenti approcci e mezzi per la realizzazione di un sistema tinyML e che è fondamentale analizzare e classificare nel suo insieme il problema che si vuole affrontare, così da definire in modo chiaro gli obiettivi da raggiungere.

Nei prossimi capitoli verrà quindi analizzato come strutturare e sviluppare un progetto tinyML, l'*hardware* e i *software* che vengono maggiormente usati e l'approccio ad oggi utilizzato per ottimizzare o comprimere i modelli di ML.

2.1 - Organizzazione e sviluppo di un progetto TinyML

Organizzare un progetto TinyML significa prima di tutto conoscere il problema che si sta cercando di risolvere e analizzare l'ambiente nel quale il sistema sviluppato dovrà operare. Come verrà discusso in un capitolo dedicato, non sempre l'approccio TinyML è la soluzione; sono state discusse altre tipologie di architetture e alcune di queste, nonostante richiedano maggiori risorse energetiche e computazionali, ai fini pratici si presentano, ad oggi, come le soluzioni più efficienti.

Un progetto, come quello che verrà analizzato in questo capitolo, si presta per tutti quei casi in cui non è possibile avere potenti elaboratori, per motivi di spazio, consumi o costi.

2.1.1 – DeepL Workflow

Ricollegandoci a quanto discusso nel capitolo 1.1, il processo che permette lo sviluppo di un modello DL comporta i seguenti passaggi:

1. *Definire gli obiettivi*

Quando si progetta un qualsiasi tipo di algoritmo, è fondamentale definire fin da subito cosa si vuole che faccia. Definire quindi cosa si vuole predire, così da avere un'idea sulla tipologia di dati da raccogliere e quindi l'architettura da utilizzare.

Per esempio, in un problema di classificazione, il nostro modello dovrà restituire, in base agli input dati, la probabilità che questi facciano parte di una classe piuttosto di un'altra. Ciò significa che l'obiettivo sarà quello di avere un modello che riesca a riconoscere il dato in ingresso e per fare questo sarà necessario allenarlo con un insieme di dati che rappresentino le classi di interesse.

2. *Raccogliere dati e creare un Dataset*

Definiti gli obiettivi sarà poi necessario identificare le tipologie di dati necessari per allenare il modello; infatti, il miglior modo per addestrarlo è quello di individuare e utilizzare le informazioni più rilevanti per risolvere il problema, eventualmente aiutandosi con tecniche statistiche.

È importante che la stessa tipologia di dati scelti per esercitare il modello siano disponibili quando sarà necessario effettuare una predizione. Per esempio, se utilizziamo dati relativi alla temperatura di un macchinario, sarà poi necessario prelevare gli input dalla stessa posizione fisica, altrimenti non è detto che il modello di comporterà allo stesso modo.

Una volta capito di che dati si necessità, bisognerà raccoglierne il più possibile. L'obiettivo è quello di collezionare dati che rappresentano l'intera gamma di condizioni ed eventi che possono verificarsi nel sistema. Questa varietà aiuterà il modello a rappresentare ogni possibile scenario.

Oltre che a raccogliere i dati, sarò necessario, in alcuni casi, classificarli (apprendimento supervisionato), etichettandoli in base alle classi presenti.

Il prodotto finale che si otterrà sarà un *dataset*, un insieme organizzato di dati, in modo tale che possano essere facilmente analizzati e utilizzati.

3. *Progettare l'architettura del modello*

Come è stato analizzato nel capitolo 1.1.2, ci sono diverse architetture per le ANN, progettate per risolvere un'ampia gamma di problemi. Ricordando che i dati sono molto legati all'architettura, per i casi più comuni, si possono trovare modelli pre-allenati online, altrimenti sarà necessario elaborare da zero la propria rete neurale.

Questo punto verrà elaborato meglio nei capitoli successivi, visto che i vincoli dei dispositivi, dove il modello dovrà verrà eseguito, sono un punto cardine del tinyML. Verrà quindi analizzato come gli approcci possano essere differenti, riducendo il numero di neuroni (caratteristica che determina il peso e la dimensione del modello), oppure utilizzando degli acceleratori hardware per velocizzare l'esecuzione di quest'ultimo.

4. *Allenare il modello*

Attraverso la scelta di obiettivi concreti, la raccolta e la classificazione di dati pertinenti e la scelta della giusta architettura, si riesce a definire il modello di ML che può fare al caso del

programmatore. Questo è processo iterativo, richiederà di andare avanti e indietro attraverso gli step del flusso di lavoro, finché non si raggiungerà un modello funzionante.

L'allenamento è il passaggio nel quale il modello impara a produrre l'output corretto per un insieme di *input*. Quando un dato viene inserito all'interno di una ANN, questo lo trasforma in una successione di operazioni matematiche che coinvolgono *pesi* e *bias* di ogni *strato*. Durante l'addestramento, un algoritmo chiamato *backpropagation* aggiusta i *pesi* e i *bias* incrementandoli fin tanto che l'output del modello non si avvicina a quello desiderato.

I due motivi più comuni per cui un modello fallisce sono per "*underfitting*" e "*overfitting*". Nel primo caso si ha una situazione nella quale il modello non ha una conoscenza abbastanza profonda per effettuare delle buone predizioni; nel secondo caso invece il modello è troppo specifico rispetto ai dati fornitogli nella fase di addestramento e quindi non è capace di generalizzare il suo comportamento su *input* che non siano quelli dati nella fase di allenamento. In una situazione l'apprendimento non è sufficiente e nell'altra invece è troppo dettagliato.

5. *Validazione del modello*

Per valutare le prestazioni del modello si utilizzano nuovi dati che non sono stati utilizzati nella fase di addestramento. Comunemente di divide il dataset iniziale in tre parti: allenamento (60%), convalida (20%) e test (20%). I dati di convalida servono per verificare se durante la fase di *training*, il modello si sta adattando eccessivamente (*overfitting*).

Una volta completato il processo di allenamento, eventualmente affinato in base ai risultati ottenuti dai dati di convalida, si procede con la verifica sui dati di test per valutare le prestazioni e l'accuratezza del modello [1].

2.1.2 – TinyML Workflow

Dopo aver costruito un modello ML si cerca di adattarlo a un sistema embedded con le limitazioni spiegate nel capitolo 1.2. I due ingredienti intriseci del tinyML sono il modello ML e la piattaforma hardware; quindi, gli approcci tradizionali per la progettazione e lo sviluppo di un progetto si orientano verso questi due elementi.

- **ML-oriented:** Il flusso di lavoro che si concentra sul modello ML si basa sulla progettazione, sull'adattamento, sull'allenamento e sulla sua valutazione; mentre la scelta della piattaforma hardware è decisa a priori o limitata.

In [1], il modello costruito e addestrato viene successivamente convertito per l'utilizzo all'interno di un microcontrollore. A tal fine, si ricorre all'uso di un interprete e di uno strumento

specifico per l'installazione. Nel capitolo 2.3 verranno presentati e analizzati alcuni di questi strumenti.

Altri approcci invece di basano sui seguenti passaggi:

- **Progettazione del modello:** Il modello viene progettato e allenato in modo tale che adempi nel miglior modo possibile alle specifiche del problema, trascurando in un primo momento la piattaforma hardware.
- **Ottimizzazione del modello:** Scegliendo tra differenti strategie, il modello viene ottimizzato compromettendo le prestazioni a favore della sua efficienza.
- **Valutazione on-host:** Il modello ottimizzato viene valutato rispetto ai parametri prestazionali richiesti nelle specifiche e, se riscontrato carente, viene riprogettato.
- **Target deployment:** Vengono applicati al modello degli ottimizzatori specializzati per migliorare efficienza dell'inferenza, andando a sfruttare delle specifiche del dispositivo hardware utilizzato.
- **Valutazione:** L'ultimo passaggio è quello di valutare le prestazioni nel complesso il sistema.

❖ **HW-oriented:** Il flusso di lavoro che si concentra sul dispositivo cerca di migliorare la piattaforma hardware in modo tale che il modello possa essere eseguito nella sua versione originale. Questo spesso significa che è da prima necessaria una fase investigativa, nella quale si cerca di capire dove possono esserci i "colli di bottiglia" nell'architettura utilizzata, per poi proseguire verso la progettazione di moduli di accelerazione hardware per migliorare i rendimenti e i consumi complessivi; oppure nella progettazione di nuove piattaforme hardware ottimizzate per applicazioni embedded e modelli ML specifici.

I passaggi tipici sono i seguenti:

- **Progettazione hardware:** Si progetta il design di un'architettura, o di un modulo di accelerazione al suo interno, che migliora le prestazioni per una determinata classe di problemi di calcolo.
- **Target deployment:** Valutazione delle prestazioni dell'hardware ottimizzato in un ambiente simulato. In caso di risultati insoddisfacenti, si ritorna alla fase di progettazione.
- **Valutazione:** Produzione e valutazione dei dispositivi hardware fisici.

2.2 - MCU e SoC per progetti TinyML

Il successo di un progetto tinyML dipende in gran parte dalle capacità del sistema di riuscire, con le ridotte capacità di calcolo e memoria, a elaborare in tempi brevi un output ai segnali in ingresso ricevuti.

La scelta dei componenti hardware, come visto nel capitolo precedente, si rifà alla tipologia di progettazione decisa e quindi alle modalità con cui organizzare il progetto (SW-oriented o HW oriented).

In tal senso, questo capitolo analizzerà componenti hardware in commercio e quindi sistemi adattati o già progettati per poter soddisfare le necessità di elaborazione dei principali modelli di ML.

2.2.1 – L'hardware

Dal punto di vista della tecnologia hardware, possiamo distinguere, come soluzioni alle esigenze di progetti tinyML, sistemi basati su *Application specific integrated circuit* (ASIC), (MCU) e "Field Programmable Gate Array" (FPGA).

La scelta può essere veicolata da diversi fattori, tra i tanti ci sono l'ambiente nel quale il sistema deve operare e i costi da sostenere. Nel bilanciare tutti i requisiti necessari per una determinata applicazione embedded, ci sono alcuni fattori da considerare nella scelta di un processore per eseguire l'inferenza dei modelli ML:

1. Considerare l'intera applicazione: quindi comprendere da subito l'ambito dell'intero progetto
2. Scegliere il giusto livello di prestazioni: una volta compreso l'ambito dell'intera applicazione, bisognerà prevedere la potenza di elaborazione necessaria e il livello di accuratezza richiesto dalle specifiche di progetto
3. Garantire la facilità d'uso: la facilità d'uso si riferisce sia alla facilità di sviluppo che alla facilità di valutazione. Alcune soluzioni offrono strumenti che, data una topologia di rete, mostrano le prestazioni e la precisione ottenibili su un determinato processore, consentendo così una valutazione delle prestazioni senza la necessità di hardware effettivo o della finalizzazione di una rete. [22]

Come è stato ampiamento analizzato nei capitoli precedenti, con l'avvento dell'intelligenza artificiale, molte aziende (Nvidia, Intel, AMD, ARM, TSMC…) hanno cominciato ad affrontare, se pur in modo differente, la progettazione e la produzione di chip e architetture ottimizzate per l'allenamento e la gestione di reti neurali e modelli di ML. Anche a livello

didattico società come Arduino, propongono kit per lo studio del Tiny Machine Learning, con un microcontrollore nRF52840 di fabbricazione Nordic Semiconductor [23].

Vi sono moli esempi di ricercatori e prodotti commerciali che hanno adottato l'architettura RISC-V, ossia la quinta generazione RISC, per la progettazione, non banale, di architetture, configurazioni e framework software per il *machine learning*. Per facilitare quindi il processo, vi sono molti framework *open-source* per la gestione di SoC nelle applicazioni di ML [12], [24].

Un esempio di full-stack open-source framework è il "CFU Playground", che consente la progettazione rapida e iterativa di acceleratori ML per sistemi embedded [25]. Gli autori del framework si concentrano su piattaforme FPGA per la possibilità di personalizzare il processore per adattarlo a eseguire in modo efficiente i calcoli dell'applicazione (HW-oriented) e integrare un'unità di accelerazione nella pipeline della CPU. Inoltre, le piattaforme basate su FPGA offrono molteplici vantaggi, non solo per l'efficienza computazionale, ma anche perché, se il sistema originale non fosse più sufficiente o disponibile, con un lavoro minimo si potrebbe passare a un altro dispositivo; infatti, il codice VHDL o Verilog può essere distribuito su qualsiasi FPGA con risorse sufficienti.

Sebbene ci siano molti vantaggi nel tinyML, l'eterogeneità dell'hardware dei MCU e le risorse limitate disponibili su di essi rappresentano una la vera sfida. Tipicamente gli MCU hanno dai dieci alle poche centinaia di KB di SRAM e uno o due MB di memori Flash. Ciò limita in modo importante le dimensioni dei modelli ML che il dispositivo può permettersi. Per quanto questi dispositivi si siano dimostrati capaci di svolgere modelli ML base come il *Keyword Spotting*, *Visual Wake Words* e *l'Anomaly Detection*, per supportare applicazioni più avanzate, mantenendo un basso consumo energetico, è necessario dell'hardware specializzato [25].

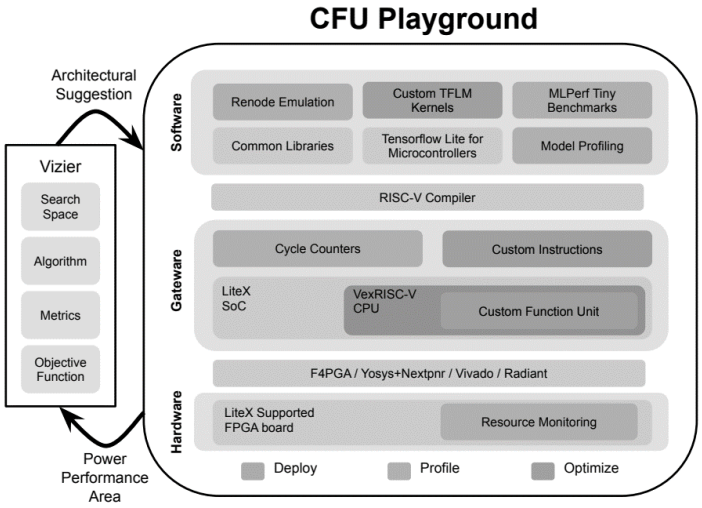

Figura 10 – CFU Playground [25]

L’architettura degli FPGA, invece, è composta da blocchi logici configurabili (CLB), che possono eseguire varie funzioni digitali e operazioni matematiche complesse tipiche degli algoritmi di ML. Un FPGA progettato per il ML spesso incorpora blocchi hardware specializzati: elaboratori per segnali digitali (DSP) e blocchi di memoria ad alte prestazioni. Migliorando così la velocità di esecuzione delle operazioni matematiche complesse e l’accesso ai dataset.

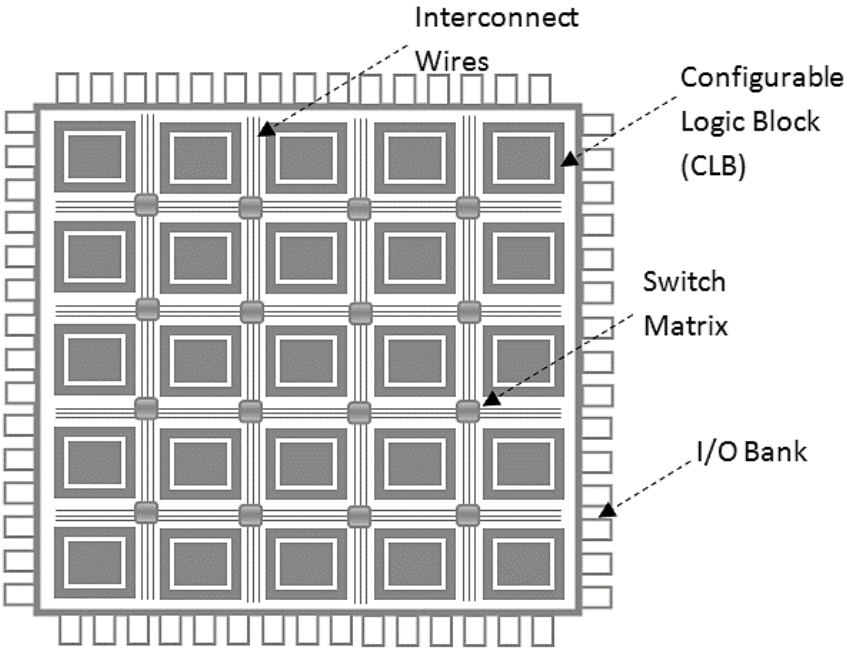

Figura 11 – Architettura FPGA

2.2.2 – Sistemi a confronto

Il continuo sviluppo della materia rende complesso l'elenco delle principali piattaforme utilizzate per progetti tinyML; l'innovazione e il progresso tecnologico si sono sviluppati di anno in anno, proponendo hardware sempre più performanti dal punto di vista dei consumi in relazione alla potenza di calcolo (TOPs/W: *Tera Operations Per Second* su Watt) [Figura 12].

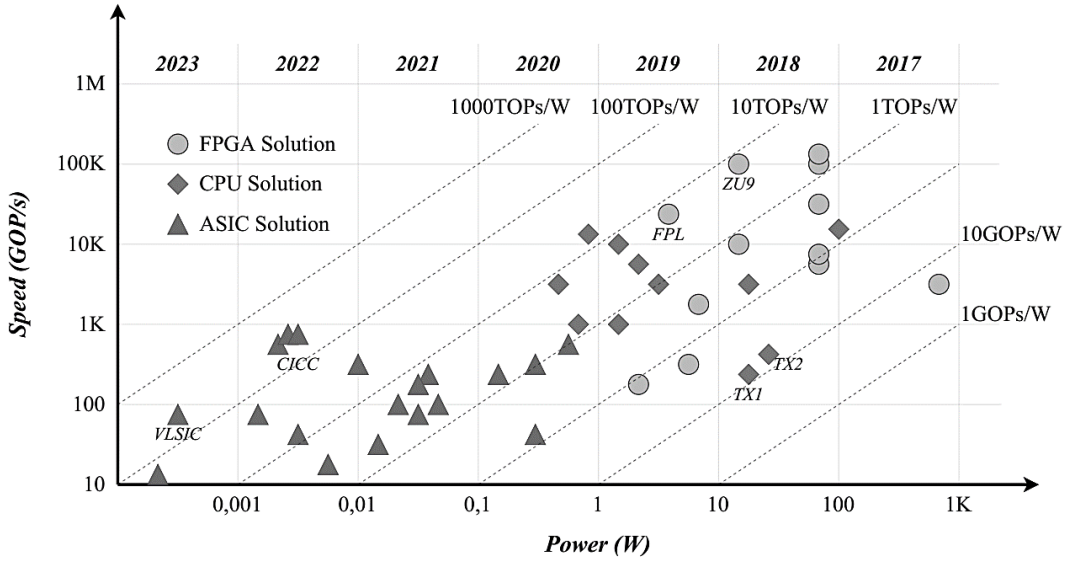

Figura 12 – Sviluppo hardware per il TinyML [12]

I microcontrollori, in particolare la famiglia ARM Cortex-M, rappresentano una piattaforma molto diffusa per l'implementazione di modelli ML [26]. Inoltre, i requisiti bassi di potenza, permettono l'alimentazione tramite batterie. Tuttavia, nonostante i loro vantaggi, i microcontrollori includono operazioni e controlli logici, che potrebbero limitare le prestazioni computazionali.

Invece, come spiegato precedentemente, gli FPGA possono offrire prestazioni, efficienza e scalabilità, soddisfacendo i complessi requisiti computazionali dei modelli ML. Da notare però che un importante svantaggio è la complessità d'uso di questi componenti e il limitato numero di librerie disponibili.

Di seguito viene proposta una tabella di confronto tra piattaforme hardware per progetti TinyML [Tabella 1].

Tabella 1 – Elenco delle principali piattoforme per progetti TinyML

Hardware	Processor	CPU clock	Flash	SRAM	Power	Produttore
Apollo3	32bit ARM Cortex-M4F	48 MHz 96 MHz	1 MB	384 KB	3-5 V	SparkFun
STM32F Discovery	32bit ARM Cortex-M4	48 MHz	1 MB	192 KB	3-5 V	STMicroelectronics
Arduino Nano 33 BLE Sense	nRF52840	64 MHz	1 MB	256 KB	3.3 V	Arduino
Himax EW-I Plus	32bit ARC EM9D	400 MHz	2 MB	2 MB	1.2–3.3 V	SparkFun
Thunderboard Sense 2	EFR32 Mighty Gecko	38.4 MHz	1 KB	256 KB	3.3–5 V	Silicon Labs
Sony's Spresense	ARM Cortex-M4F	156 MHz	8 MB	1.5 MB	3.3–5 V	Sony
Arduino Portenta H7	ARM Cortex-M7	480 MHz	16 MB	8 MB	3.7–5 V	Arduino
Raspberry Pi 4B	64bit ARM Cortex-A72	1.5 GHz	-	256 KB	3.3–5 V	Raspberry Pi
Nicla Sense ME	ARM Cortex-M4	64 MHz	512 KB	64 KB	3.7 V	Arduino
GAP9	RISC-V	400 MHz, 150.8GOPs	1.5 MB	128 KB	1.8–3.3 V, 0.33mW/GOP	Green Wave Technologies
Nordic Semi nRF52840 DK	ARM Cortex-M4	64 MHz	192 KB	24 KB	1.7–5 V	Nordic
Arty A7-100T	Xilinx MicroBlaze	450 MHz	16 MB	607.5 KB	7-15V	Digilent
OrangeCrab	Lattice ECP5-85F	48 MHz	128 Mb	194 Kb	1.35 V	Lattice Semiconductor
ULX3S (12F)	Lattice LFE5U-12F	25 MHz	16 MB	32 MB	3.3 V	Radiona
Fomu	Lattice ICE40UP5K	48 MHz	1MB	128 KB	5 V	Tomu project

2.2.3 – Acceleratori hardware (NPU – Neural Processing Unit)

Gli acceleratori hardware non sono nuovi nel mondo dell'informatica e dell'elettronica, essi infatti sono comunemente utilizzati per migliorare le prestazioni dei PC in settori specifici (operazioni in virgola mobile o elaborazioni grafiche), per permettere di alleggerire la CPU dalle attività che richiedono eccessive risorse computazionali.

Nel ML gli NPU sono acceleratori hardware specializzati per accelerare le applicazioni di intelligenza artificiale. Molti esempi sono tutt'ora utilizzati su dispositivi mobile come gli iPhone e Huawei o nei laptop AMD.

- *Graphics Processing Unit* (GPU):

Originariamente progettate per accelerare l'elaborazione grafica, le GPU differiscono dalle CPU per la loro composizione. Le GPU sono composte da alcune migliaia di ALU che permettono l'esecuzione parallela di molte operazioni. L'architettura parallela rende le GPU particolarmente adatte all'inferenza dei modelli ML e all'allenamento delle reti neurali, dove spesso vengono coinvolte operazioni su matrici e vettori.

Negli ultimi anni alcune aziende come Nvidia hanno commercializzato prodotti (Jetson Nano) specificamente progettati per progetti tinyML, nei quali l'architettura del processore è molto simile a quella delle più moderne GPU. Inoltre, le GPU sono progettate specificamente per gestire in modo efficiente grandi dataset e facilitare il rapido trasferimento dei dati tra la memoria principale del sistema e le unità di elaborazione, un aspetto significativo poiché il ML opera tipicamente su ampi dati in tempo reale.

- *Tensor Processing Unit* (TPU)

La TPU è un processore, sviluppato da Google, specificatamente progettato per compiti di ML. Google ha rilasciato le Edge TPU utilizzando la piattaforma Coral in vari formati, che vanno da una scheda di sviluppo simile al Raspberry Pi a moduli indipendenti saldabili.

Rispetto alla GPU, le TPU sono progettate per un elevato volume di calcoli a bassa precisone ed efficienti nella gestione delle operazioni tensoriali[4]. Le TPU sono particolarmente adatte per le reti neurali convoluzioni (CNN), mentre GPU e CPU possono avere vantaggi per le reti neurali ricorrenti (RNN) [12]. Altri esempi possono essere i chip Trainium e Inferentia di Amazon Web Service.

[4] Una TPU, infatti, ha al suo interno unità moltiplicative delle matrici e unità di elaborazione vettoriale; oltre che ad avere memorie ad elevata larghezza di banda per il rapido spostamento di dati del modello dentro e fuori le unità di elaborazione.

❖ *Vision processing unit* (VPU) e *Neural network processors* (NNP)

Per completezza si citano le VPU, che si differenziano dalle GPU per la loro progettazione specifica nell'eseguire modelli di ML dedicati all'elaborazione delle immagini; possono quindi includere un'interfaccia diretta con dispositivi ottici come fotocamere.

Le NNP invece sono una famiglia di processori neurali progettati da Intel per l'accelerazione nell'elaborazione dei codici nei progetti di intelligenza artificiale. L'ultima architettura è stata presentata nel 2019 con il nome di Spring Hill (Intel Nervana NNP-I 1000 e Intel Nervana NNP-T 1000).

2.3 - Piattaforme di sviluppo e programmazione

In relazione allo sviluppo hardware nel paradigma TinyML vi è la conseguente crescita a livello software; infatti, si può notare come i framework prevalenti si basino fortemente su librerie distribuite dai fornitori. Sarà scopo di questo capitolo fornire una panoramica dei principali framework presenti e degli elementi abilitanti per progetti TinyML.

2.3.1 – Piattaforme di sviluppo a confronto

- TensorFlow Lite

È un *framework open-source* che permette ai MCU con capacità di memoria limitata di eseguire modelli ML. Consente lo sviluppo di machine learning su dispositivi *edge* utilizzando diversi linguaggi come C, C++, Python, Java e Swift. TFL facilita l'ottimizzazione dei modelli per l'accelerazione hardware.

Il framework richiede una piattaforma a 32 bit ed è compatibile con la maggior parte dei processori della serie ARM Cortex-M, che, come si può notare dalla tabella 1, sono molto diffusi nei progetti tinyML.

Il processo di lavoro si basa sulla conversione del modello in un file *.tflite*, il quale verrà caricato all'interno della memoria del dispositivo. Inoltre, l'estensione TensorFlow Lite Micro, ottimizza l'esecuzione di modelli ML su processori ARM Cortex, ma funziona anche su piattaforme a 32 bit come ESP32, Arduino Nano 33 BLE Sense o Sony Spresense.

- uTensor

È un ambiente di apprendimento embedded gratuito ed estremamente leggero, che facilita la prototipazione e il rapido deployment sui dispositivi *IoT-edge* (ottimizzato per dispositivi ARM). Utilizza un modello di rete neurale addestrato con Keras. Successivamente, converte il modello in file sorgente C++.

- Edge Impulse

In questo servizio, il processo di addestramento avviene su una piattaforma cloud e il modello addestrato può essere poi esportato facilmente su un dispositivo *edge*. Inoltre, Edge Impulse semplifica la raccolta di dati dei sensori reali, abilita l'elaborazione in tempo reale dei segnali dai dati grezzi alle reti neurali e semplifica le procedure di testing.

❖ STM32Cube.AI

È un software per la compilazione e l'ottimizzazione del codice ML per schede basta su STM32 ARM Cortex-M. L'implementazione delle reti neurali su una scheda STM32 può essere ottenuta direttamente utilizzando STM32Cube.AI, che converte le reti neurali in codice ottimizzato per il microcontrollore più appropriato. Questo strumento è in realtà un'estensione del framework originale STM32CubeMX, che collabora per la generazione del codice finale.

❖ uTVM

Come estensione della macchina virtuale tensoriale TVM, permette l'ottimizzazione dei programmi tramite la piattaforma AutoTVM. Permette (tramite OpenOCD) di ottenere portabilità delle prestazioni su una vasta gamma di hardware.

❖ PyTorch Mobile

Appartiene all'ecosistema PyTorch e mira a supportare tutte le fasi, dalla formazione al deployment dei modelli di machine learning su smartphone. Supporta anche GPU, DSP e NNP.

3 – Discussione

Con l'importante aumento degli studi e delle ricerche che si occupano dei metodi, delle ottimizzazioni e delle applicazioni del tinyML, quest'ultimo sta avendo un considerevole sviluppo. Una crescita che riflette l'aumento della produzione di soluzioni in tempo reale per applicazioni industriali e no.

In questo capitolo si discuterà del lavoro che la comunità scientifica sta affrontando e degli obiettivi che si è posta; andando poi ad affrontare il discorso della tecnologia tinyML in ambito industriale e quindi delle sue criticità.

3.1 – Sfide e Limitazioni

Una delle principali problematiche di ricerca per il tinyML è la limitata disponibilità energetica dei dispostivi *edge*. I modelli richiedono un certo livello di energia per mantenere elevata l'accuratezza degli algoritmi ed è complicato definire un loro modello dei consumi energetici. Inoltre, progettare un modulo universale per la gestione energetica risulta complesso, per via delle differenze dei design e dei processi che caratterizzano le diverse piattaforme hardware.

Un altro aspetto che ostacola la crescita del paradigma sono le dimensioni estremamente ridotte della SRAM e delle memorie flash dei dispositivi. L'inferenza tradizionale del ML può richiedere picchi di memoria più elevati (ad esempio, GB), che rendono le piattaforme *edge* incompatibili. Inoltre, la maggior parte dei dispositivi *edge* opera a velocità di clock comprese tra 10 e 1000 MHz, il che è limitante per l'esecuzione efficace di modelli di apprendimento complessi.

A tutto questo si unisce l'eterogeneità delle piattaforme hardware e software, rendendo così più insidiosa la definizione di un metodo di apprendimento e una strategia di "*deployment*" comune. Il "fattore di forma" delle varie piattaforme hardware rende impossibile uniformare i risultati delle prestazioni dei i modelli su dispositivi *edge*.

Altre importanti limitazioni sono la mancanza di strumenti di *benchmarking* e la mancanza di dataset adatti a progetti tinyML. Quest'ultimi dovrebbero avere una risoluzione temporale e spaziale adeguata a corrispondere le caratteristiche di generazione dei dati dai sensori esterni. Inoltre, il rumore (soprattutto in ambito industriale) e le diversità tra i dataset presenti, richiedono un lavoro aggiuntivo non indifferente, che talvolta pregiudica l'efficienza e l'efficacia del modello stesso.

Nel contesto più generale invece, progettare software per dispositivi *edge* richiede competenze specialistiche e inoltre, strutturare l'architettura di rete e quindi organizzare il collegamento dei

dispositivi è un'altra sfida chiave, che rende ancor di più la materia un ambito settoriale. Anche perché il "*deployment*" dei progetti tinyML non basta se i dati e la rete non sono gestiti correttamente, soprattutto nel caso in cui i dati raccolti siano di tipo eterogeneo. Non comprendere appieno il ciclo di vita dei dati raccolti dai sensori aggrava la problematica della gestione delle informazioni e quindi anche sugli eventuali guasti di rete.

È essenziale sviluppare nuovi modelli di ML specifici per l'ecosistema tinyML, sfruttando approcci come l'apprendimento federato, l'apprendimento rinforzato o l'apprendimento online. Assicurandosi inoltre di ridurrei complessivamente i costi e coordinando in modo efficiente l'interazione tra i dispositivi della rete.

3.2 – Tendenze future

Il tinyML sta gradualmente diventando una necessità sempre più stringente. Infatti, i dispostivi *smart* richiederanno sempre più la presenza dell'intelligenza artificiale al loro interno, pur mantenendo consumi energetici ridotti e funzionamenti in *Real-time Computing* (RTC).

La tendenza è quindi quella di rendere il paradigma il più popolare possibile, semplificandone la grammatica, sviluppando flussi di processo standardizzati per aiutare gli sviluppatori a realizzare scenari di *deployment* pronti per il mercato e popolando *repository* di *dataset* appropriati e strumenti di *benchmarking* leggeri.

Per contrastare l'eterogeneità dei dispositivi hardware e facilitare la produzione su grande scala, si stanno sviluppando paradigmi come quelli dei *Container*, unità eseguibili di *software* che contengono tutti gli elementi necessari per l'esecuzione in qualsiasi ambiente [27]. I *Container* (e.g. Docker) utilizzano al meglio una forma di virtualizzazione del sistema operativo (SO) in cui le funzionalità kernel dell'SO possono essere utilizzate per isolare i processi e controllare la quantità di CPU, memoria e disco a cui tali processi possono accedere [28].

Come analizzato nei capitoli precedenti, i produttori di microcontrollori hanno cominciato a concentrarsi sulla progettazione di design adatti al ML; questo è indice del fatto che le richieste del mercato tendono un'integrazione sempre più importante di modelli ML all'interno delle piattaforme hardware più utilizzate, soprattutto nell'ambito IoT.

Tabella 2 – Confronto tra sfide e tendenze future

N.	*Limitazioni*	*Sfide*	*Tendenze future*
1.	Risorse	Disponibilità limitata di energia e dimensioni della memoria	Progettazione di dispositivi edge tramite un approccio di co-design
2.	Hardware e Software	Eterogeneità hardware e software	Sviluppare modelli generalizzati per lavorare in modo efficiente con sistemi eterogenei
3.	Data-set	Dataset esistenti non utilizzabili direttamente per addestrare i modelli TinyML	Sviluppare dataset appropriati e strumenti di benchmarking leggeri
4.	Progettazione modelli ML	Tempi di risposta dei modelli ML molto lunghi	Incorporare la riduzione del modello e la quantificazione come parte della progettazione del modello
5.	Affidabilità	Problemi di affidabilità dell'hardware e del modello	Sistemi per garantire che il dispositivo edge sia affidabile e accurato prima di distribuirlo nel campo di applicazione

3.3 – Considerazioni e criticità in ambienti industriali

Il vantaggio principale del tinyML in ambienti industriali è quello di poter usufruire di modelli ML direttamente su dispositivi *edge* localizzati direttamente nell'impianto. Riducendo così i tempi di latenza e quindi dotando il sistema della capacità di avere una risposta rapida agli eventi, migliorando la sicurezza e l'efficienza operativa.

Nonostante i benefici, l'adozione di progetto tinyML in ambienti industriali presenta delle criticità tecniche che rendono ancora complessa la vasta adozione di tale paradigma nei processi industriali più comuni. Tra queste criticità si evidenzia, come discusso nei capitoli precedenti, la minore precisione dei modelli, che però non può più essere sottovalutata vista anche la variabilità della qualità dei dati raccolti dai sensori, tipica degli ambienti industriali. Infatti, i sensori possono essere soggetti a rumore e degradazione. Sarà quindi necessario eseguire del lavoro aggiuntivo per rendere i modelli particolarmente robusti e in grado di gestire dati imperfetti o incompleti, mantenendo così le prestazioni in linea con le richieste del progetto.

Localizzare i dispositivi hardware nell'impianto comporta ovviamente l'adattamento di quest'ultimi al processo produttivo e quindi allo sviluppo di interfacce e protocolli specifici, in modo che le piattaforme e i modelli possano interagire efficacemente con i sistemi esistenti.

Ridurre al minimo la manutenzione del sistema e mantenerlo aggiornato per tutto il periodo funzionamento, sono elementi chiave per un buon progetto tinyML, permettendo quindi un monitoraggio da remoto e un intervento immediato in caso di malfunzionamenti.

Rispetto quindi a semplici progetti, quelli sviluppati all'interno di ambienti industriali richiedono che l'hardware e il modello adempiano agli scopi del progetto mantenendo il più possibile invariata la struttura del processo produttivo nel quale vengono inseriti.

Il paradigma ha la possibilità di rivoluzionare l'ambiente industriale e quindi collaborare alla sua quinta rivoluzione solo se sarà capace di offrire soluzioni efficaci e reattive, che permettano di supportare il processo decisionale all'interno delle aziende e quindi risolvendo le criticità dovute alla qualità e alla scarsità dei dati e all'affidabilità dei modelli. La ricerca di soluzioni innovative contribuirà a superare queste sfide, aprendo la strada a un'adozione più ampia ed efficace nel settore industriale.

Citando l'esempio della società Etrash srl con sede a Venezia e del suo cestino intelligente per la raccolta differenziata automatica dei rifiuti, l'innovazione in ambito tinyML, allo stato dell'arte, può avvenire solo tramite una prima e ben strutturata fase di sperimentazione e verifica (*System-under-test*, SUT); nella quale la raccolta delle informazione permette non solo la collezione dei dati fondamentali per l'addestramento dei modelli, ma anche di studiare al

meglio l’ambiente nel quale il prodotto finale dovrà operare e quindi strutturare la migliore strategia per rendere il sistema il più affidabile possibile.

La società, pioniera in Italia nell'adozione del paradigma tinyML nei propri prodotti e servizi, propone una visione innovativa per guidare le imprese verso l'industria 5.0, ponendo il focus sull'impatto ambientale delle attività aziendali. Insieme all’aeroporto di Venezia sta procedendo, nell’ambito del bando LIFE indetto dall’Unione Europea, a dare un esempio di come il futuro delle società sia sempre più interessato al *Machine Learning* e al TinyML [29].

4 – Conclusioni

È stato necessario soffermarsi sulle definizioni più importanti del *Machine Learning* prima di introdurre il paradigma del tinyML. In particolare, sono state analizzate le reti neurali e i loro principali metodi di allenamento.

Successivamente, il tinyML è stato esplorato partendo dalle sue definizioni base e tracciandone l'evoluzione, con un focus sulle sue applicazioni. A supporto, sono state fornite le definizioni dei principali componenti elettronici utilizzati nei progetti tinyML.

La discussione, spostandosi sulla creazione di progetti tinyML, ne ha analizzato la loro organizzazione tipica, mettendo a confronto i framework più utilizzati allo stato dell'arte. È stato sottolineato come l'organizzazione di un progetto possa focalizzarsi in misura maggiore sulla componente hardware oppure software.

Infine, l'analisi si conclude con alcune osservazioni critiche sulle sfide e sulle limitazioni del paradigma, descrivendo come le tendenze puntino a superare questi ostacoli.

Il tinyML rappresenta un esempio di come l'intelligenza artificiale stia rivoluzionando il mondo dell'elettronica. Un ingegnere deve essere preparato a conoscere questa disciplina e a gestirne sia i vantaggi che le criticità.

Le ragioni di base di questo testo non risiedono solo nel fatto che il tinyML si presenta come la prossima rivoluzione tecnologica, ma anche perché si è voluto creare un'opera che colmasse la scarsità, nel panorama, di articoli in lingua italiana.

L'obiettivo è quindi quello di introdurre, seppur in modo non troppo approfondito, e divulgare un paradigma alla comunità tecnico-scientifica italiana. Il testo cerca di semplificare la grammatica e il linguaggio tecnico per agevolarne la comprensione, favorendo così un dialogo aperto e una discussione più ampia, consolidando i progressi della comunità TinyML.

Una branca dell'intelligenza artificiale è difficile da descrivere in un testo imperituro. Le migliorie e i progressi tecnologici sono di anno in anno sempre più importanti, il che rende complesso accostargli definizioni univoche. Ma il futuro che ci si aspetta è quello nel quale l'ambiente universitario e industriale collaborino per creare una grammatica universale con la quale discutere del tinyML.

Ampliare il testo e perfezionarne il contenuto è un qualcosa di obbligato, cercando di riprendere i nuovi progressi tecnologici e le nuove definizioni che la comunità si farà carico di sviluppare ed è importante che rimanga una base affidabile con la quale approcciare il paradigma e comprendere testi più complessi e specialistici.

5 – Bibliografia

[1] Warden Pete e Situnayake Daniel, *TinyML*, Second Release. 2020.

[2] Oracle, «Cos'è il Machine Learning?» Consultato: 15 luglio 2024. [Online]. Disponibile su: https://www.oracle.com/it/artificial-intelligence/machine-learning/what-is-machine-learning/

[3] Wikipedia, «Apprendimento automatico». Consultato: 15 luglio 2024. [Online]. Disponibile su: https://it.wikipedia.org/wiki/Apprendimento_automatico

[4] Y. Y. Siang, M. Ridzuan Ahamd, M. Shafinaz, e Z. Abidin, «Anomaly Detection Based on Tiny Machine Learning: A Review», 2021.

[5] F. Tian *et al.*, «Prediction of tumor origin in cancers of unknown primary origin with cytology-based deep learning», *Nat Med*, vol. 30, n. 5, pagg. 1309–1319, mag. 2024, doi: 10.1038/s41591-024-02915-w.

[6] R. Kallimani, K. Pai, P. Raghuwanshi, S. Iyer, e O. L. A. López, «TinyML: Tools, applications, challenges, and future research directions», *Multimed Tools Appl*, vol. 83, n. 10, pagg. 29015–29045, mar. 2024, doi: 10.1007/s11042-023-16740-9.

[7] IBM, «Cos'è il deep learning?» Consultato: 15 luglio 2024. [Online]. Disponibile su: https://www.ibm.com/it-it/topics/deep-learning

[8] S. P. F. Meneghello, «The Role of Artificial Intelligence in Next-Generation Wireless Networks - an Overview of Technological and Law Implications».

[9] R. Y. Choi, A. S. Coyner, J. Kalpathy-Cramer, M. F. Chiang, e J. Peter Campbell, «Introduction to machine learning, neural networks, and deep learning», *Transl Vis Sci Technol*, vol. 9, n. 2, 2020, doi: 10.1167/tvst.9.2.14.

[10] E. García-Martín, C. F. Rodrigues, G. Riley, e H. Grahn, «Estimation of energy consumption in machine learning», *J Parallel Distrib Comput*, vol. 134, pagg. 75–88, dic. 2019, doi: 10.1016/j.jpdc.2019.07.007.

[11] S. K. Rethinagiri, O. Palomar, R. Ben Atitallah, S. Niar, O. Unsal, e A. C. Kestelman, «System-level power estimation tool for embedded processor based platforms», *ACM International Conference Proceeding Series*, 2014, doi: 10.1145/2555486.2555491.

[12] L. Capogrosso, F. Cunico, D. S. Cheng, F. Fummi, e M. Cristani, «A Machine Learning-oriented Survey on Tiny Machine Learning», set. 2023, [Online]. Disponibile su: http://arxiv.org/abs/2309.11932

[13] «tinyML Foundation». Consultato: 29 luglio 2024. [Online]. Disponibile su: https://www.tinyml.org/

[14] W. ’P. e S. ’D., *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. 2019. Consultato: 29 luglio 2024. [Online]. Disponibile su: https://cms.tinyml.org/wp-content/uploads/emea2021/tinyML_Talks_Felix_Johnny_Thomasmathibalan_and_Fredrik_Knutsson_210208.pdf

[15] P. P. Ray, «A review on TinyML: State-of-the-art and prospects», 1 aprile 2022, *King Saud bin Abdulaziz University*. doi: 10.1016/j.jksuci.2021.11.019.

[16] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, e S. Han, «Tiny Machine Learning: Progress and Futures», mar. 2024, doi: 10.1109/MCAS.2023.3302182.

[17] IEEE - Dr. Karim Arabi, « Mobile Computing Opportunities, Challenges and Technology Drivers», 2014. Consultato: 29 luglio 2024. [Online]. Disponibile su: https://web.archive.org/web/20200730234708/http://www2.dac.com/events/videoarchive.aspx?confid=170&filter=keynote&id=170-103--0&

[18] MIT - Dr. Karim Arabi, «Trends, Opportunities and Challenges Driving Architecture and Design of Next Generation Mobile Computing and IoT Devices», 2015, Consultato: 29 luglio 2024. [Online]. Disponibile su: https://www.mtl.mit.edu/events-seminars/seminars/trends-opportunities-and-challenges-driving-architecture-and-design-next

[19] H. Han e J. Siebert, «TinyML: A Systematic Review and Synthesis of Existing Research», in *4th International Conference on Artificial Intelligence in Information and Communication, ICAIIC 2022 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2022, pagg. 269–274. doi: 10.1109/ICAIIC54071.2022.9722636.

[20] R. Kallimani, K. Pai, P. Raghuwanshi, S. Iyer, e O. L. A. López, «TinyML: Tools, Applications, Challenges, and Future Research Directions», mar. 2023, doi: 10.1007/s11042-023-16740-9.

[21] Wikipwdia, «FPGA». Consultato: 10 settembre 2024. [Online]. Disponibile su: https://it.wikipedia.org/wiki/Field_Programmable_Gate_Array

[22] «Mark Nadeski Embedded Processing Texas Instruments Bringing machine learning to embedded systems».

[23] Arduino, «Arduino Tiny Machine Learning Kit». Consultato: 4 settembre 2024. [Online]. Disponibile su: https://store.arduino.cc/products/arduino-tiny-machine-learning-kit?srsltid=AfmBOoo5B8Qb4tXh2KQNMLTALtj9EwWXwlfTcKeKIzlIxSIKs80Lxe1-

[24] S. Kalapothas, M. Galetakis, G. Flamis, F. Plessas, e P. Kitsos, «A Survey on RISC-V-Based Machine Learning Ecosystem», 1 febbraio 2023, *MDPI*. doi: 10.3390/info14020064.

[25] S. Prakash *et al.*, «CFU Playground: Full-Stack Open-Source Framework for Tiny Machine Learning (tinyML) Acceleration on FPGAs», gen. 2022, doi: 10.1109/ISPASS57527.2023.00024.

[26] Naveen Suda e Danny Loh, «Machine Learning on Arm Cortex-M Microcontrollers», *Cambridge,UK*, 2019.

[27] Google, «Che cosa sono i container?» Consultato: 9 settembre 2024. [Online]. Disponibile su: https://cloud.google.com/learn/what-are-containers?hl=it

[28] IBM, «Cosa sono i contenitori?» Consultato: 9 settembre 2024. [Online]. Disponibile su: https://www.ibm.com/it-it/topics/containers

[29] Etrash, «Etrash». Consultato: 10 settembre 2024. [Online]. Disponibile su: https://www.etrash.it/

[30] Microsoft, «Inferenza e valutazione dei modelli di previsione». Consultato: 15 luglio 2024. [Online]. Disponibile su: https://learn.microsoft.com/it-it/azure/machine-learning/concept-automl-forecasting-evaluation?view=azureml-api-2