Integration of TinyML and LargeML: A Survey of 6G and Beyond

Thai-Hoc Vu, Ngo Hoang Tu, *Member, IEEE*, Thien Huynh-The, *Senior Member, IEEE*, Miroslav Voznak, *Senior Member, IEEE*, Kyungchun Lee, *Senior Member, IEEE*, Sunghwan Kim, *Senior Member, IEEE*, and Quoc-Viet Pham, *Senior Member, IEEE*

***Abstract*—The evolution from fifth-generation (5G) to sixth-generation (6G) networks is driving an unprecedented demand for advanced machine learning (ML) solutions. Deep learning (DL) has already demonstrated significant impact across mobile networking and communication systems, enabling intelligent services such as smart healthcare, smart grids, autonomous vehicles, aerial platforms, digital twins, and the metaverse. At the same time, the rapid proliferation of resource-constrained Internet of Things (IoT) devices has accelerated the adoption of tiny machine learning (TinyML) for efficient on-device intelligence, while large machine learning (LargeML) models continue to require substantial computational resources to support large-scale IoT services and ML-generated content. These trends highlight the need for a unified framework that integrates TinyML and LargeML to achieve seamless connectivity, scalable intelligence, and efficient resource management in future 6G systems. This survey provides a comprehensive review of recent advances enabling the integration of TinyML and LargeML in next-generation wireless networks. In particular, we: 1) provide an overview of TinyML and LargeML; 2) analyze the motivations and requirements for unifying these paradigms within the 6G context; 3) examine efficient bidirectional integration approaches; 4) review state-of-the-art solutions and their applicability to emerging 6G services; and 5) identify key challenges related to performance optimization, deployment feasibility, resource orchestration, and security. Finally, we outline promising research directions to guide the holistic integration of TinyML and LargeML for intelligent, scalable, and energy-efficient 6G networks and beyond.**

***Index Terms*—Artificial intelligence (AI), deep learning (DL), federated learning (FL), Internet of Things (IoT), large machine learning (LargeML), sixth-generation (6G), tiny machine learning (TinyML).**

Received 4 October 2025; revised 6 March 2026; accepted 12 March 2026. Date of publication 16 March 2026; date of current version 11 May 2026. This work was supported in part by the European Union under the REFRESH—Research Excellence For REgion Sustainability and High-tech Industries via the Operational Program Just Transition under Project CZ.10.03.010022_0030000048; and in part by the Ministry of Education, Youth and Sports of the Czech Republic (MEYS CZ) within a Student Grant Competition in the VSB-Technical University of Ostrava under Project SGS SP2026004. The work of Kyungchun Lee was supported in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant NRF-2019R1A6A1A03032119 and in part by the NRF Grant funded by Korean Government [Ministry of Science and ICT (MSIT)] under Grant RS-2025-21812978. The work of Sunghwan Kim was supported by the Research Program through NRF under Grant NRF-2023R1A2C1003546. The work of Quoc-Viet Pham was supported in part by the Research Ireland under the European Innovation Council through the CHIST-ERA SHIELD Project under Grant 216449 and Grant 19226. *(Thai-Hoc Vu and Miroslav Voznak are co-first authors.) (Corresponding author: Sunghwan Kim.)*

Thai-Hoc Vu and Miroslav Voznak are with the Faculty of Electrical Engineering and Computer Science, VSB-Technical University of Ostrava, 708 00 Ostrava, Czechia (e-mail: thai.hoc.vu@vsb.cz; miroslav.voznak@vsb.cz).

Ngo Hoang Tu is with the Faculty of Information Technology, Van Lang School of Technology, Van Lang University, Ho Chi Minh City 70000, Vietnam (e-mail: tu.nh@vlu.edu.vn).

Thien Huynh-The is with the Department of Electronics and Information Engineering, Ho Chi Minh City University of Technology and Engineering, Ho Chi Minh City 71307, Vietnam (e-mail: thienht@hcmute.edu.vn).

Kyungchun Lee is with the Department of Electrical and Information Engineering and the Research Center for Electrical and Information Technology, Seoul National University of Science and Technology, Seoul 01811, South Korea (e-mail: kclee@seoultech.ac.kr).

Sunghwan Kim is with the Department of Artificial Intelligence and Information Technology, Sejong University, Seoul 05006, Republic of Korea (e-mail: ks@sejong.ac.kr).

Quoc-Viet Pham is with the School of Computer Science and Statistics, Trinity College Dublin, Dublin 2, D02 PN40 Ireland (e-mail: viet.pham@tcd.ie).

Digital Object Identifier 10.1109/JIOT.2026.3674358

I. Introduction

THE evolution of communication technologies has been characterized by extraordinary advances every decade. From the first-generation (1G) analog systems, which introduced mobile voice communications, to fifth-generation (5G) networks currently transforming industries with ultrafast speeds and low latency, each generation has delivered major transformative impacts [1]. Looking ahead, sixth-generation (6G) networks are expected to push technological boundaries further, offering unprecedented capabilities and driving global innovation. The core features of 6G will clearly differentiate it from previous generations (Fig. 1), positioning the paradigm as a transformative technology for diverse applications. These features address the escalating demands for higher data rates, superior reliability, efficient spectrum management, and super-intelligence capabilities. Furthermore, 6G aims to seamlessly integrate advanced artificial intelligence (AI) and machine learning (ML) technologies, expanding the possibilities of wireless networks and significantly improving the delivery of high-end applications and services to users.

The integration of AI and ML into 6G will enable new levels of automation and intelligence, unlocking numerous new applications and high-end services [2]. AI already operates on multiple network layers to improve dynamic resource management, predict maintenance needs, and personalize user experiences. Advances in ML, particularly deep learning (DL) architectures that execute computer vision and natural language processing (NLP) tasks, are expected to support high-fidelity holographic communications. This capability will allow users to interact in immersive 3-D environments and could revolutionize remote collaboration. In a hyper-connected future, 6G will unlock the full potential of the Internet of Things (IoT) and mobile big data to drive innovation and lead the world into smarter, deeper connectivity.

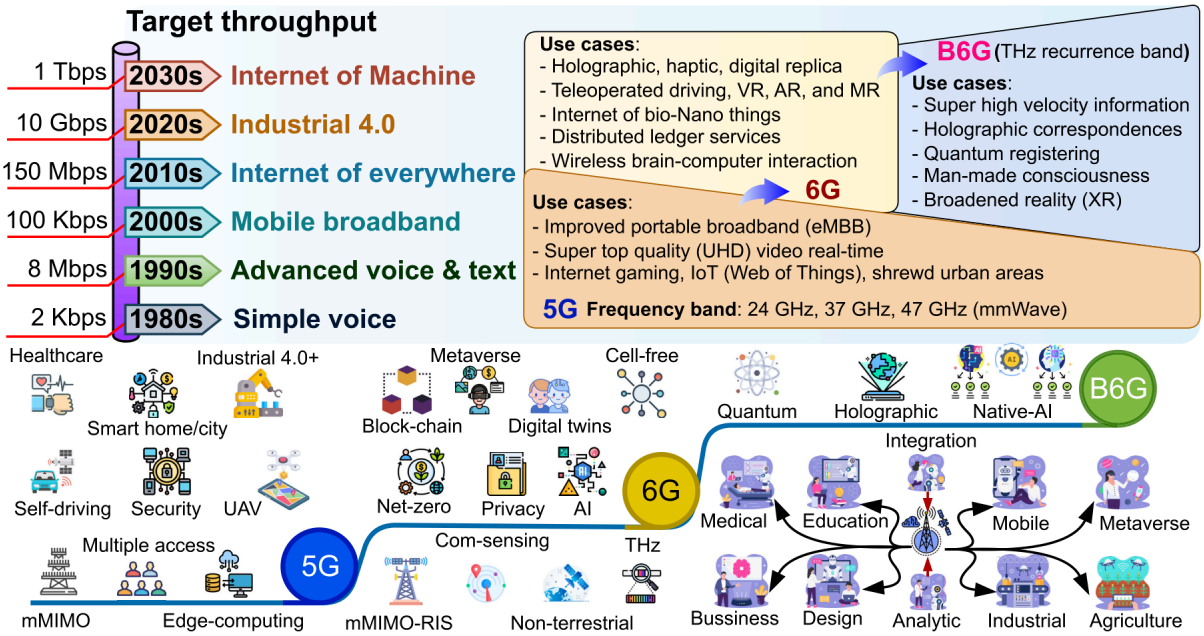

Fig. 1. Transition from 5G to 6G and beyond 6G (B6G): target throughput, use cases, core technologies, and applications.

TABLE I
COMPARISON OF TINYML AND LARGEML FEATURES

Aspects	TinyML	LargeML
Definition	ML on resource-constrained devices	Large-scale AI models with vast numbers of parameters and large data processing capacity
Computational requirements	Limited power, memory, and processing capability	Substantial power, memory, and computational resources
Deployment environment	Smart sensors, edge devices, and microcontrollers	Cloud servers and specialized hardware (e.g., GPUs, TPUs)
Latency	Ultra-low latency for real-time processing	Higher latency, acceptable for batch processing
Power consumption	Very low, suitable for battery-powered devices	High, requiring a robust power supply
Data processing	Local (on-device) processing	Centralized (large-scale) data processing
Typical applications	Real-time analytics and sensor data processing in IoT environments (e.g., smart homes, industrial monitoring)	Complex data analysis and decision-making (e.g., NLP, image recognition, autonomous systems)
Advantages	Low latency, reduced data transmission, enhanced privacy	High accuracy, with the ability to process and analyze large datasets
Challenges	Constrained by hardware limitations and task complexity	Requires extensive computational infrastructure and incurs high costs
Complementary roles	Supports real-time, lightweight, and edge-based tasks	Enables deep, comprehensive analysis and cloud-based decision-making
Examples	Keyword spotting, anomaly detection in sensor data	GPT-4, BERT, large-scale image classifiers

A. Context and Motivation

In the last decade, AI has converged around two contrasting approaches: tiny ML (TinyML) and large machine learning (LargeML). TinyML focuses on creating efficient models for resource-constrained edge devices, allowing real-time analytics close to data sources (i.e., bringing analytics closer to data sources) [3]. In contrast, LargeML leverages powerful cloud servers to handle complex, data-intensive tasks, exemplified by models like GPT-4 and BERT, which excel in applications such as sentiment analysis and creative text generation [4] (e.g., large language models (LLMs), generative AI (GenAI), and agentic AI).

However, LargeML's heavy computational demands limit real-time use on constrained devices and raise privacy concerns, while TinyML mitigates these issues by preprocessing data at the edge and enabling LargeML to deploy compressed models through knowledge distillation [5]. This collaborative strategy improves flexibility for IoT services: TinyML gains accuracy from LargeML's advanced learning, and LargeML benefits from TinyML's real-time processing and privacy preservation, as summarized in Table I. Their integration strengthens AI capabilities in 6G systems. For example, wearable health monitors can analyze vital signs locally with TinyML [6] while LargeML detects medical risks from historical data, and TinyML can process electroencephalography (EEG) signals for emotional state recognition [7] while LargeML translates them into metaverse actions [8].

Despite these advantages, challenges remain, including heightened security risks from massive device connectivity and ethical concerns around explainability and data privacy, yet combining TinyML and LargeML ultimately enables seamless device–server interaction and advances in data-driven innovation and intelligent living environments.

B. State-of-the-Art and Contributions

Over the past decade, numerous surveys have examined TinyML's role in edge computing, offering comprehensive

TABLE II
SUMMARY OF RELATED SURVEYS ON TINYML AND LARGEML

Refs.	Review Topics		Key Contributions	Limitations
	TinyML	LargeML		
[9]	✓		Highlights hardware-software co-design for optimizing TinyML frameworks on resource-limited devices and reviews ML algorithms optimized for low-power deployment.	Focuses on TinyML and 5G, without discussion of integration with LargeML in 6G systems.
[10]	✓		Introduces a classification system for TinyML applications based on an extensive literature review, exploring benefits and use cases.	Does not address TinyML–LargeML integration or the specific challenges of 6G integration.
[11]	✓		Surveys TinyML frameworks, development tools, enablers, and applications, and outlines current research challenges.	Lacks discussion of TinyML–LargeML integration in the context of 6G networks.
[12]	✓		Proposes a taxonomy for reformable TinyML solutions and evaluates the suitability of TinyML layers for reformability.	Does not explore TinyML–LargeML integration in the context of 6G networks.
[13]	✓		Reviews neural network (NN) compression techniques and categorizes TinyML hardware and software resources.	Examines the foundational aspects of TinyML, with limited discussion on integration with LargeML or implications for 6G environments.
[14]	✓		Reviews TinyML research, workflows, algorithms, and development tools, with an emphasis on recent ML advances.	Lacks analysis of integration with LargeML in the context of 6G networks.
[15]		✓	Reviews transformer architectures in LLMs, discussing training methods, applications, societal impact, and deployment challenges.	Does not address 6G-related challenges or TinyML–LargeML integration for enhancing 6G capabilities.
[16]		✓	Examines LLM evaluation methods, benchmarks, and performance across domains.	Omits discussion of ML integration and analysis of challenges in 6G applications.
[17]		✓	Analyzes fine-tuning algorithms for LargeML, detailing performance metrics and deployment scenarios.	Lacks the investigation of fine-tuning algorithms with TinyML for 6G applications.
[18]		✓	Proposes a taxonomy of explainability techniques for LLMs, with evaluation metrics and implementation challenges.	Concentrates on LLM explainability without addressing its integration with TinyML in collaborative 6G systems.
[19]		✓	Outlines architectures, techniques, and challenges in PLMs.	Focuses on NLP applications of PLMs; lacks discussion of TinyML integration in 6G systems.
This work	✓	✓	Offers the first survey of the integration of TinyML and LargeML for 6G systems. Specifically, (*i*) Introduces fundamentals of TinyML and LargeML; (*ii*) Discusses motivations and requirements for integrating TinyML and LargeML in 6G; (*iii*) Proposes bidirectional integration frameworks for enhancing future network performance and capabilities; (*iv*) Explores potential applications of integrated TinyML–LargeML systems; and (*v*) Identifies challenges and future research directions for TinyML–LargeML integration.	

overviews of its fundamentals, development tools, applications, and future directions [9], [10], [11], [12], [13], [14], with key contributions including hardware–software co-design for efficient deployment on constrained devices [9] and systematic reviews of TinyML toolchains spanning hardware platforms, software frameworks, and libraries [11]. Other works [10], [13], [14] highlight TinyML's broad application landscape, from anomaly detection and predictive maintenance to intelligent sensor networks, where models monitor sensor data for deviations, forecast equipment failures, and enable fast, local decision-making. Collectively, these surveys highlight TinyML's transformative potential across industrial automation, environmental monitoring, smart infrastructure, and other data-driven domains.

Despite these advances, the existing survey literature on TinyML reveals some limitations when considered in the context of 6G integration. Although these surveys examine TinyML applications within general IoT frameworks, they overlook the specific challenges and opportunities presented by integrating TinyML with LargeML in 6G networks [10]. Some surveys [13], [14] provide limited clarification on fine-tuning TinyML models to the specific resource constraints of 6G environments. For example, 6G may require the development of lightweight communication protocols tailored to TinyML models for efficient data exchange with server-side LargeML systems. These surveys also fall short of exploring optimization strategies for minimizing power consumption in TinyML models while preserving the accuracy required for effective collaboration with LargeML [14].

In contrast, LargeML surveys outline complementary strengths and limitations, with works such as [15], [16], [17], [18], and [19] examining architectures, applications, evaluation methods, and open challenges. Reviews like [15] analyze transformer-based models and LLM development, while [19] highlights the impact of large pretrained language models (PLMs) on diverse NLP tasks, emphasizing their strong performance in text generation, translation, and sentiment analysis. Other surveys [16], [17], [18] focus on challenges such as explainability and the need for parameter-efficient fine-tuning (PEFT) [17] to reduce computational overhead. However, despite their breadth, current LargeML surveys do not address 6G-specific constraints: they overlook deployment and coordination challenges in real-time, latency-sensitive, edge-intelligent networks, and the need to compress or adapt LargeML models for efficient operation alongside TinyML systems. They also omit the implications of emerging 6G communication protocols and strict latency requirements for training and inference. Table II summarizes these surveys and situates the contributions of this work.

The previous discussion reveals a significant gap in surveys addressing the integration of TinyML and LargeML in the context of 6G and beyond. Although recent advancements in LargeML [20], [21] are vital for developing efficient 6G networks, they have not been thoroughly analyzed. Most

TABLE III
TINYML CLASSIFICATIONS, HARDWARE DEVICES, ENVIRONMENTAL DEPLOYMENT, AND SOFTWARE TOOLS

Classification	Hardware Device [14]	Environmental Deployment [10]	Software Tools [12]
• **Model size**: (1) Compression methods (pruning, quantization, knowledge distillation), (2) Memory/storage design (kilobytes to megabytes), (3) Working modes (centralized, distributed, decentralized). • **Architecture**: (1) Convolutional neural network (CNN) for image processing, (2) Recurrent neural network (RNN) for sequential data, (3) Fully connected networks for certain tasks, (4) Transformer [23] for data generation, context awareness, translation, and summarization. • **Training data**: (1) Quality assurance and data labeling, (2) Data augmentation (rotation, scaling, flipping), and (3) Fine-tuning pre-trained models on large datasets. • **Algorithms**: Supervised, weakly supervised, unsupervised, self-supervised, meta-learning, continual learning, deep reinforcement learning (DRL), transfer learning (TL), split learning (SL), and federated learning (FL) [10] (readers may refer to [24]–[28] for additional foundations and core concepts).	• **Central processing unit (CPU)**: Arm Cortex M family offers ultra-low-power operation but limits to parallel processing. • **Graphics processing unit (GPU)**: Supports parallel computing and optimized sequential processing of large datasets (e.g., NVIDIA Jetson family, AMD, Intel, Arm), but is not cost-effective. • **Field-programmable gate array (FPGA)**: Enables model customization and hardware-level acceleration for complex operations and large dataset access, but lacks broad library support. • **Tensor processing unit (TPU)**: Accelerates ML workloads, particularly those involving NNs, with two representations of Edge TPUs and Cloud TPUs.	• **Atmospheric**: Low-power sensors to monitor air quality in urban, forest, and industrial areas, detecting pollutants by measuring particulate matter, carbon dioxide levels, and others. • **Hydrosphere**: Sensors in aquatic environments to detect pollutants, measure pH, and monitor temperature changes. • **Biosphere**: Device used for wildlife conservation, monitoring biological activity, or classifying sounds (e.g., animal habitats and anti-poaching surveillance). • **Lithosphere**: Applied in agriculture for environmental monitoring (e.g., soil moisture, nutrient levels, pests) or earthquake detection. • **Human-related**: Wearable health monitor (e.g., blood pressure) and assistive technologies for humans (e.g., mobility).	• **Mobile device**: TensorFlow Lite for model optimization and PyTorch for programming. • **Embedded devices**: TensorFlow Lite for model optimization, PyTorch for programming, and Qeexo AutoML for end-to-end customization. • **Edge devices**: PyTorch for programming, NanoEdgeAIStudio for low/no-code solution, and Imagimob/Edge Impulse for end-to-end customization. • **Microcontroller**: Arm NN for network architecture optimization, MinUn for bandwidth/memory management, STM32CubeA and uTensor for pre-trained model conversion, and NXP eIQ as a development environment.

studies focus on either TinyML or LargeML [22], lacking a cohesive examination of their combined potential to tackle emerging network challenges. This survey aims to fill that gap by exploring the integration of TinyML and LargeML for 6G and future networks. A key contribution of this work is a thorough exploration of the motivations and requirements for such integration. Our contributions are summarized below.

1) *Background:* This work first provides the background of TinyML and LargeML, highlighting their individual advances and potential for cooperation.
2) *Integration Requirements:* We outline the motivations and requirements for integrating TinyML and LargeML to emphasize their importance for 6G networks.
3) *Efficient Solutions:* We propose and discuss efficient bidirectional integration solutions aimed at enhancing the performance and capabilities of future networks.
4) *Applications of Integrated Systems:* We conduct an extensive review of the applications of integrated TinyML and LargeML systems across various domains to demonstrate their practical benefits and potential impacts.
5) *Discussion of Challenges and Future Research:* From our extensive review, we identify critical challenges and propose several potential directions for future research, including resource allocation, communication efficiency, interplay with LargeML, and standardization.

II. OVERVIEW OF TINYML AND LARGEML

A. Overview of TinyML

TinyML is a lightweight, resource-constrained class of ML designed for environments with limited memory and processing power [3]. TinyML supports diverse communication tasks, and its characteristics are summarized in Table III.

As [29], the TinyML life cycle has three stages as follows.

1) *Stage 1:* Data collection from peripheral devices (sensors or production environments).
2) *Stage 2:* Model deployment using quantization and compression to address hardware heterogeneity and resource constraints.
3) *Stage 3:* Model operationalization, converting trained models into interpretable forms for target devices. TinyML relies on application-specific data formats, prioritizing model customization for device capacity and efficient data transmission (see Fig. 2 for a short overview).

1) TinyML Design: While no standard design guidelines exist, several best practices are applied to guide development.

a) Quantization: This reduces the memory footprint and computation by representing parameters in lower precision (e.g., 8-bit integers versus 32-bit floats) [30]. It can be applied during training (quantization-aware training) or after training (post-training quantization). Variants include uniform and nonuniform quantization for optimizing accuracy, symmetric or asymmetric schemes to reduce quantization error, and Bayesian inference to minimize bit encoding requirements.

b) Pruning: This removes redundant weights, neurons, or layers to shrink NN size [31]. Pruned models typically require fine-tuning to restore accuracy. Common strategies include threshold-based pruning, in which weights below a specified magnitude are discarded. From differences in the model weight matrix, pruning is classified as unstructured (removal of individual weights) or structured (removal of neurons, filters, rows, or columns), guided by redundancy or regularization metrics.

c) Low-Rank Matrix Decomposition: This approximates large weight matrices via techniques like singular value decomposition, with lower rank components. The resulting factorized forms contain fewer parameters, reduce computational complexity, and simplify matrix operations [32]. Albeit this technique can be computationally intensive and more difficult to implement, as the decomposed network requires

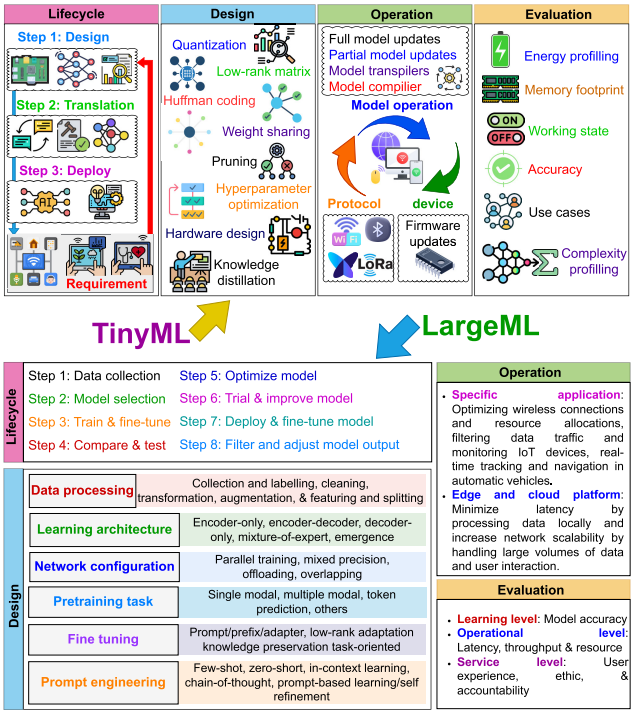

Fig. 2. Overview of TinyML and LargeML, including their life cycle, design, operation, and performance evaluation.

additional fine-tuning to recover accuracy loss, similar to model pruning.

d) Weight Sharing: This reduces parameters across model components to cut storage and computation [33]. For example, shared weights can be arranged by row or column within a weight matrix. However, designing effective sharing configurations is difficult and can lead to unpredictable performance.

e) Algorithm-Architecture Co-Design: This jointly optimizes algorithms with specialized hardware (e.g., TPUs, Cortex-M7, Cortex-M4, Cortex-M0 +, and eDMPv1), for inference efficiency [34]. This joint optimization also enhances computational performance and energy efficiency.

f) Huffman Coding: This compresses model parameters using variable-length encoding based on symbol frequency [35]. When applied to model weights and parameters, this technique can reduce overall model size through efficient encoding.

g) Hyperparameter Optimization: This tunes model parameters, such as learning rate, batch size, activation functions, number of neurons, and network depth, typically via grid search, random search, Bayesian optimization, hyperband (which dynamically allocates resources to promising configurations while terminating underperforming configurations early), genetic algorithms, tree-structured Parzen estimators, and hyperdimensional computing [36]. These techniques aim to improve model performance by minimizing memory and computational overhead while ensuring user-defined accuracy.

h) Knowledge Distillation: This transfers knowledge from a larger "teacher" model to a smaller "student" via: 1) response-based, where students mimic the teacher's final output; 2) feature-based, where students replicate intermediate feature representations; and 3) relation-based, where students learn the structural relationships within the teacher model [37].

2) TinyML Operation: From an operational standpoint, TinyML can be categorized into three main levels: firmware updates, model operation, and communication protocols [38].

a) Firmware Updates: Replace the entire software stack, including the TinyML model and system components. While comprehensive, this process is resource- and time-intensive. Over-the-air (OTA) delivery via lightweight M2M protocols (push/pull) is common [39]. Tools such as RIOT-ML facilitate secure OTA updates, enabling remote, unattended deployment.

b) Model Operations: Several strategies optimize TinyML deployment [38]. *Full model updates* replace the entire ML model via OTA delivery, ideal for major changes, ensuring devices run the latest version. *Partial model updates* modify only specific layers or parameters, reducing network traffic and power use, suitable for fine-tuning on constrained devices [40]. *Model transpilers* (e.g., TensorFlow → TensorFlow Lite Micro), improve cross-hardware compatibility and resource efficiency [41]. *Model compilers* (e.g., uTensor and TinyTS) generate optimized binaries using quantization and pruning, reducing size and improving inference efficiency, fitting within tight memory and computational limits.

c) Communication Protocols: TinyML relies on specialized protocols to optimize data transfer under hardware constraints in IoT ecosystems [14]. For *intradevice* links, inter-integrated circuit and serial peripheral interface protocols support efficient data exchange between MCU-peripheral exchange. For *device-cloud* links, message queuing telemetry transport offers lightweight transfer in low-bandwidth, high-latency environments. *Short-range* links are met by low-power wide-area network (LoWPAN) protocols (Bluetooth, Zigbee, and 6LoWPAN) for wearable and smart sensors, while Wi-Fi supports higher data rates. For *long-range, low-power* links, long-range wide area network excels in remote applications like agriculture and smart cities, offering energy-efficient, kilometer-scale coverage. Sigfox, designed for ultranarrowband IoT and M2M tasks, transmits small, infrequent data packets on unlicensed bands, ensuring low-cost, secure deployments. Similarly, narrowband IoT (3GPP) supports low-data-rate, energy-efficient communication in licensed/unlicensed spectra, enabling scalable, reliable connectivity for massive IoT deployments in challenging environments like underground or indoor settings.

3) Performance Evaluation: Given the diversity of application contexts and strict resource limitations, performance evaluation of TinyML models requires the consideration of both device constraints and learning model characteristics.

a) Memory Footprint: TFLite Micro targets microcontrollers with limited ROM and RAM, using quantization and pruning to reduce model size with minimal accuracy loss [42]. Evaluating ROM and RAM consumption is crucial for assessing TinyML deployments on IoT devices.

b) On/Off Characteristics: TinyML components (sensors, protocols, inference engines) switch between sleep and active power states to save energy [41]. Optimizing the duration and frequency of these transitions minimizes active time, enhancing energy efficiency while preserving functionality.

c) Complexity Profiling: This refers to the analysis of the computational demands of TinyML models. Floating-point operations are more computationally intensive than quantized int8 operations [14], which reduce weight and activation precision to shrink model size and accelerate inference. Complexity

profiling helps quantify the tradeoffs between model size, accuracy, and inference performance [3].

d) Energy Profiling: Energy profiling measures CPU cycles, latency, and total energy consumption during inference [43]. Floating-point models typically consume more energy than quantized int8 models due to the greater computational overhead. Thus, it helps identify energy-efficient TinyML models.

e) Model Evaluation Metrics: Several metrics are used to evaluate TinyML model performance: accuracy, mean absolute error (MAE), root-mean-square error (RMSE), R-Squared, precision, recall, $F1$-score, and cross-entropy loss.

f) Deployment Robustness: Deployment durability ensures the reliability of the TinyML model under a variety of environmental conditions. Two evaluation methods are used [38]: *per-model evaluation* assesses performance in controlled predeployment tests, while *per-operator evaluation* monitors adaptability in real-world environments. Durability testing encompasses diverse datasets, hardware, and environments to ensure consistent performance.

B. Overview of LargeML

LargeML is increasingly integrated into wireless networks to enable autonomous operations via cognitive modules for resource allocation, channel estimation, and mobility management [4]. Its life cycle consists of eight stages as follows.

1) *Stage 1:* Data collection and preprocessing from multiple domains for high-quality training corpus.
2) *Stage 2:* Model architecture design regarding layer types, parameters, and training algorithms.
3) *Stage 3:* Training through pretraining on broad data and fine-tuning for task-specific applications.
4) *Stage 4:* Evaluation with benchmarks, refining architecture, hyperparameters, and datasets.
5) *Stage 5:* Compression to reduce memory and latency while retaining performance.
6) *Stage 6:* Testing and optimization under controlled conditions for inference efficiency.
7) *Stage 7:* Deployment with continuous monitoring, fine-tuning, and adaptation.
8) *Stage 8:* Risk mitigation using adversarial training, fairness constraints, and curated datasets, addressing bias, privacy, security, and misuse concerns.

1) LargeML Design: The design process for LargeML covers six key areas, outlined below.

a) Data Processing: Training LargeML models depends on diverse IoT data, as quality impacts prediction accuracy. The process involves three key steps.

Collection and Labeling: Data is gathered from heterogeneous sources (real-time sensors, logs, and user feedback) and annotated with relevant features (e.g., signal strength, noise, and mobility patterns [44]). Accurate labeling prevents bias and ensures models generalize across operational scenarios [45].

Cleaning, Transformation, and Augmentation: Raw data is cleaned by removing duplicates, correcting errors, and handling missing values. It is then transformed through normalization, encoding, and feature engineering, and represented in formats suitable for models, such as token-based input for text, tree structures for hierarchical relationships, and pixel-based representations for visual data [16]. Augmentation methods, like rotation, flipping, and noise injection, enhance sample diversity and robustness, resulting in a clean and well-structured dataset for downstream tasks.

Featuring and Splitting: It involves feature selection to identify high-impact variables, feature extraction to derive new indicators (e.g., signal variation rates) and capture deeper patterns, as well as data partitioning for training, validation, and test sets to ensure robustness and generalizability [46].

b) Learning Architecture: Future 6G networks aim to support integrated sensing and communication (ISAC), task-oriented communications, and digital twins, requiring LargeML models for efficient real-time data processing [4]. Selecting suitable learning architectures is challenging due to application-specific constraints. Transformers, leveraging attention mechanisms, outperform RNNs in sequence data processing [23]. They use an encoder–decoder structure with stacked sublayers: encoders create context-aware representations, positional embeddings preserve token order, and decoders apply masked or multihead self-attention to assess token relevance. Aside from three typical architectures (encoder only [47], encoder–decoder [23], and decoder only with types of causal decoders [48] and decoders [49]), below are new variants of transformer architectures.

Mixture of Experts: Mixture of Experts (MoE) activates only subsets of model weights, improving efficiency as in switch transformer [50] and GLaM [51]. Performance scales with more experts or larger parameter size, but unstable training can occur due to complex routing operations. This issue can be mitigated by precision tensors, initialization adjustments, and hybrid resource-conditional computation [52].

Emergence Architectures: Standard transformers face quadratic complexity with sequence length. Motivated by this, parameterized state-space models [53] integrate RNN- and CNN-like properties: they process outputs recursively (like RNNs) while encoding sequences in parallel (like CNNs). Techniques such as fast Fourier transform and chunk-wise recurrence can further improve efficiency.

c) Network Configuration: Network setup strongly affects training performance. Common strategies address parallelism, precision, memory, and computation [54].

Parallel Training: Data parallelism synchronizes GPUs using a parameter server/all-reduce gradients; pipeline parallelism splits models across devices; and model parallelism distributes layers/neurons, exchanging intermediate results.

Mixed Precision and Offloading: Mixed precision stores gradients in single-precision (32-bit) while training in half-precision (16-bit) to prevent underflow and ineffective parameter updates. Offloading optimizer states from the GPU to multiple CPUs alleviates memory bottlenecks.

Overlapping and Check-Pointing: Overlapping fetches parameters while computing on previously loaded ones, and checkpointing stores only minimal activations and recomputes the rest to trade memory for compute.

d) Pretraining Tasks: Pretraining equips models with general knowledge that minimizes the need for task-specific training. It involves large-scale data collection, self-supervised learning to capture foundational representations, and fine-tuning for target tasks. LargeML models are pretrained on extensive corpora to learn universal syntactic and semantic features, which are later adapted for downstream applications. Common pretraining workflows are outlined below [55].

Single-Modal: This trains on a single modality (e.g., text, images, and audio) to capture domain-specific patterns. For instance, pretraining on large text corpora enables strong

performance in sentiment analysis, speech classification, and translation.

Multimodal: This combines different modalities to learn richer representations. For example, training image captions with visuals improves tasks such as captioning, visual question answering, and cross-modal retrieval.

Token Prediction: This trains models to predict missing or next tokens, enhancing contextual understanding. Masked language modeling (e.g., BERT) predicts hidden random tokens, while auto-regressive modeling (e.g., GPT) generates tokens sequentially for coherent outputs.

e) Fine-Tuning: After pretraining, a model is fine-tuned on a smaller, task-specific dataset to adjust its parameters for the target application. This process builds on the knowledge acquired during pretraining. In addition to the TinyML strategies discussed in Section II-A1, this section reviews the fine-tuning techniques relevant to LargeML systems [56].

Prompt Tuning: Prompt tuning introduces trainable tokens (prompts) to guide the model's output without modifying its architecture or parameters.

Prefix Tuning: This technique prepends a set of trainable tokens (prefixes) to the model's input and intermediate layers. These prefixes steer the model's attention and interpretative processes without updating the core parameters, enabling task adaptation with low computational overhead.

Adapter Tuning: Adapter tuning inserts small NN modules (adapters) into the model's architecture. During fine-tuning, only the adapters are trained to capture task-specific information [57], while the original model remains unchanged.

Low-Rank Adaptation: Low-rank adaptation (LoRA) enhances transformer models by introducing trainable low-rank matrices to selected layers, keeping original weights frozen [58]. Only lightweight matrices are trained to approximate weight updates, thereby reducing memory and computational costs.

Knowledge Preservation: This technique adapts a pretrained model to new tasks while preserving its valuable preexisting knowledge, mitigating *catastrophic forgetting* through causal-aware methods and PEFT techniques such as LoRA and half fine-tuning. The result is a model that stays effective across tasks without overfitting to new data.

Task Orientation: This technique adapts pretrained models to specific tasks (e.g., dialog systems) by fine-tuning the model's parameters on task-specific data. The incorporation of techniques such as adapter layers and prefix tuning reduces the overall parameter count for training, improving efficiency and lowering resource demands.

f) Prompt Engineering: After fine-tuning, prompt engineering refines how the model interprets and responds to inputs according to task requirements and application contexts. Common types include multiturn instructions, multiple modeling, task planning, tool augmented alignment, and retrieval augmentation. The most widely used prompting techniques in wireless networks are described below [4].

Few-Shot Prompting: This enables the model to generalize from a small number of examples. It is particularly useful when large annotated datasets are unavailable, allowing the model to perform new tasks with minimal additional data.

Zero-Shot Prompting: This exploits the model's pretrained knowledge to perform tasks without prior examples or specific training. The model generates outputs based solely on instructions, drawing from its existing knowledge. In contrast to few-shot prompting, zero-shot prompting provides no examples to guide the model's output.

In-Context Learning: This enables models to perform tasks from prompts alone, without parameter updates. By analyzing examples in the prompt, the model infers task structure and applies it to new inputs, e.g., detecting network intrusions, adapting to threats, or managing resources in real time.

Chain-of-Thought: This enhances reasoning by breaking tasks into intermediate steps, ensuring logical and coherent decision-making. Useful in applications where stepwise analysis accelerates repairing and improves resilience, e.g., fault diagnosis.

Prompt-Based Learning: This leverages pretrained knowledge to generate accurate, contextually relevant responses without further training or parameter adjustments. Well-designed prompts guide task execution, e.g., predicting maintenance needs and preventing outages by identifying failure patterns.

2) LargeML Operation: LargeML follows a multistage process: large-scale data collection and preprocessing, initial training on high-performance hardware, fine-tuning for specific tasks, and postdeployment adaptation through real-time inputs and feedback. This ensures both accuracy and long-term relevance. Its operation varies across domains and input data characteristics from sensors and devices [4]. In wireless communication, LargeML optimizes connectivity and manages network traffic across mobile, Wi-Fi, and satellite links. In the IoT domain, it processes data from smart homes, connected vehicles, and healthcare monitors to enable intelligent automation and real-time decision-making. For example, it serves as an AI control hub in smart homes, supports navigation and diagnostics in vehicles, and enables remote monitoring and personalized care in healthcare. Deployment strategies include edge-based processing to reduce latency and cloud integration for scalability [56]. Continuous monitoring and updates based on user feedback and new data sustain model performance across applications. Integrated into wireless and IoT systems, LargeML supports smart city infrastructure (e.g., traffic and energy optimization), predictive industrial maintenance, and smart agriculture through environmental monitoring.

3) Performance Evaluation: The performance of LargeML can be assessed at three levels, described below.

At the learning level, the focus is on accuracy, output quality, and efficiency. Accuracy is assessed using precision, recall, $F1$-score, and area under the receiver operating characteristic (AUC), which measure classification and discrimination ability [59]. Output quality is gauged using perplexity (confidence in predictions), bilingual evaluation understudy (BLEU), and recall-oriented understudy for gisting evaluation (ROUGE), which compare generated text against reference outputs.

At the *operational level*, performance is measured by typical metrics, such as latency (response time), throughput (requests processed per unit time), and resource use (computational cost), ensuring scalable and reliable operation.

At the *service level*, evaluation focuses on user experience, ethics, and accountability [16], [18]. User experience includes user satisfaction (perception feedback), engagement (interaction depth), and retention rate (continued usage's proportion). Ethical and accountability metrics assess fairness (bias reduction), transparency (explainable decisions), and ethical compliance (privacy, security, and accountability standards).

III. Motivations and Requirements for TinyML–LargeML Integration in 6G Networks

A. Challenges in TinyML

1) Resource Constraints and Energy Efficiency: TinyML devices typically operate with limited processing power, memory, and energy resources, complicating the deployment of complex models. For instance, microcontrollers often lack DRAM and an operating system, with memory capacities ranging from a few hundred kilobytes to several megabytes [10], [60]. Such limitations preclude the use of LargeML without aggressive compression, often forcing tradeoffs between accuracy and feasibility. Furthermore, many TinyML devices are battery-powered, requiring energy-efficient models that minimize inference overhead to extend device lifespan. Achieving low-latency inference while maintaining energy efficiency remains a key challenge, especially for mission-critical 6G applications such as autonomous driving and industrial automation [61].

2) Model Deployment and Data Preprocessing: The deployment of ML models on TinyML devices necessitates compression techniques such as pruning, quantization, and knowledge distillation [62]. Although these approaches reduce memory and computation costs, they may compromise accuracy, particularly when parameters essential for prediction are removed. Quantized optimization introduces additional difficulties due to mixed-precision tensors and the absence of batch normalization layers [63]. Beyond model compression, preprocessing tasks such as normalization, augmentation, and feature extraction require computational and memory resources that are typically unavailable on TinyML devices. This makes handling complex datasets, such as high-resolution images, particularly challenging.

3) Reliability, Privacy, and Security: TinyML devices suffer from limited numerical precision and fabrication-related errors, which can undermine system reliability, especially in mission-critical settings like healthcare, where failures may endanger patient safety [6]. They also face significant privacy and security risks: although they often handle sensitive audio or visual data, their constrained compute capacity prevents the use of stronger protections, such as differential privacy or secure multiparty computation, leaving them more exposed to data leakage and adversarial attacks [64].

B. Challenges in LargeML

1) Resource and Infrastructure Demands: Training and inference of LargeML require specialized hardware such as GPUs or TPUs, along with extensive RAM and storage for managing parameters and activation states [65]. Training often spans days or weeks, incurring high financial and energy costs. Such requirements limit accessibility, particularly for small and medium enterprises or researchers in developing regions, where infrastructure and expertise may be insufficient [66].

2) Latency and Scalability Issues: Due to their size and complexity, LargeML models are prone to high inference latency. Offloading computation to cloud servers introduces further network delays, which are unacceptable for latency-sensitive 6G applications such as augmented reality, virtual reality, and autonomous vehicles [67]. Scaling LargeML is further hindered by the need to manage vast, heterogeneous, and often non-independent and identically distributed (non-IID) datasets across distributed infrastructures. This increases the risks of bottlenecks, degraded model performance, and challenges in real-time processing [68].

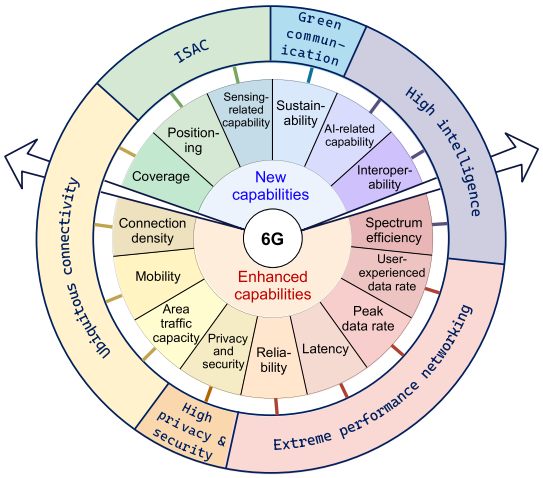

Fig. 3. Envisaged 6G capabilities and pivotal requirements.

3) Deployment, Interpretability, and Adaptability: Deploying LargeML involves costly and complex system integration, including hardware configuration, software optimization, and application-specific tuning. Moreover, these models often operate as "closed boxes," with opaque decision-making processes. This lack of interpretability undermines trust in mission-critical domains such as healthcare and finance, where explainability and accountability are essential [69]. Effective fine-tuning is also challenging: while adaptation to new data domains is necessary, it requires careful balancing to avoid overfitting or underfitting [70].

4) Privacy and Security Concerns: LargeML models typically rely on centralized data aggregation, which concentrates sensitive information in repositories vulnerable to cyberattacks [71]. Inference tasks may also expose confidential data, complicating real-time privacy protection. Furthermore, these models are susceptible to threats such as adversarial attacks, model extraction, and reverse engineering, which can compromise both model integrity and the confidentiality of training data.

C. Vision for 6G

IMT-2030 framework identifies six overarching requirements for 6G networks: ubiquitous connectivity, extreme performance, high intelligence, strong security, green communication, and efficient ISAC (see Fig. 3) [72]. Standalone approaches (whether TinyML or LargeML) are insufficient to satisfy these requirements, necessitating a bidirectional integration paradigm. TinyML supports local data processing at the edge, while LargeML leverages high-capacity infrastructure for global-scale optimization. Together, they enable complementary capabilities aligned with the 6G vision.

1) Ubiquitous Connectivity: 6G must support ultradense networks, with connection densities up to 10^6–10^8 devices/km^2 and data volumes five times higher than 5G [73]. TinyML enables edge devices to process local data in real time, reducing network load, while LargeML, deployed across a *ubiquitous server set* (cloud, edge, and mobile devices) [74], aggregates distributed information to optimize resource allocation and mobility management. This integration supports global coverage across terrestrial, aerial, satellite, and underwater systems, with intelligent handover strategies for high-mobility platforms [75], [76].

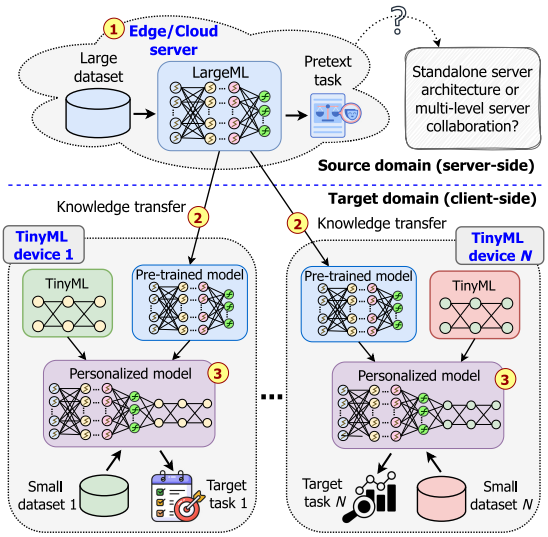

Fig. 4. TinyML–LargeML system with TL.

2) Extreme Performance Networking: Target metrics for 6G include latencies as low as 0.1 ms and reliability levels up to 99.99999, with peak data rates projected to reach 1 Tbps [77]. Bidirectional integration enhances performance by reducing latency at the edge (TinyML) while maintaining complex server-side analytics (LargeML). Model splitting, model transfer, hierarchical distribution, and lightweight fine-tuning enable real-time collaboration across devices and servers, improving throughput, resilience, and accuracy.

3) High Intelligence: 6G envisions distributed intelligence across networks, combining localized processing and global decision-making [77]. TinyML provides real-time inference for edge devices, while LargeML ensures resource-intensive optimization and personalization. This collaboration supports bandwidth-heavy applications such as ISAC and holographic communication [73], where AI-driven spectrum allocation and cognitive technologies improve efficiency and responsiveness.

4) Strong Privacy and Security: As 6G interconnects billions of devices, ensuring data confidentiality and system robustness is paramount. TinyML enhances privacy by performing local preprocessing and transmitting only essential features, thereby reducing exposure of raw data. Concurrently, LargeML systems, supported by high-performance servers, can conduct sophisticated threat detection and adaptive defense using aggregated data from multiple devices [78].

5) Green Communications: With the proliferation of connected devices, energy efficiency is central to sustainable 6G systems. TinyML reduces reliance on energy-intensive servers by enabling low-power, on-device processing. LargeML complements this by optimizing network-wide resource management, minimizing redundant processing, and reducing overall CO_2 emissions [79]. Bidirectional integration also supports predictive maintenance of network infrastructure and the use of energy-harvesting technologies.

6) Efficient ISAC: ISAC represents a defining requirement of 6G, merging communication, sensing, and localization into a unified framework [80]. TinyML supports on-device feature extraction and preliminary sensing tasks, while LargeML performs advanced analysis for object detection, imaging, and mapping. This integration enhances precision, reduces false positives, and enables real-time navigation and environmental awareness, being key for autonomous systems and smart cities.

D. Discussions and Lessons Learned

The transition from 5G to 6G creates a significant opportunity to integrate TinyML and LargeML to enhance AI capabilities in next-generation networks. Section III-A discusses the key challenges of TinyML, while Section III-B examines the limitations of LargeML. As highlighted in Section III-C, a hybrid framework that combines TinyML and LargeML leverages the strengths of distributed and centralized learning, forming a promising paradigm aligned with the 6G vision. In essence, TinyML enables edge-level data preprocessing to reduce bandwidth consumption and latency, whereas LargeML performs deeper analysis for high-accuracy insights. This bidirectional integration, powered by emerging technologies, helps fulfill the six core requirements of 6G networks. Detailed discussions of specific integration approaches are provided in Section IV.

IV. Efficient Bidirectional Integration Solutions for 6G and Beyond Networks

A. Transfer Learning

TL, a subset of ML methods, enables knowledge reuse from pretrained models across related tasks and domains (readers can find more details of this technique in [27]), providing an effective mechanism for TinyML–LargeML integration in 6G networks. By combining large-scale server-based training with lightweight on-device adaptation, TL supports both scalability and personalization. A generic TL framework for integration consists of three main stages (see Fig. 4).

① *Server-Side Training:* A LargeML model f_S is trained on a source-domain dataset $\mathcal{D}_S$ using high-performance cloud or edge servers. The objective is to optimize model parameters, comprising weights and biases, by minimizing the loss function

$$f_S = \arg\min_{f_S} \frac{1}{|\mathcal{D}_S|} \sum_{i \in \mathcal{D}_S} \mathcal{L}\left(\mathbf{y}_i, f_S\left(\mathbf{x}_i\right)\right) \tag{1}$$

where $\{\mathbf{x}_i, \mathbf{y}_i\}_{i \in \mathcal{D}_S}$ represents the training samples, and $f_S(\mathbf{x}_i)$ denotes the model's activation after processing input feature $\mathbf{x}_i$. The loss function $\mathcal{L}(\cdot)$ varies depending on the learning task and may include metrics such as cross-entropy, mean squared error, MAE, or log-likelihood.

Different server-side training architectures can be adopted [81], including: 1) hybrid training (cloud pretraining with edge fine-tuning); 2) distributed training (workload split between cloud and edge); 3) hierarchical processing (edge preprocessing before cloud training); and 4) enhanced availability (separating edge servers during cloud outages). Compared to standalone setups (e.g., two-tiered client-edge/client-cloud), multilevel collaboration offers greater flexibility in balancing communication and computational overhead.

② *Pretrained Knowledge Transfer:* After training in the source domain, selected model components $t_S \subset f_S$ are transferred to the target domain. The effectiveness of this process relies on answering four guiding questions [82], [83] as follows.

1) *What to Transfer:* The transferred knowledge may include model parameters (weights and biases), feature extractors, embeddings, or even relational structures. The choice depends on the similarity between source and target tasks. Parameters are most effective when

tasks are closely related, while feature- or relational-based knowledge suits domains with shared patterns but different label spaces.

2) *When to Transfer:* Knowledge can be transmitted within [84]: 1) *real-time*, immediately after model updates, enabling rapid adaptation in dynamic environments; 2) *periodic/semi-real-time*, following scheduled intervals (e.g., daily and weekly), useful in stable or predictable contexts; and 3) *on-demand*, initiated by the TinyML device when additional knowledge is required, reducing communication overhead in autonomous scenarios.
3) *Where and How to Transfer:* The "where" aspect refers to identifying the target domain where the knowledge will be applied (e.g., TinyML devices). The "how" aspect ensures compatibility and maximizes the effectiveness of knowledge use in the target domain. Effective transfer depends on both domain similarity and the selection of optimization techniques. Transfer is typically more successful between domains with overlapping characteristics.

Examined through these guiding questions, TL can be categorized according to several criteria [83] as follows.

1) *Label Availability:* Transductive (labels only in source), inductive (labels only in target), or unsupervised (no labels in either domain).
2) *Domain Alignment:* Homogeneous (shared feature and label spaces) and nonhomogeneous (otherwise).
3) *Transfer Mechanism:* It includes approaches: 1) *instance-based TL*, reweighting or selecting relevant source samples; 2) *feature-based TL*, learning shared representations via dimensionality reduction or transformations; 3) *parameter-based TL*, reusing pretrained weights and biases as initialization for target training; and 4) *relational-based TL*, transferring structural knowledge such as graphs, relationships, or relational patterns.

③ *Model Personalization and On-Device Adaptation:* Each TinyML device $n \in \mathcal{N} = \{1, \ldots, N\}$ adapts the transferred model using its local dataset $\mathcal{D}_n$. Fine-tuning typically involves freezing pretrained layers t_S while updating TinyML lightweight layers g_n for device-specific tasks. Domain adaptation methods, such as adaptation layers [85] or correlation alignment (CORAL) [86], help reduce distributional shift between $\mathcal{D}_S$ and $\mathcal{D}_n$ such that

$$g_n = \arg\min_{g_n} \frac{1}{|\mathcal{D}_n|} \sum_{j \in \mathcal{D}_n} \mathcal{L}\left(\mathbf{y}_j^n, t_S\left(\mathbf{x}_j^n\right)\right) + \lambda \ell_{\mathcal{D}_S, \mathcal{D}_n} \tag{2}$$

where $\lambda > 0$ controls the tradeoff, and $\ell_{\mathcal{D}_S, \mathcal{D}_n}$ quantifies the discrepancy between domains, enabling personalized learning while keeping all sensitive data on-device.

Recent studies have explored TinyML–LargeML integration via TL across diverse applications. For example, TinyTL [42] uses ProxylessNAS-Mobile for source-domain pretraining and fine-tunes TinyML devices with a once-for-all network, applying parameter-based TL for tasks like object and facial attribute recognition of cars, aircraft, flowers, birds, pets, food, and celebrities. Similarly, Ostrovan [40] introduces a bidirectional integration using CNN pretraining with TensorFlow Lite Micro, pruning, and quantization for in-car presence detection and classification. In [87], a bidirectional integration system with feature-based TL leverages a deep neural network (DNN) source-domain backbone and a CMSIS-NN target

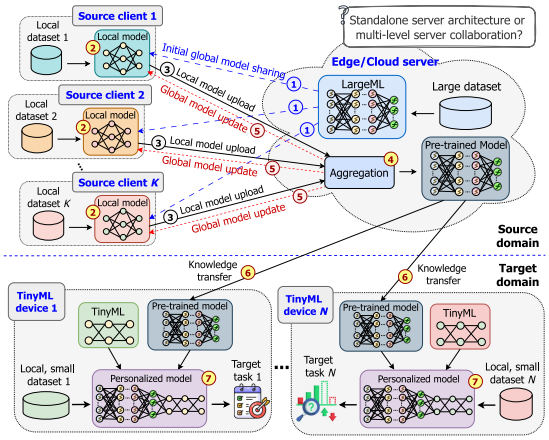

Fig. 5. TinyML–LargeML system with FTL.

model for IoT health trackers, supporting object and activity recognition. Furthermore, Azevedo et al. [88] apply feature-based TL with VGG-16, DenseNet, and MobileNet backbones, deployed through Edge Impulse's EON Tuner for face mask detection. Another study [89] combines a DNN-long short-term memory (LSTM) model with parameter-based TL for soil humidity prediction in smart agriculture, fine-tuned on TinyML devices with TensorFlow Lite Micro and quantization. Hayajneh et al. [90] introduce a feature-based TL bidirectional system for motion classification, using principal component analysis (PCA) for dimensionality reduction and a CNN-LSTM trained on large datasets, with Raspberry Pi as an edge intermediary platform. Finally, Kwon et al. [91] present a feature-based TL framework enhanced with meta-learning, using pretrained MCUNet, MobileNetV2, and ProxylessNAS, then fine-tuned on TinyML devices via a task-adaptive sparse-update method to support few-shot cross-domain classification under dynamic constraints.

B. Federated Transfer Learning

TL methods (Section IV-A) work only when source and target domains share related tasks and some dataset similarities. In 6G systems, these conditions rarely hold because massive device populations generate highly non-IID, heterogeneous, and context-dependent data. Federated transfer learning (FTL overcomes this by combining FL's collaborative training with TL's cross-domain adaptation: source-domain clients train the initial global model, and target-domain clients then adapt and personalize it to their own data and tasks (see [28] for more details).

1) Vanilla FTL: Vanilla FTL is the core form of FTL in which multiple parties jointly train a model without sharing raw data, even when their feature spaces and data distributions differ. It links local feature extractors to a shared prediction model optimized through privacy-preserving gradient exchange. This separation supports scalable knowledge reuse and fine-grained personalization across heterogeneous 6G devices, making it well-suited for privacy-sensitive TinyML–LargeML deployments.

Fig. 5 illustrates the workflow of this FTL-based approach. The FTL process begins with the standard FL protocol, differing primarily in the initialization of the global model. Specifically, the FL phase includes the following steps.

① *Initial Global Model Sharing:* Unlike conventional FL, where the global model starts with random weights, FTL

initializes it with a pretrained LargeML model f_S, trained on the server's large dataset $\mathcal{D}_S$ such that $f_S^{(0)} = \arg\min_{f_S} 1/|\mathcal{D}_S| \sum_{i\in\mathcal{D}_S} \mathcal{L}(\mathbf{y}_i, f_S(\mathbf{x}_i))$. Once trained, $f_S^{(0)}$ is distributed to a source client set $\mathcal{K} = \{1, \ldots, K\}$.

② *Local Training:* In round t, each source client $k \in \mathcal{K}$ exploits its dataset $\mathcal{D}_k$ to train a local copy of the model using the loss function $f_{S,k}^{(t)} = \arg\min_{f_S^{(t-1)}} 1/|\mathcal{D}_k| \sum_{j\in\mathcal{D}_k} \mathcal{L}(\mathbf{y}_j^k, f_S^{(t-1)}(\mathbf{x}_j^k))$, adapting the model to local patterns.

③ *Local Model Update:* Subsequently, each client uploads its updated parameters $f_{S,k}^{(t)}$ to the server for aggregation.

④ *Server-Side Aggregation:* The server aggregates client updates into a new global model using either FedAvg [92], proportion-based weighting [93], or advanced heterogeneity-aware aggregation methods to minimize the global loss: $f_S^{(t)} = \arg\min_{f_{S,k}^{(t)}} \sum_{k\in\mathcal{K}} r_k/|\mathcal{D}_k| \sum_{j\in\mathcal{D}_k} \mathcal{L}(\mathbf{y}_j^k, f_{S,k}^{(t)}(\mathbf{x}_j^k))$, where $r_k = |\mathcal{D}_k|/\sum_{k=1}^{K} |\mathcal{D}_k|$ is the ratio of client k's data size to the total data size.

⑤ *Global Model Update:* The updated global model is then sent back to the source clients.

Steps ② to ⑤ are repeated until the global model reaches satisfactory performance.

Step ⑥ presents a *knowledge transfer* phase. Once the global model has been sufficiently trained and demonstrates strong generalization, the server transfers the useful components of the pretrained global model f_S, denoted $t_S \subset f_S$, to the target TinyML devices (target clients). This transferred model encapsulates the knowledge learned from source clients.

Step ⑦ stands for *model personalization and on-device adaptation*. Each target TinyML device $n \in \mathcal{N} = \{1, \ldots, N\}$ fine-tunes t_S with its smaller, domain-specific dataset $\mathcal{D}_n$. Depending on task similarity, adaptation may involve updating all layers t_S or only a subset. TinyML devices employ lightweight models g_n, trained with the objective function (2). This enables efficient personalization without retraining from scratch, conserving computational resources and reducing latency.

Recent studies have applied FTL to integrate TinyML and LargeML across diverse domains. For example, TinyFedTL [94], a parameter-based FTL framework, pretrains MobileNetV2 and adapts Perf-MobileNet (part of the TinyML Perf Benchmark) for personalized model adaptation on IoT edge devices, improving the quality of experience (QoE) under resource constraints without compromising privacy. FedHealth [95], a parameter-based FTL framework, employs a CNN backbone and fine-tuning for wearable healthcare, supporting activity recognition and auxiliary Parkinson's disease diagnosis with high accuracy and privacy preservation. ACGAN-FTL [96] integrates auxiliary classifier generative adversarial networks for Raspberry Pi-based TinyML devices, predicting the quality of prebaked carbon anodes in industrial production. In [97], a feature-based FTL bidirectional integration system using Genann outperformed standalone FL, TL, and TensorFlow Lite fusion across classification and regression tasks, demonstrating effective device-specific adaptation on Arduino Wi-Fi Rev2, MKR1010, ESP8266, and ESP32.

From a hierarchical FTL perspective, IoTDefender [98] employs three edge servers running FL, aggregated at a cloud server, with CNN-based pretraining and parameter-based TL for intrusion detection in resource-constrained IoT networks. Upon this, HFTL [99] introduces a hierarchical FTL framework that incorporates additional fog servers. These fog servers aggregate models from associated edge servers before forwarding them to a cloud server for further aggregation. Using VGG-16, ResNet50V2, and MobileNet as pretrained models, fine-tuned with an additional DNN, HFTL achieves secure and efficient fault classification in additive manufacturing, with parameter-based TL as the core approach.

2) Advanced FTL: Federated meta-learning (FML) aims to achieve a higher level of generalization and adaptability by enabling models to learn new tasks rapidly through meta-knowledge acquired from diverse clients or domains [100]. Unlike standard FTL, transferring knowledge across related tasks, FML trains models to adapt to previously unseen tasks. This "learning how to learn" capability improves robustness and broadens applicability, supporting both knowledge transfer and efficient task-specific adaptation.

Several studies have explored FML for TinyML–LargeML integration. TinyReptile [101] combines FL and online learning to collaboratively train RNN initializations on resource-constrained IoT devices, enabling rapid adaptation to new targets. Evaluated on sine-wave, Omniglot, and keyword spotting datasets, it demonstrated efficient implementation on Raspberry Pi 4 and Cortex-M4 MCU using parameter-based TL. TinyMetaFed [102] extends this approach with an algorithm that reduces energy consumption and communication overhead, accelerates convergence, and stabilizes training on similar TinyML devices. Building on both TinyReptile and TinyMetaFed, the framework in [103] introduces an advanced prototype with a CNN backbone and MLPerf Tiny Benchmark, supporting personalized adaptation for handwritten character recognition, keyword spotting, and smart-building presence detection. Finally, PMFed [104] applies a pretrained CNN fine-tuned on the Raspberry Pi 3B for personalized adaptation in IoT health monitoring, achieving efficient performance in defibrillation support, atrial fibrillation detection, and human activity recognition.

On the other hand, federated foundation models (FFMs) integrate the strengths of foundation models (FMs) and FL, enabling decentralized clients to collaboratively adapt pretrained LargeML models without exchanging raw data. This approach combines the privacy advantages of FL with the TL benefits of TFMs [21]. Federated fine-tuning (FedFT) of FMs allows each client to locally personalize a shared model while maintaining data confidentiality. To reduce computational overhead, PEFT techniques such as LoRA, adapter modules, and prompt tuning are widely applied [21].

Recent work has advanced FFMs and FedFT for FMs on resource-constrained edge devices. FedPT [105] introduces a federated proxy-tuning method, reconstructing full-sized LoRA modules from predictions for server-side aggregation, reducing computational and communication overhead but struggling under non-IID data. To address heterogeneity, HetLoRA [106] standardizes LoRA modules via zero-padding and rank truncation, improving convergence in tasks such as multisession chat dialog and text summarization. CAFF [107] further improves accuracy and fairness in language and vision classification tasks by combining tiny transformers with a layer-wise FedFT and NN selection strategy. FedD2P [108] distills aggregated knowledge into a prompt generator, enabling frozen FMs to adapt efficiently to downstream vision tasks (e.g., image classification or satellite imagery analysis). Moreover, FwdLLM [109] reduces footprint and accelerates convergence by using BP-free training with PEFT and injecting minor self-generated perturbations into model parameters.

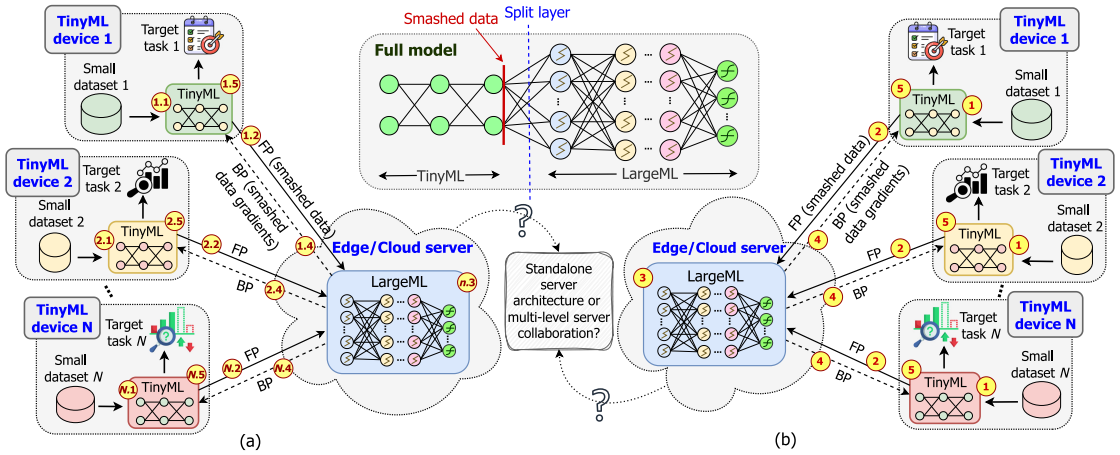

Fig. 6. TinyML–LargeML systems. (a) VSL. (b) PSL.

To further improve efficiency in heterogeneous environments, several studies integrate sparse architectures like MoE with PEFT. For instance, FEDFMSL [110] incorporates a two-stage MoE-based FedFT, training global experts before fine-tuning local experts for improved personalization, with sparsely activated LoRA matrices guided by historical performance tracked in a capability queue. This approach yields performance gains and resource savings in various image classification tasks, such as food recognition, land use, land cover, and human action detection. FEDMOE [111] extends FEDFMSL by dynamically activating submodels via global expert recommendations, achieving faster convergence in NLP tasks such as text classification, summarization, and reading comprehension. Finally, A^{3}SMOE [52] introduces resource-conditional computation, activating sparsely activated MoEs based on client resource availability, surpassing prior LoRA-based techniques in instruction-tuning and complements existing heterogeneous LoRA methods.

C. Split Learning

SL is an ML paradigm for collaborative model training in resource-constrained environments, allowing devices to jointly train a complete model while enhancing learning efficiency (see [74] for more details). SL is particularly suitable for integrating TinyML and LargeML within 6G and future network architectures. By partitioning the model across client devices and servers, SL allows TinyML devices to participate in training and inference tasks without becoming overwhelmed by the computational demands of LargeML while also preserving data privacy. Two main SL variants can be applied to TinyML–LargeML systems: vanilla SL (VSL) and parallel SL (PSL).

1) Vanilla Split Learning: VSL is the foundational form of SL in which the training process proceeds sequentially between clients and a coordinating server [112]. The server can follow a standalone architecture (e.g., client-edge or client-cloud) or a multilevel collaborative model.

Fig. 6(a) illustrates a VSL-based integration of TinyML–LargeML, in which a standalone cloud or edge server collaborates with multiple TinyML devices during training. Typically, a smaller submodel $\mathbf{T}_n$ resides on each TinyML device, and a larger submodel $\mathbf{L}$ is hosted on the server [112]. Each client $n \in \mathcal{N}$ trains on its local dataset $\mathcal{D}_n = \{\mathbf{x}_j^n, \mathbf{y}_j^n\}_{j=1}^{|\mathcal{D}_n|}$. Together, they form a temporary global model $\mathbf{G}_n = \{\mathbf{T}_n; \mathbf{L}\}$. Training proceeds over R outer rounds, where clients interact with the server in a round-robin fashion. For each client, T inner iterations are executed before moving to the next. During each inner iteration, it proceeds as follows.

1) *Step* (n.1)*:* Client n performs forward propagation (FP) through its local model $\mathbf{T}_n$, up to the designated split (or cut) layer. The objective is to minimize its local loss function $\bar{\mathcal{L}}(\mathbf{G}_n|\mathcal{D}_n) \triangleq 1/|\mathcal{D}_n| \sum_{j\in\mathcal{D}_n} \mathcal{L}(\mathbf{y}_j^n, \mathbf{G}_n(\mathbf{x}_j^n))$. This produces intermediate feature representations, called *smashed data*, $\mathbf{T}_n(\mathbf{x}_n)$.
2) *Step* (n.2)*:* The smashed data and labels, $\{\mathbf{T}_n(\mathbf{x}_n), \mathbf{y}_n\}$, are transmitted to the server. Data may be sent in full batches or mini-batches depending on optimal hyperparameters or task-specific requirements [113].
3) *Step* (n.3)*:* The server continues FP using $\mathbf{L}$, computes the gradients of the loss function $\bar{\mathcal{L}}(\mathbf{G}_n|\mathcal{D}_n)$ to generate $\nabla\bar{\mathcal{L}}(\mathbf{L})$ and $\nabla\bar{\mathcal{L}}(\mathbf{T}_n(\mathbf{x}_n))$. Then, it performs server-side local backward propagation (BP) by updating $\mathbf{L} \leftarrow \mathbf{L} - \eta_S \nabla\bar{\mathcal{L}}(\mathbf{L})$, where η_S is the server learning rate.
4) *Step* (n.4)*:* The server sends the smashed data gradients $\nabla\bar{\mathcal{L}}(\mathbf{T}_n(\mathbf{x}_n))$ back to client n.
5) *Step* (n.5)*:* Client n updates its local submodel as $\mathbf{T}_n \leftarrow \mathbf{T}_n - \eta_n \nabla\bar{\mathcal{L}}(\mathbf{T}_n(\mathbf{x}_n))$, where η_n is the client learning rate. For consistency, the terms η_S and η_n are reused with the same definitions in subsequent sections.

This exchange completes one inner iteration. During the outer iterations, the aim is to minimize the global loss function $\min_{\{\mathbf{G}_n\}\forall n} \sum_{n\in\mathcal{N}} |\mathcal{D}_n| \bar{\mathcal{L}}(\mathbf{G}_n|\mathcal{D}_n) \big/ \sum_{n\in\mathcal{N}} |\mathcal{D}_n|$. After convergence, TinyML devices can employ the global model $\mathbf{G}$ for inference on a designated target task.

The integration of VSL-based TinyML with LargeML has enabled a diverse range of applications for tasks in resource-constrained environments. Specifically, a two-stage wireless channel adaptive split inference method [114] addresses edge device memory and energy limits by dynamically adjusting split points based on real-time channel gain, improving privacy, inference accuracy, and adaptability. Similarly, ARES [115], an adaptive resource-aware SL scheme, mitigates stragglers through device-specific split layers, balancing energy use, training time, and throughput variability. An adaptive SL algorithm based on Lyapunov theory [116] further reduces training delay and energy consumption by dynamically selecting split layers and allocating server-side resources in edge networks.

When extended to multilevel server collaboration, VSL can be deployed via multihop SL [74], where cloud and edge servers form a hierarchical chain. In this setup, initial layers run on TinyML devices, intermediate layers on distributed servers, and final layers on a central server [117]. During training, inference proceeds forward through the network while BP sends gradients backward to update each node's parameters [118], [119]. With N TinyML devices and K servers, each inner iteration requires $2NK$ inference and BP steps, highlighting the need for latency-efficient algorithms.

Beyond serial data flow, centralized smashed data routing [74] can also be implemented in 6G networks by integrating ad hoc data routing and centralized model partitioning, enabled through software-defined networking (SDN). This approach must consider bandwidth, memory, and computing constraints. Effective server coordination is crucial for compatibility between model segments, especially with non-IID data distribution across TinyML devices, necessitating periodic synchronization of model parameters for consistency.

2) Parallel Split Learning: The sequential training process of VSL leads to significant training latency and struggles to scale with more TinyML devices, particularly with the anticipated connectivity in 6G networks. In addition, the non-IID nature of client data can bias model performance. PSL addresses these issues by allowing multiple TinyML devices to train their models concurrently, which reduces training time [120], offers "semi-scalability" with more devices, and helps mitigate the effects of non-IID data in 6G environments.

Fig. 6(b) illustrates the workflow of PSL-based TinyML–LargeML systems under a standalone server architecture. Unlike VSL-based bidirectional integration, which requires N sequential rounds of FP and BP, PSL completes training with a single round, regardless of N. This is the basis of semi-scalability, meaning the number of propagation rounds is scalable on the client side. However, the server-side training workload (i.e., smashed data gradients and computational workload) still scales proportionally with N. The inner-iteration training process of a PSL-based TinyML–LargeML system proceeds as follows.

① *Simultaneous Client-Side Local FP:* Each TinyML device $n \in \mathcal{N}$ trains on its local dataset $\mathcal{D}_n$ using its model segment $\mathbf{T}_n$, generating smashed data $\mathbf{T}_n(\mathbf{x}_n)$ at the designated split layer.
② *FP of Smashed Data:* Each device transmits its smashed data and labels, $\{\mathbf{T}_n(\mathbf{x}_n), \mathbf{y}_n\}$, to the server. The server receives these inputs concurrently from all devices.
③ *Server-Side Local FP and BP:* The server trains the concatenated smashed data $\mathbf{S} = [\mathbf{T}_1(\mathbf{x}_1); \ldots; \mathbf{T}_N(\mathbf{x}_N)]$ using its model segment $\mathbf{L}$. Concurrent training accelerates the overall local FP and generates the model output $\hat{\mathbf{y}} = \mathbf{L}(\mathbf{S})$. Given the predicted result $\hat{\mathbf{y}}$ and the concatenated ground-truth labels $\mathbf{y} = [\mathbf{y}_1; \ldots; \mathbf{y}_N]$, the server computes the loss value $\bar{\mathcal{L}}(\mathbf{y}, \hat{\mathbf{y}})$. Further process includes calculating gradients for updating $\mathbf{L}$, i.e., $\nabla\bar{\mathcal{L}}(\mathbf{L}(\mathbf{S}))$ and for each set of smashed data, i.e., $\nabla\bar{\mathcal{L}}(\mathbf{T}_n(\mathbf{x}_n))$ before performing the local BP as $\mathbf{L} \leftarrow \mathbf{L} - \eta_S \nabla\bar{\mathcal{L}}(\mathbf{L}(\mathbf{S}))$.
⑤ *BP of Smashed Data Gradients:* The computed gradients $\nabla\bar{\mathcal{L}}(\mathbf{T}_n(\mathbf{x}_n))$ are returned to each respective TinyML device in parallel.
⑥ *Simultaneous Client-Side Local BP:* Each TinyML device performs local BP and updates its model parameters, i.e., $\mathbf{T}_n \leftarrow \mathbf{T}_n - \eta_n \nabla\bar{\mathcal{L}}(\mathbf{T}_n(\mathbf{x}_n))$.

These training steps are repeated over consecutive rounds until the model converges, i.e., by minimizing the global loss function

$$\min_{\mathbf{L},\{\mathbf{T}_n\}_{\forall n}} \sum_{n\in\mathcal{N}} \sum_{j\in\mathcal{D}_n} \mathcal{L}\left(\mathbf{y}_j^n, \mathbf{L}\left(\mathbf{T}_n\left(\mathbf{x}_j^n\right)\right)\right) \frac{1}{\sum_{n\in\mathcal{N}} |\mathcal{D}_n|}. \tag{3}$$

After convergence, each TinyML device fine-tunes the model to its specific task. However, two potential issues arise as follows.

1) Although PSL helps prevent overfitting caused by aligning the end-to-end output model too closely with the distribution of the previous client, the highly non-IID nature of data across TinyML devices can reduce the convergence rate. This may require server or client-side aggregation strategies. Relevant solutions include EDGESPLIT [121] and FSL.
2) The server assumes the primary training workloads from N TinyML devices, which does not scale as $N \to \infty$. Although cloud/edge servers are generally powerful, they could still become bottlenecks in the massive connectivity expected in 6G systems. An efficient variant, PSL (EPSL), addresses this issue [122].

Further discussion of EDGESPLIT and EPSL is given in Section IV-C3; details of FSL are provided in Section IV-D.

Recent studies have applied PSL to TinyML–LargeML systems [123], [124], [125]. A privacy-sensitive PSL framework in [123] enables PSL training across multiple resource-constrained devices, mitigating overfitting from non-IID training orders and varying dataset sizes on resource-constrained devices while preserving privacy. ADASPLIT [124] enhances scalability in heterogeneous, resource-constrained systems by minimizing on-device computation, optimizing server-side processing, and reducing via sparse model updates without requiring server gradients on clients. In [125], PSL-based bidirectional integration is modeled as a multiplayer bargaining problem to identify optimal split-layer strategies, balancing computation, transmission energy, training time, and data privacy, while showing robustness on non-IID datasets.

In PSL-based TinyML–LargeML systems involving multilevel server collaboration, the architecture extends beyond simple client-edge or client-cloud configurations by incorporating multiple servers or hierarchical layers of servers that collaborate during the training process. Such setups require server orchestration and aggregation strategies, necessitating advanced PSL schemes to coordinate interactions between server layers and integrate TinyML with LargeML seamlessly. Detailed in Section IV-C3, this class of schemes is referred to as advanced PSL, where parallel-server PSL [126] and multihop PSL [127] serve as pioneering frameworks. These frameworks support the development of more sophisticated server orchestration and aggregation methods.

3) Advanced Parallel Split Learning: This section introduces EDGESPLIT and EPSL, which are advanced PSL schemes designed to address the highly non-IID nature of data across TinyML devices. Both schemes consider model aggregation at the server side. EPSL further provides full scalability. The details of each scheme are described below.

EDGESPLIT [121] is an advanced PSL variant designed to address the challenge of highly non-IID data distributed across TinyML devices. Non-IID data causes local models trained on individual TinyML devices to diverge, reducing generalization and degrading global model performance. EDGESPLIT introduces a coordinated learning approach across diverse datasets

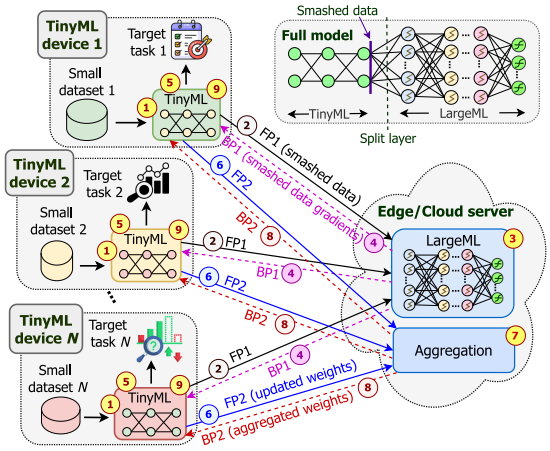

Fig. 7. Integrated TinyML–LargeML system with EDGESPLIT.

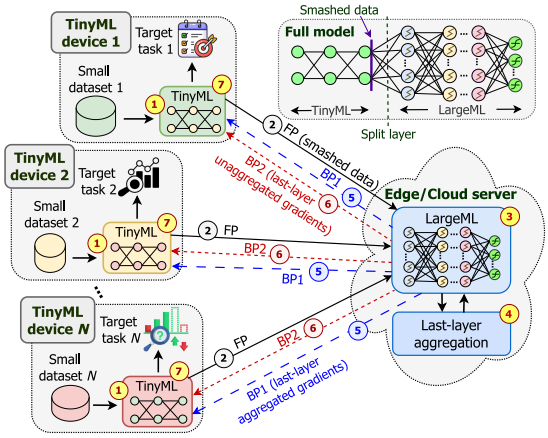

Fig. 8. Integrated TinyML–LargeML system with EPSL.

while preserving the core benefits of PSL. Fig. 7 illustrates the inner-iteration training steps of EDGESPLIT, which begins by a standard PSL procedure: ① Simultaneous client-side local FP; ② FP1 of smashed data; ③ server-side local FP and local BP; ④ BP1 of smashed data gradients; and ⑤ simultaneous client-side local BP. The subsequent steps introduce the key advances of EDGESPLIT over conventional PSL.

① *FP2 With Updated Weights:* After client-side local BP, each TinyML device forward its updated model weights $\mathbf{T}_n$ to the server. These weights incorporate the local learning adjustments made by each device based on its non-IID data.
② *Server-Side Aggregation:* The server aggregates all updated weights received from TinyML devices using an algorithm, such as FedAvg [92]

$$\mathbf{T}_{\text{FedAvg}} = \sum_{n\in\mathcal{N}} |\mathcal{D}_n|\mathbf{T}_n \Big/ \sum_{n\in\mathcal{N}} |\mathcal{D}_n|. \tag{4}$$

This step harmonizes learning across all devices by combining their weights and enables the server to produce a more generalized model that reflects the diversity of data distributions.
③ *BP2 of Aggregated Weights:* The server broadcasts the aggregated weights $\mathbf{T}_{\text{FedAvg}}$ to all TinyML devices, ensuring that each device updates its model with globally informed weights that reduce the impact of non-IID data.
④ *Updated Client-Side Model:* Each TinyML device receives the globally aggregated model $\mathbf{T}_{\text{FedAvg}}$ and updates its weights accordingly, i.e., $\mathbf{T}_n \leftarrow \mathbf{T}_{\text{FedAvg}}\ \forall n$, to synchronize all devices with a consistent model.

This iterative process continues until convergence, guided by the global objective function defined in (3). During the inference phase, each TinyML device adapts the converged model to its specific target task.

EDGESPLIT is used for selected TinyML devices. In the context of widespread 6G connectivity, device heterogeneity is common, leading to variations in computing power, data distribution, and channel conditions. Cluster-based PSL (CPSL) [128] addresses this by clustering devices based on shared characteristics like computing capabilities. This results in clusters with stable data distributions, enabling more efficient training and enhanced model performance.

Conventional PSL methods like EDGESPLIT and CPSL scale well on the client side but still leave heavy computation on the server. EPSL [122] addresses this imbalance with a revised aggregation strategy that lowers server-side cost and training latency, achieving full end-to-end scalability. Its main steps are shown in Fig. 8 and summarized below.

① *Simultaneous Client-Side Local FP:* Each TinyML device $n \in \mathcal{N}$ independently trains on its local dataset $\mathcal{D}_n$ using its local model segment, $\mathbf{T}_n$. The device generates smashed data $\mathbf{T}_n(\mathbf{x}_n)$ at the designated split layer. This step is executed in parallel across all devices.
② *FP of Smashed Data:* Devices transmit their smashed data and labels, $\{\mathbf{T}_n(\mathbf{x}_n), \mathbf{y}_n\}$, to the server. The server processes these multiple smashed data streams concurrently.
③ *Server-Side Local FP:* The server concatenates the smashed data $\mathbf{S} = [\mathbf{T}_1(\mathbf{x}_1); \ldots; \mathbf{T}_N(\mathbf{x}_N)]$, then forward this data through its model segment $\mathbf{L}$ to generate the predicted output $\hat{\mathbf{y}} = \mathbf{L}(\mathbf{S})$.
④ *Gradient Aggregation and Server-Side Local BP:* The server uses the loss $\bar{\mathcal{L}}(\mathbf{y}, \hat{\mathbf{y}})$ to compute the last-layer activation gradients $\mathbf{G}_a$, where $\mathbf{y} = [\mathbf{y}_1; \ldots; \mathbf{y}_N]$ is the concatenated ground-truth label vector. Aggregating these activation gradients compresses the information passed back to clients, which substantially reduces server-side computation, lowers training latency, and improves scalability [122]. The server then computes its own model gradients, denoted $\mathbf{G}_S$. The parameter update for the server-side model is expressed as $\mathbf{L} \leftarrow \mathbf{L} - \eta_S \mathbf{G}_S$.
⑤ *BP1 (Aggregated Gradients):* The server broadcasts the aggregated activation gradients $\mathbf{G}_a$, computed at the split layer by performing BP on the last-layer aggregated activation gradients, to all participating TinyML devices.
⑥ *BP2 (Unaggregated Activation Gradients):* To enable more precise updates, the server also sends the unaggregated activation gradients, denoted $\mathbf{G}_u$, computed at the split layer by performing BP on the last-layer aggregated activation gradients, to the respective TinyML devices.
⑦ *Client-Side Local BP:* Each TinyML device uses the received gradients $\mathbf{G}_a$ and $\mathbf{G}_u$ to compute its local gradient $\mathbf{G}_n$ and update its model segment $\mathbf{T}_n \leftarrow \mathbf{T}_n - \eta_n \mathbf{G}_n$.

The EPSL training process is repeated until convergence. Similarly, (3) can be used as the global objective function for the EPSL training process. During inference, each TinyML device adapts the trained model to its specific task, enabling personalized performance on its local data. Compared to standard PSL, EPSL reduces BP computation and communication complexity from $\mathcal{O}(N)$ to $\mathcal{O}(1)$, achieving full scalability [74].

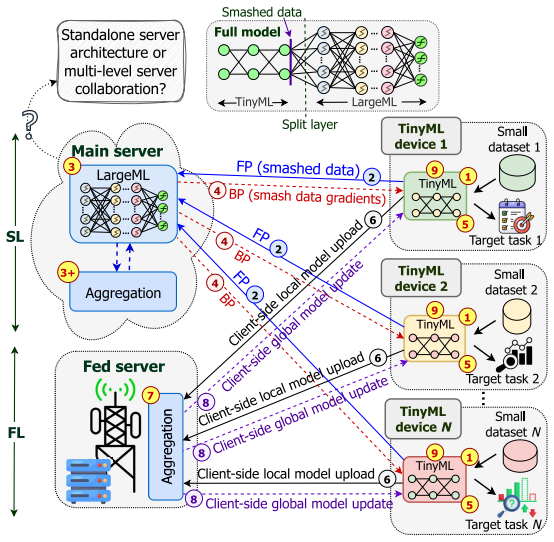

Fig. 9. Integrated TinyML–LargeML system with FSL.

EPSL also introduces a tunable aggregation ratio, denoted r, in the BP phase, enabling a tradeoff between computational and communication efficiency while maintaining learning accuracy. When $r = 0$, EPSL reverts to PSL.

From the PSL-based multilevel server collaboration framework introduced in Section IV-C2, two extensions have been proposed: parallel-server PSL and multihop PSL. In parallel-server PSL [126], multiple servers concurrently process model segments for resource-constrained clients, requiring orchestration management and joint optimization of client-server assignments and scheduling to minimize training time. Due to its NP-hard nature, two scalable solutions are suggested: 1) a decomposition-based method utilizing structural symmetry, and 2) a lightweight alternative with low computational overhead. Conversely, multihop PSL [127] enhances traditional multihop SL with pipelined, parallel execution. Each locally trained TinyML device generates smashed data at a split layer and sends it to the first server, where inference and backpropagation occur sequentially, akin to multihop SL in Section IV-C1. After backpropagation, devices receive synchronized gradient updates for aggregation. Unlike multihop SL, multihop PSL reduces inference and backward steps to $(2K + 2)$, independent of the number of devices N.

D. Federated Split Learning

PSL parallelizes client-side model training (see Section IV-C2) while handling highly non-IID client data, effectively blending the advantages of split learning and federated learning [129]. Federated split learning (FSL) differs from PSL, EDGESPLIT, and EPSL by distributing responsibilities across two cooperating servers: a main cloud server and a Fed server that runs FedAvg. In FSL, the Fed server aggregates updated local models from clients using FL-based techniques (see [130] for more details).

1) Vanilla FSL: Vanilla FSL, the simplest and foundational form of FSL, merges the core ideas of SL, where the model is cut at a chosen layer, and clients only compute the early part of the network, with FL, where many clients participate in training while keeping their data local. This process is illustrated in Fig. 9, with the key steps as follows.

① *Simultaneous Client-Side Local FP:* Each TinyML device $n \in \mathcal{N}$ performs local FP on its model segment, $\mathbf{T}_n$, using its local dataset $\mathcal{D}_n$ to generate smashed data $\mathbf{T}_n(\mathbf{x}_n)$ at the split layer.
② *FP of Smashed Data:* Devices concurrently transmit the smashed data and corresponding labels $\{\mathbf{T}_n(\mathbf{x}_n), \mathbf{y}_n\}$ to the main server for further centralized processing.
③ *Main-Server Processing:* The main server processes the smashed data in both local FP and BP using either strategies of SplitFedV1 (when the server has substantial computing capacity and can efficiently execute parallel processing) or SplitFedV2 (personalized updates on the main server are essential, and client data is relatively homogeneous) [129], [131]. In SplitFedV1, each client's smashed data is separately processed and parallelized on its model segment $\mathbf{L}$, where the loss value $\bar{\mathcal{L}}(\mathbf{y}_n, \hat{\mathbf{y}}_n)$ and its gradient $\nabla\bar{\mathcal{L}}(\mathbf{y}_n, \hat{\mathbf{y}}_n)$ are calculated based on the predicted output $\hat{\mathbf{y}}_n = \mathbf{L}(\mathbf{T}_n(\mathbf{x}_n))$ and ground-truth label $\mathbf{y}_n$. In the subsequent steps, the server aggregates the gradients of the smashed data and updates the server-side model using the FedAvg algorithm [92], expressed as $\mathbf{L} \leftarrow \mathbf{L} - \eta_S \sum_{n\in\mathcal{N}} |\mathcal{D}_n| \nabla\bar{\mathcal{L}}(\mathbf{y}_n, \hat{\mathbf{y}}_n) \Big/ \sum_{n\in\mathcal{N}} |\mathcal{D}_n|$. In SplitFedV2, the server processes client smashed data sequentially on $\mathbf{L}$, following a randomly determined client order represented by the list $\mathcal{S}$ and the computation of $\nabla\bar{\mathcal{L}}(\mathbf{y}_n, \hat{\mathbf{y}}_n)$ akin to SplitFedV1. Instead of aggregation, the server sequentially process the data in the order defined by $\mathcal{S}$ to update the global model after each local FP using the rule $\mathbf{L} \leftarrow \mathbf{L} - \eta_S \nabla\bar{\mathcal{L}}(\mathbf{y}_n, \hat{\mathbf{y}}_n)$ and continuing until all clients in the list $\mathcal{S}$ are processed.
④ *BP of Smashed Data Gradients:* After performing local FP and BP on the server-side model, the main server returns the evaluation of $\nabla\bar{\mathcal{L}}(\mathbf{y}_n, \hat{\mathbf{y}}_n)$ to TinyML devices.
⑤ *Client-Side Local BP:* Upon receiving the gradients from the main server, each TinyML device $n \in \mathcal{N}$ calculates $\nabla\bar{\mathcal{L}}(\mathbf{T}_n(\mathbf{x}_n))$ and performs local BP, updating its model as $\mathbf{T}_n \leftarrow \mathbf{T}_n - \eta_n \nabla\bar{\mathcal{L}}(\mathbf{T}_n(\mathbf{x}_n))$.
⑥ *Client-Side Local Model Upload:* Devices upload their updated local model to the Fed server for aggregation.
⑦ *Fed-Server Aggregation:* The Fed server aggregates the client-side local models using (4) to generate a global client-side model $\mathbf{T}_{\text{FedAvg}}$ that incorporates updates from all participating TinyML devices.
⑧ *Client-Side Global Model Update:* $\mathbf{T}_{\text{FedAvg}}$ is returned to all TinyML devices.
⑨ *Client-Side Local Model Synchronization:* Device update their local model with the rule $\mathbf{T}_n \leftarrow \mathbf{T}_{\text{FedAvg}}$ $\forall n$. This step ensures that all devices converge toward improved and consistent model performance.

This FSL cycle repeats iteratively until the overall model converges. During inference, each TinyML device adapts the converged model to its specific target task.

Several recent studies have explored the use of FSL for integrating TinyML and LargeML in a range of applications [132], [133], [134], [135]. Building upon SplitFedV1, SPLITGP supports bidirectional training for personalized client models and generalized server models, surpassing baselines [132]. Similarly, ESFL enhances training stability by incorporating prior global models, using an iterative algorithm to solve its NP-hard optimization, improving efficiency over standard FL, SL, and FSL [133]. For SplitFedV2, ADAPTSFL dynamically adjusts model splitting and aggregation to optimize latency and convergence on resource-constrained devices, outperforming benchmarks [134]. Meanwhile, FEDSL enables

robust healthcare analytics on wearables, excelling in medical imaging across diverse data [135]. Furthermore, hierarchical FSL frameworks boost efficiency: one trains multiple edge server-client pairs with cloud aggregation for resilience [136], while another uses parallel multiclient setups for enhanced performance on IoT devices [137].

2) Federated Split Models and Fine-Tuning: Federated split FMs (FSFMs) extend the FL framework by partitioning large FMs across clients and servers, enabling distributed training and inference with reduced local resource requirements [138]. Clients train or fine-tune distinct FM segments collaboratively, preserving privacy by not exposing raw data and lowering computational overhead. Federated split fine-tuning (FedSFT) further enables client-side personalization of model components while retaining global knowledge through coordinated updates. Compared to standard FL and FFMs, FSFMs improve scalability by exchanging only compact smashed data rather than full model parameters [138]. These features position FSFMs and FedSFT as particularly suitable approaches in 6G scenarios characterized by device heterogeneity, limited computational capacity, and stringent privacy requirements.

For instance, SFPROMPT [139] is a prompt-based FedSFT framework for privacy-sensitive, resource-constrained edge environments. By dividing pretrained FMs between server and client and integrating soft prompts with local data pruning, it improves FedFT efficiency while significantly reducing computation and communication costs in image classification. FEDSPLITX [140] introduces fine-grained FSFM with multiple model partition points to accommodate diverse client capabilities in heterogeneous, resource-constrained edge environments. Auxiliary networks at each partition reduce latency and communication overhead, enhancing image classification accuracy. FEDVZ [141] defends against gradient inversion in FSFMs using vision transformers by replacing BP with a zeroth-order optimization forward pass, enabling secure, efficient training in constrained environments. Finally, PRINCE [142] incentivizes high-quality device participation from multiple FSL tenants via strategic pricing, boosting FedSFT performance in tasks such as image classification, sentiment analysis, speech-to-text, and question answering.

E. Discussions and Lessons Learned

While Table IV outlines their key features and advances, lessons learned from the TinyML–LargeML integration provide valuable insights for future research.

First, TL-based integration (Section IV-A) fine-tunes pretrained models on TinyML devices by updating only a subset of parameters, reducing computational overhead and enabling on-device adaptation. However, it assumes closely related source and target domains, which limits generalizability.

Second, FTL-based integration (Section IV-B) enables client-level personalization under highly non-IID data, aligning well with 6G ubiquitous connectivity. FML further enhances adaptability through meta-learning, while FFMs combine FMs and FL to support collaborative, privacy-preserving, and decentralized FedFT; efficiency is improved via PEFT, prompt tuning, and MoE to handle model heterogeneity.

Third, SL-based integration (Section IV-C) allows resource-constrained TinyML devices to participate in full model training, but VSL suffers from high latency, limited scalability, and overfitting risks due to sequential training. PSL mitigates these issues through parallelization and partial non-IID handling. Building on PSL, EDGESPLIT improves convergence under highly non-IID settings, while EPSL further reduces server-side workload and training latency, achieving full scalability.

Finally, FSL-based integration (Section IV-D) combines SL and FL to enable flexible aggregation and improved convergence under highly non-IID data, with SplitFedV1/V2 serving as foundational designs. Extensions such as FSFMs and FedFT of split FMs support scalable, privacy-preserving personalization via lightweight smashed-data exchange, though their SL-based architectures remain sensitive to communication latency and network conditions.

Notably, interdisciplinary collaboration continues to advance and refine TinyML–LargeML integration strategies. Active industry participation is equally important to ensure that proposed solutions are both innovative and practical for real-world deployment. Although 6G has not yet been officially standardized, and it remains uncertain whether these approaches will fully satisfy the requirements of 6G and future networks, they represent meaningful progress toward next-generation intelligent wireless networks.

V. APPLICATIONS OF TINYML–LARGEML INTEGRATION

A. Data Privacy and Network Security

Toward Network Privacy: Model inversion attacks pose major risks when handling sensitive data, and integrating TinyML with LargeML provides a privacy-preserving architecture by keeping inference and feature extraction on-device while sending only abstracted representations or model updates to the cloud [10], [144]. TinyML supports established techniques such as federated learning [10] and differential privacy [145], while its combination with LargeML enables stronger ecosystem-wide protection through FTL [146], FML [100], and FSL [129] (see Section IV). This integration is especially important in healthcare [87], [95], [104], [135], where TinyML wearables process physiological signals locally and LargeML servers use aggregated insights for diagnosis and prediction; similar benefits appear in smart agriculture [89], where on-site analysis protects proprietary practices while cloud models improve forecasting, and in smart homes and cities [103], where local inference preserves user privacy while LargeML enhances personalization and resource optimization.

Toward Network Security: TinyML devices operate under strict memory and compute limits, often two to three orders of magnitude below typical IoT or edge hardware [10], [60], [147], making conventional security mechanisms impractical, yet their integration with LargeML enables a multilayered 6G defense in which TinyML performs lightweight, real-time anomaly detection at the edge [148] while LargeML correlates traffic patterns across devices to uncover complex attacks [143], as demonstrated by IOTDEFENDER [98], which applies FTL across edge servers and cloud CNNs for intrusion detection under hardware constraints. To secure communication, TinyML supports lightweight encryption on-device [149], while LargeML manages key distribution and protocol orchestration, dynamically updating keys to reinforce communication channels [150]. This hybrid approach balances strong security with device limitations and underpins the 6G privacy-and-security ecosystem illustrated in Fig. 10.

B. Network Management

The rise of IoT devices and mobile users has pushed 5G networks to support ultrareliable low-latency communication

TABLE IV
SUMMARY OF TINYML–LARGEML INTEGRATION SOLUTIONS

Sol.	Procedures	Characteristics	Representatives	Advances
TL	(1) Server-side training → (2) Pre-trained knowledge transfer → (3) Model personalization and on-device adaptation	- Aligns well with resource-constrained TinyML devices - Requires ideal conditions: source and target domains with distinct yet closely related datasets and tasks	TINYTL [42], [40], [87], [88], [89], [90], [91], and [143]	Multi-level server collaboration: Hybrid training, distributed fashion, hierarchical processing, and enhanced availability
FTL	(1) Model initialization → (2) Source-client-side local training → (3) Local model upload → (4) Aggregation → (5) Global model update → (6) Knowledge transfer → (7) Model personalization and on-device adaptation (Steps (2) – (5) iterate until convergence)	- Enables collaborative learning and distinguishes between source and target clients - Combines the strengths of FL and TL, efficient in highly non-IID data across TinyML devices	TINYFEDTL [94], FEDHEALTH [95], ACGAN-FTL [96], and [97]	- Hierarchical FTL: IOTDEFENDER [98], and HFTL [99] - FML: [101]–[104] - FFM and FedFT of FMs: [52], [105]–[111]
VSL	(1.1) First client local FP → (1.2) FP → (1.3) Server-side local FP and BP → (1.4) BP → (1.5) First client local BP → ⋯ → (N.1) Last client local FP → ⋯ → (N.5) Last client local BP → (6) On-device adaptation (Steps (1.1) – (N.5) iterate until convergence)	- Enables collaborative learning with sequential training among participating TinyML devices - Incurs high training latency, lacks scalability, overfits to the last trained client	ARES [115], frameworks in [114] and [116]	Multi-level server collaboration: Multi-hop SL [117] and centralized ad-hoc data routing [74]
PSL	(1) Simultaneous client-side local FP → (2) FP → (3) Server-side local FP and BP → (4) BP → (5) Simultaneous client-side local BP → (6) On-device adaptation (Steps (1) – (5) iterate until convergence)	- Enables collaborative learning with parallel training among TinyML devices - Significantly reduces training latency, offers "semi-scalability", and partially addresses non-IID data issues	ADASPLIT [124], frameworks in [123] and [125]	- Multi-level server collaboration: Parallel-server PSL [126] and multi-hop PSL [127] - Advanced PSL: EDGESPLIT [121] and EPSL [122]
EDGE SPLIT	(1) Simultaneous client-side local FP → (2) FP1 → (3) Server-side local FP and BP → (4) BP1 → (5) Client-side local BP → (6) FP2 → (7) Server-side aggregation → (8) BP2 → (9) Updated client-side model → (10) On-device adaptation (Steps (1) – (9) iterate until convergence)	- An advanced variant of PSL with additional server-side aggregation - Improves convergence rate in highly non-IID data environments while maintaining PSL benefits	[121]	CPSL [128] for clustering multiple groups of TinyML devices
EPSL	(1) Simultaneous client-side local FP → (2) FP → (3) Server-side local FP → (4) Last-layer gradient aggregation and BP → (5) BP1 → (6) BP2 → (7) Client-side local BP → (8) On-device adaptation (Steps (1) – (7) iterate until convergence)	- An advanced variant of PSL with modified server-side aggregation - Maintains PSL benefits, reduces computational workload and training latency, offers full scalability	[122]	
FSL	The following procedures are for both **SplitFedV1** and **SplitFedV2**, except that step (3+) is specified to SplitFedV1: (1) Simultaneous client-side local FP → (2) FP → (3) Main-server-side local FP → (3+) *Main-server-side aggregation and local BP (only for SplitFedV1)* → (4) BP → (5) Client-side local BP → (6) Client-side local model upload → (7) Fed-server aggregation → (8) Client-side global model update → (9) Client-side local update → (10) On-device adaptation (Steps (1) – (9) iterate until convergence)	- **SplitFedV1**: An advanced variant of PSL, with aggregation on both server and client sides through main and Fed servers; Combines the strengths of FL and SL, improves convergence rate in highly non-IID data environments - **SplitFedV2**: An advanced variant of PSL with client-side aggregation through Fed server support; Combines the strengths of FL and SL, improves convergence in highly non-IID data environments	ADAPTSFL [134], FEDSL [135], SPLITGP [132], and ESFL [133]	- Hierarchical FSL: Frameworks in [136] and [137] - FSFMs and FedSFT of FMs: SFPROMPT [139], FEDSPLITX [140], FEDVZ [141], and PRINCE [142]

(uRLLC), massive machine-type communication (mMTC), and enhanced mobile broadband (eMBB) using network slicing, SDN, network function virtualization (NFV), and multiaccess edge computing (MEC) [151]. Here, network slicing creates virtual networks on shared infrastructure, SDN and NFV provide flexibility through decoupling and virtualization, and MEC reduces latency via edge processing. As 6G moves toward user-centric and intelligent management, these traditional mechanisms become insufficient, necessitating closed-loop automation powered by ML and big data analytics [152].

TinyML–LargeML integration offers a transformative approach by combining edge efficiency with large-scale analytics for adaptive management. TinyML enables localized optimization by monitoring metrics, like latency and traffic, to sustain low-latency quality-of-service (QoS) in applications such as smart factories [153]. LargeML aggregates data across nodes, detects complex patterns, predicts failures, and orchestrates network-wide resources, for instance, rerouting traffic to preserve uRLLC, mMTC, and eMBB [154]. A notable example is a Lyapunov-based adaptive split learning framework that dynamically assigns split layers to TinyML devices, reducing latency and energy consumption [116]. Fig. 11 illustrates such bidirectional integration across domains. Applications span major technologies: 1) in *network slicing*, TinyML monitors metrics for real-time resource reallocation, while LargeML predicts congestion and optimizes slice provisioning [155], [156]; 2) in *SDN*, TinyML detects anomalies (e.g., DDoS attacks) at the edge, and LargeML enhances traffic control by predicting congestion and optimizing routes [157], [158]; 3) in *NFV*, TinyML mitigates performance degradation, while LargeML automates resource scaling [159]; and 4) in *MEC*, TinyML minimizes latency through on-device inference, complemented by LargeML's orchestration to ensure efficient,

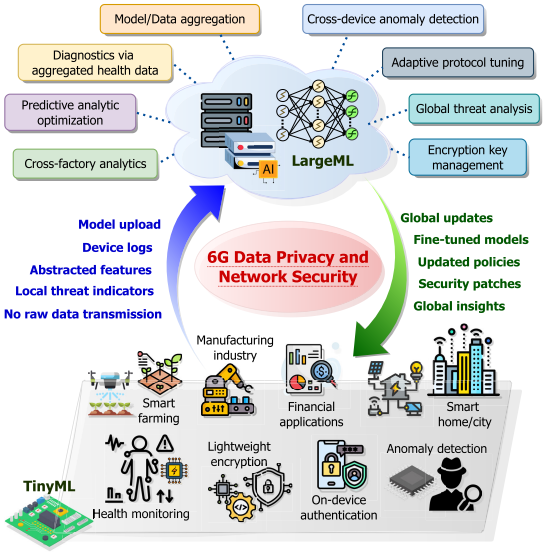

Fig. 10. Complementary roles of TinyML and LargeML in enhancing data privacy and network security in 6G systems.

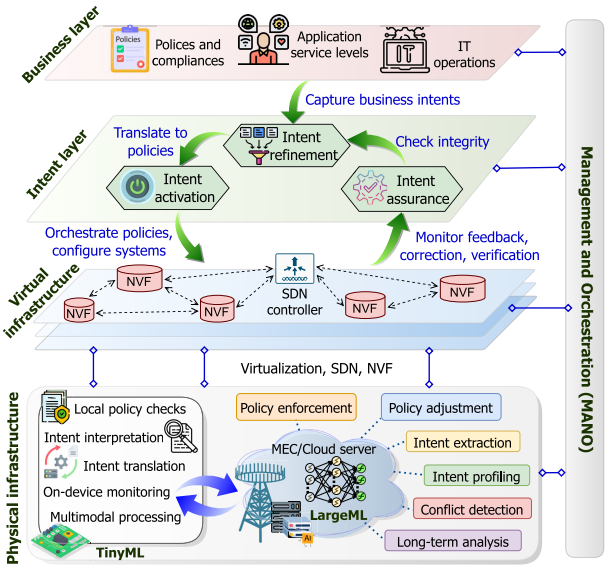

Fig. 12. IBNs with TinyML–LargeML integration.

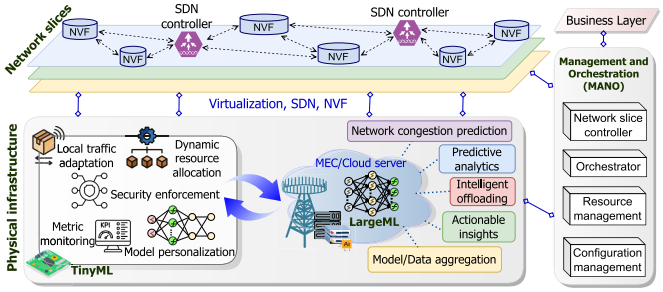

Fig. 11. Applications of TinyML–LargeML integration in 6G network slicing, SDN, NFV, and MEC management.

low-latency operation [160]. Examples include SPLITGP [132], an FSL strategy that balances personalization (on-device TinyML) with generalization (MEC-side LargeML), enabling robust training-inference optimization across diverse inputs.

C. Intent-Based Networking

Intent-based networking (IBN) is a management paradigm that translates high-level business objectives, or human-readable "intents," into automated network configurations [161]. Using closed-loop automation, IBN continuously verifies and enforces these intents through three processes [162]: 1) *intent refinement*, where user-defined intents are translated into maintainable and adaptable network policies; 2) *intent activation*, which leverages analytics and ML to detect and resolve policy conflicts; and 3) *intent assurance*, which integrates with SDN controllers and orchestration systems, using telemetry for continuous monitoring and adjustment.

The effectiveness of IBN depends on the accurate interpretation of user inputs (e.g., voice, text, and multimodal data). As shown in Fig. 12, TinyML–LargeML integration supports this objective. TinyML provides lightweight, on-device AI for real-time intent detection and localized processing [163], while LargeML aggregates intents across devices and translates them into network actions and policies using techniques such as TL, FTL, FFMs, SL, FSL, and FSFMs [164]. A notable example is an intent-based on-device AI framework [165], where TinyML processes local sensor data (e.g., speech-to-text conversion and intent translation), while LargeML manages centralized policy coordination. This reduces reliance on cloud-only processing, improving security, latency, and energy efficiency. Another application is intent-based management for autonomous aerial vehicle swarm networks [166], which combines TinyML with explainable AI, blockchain, and zero-trust architectures. This system supports interpretable, tamper-proof intent predictions and detects false communications that may distort shared positional data. Results in [166] demonstrate a 2.34% accuracy gain, 2.97% sensitivity improvement, latency reduced to 43.1 ms, and storage savings down to 0.0013 MB.

D. Zero-Touch Network

A zero-touch network (ZTN) leverages AI, ML, and IBN to automate operations with minimal human intervention [167]. By handling tasks such as configuration, monitoring, and troubleshooting, ZTNs reduce operational costs, improve efficiency, and free administrators for strategic planning. They also enhance reliability by analyzing traffic and user requests in real time, adjusting parameters, resolving faults, and restoring services autonomously. Moreover, ZTNs align network behavior with organizational policies while adapting to growing demands and evolving requirements. Importantly, ZTNs integrated security mechanisms further enable real-time threat detection and protection.

Achieving zero-touch automation with net-zero emissions in AI-driven wireless networks demands energy-efficient TinyML and LargeML integration, low-power transceivers, lightweight AI models, and adaptive resource allocation [168]. Recent research highlights a TinyML-LargeML ecosystem for 6G ZTNs (see Fig. 13). In self-organizing networks, TinyML optimizes neural network training on IoT devices, while LargeML uses federated learning and DL at base stations or clouds to balance computational loads and minimize energy consumption [169]. In mobile networking, TinyML allows autonomous IoT operations, while LargeML develops coverage maps and offers AI-as-a-service (AIaaS), as shown in NanoDeploy Automator [170]. In industrial IoT, TinyML processes data locally, while LargeML enhances efficiency

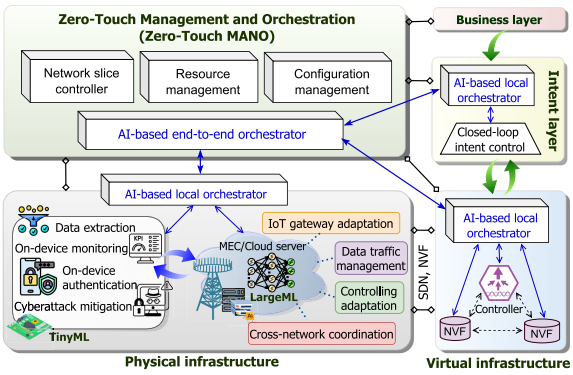

Fig. 13. ZTNs with TinyML–LargeML integration.

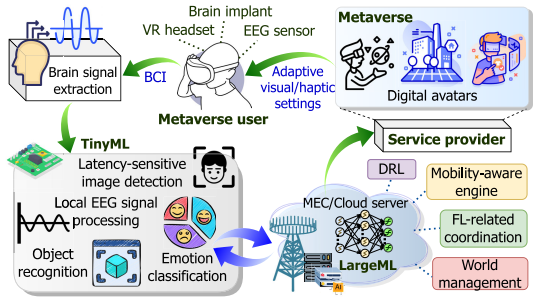

Fig. 14. BrainMeta with TinyML and LargeML.

and traffic management at gateways and clouds, as seen in O-RAN-based ZTNs for smart factories [171]. Cross-layer cybersecurity frameworks, like SH-CASH, utilize AutoML to secure layers, with TinyML ensuring service-level agreement compliance in O-RAN architectures [172].

E. Brain-Level Metaverse

The brain-level metaverse (BrainMeta) links brain–computer interfaces (BCIs) with immersive virtual worlds, allowing users to control avatars, interact with objects, and navigate environments through brain signals [173]. BCIs range from noninvasive EEG headsets [174] to invasive implants [175], enabling personalized experiences in virtual learning, gaming, remote work, and neurorehabilitation while expanding accessibility for users with disabilities. Yet BrainMeta remains early stage and must overcome challenges in designing brain-inspired cognitive architectures under strict size, weight, power, and processing limits [176], as well as the need for efficient, high-performance models for neural signal interpretation, cognition support, and adaptive content generation, motivating integrated TinyML–LargeML deployment (see Fig. 14).

TinyML can leverage neuromorphic chips [177] to support diverse neural message formats, customize processing in brain-inspired devices, localize memory, and accelerate tasks such as image recognition and classification, while edge-based TinyML processors reduce latency and resource use through circuits like winner-take-all implementations on FPGA hardware [178]. LargeML complements this with cloud-edge-end collaborative models, mobility-aware prerendering, and diffusion-based adaptive rendering [8], enhancing data collection from TinyML devices and improving metaverse quality. Techniques include extracting generalized representations from large-scale EEG datasets, segmenting signals into channel arrays, and applying vector-quantized neural spectrum prediction for training neural analyzers [179]. In addition, AMFL [180] combines FL and DRL to handle non-IID data, reduce resource overhead, and improve QoE for resource-constrained, human-centric AR devices.

VI. Challenges and Future Research Directions

A. Standardization

5G faces challenges with dynamic user demands and traffic, while 6G aims to enable sustainable applications like holography and the metaverse through TinyML–LargeML integration [181]. This integration is designed for user-centric services that adapt to changing behaviors and networks. However, standardization issues impede the development of a fully integrated ecosystem. The ITU-R identifies RAN slicing as crucial for 6G virtual network sharing [182], requiring a unified O-RAN architecture that supports diverse air interfaces and AI functions. This architecture should offer customizable service parameters, but shared infrastructure raises concerns about transparency, reliability, and privacy. Standardization bodies are advancing AI-driven wireless architectures. 3GPP's Releases 17 [183] and 18 [184] enhance 5G RAN with AI for intelligence and efficiency, while the O-RAN Alliance integrates AI for orchestration and control [185].

However, challenges remain in integrating TinyML and LargeML across the network lifecycle. Proposals include a secure TinyML edge AI architecture [64], explainable AI for decision transparency [186], and a testbed for trustworthy network management [187]. Sustainability efforts are supported by an energy cost-of-AI metric [188]. At a higher level, a shift to a multitier mNode model is proposed [181], featuring distributed nodes and network AI logic. In addition, a digital twin framework for AI-RAN can manage digital twins and enable real-time adaptability in dense networks [189].

B. Resource Management and Orchestration

6G networks face rising orchestration and resource management pressures due to wireless channel uncertainty, dense ISAC deployments, and carbon-reduction requirements. AI/ML improves interference control, energy efficiency, and reliability [190], but also raises concerns about privacy, computational load, and scalability. New directions include DRL paired with diffusion models for adaptive optimization [190] and FTL with blockchain-based incentives for low-latency, privacy-preserving coordination [191]. TinyML remains constrained by compute tradeoffs, vanishing gradients that demand hardware acceleration, memory limits for semantic communication, limited real-world data, and environmental variability [192]. LargeML, despite distributed learning, still struggles with data segmentation, transmission errors in FL, and strict privacy and coordination needs. Future methods such as model-agnostic meta-learning [193] and accretionary learning [194] may help alleviate these challenges.

At the orchestration layer, integrating AI into 6G network slicing blends NFV-based virtualization with SDN's centralized control: a global SDN controller handles planning and provisioning, while AI-enhanced local controllers optimize slice-level performance. Cross-domain AI coordination introduces challenges in data and knowledge sharing, training alignment, inference synchronization, and resource allocation

[195]. As 6G shifts from centralized 5G models to distributed learning, learning-model orchestration becomes more complex and depends on node capacity, data properties, resource availability, model size, latency, and network dynamics. Assigning specialized roles across slices enables scalable, efficient learning in 6G environments, e.g., horizontal FL for decentralized data, hierarchical FL for reduced communication, TL for efficient reuse, vertical FL for cross-domain learning, SL for heterogeneous resources, multiagent RL for local coordination, and swarm learning for secure joint training.

C. Advanced Security and Privacy-Preserving Techniques

As TinyML–LargeML integration becomes central to next-generation wireless networks, security and privacy concerns intensify, especially around adversarial threats, data protection, model integrity, and user privacy. TinyML's limited edge-device resources make sensors, smartphones in FL, and wearables vulnerable to attacks, but adversarial training strengthens robustness [196], homomorphic encryption protects data confidentiality and integrity [197], differential privacy obscures sensitive information [198], and advanced encryption secures model parameters during transmission [199]. LargeML, operating across broader and multistakeholder environments, also requires strong safeguards: model distillation reduces computational demands while preserving performance [37], GAN-based simulations improve cyber-resilience [200], and blockchain ensures model integrity by hashing and tracking TinyML model versions [198]. Additional privacy-preserving techniques, including data shuffling, machine unlearning [201], synthetic data generation, private teacher-ensemble aggregation, Rényi differential privacy, secure cross-silo FL, and decentralized knowledge distillation, further reinforce LargeML deployments.

D. Real-Time, Lightweight Intelligence

Beyond-5G and 6G networks are shifting from traditional M2M automation toward human-centric architectures, demanding AI systems that provide real-time, lightweight intelligence while meeting user-experience and sustainability goals. Joint TinyML–LargeML deployment expands these capabilities but also introduces challenges. TinyML devices span a wide range of protocols, latency profiles, and hardware, from Arduino Nano (64 MHz, 1-MB flash, and 256-kB RAM) to Raspberry Pi (1.8 GHz, 32-GB flash, and 8-GB RAM) [3], [10], emphasizing the need for compact, unified frameworks that support plug-and-play deployment, efficient processing, and lightweight runtimes [202]. LargeML, by contrast, handles massive multimodal inputs and depends on prompt engineering, model partitioning, and hyperparameter tuning, all of which incur high training costs, power consumption, and memory demands [15]. Emerging techniques such as model-adaptive prompts [203] and pluggable lightweight tuning [204] can improve explainability and accelerate inference without disrupting life-cycle pipelines.

Integration remains limited by the lack of standardized platforms and tools for workflow management, data coordination, and runtime compatibility. Large models often require specialized libraries, compilers, and runtimes with inconsistent input/output specifications [29], making iterative refinement necessary for reliable outputs. As a result, progress depends not only on optimizing TinyML and LargeML individually but also on developing guidelines and evaluation frameworks that define task-specific roles, interoperability requirements, and life-cycle management strategies across domains [205].

E. AI-Enabled 6G Communication-Sensing-Computing

Integrating TinyML's real-time, low-power intelligence with LargeML's high-capacity reasoning creates a unified AI layer for 6G, where communication, sensing, and computation operate as a single adaptive system [206]. In this hierarchy, TinyML provides fast on-device perception through lightweight feature extraction and inference, while LargeML offers deeper interpretation and coordination through global optimization, multimodal fusion, and long-horizon prediction. To do so, TinyML first semantically compresses sensed data for efficient transmission [207]. LargeML then uses these representations to optimize routing, beamforming, and spectrum allocation, and finally, LargeML's predictions guide TinyML nodes to adjust sensing rates or modalities, forming adaptive networks responsive to mobility, interference, and environmental changes. This synergy supports scalable multimodal sensing, energy-efficient operation, and human-centric services [208], where edge devices capture fine-grained context and cloud models infer intent or anticipate user needs.

However, integrating TinyML and LargeML in 6G introduces significant challenges. First, model partitioning must balance latency, energy, and accuracy across heterogeneous devices [207]. Second, synchronizing TinyML's millisecond-scale inference with LargeML's slower global reasoning requires new timing and scheduling mechanisms. Third, interoperability is hindered by diverse hardware, runtime environments, and data formats across edge devices [209]. Fourth, security and privacy risks increase as sensitive edge data interacts with large-scale models, demanding robust protection against adversarial attacks, leakage, and model manipulation. Finally, additional 6G-specific constraints (i.e., URLLC requirements, dynamic spectrum conditions, mobility-induced channel variation, and sustainability targets [206], [208]) further complicate joint deployment.

F. AI-Native and Collective AI for 6G

AI-native systems embed intelligence at the core of their operation, aiming to solve complex tasks through data-driven reasoning, adapt to dynamic environments, fuse foundational knowledge with new insights, and maintain transparency and trustworthiness for resilient autonomous networks [210]. Integrating TinyML and LargeML into these architectures introduces challenges such as coordinating constrained TinyML with powerful LargeML for zero-touch automation, defining their roles across heterogeneous infrastructures, ensuring interoperable lifelong learning, managing AI life cycles for fairness and reliability, and balancing joint performance without degrading TinyML accuracy. Collective AI extends this vision through multiagent collaboration, where agents learn locally toward shared goals without direct communication [211]; platform-level collective reinforcement learning accelerates convergence and reduces redundant training [212], while system-level swarm intelligence supports IoT applications in dynamic environments [213], though it still lacks unified reasoning and coordination frameworks.

Integrating TinyML and LargeML into collective AI creates new opportunities but also raises several challenges: sustaining agent participation through incentive mechanisms, ensuring efficient spectrum use in dense 6G IoT deployments, and

addressing security, trust, and privacy concerns, as shared outputs may expose sensitive information while blockchain remains resource-intensive for constrained devices. Human–AI collaboration further requires cognitive architectures capable of interpreting social signals, and ethical considerations, including transparency, bias mitigation, and accountability, are essential, especially as errors in LargeML outputs can have significant real-world consequences.

G. Application Prospects and Outlook for 6G and AI

6G will unify TinyML at the edge with large-scale AI models deployed across the network and cloud, creating an intelligent and adaptive wireless environment. This integration will enable real-time sensing, IBN, semantic communication, and automated SDN/NFV orchestration, thereby forming the foundation for immersive XR, holographic telepresence, and large-scale digital twins [78], [214]. Early scenarios already reflect this direction: in smart factories, TinyML sensors detect anomalies locally and send only semantic insights, while edge AI coordinates predictive maintenance and robotic actions with subsecond latency. At the city scale, distributed TinyML nodes monitor environmental and public-safety conditions and feed compact features into collective AI systems that coordinate drones, traffic flows, and emergency responses through privacy-preserving FL.

Over the longer term, 6G is expected to evolve into an AI-native network where intelligence is embedded across all layers, and devices collaborate through distributed learning and collective decision making [214]. This will enable ZTN operation, self-healing infrastructures, and brain-level Metaverse experiences that tightly integrate communication, sensing, and computation. Realizing this vision requires advances in O-RAN standardization, trustworthy and explainable AI, secure FL, and efficient orchestration of communication, computing, and energy resources. As these foundations mature, the fusion of 6G and AI will support resilient industrial automation, new service models, and globally scalable intelligence.

VII. Concluding Remarks

TinyML–LargeML integration is set to reshape AI design for 6G by enabling efficient, intelligent solutions to challenges in resource allocation, network management, security, and privacy. This survey reviews the foundations of this convergence, covering design principles, operational mechanisms, and performance evaluation while outlining the motivations and technical requirements driving bidirectional integration and its emerging application scenarios across 6G ecosystems. This survey also identifies key open challenges and promising research directions, including O-RAN standardization, cross-layer resource orchestration, advanced security and privacy-preserving methods, real-time lightweight AI, AI-enabled 6G communication-sensing-computing, the shift toward AI-native and collective AI architectures, and application prospects and outlooks for 6G and AI. Continued progress in these areas is essential, and the insights provided here aim to guide and support future 6G-enabled intelligent systems.

References

[1] N.-N. Dao et al., "A review on new technologies in 3GPP standards for 5G access and beyond," *Comput. Netw.*, vol. 245, May 2024, Art. no. 110370.

[2] S. Zhang and D. Zhu, "Towards artificial intelligence enabled 6G: State of the art, challenges, and opportunities," *Comput. Netw.*, vol. 183, Dec. 2020, Art. no. 107556.

[3] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, and S. Han, "Tiny machine learning: Progress and futures [Feature]," *IEEE Circuits Syst. Mag.*, vol. 23, no. 3, pp. 8–34, Mar. 2023.

[4] M. Xu et al., "When large language model agents meet 6G networks: Perception, grounding, and alignment," *IEEE Wireless Commun.*, vol. 31, no. 6, pp. 63–71, Dec. 2024.

[5] Z. Zhu, J. Hong, and J. Zhou, "Data-free knowledge distillation for heterogeneous federated learning," in *Proc. Int. Conf. Mac. Learn.*, vol. 139, 2021, pp. 12878–12889.

[6] V. Tsoukas, E. Boumpa, G. Giannakas, and A. Kakarountas, "A review of machine learning and TinyML in healthcare," in *Proc. ACM Pan-Hellenic Conf. Inform.*, 2021, pp. 69–73.

[7] H.-T. Nguyen, N.-D. Mai, B. G. Lee, and W.-Y. Chung, "Behind-the-ear EEG-based wearable driver drowsiness detection system using embedded tiny neural networks," *IEEE Sensors J.*, vol. 23, no. 19, pp. 23875–23892, Oct. 2023.

[8] Y. Wang, Q. Hu, Z. Su, L. Du, Q. Xu, and W. Li, "Large model empowered metaverse: State-of-the-art, challenges and opportunities," 2025, *arXiv:2502.10397*.

[9] D. L. Dutta and S. Bharali, "TinyML meets IoT: A comprehensive survey," *Internet Things*, vol. 16, Dec. 2021, Art. no. 100461.

[10] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, and A. S. Hafid, "A comprehensive survey on TinyML," *IEEE Access*, vol. 11, pp. 96892–96922, 2023.

[11] P. P. Ray, "A review on TinyML: State-of-the-art and prospects," *J. King Saud Univ. -Comput. Inf. Sci.*, vol. 34, no. 4, pp. 1595–1623, Apr. 2022.

[12] V. Rajapakse, I. Karunanayake, and N. Ahmed, "Intelligence at the extreme edge: A survey on reformable TinyML," *ACM Comput. Surveys*, vol. 55, no. 13s, pp. 1–30, Dec. 2023.

[13] V. Tsoukas, A. Gkogkidis, E. Boumpa, and A. Kakarountas, "A review on the emerging technology of TinyML," *ACM Comput. Surveys*, vol. 56, no. 10, pp. 1–37, Oct. 2024.

[14] L. Capogrosso, F. Cunico, D. S. Cheng, F. Fummi, and M. Cristani, "A machine learning-oriented survey on tiny machine learning," *IEEE Access*, vol. 12, pp. 23406–23426, 2024.

[15] M. A. K. Raiaan et al., "A review on large language models: Architectures, applications, taxonomies, open issues and challenges," *IEEE Access*, vol. 12, pp. 26839–26874, 2024.

[16] Y. Chang et al., "A survey on evaluation of large language models," *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 3, pp. 1–45, 2024.

[17] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, "Parameter-efficient fine-tuning for large models: A comprehensive survey," 2024, *arXiv:2403.14608*.

[18] H. Zhao et al., "Explainability for large language models: A survey," *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 2, pp. 1–38, 2024.

[19] B. Min et al., "Recent advances in natural language processing via large pre-trained language models: A survey," *ACM Comput. Surveys*, vol. 56, no. 2, pp. 1–40, Feb. 2024.

[20] W. Zhuang, C. Chen, J. Li, C. Chen, Y. Jin, and L. Lyu, "When foundation model meets federated learning: Motivations, challenges, and future directions," 2023, *arXiv:2306.15546*.

[21] C. Ren et al., "Advances and open challenges in federated learning with foundation models," *IEEE Commun. Surveys Tuts.*, vol. 28, pp. 2087–2126, 2026.

[22] M. Wang, W. Fu, X. He, S. Hao, and X. Wu, "A survey on large-scale machine learning," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 6, pp. 2574–2594, Jun. 2022.

[23] A. Vaswani, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 30, 2017, pp. 6000–6010.

[24] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. Cambridge, MA, USA: MIT Press, 2018.

[25] C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*. Cham, Switzerland: Springer, 2023.

[26] M. A. Wani, B. Sultan, S. Ali, and M. A. Sofi, *Advances in Deep Learning*, vol. 2. Cham, Switzerland: Springer, 2025.

[27] J. Wang, *Introduction To Transfer Learning*. Los Alamitos, CA, USA: Publishing House of Electronics Industry, 2023.

[28] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–19, 2019.

[29] G. Wu, S. Tarkoma, and R. Morabito, "Consolidating TinyML lifecycle with large language models: Reality, illusion, or opportunity?," 2025, *arXiv:2501.12420*.

[30] B. Rokh, A. Azarpeyvand, and A. Khanteymoori, "A comprehensive survey on model quantization for deep neural networks in image classification," *ACM Trans. Intell. Syst. Technol.*, vol. 14, no. 6, pp. 1–50, Dec. 2023.

[31] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," 2017, *arXiv:1710.01878*.

[32] M. Udell, C. Horn, R. Zadeh, and S. Boyd, "Generalized low rank models," *Found. Trends Mach. Learn.*, vol. 9, no. 1, pp. 1–118, Jun. 2016.

[33] J. Ott, E. Linstead, N. LaHaye, and P. Baldi, "Learning in the machine: To share or not to share?," *Neural Netw.*, vol. 126, pp. 235–249, Jun. 2020.

[34] M. T. Lê, P. Wolinski, and J. Arbel, "Efficient neural networks for tiny machine learning: A comprehensive review," 2023, *arXiv:2311.11883*.

[35] A. Moffat, "Huffman coding," *ACM Comput. Surv. (CSUR)*, vol. 52, no. 4, pp. 1–35, 2019.

[36] M. Feurer, *Hyperparameter Optimization*. Cham, Switzerland: Springer, 2019.

[37] C. Yang, X. Yu, Z. An, and Y. Xu, "Categories of response-based, feature-based, and relation-based knowledge distillation," in *Proc. Adv. Knowl. Distillation, Towards New Horizons Intell. Syst.*, 2023, pp. 1–32.

[38] Z. Huang, K. Zandberg, K. Schleiser, and E. Baccelli, "RIOT-ML: Toolkit for over-the-air secure updates and performance evaluation of TinyML models," *Ann. Telecommun.*, vol. 80, nos. 3–4, pp. 283–297, Apr. 2025.

[39] B. Sudharsan et al., "OTA-TinyML: Over the air deployment of TinyML models and execution on IoT devices," *IEEE Internet Comput.*, vol. 26, no. 3, pp. 69–78, May 2022.

[40] E. Ostrovan, "TinyML On-Device Neural Network Training," *Master's thesis, Dept. Comput. Sci. Eng., Politecnico di Milano, Milan, Italy*, Apr. 2022. [Online]. Available: https://hdl.handle.net/10589/187690

[41] R. David et al., "Tensorflow lite micro: Embedded machine learning for TinyML systems," *Proc. Mach. Learn. Syst.*, vol. 3, pp. 800–811, 2021.

[42] H. Cai, C. Gan, L. Zhu, and S. Han, "TinyTL: Reduce memory, not parameters for efficient on-device learning," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 11285–11297.

[43] A. Sabovic, M. Aernouts, D. Subotic, J. Fontaine, E. De Poorter, and J. Famaey, "Towards energy-aware TinyML on battery-less IoT devices," *Internet Things*, vol. 22, Jul. 2023, Art. no. 100736.

[44] M. Le, T. Huynh-The, T. Do-Duy, T.-H. Vu, W.-J. Hwang, and Q.-V. Pham, "Applications of distributed machine learning for the Internet-of-Things: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 27, no. 2, pp. 1053–1100, Apr. 12, 2025, doi: 10.1109/COMST.2024.3427324.

[45] Q. Huang and T. Zhao, "Data collection and labeling techniques for machine learning," 2024, *arXiv:2407.12793*.

[46] I. Muraina, "Ideal dataset splitting ratios in machine learning algorithms: General concerns for data scientists and data analysts," in *Proc. 7th Int. Mardin Artuklu Sci. Res. Conf.*, 2022, pp. 496–504.

[47] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[48] A. Radford et al., "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[49] B. Zhang et al., "Examining scaling and transfer of language model architectures for machine translation," in *Proc. PMLR Int. Conf. Mac. Learn.*, 2022, pp. 26176–26192.

[50] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *J. Mach. Learn. Res.*, vol. 23, no. 120, pp. 1–39, 2021.

[51] N. Du et al., "GLaM: Efficient scaling of language models with mixture-of-experts," in *Proc. PMLR Int. Conf. Mac. Learn.*, 2021, pp. 5547–5569.

[52] V.-T. Tran, L. H. Khiem, and V. Q. Pham, "Revisiting sparse mixture of experts for resource-adaptive federated fine-tuning foundation models," in *Proc. Mod. Collab. Decent. Cont. Deep Learn. Workshop*, Mar. 2025, p. 1.

[53] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," 2021, *arXiv:2111.00396*.

[54] Y. Liu et al., "Understanding LLMs: A comprehensive overview from training to inference," *Neurocomputing*, vol. 620, Mar. 2025, Art. no. 129190.

[55] C. Wei, Y.-C. Wang, B. Wang, and C. J. Kuo, "An overview of language models: Recent developments and outlook," *APSIPA Trans. Signal Inf. Process.*, vol. 13, no. 2, Dec. 2024.

[56] X. Hou et al., "Large language models for software engineering: A systematic literature review," *ACM Trans. Softw. Eng. Methodology*, vol. 33, no. 8, pp. 1–79, Nov. 2024.

[57] N. Houlsby et al., "Parameter-efficient transfer learning for NLP," in *Proc. PMLR Int. Conf. Mac. Learn.*, 2019, pp. 2790–2799.

[58] E. J. Hu et al., "LoRA: Low-rank adaptation of large language models," in *Proc. Int. Conf. Learn. Represent.*, 2022, pp. 1–7.

[59] Nebius. *AI Model Performance Metrics: In-depth Guide*. Accessed: Dec. 13, 2024. [Online]. Available: https://nebius.com/blog/posts/ai-model-performance-metrics

[60] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256KB memory," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 22941–22954.

[61] A. Tuama, F. Comby, and M. Chaumont, "Camera model identification with the use of deep convolutional neural networks," in *Proc. IEEE Int. Workshop Inf. Forensics Secur. (WIFS)*, Abu Dhabi, UAE, Dec. 2016, pp. 1–6.

[62] *State of the TinyAutoML Market 2022*, TinyML Foundation, Altos, CA, USA, Jun. 2020.

[63] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[64] M. Y. Shabir, G. Torta, A. Basso, and F. Damiani, "Toward secure TinyML on a standardized AI architecture," in *Device-Edge-Cloud Continuum: Paradigms, Architectures and Applications*. Cham, Switzerland: Springer, 2023, pp. 121–139.

[65] A. Koubaa. (2023). *Gpt-4 Vs. GPT-3.5: A Concise Showdown*. [Online]. Available: https://www.preprints.org/manuscript/202303.0422/v1

[66] K. Grace et al., "Thousands of AI authors on the future of AI," 2024, *arXiv:2401.02843*.

[67] S. A. Mohammed, S. Shirmohammadi, and S. Altamimi, "Artificial intelligence-based distributed network latency measurement," in *Proc. IEEE Int. Instrum. Meas. Technol. Conf. (I2MTC)*, Auckland, New Zealand, May 2019, pp. 1–6.

[68] H. Barmer et al., "Scalable AI," Softw. Eng. Inst., White Paper, Jun. 2021, pp. 1–9. [Online]. Available: https://www.sei.cmu.edu/library/scalable-ai/

[69] F. D. Protection, "General data protection regulation (GDPR)," Intersoft Consulting, Hamburg, Germany, Tech. Rep., 2018, vol. 24, no. 1.

[70] N. Ding et al., "Parameter-efficient fine-tuning of large-scale pre-trained language models," *Nature Mach. Intell.*, vol. 5, no. 3, pp. 220–235, 2023.

[71] S. Dilmaghani, M. R. Brust, G. Danoy, N. Cassagnes, J. Pecero, and P. Bouvry, "Privacy and security of big data in AI systems: A research and standards perspective," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Los Angeles, CA, USA, Dec. 2019, pp. 5737–5743.

[72] R. Liu, H. Lin, H. Lee, F. d. S. Chaves, H. Lim, and J. Skold, "Beginning of the journey toward 6G: Vision and framework," *IEEE Commun. Mag.*, vol. 61, no. 10, pp. 8–9, Oct. 2023.

[73] R. Liu, L. Zhang, R. Yu-Ngok Li, and M. Di Renzo, "The ITU vision and framework for 6G: Scenarios, capabilities and enablers," 2023, *arXiv:2305.13887*.

[74] Z. Lin, G. Qu, X. Chen, and K. Huang, "Split learning in 6G edge networks," *IEEE Wireless Commun.*, vol. 31, no. 4, pp. 170–176, Aug. 2024.

[75] N.-N. Dao et al., "Survey on aerial radio access networks: Toward a comprehensive 6G access infrastructure," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 1193–1225, 2nd Quart., 2021.

[76] N.-N. Dao, N. H. Tu, T. T. Thanh, V. N. Q. Bao, W. Na, and S. Cho, "Neglected infrastructures for 6G—Underwater communications: How mature are they?," *J. Netw. Comput. Appl.*, vol. 213, Apr. 2023, Art. no. 103595.

[77] (2023). *M.2160: Framework and Overall Objectives of the Future Development of IMT for 2030 and Beyond*. [Online]. Available: https://www.itu.int/rec/R-REC-M.2160/en

[78] B. Rong, "6G: The next horizon from connected people and things to connected intelligence," *IEEE Wireless Commun.*, vol. 28, no. 5, p. 8, Oct. 2021.

[79] T. Huang, W. Yang, J. Wu, J. Ma, X. Zhang, and D. Zhang, "A survey on green 6G network: Architecture and technologies," *IEEE Access*, vol. 7, pp. 175758–175768, 2019.

[80] F. Liu et al., "Integrated sensing and communications: Toward dual-functional wireless networks for 6G and beyond," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 6, pp. 1728–1767, Jun. 2022.

[81] Y. Tao, J. Qiu, and S. Lai, "A hybrid cloud and edge control strategy for demand responses using deep reinforcement learning and transfer learning," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 56–71, Jan. 2022.

[82] Y. Jang, H. Lee, S. J. Hwang, and J. Shin, "Learning what and where to transfer," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3030–3039.

[83] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2009.

[84] S. Parsaeefard and A. Leon-Garcia, "Efficient transfer learning in 6G," in *Proc. IEEE Future Netw. World Forum (FNWF)*, Montreal, QC, Canada, Oct. 2022, pp. 314–319.

[85] M. Ghifary, W. B. Kleijn, and M. Zhang, "Domain adaptive neural networks for object recognition," in *Proc. Pacific Rim Int. Conf. Artif. Intell.* Cham, Switzerland: Springer, 2014, pp. 898–904.

[86] B. Sun and K. Saenko, "Deep CORAL: Correlation alignment for deep domain adaptation," in *Proc. Eur. Conf. Comput. Vis. Worksh.*, Oct. 2016, pp. 443–450.

[87] C. Profentzas, M. Almgren, and O. Landsiedel, "MicroTL: Transfer learning on low-power IoT devices," in *Proc. IEEE 47th Conf. Local Comput. Netw. (LCN)*, Edmonton, AB, Canada, Sep. 2022, pp. 1–8.

[88] M. B. Azevedo, T. D. A. de Medeiros, M. D. A. Medeiros, I. Silva, and D. G. Costa, "Detecting face masks through embedded machine learning algorithms: A transfer learning approach for affordable microcontrollers," *Mach. Learn. Appl.*, vol. 14, Dec. 2023, Art. no. 100498.

[89] A. M. Hayajneh, S. A. Aldalahmeh, F. Alasali, H. Al-Obiedollah, S. A. Zaidi, and D. McLernon, "Tiny machine learning on the edge: A framework for transfer learning empowered unmanned aerial vehicle assisted smart farming," *IET Smart Cities*, vol. 6, no. 1, pp. 10–26, Mar. 2024.

[90] A. M. Hayajneh, M. Hafeez, S. A. R. Zaidi, and D. McLernon, "TinyML empowered transfer learning on the edge," *IEEE Open J. Commun. Soc.*, vol. 5, pp. 1656–1672, 2024.

[91] Y. D. Kwon, R. Li, S. Venieris, J. Chauhan, N. D. Lane, and C. Mascolo, "TinyTrain: Resource-aware task-adaptive sparse training of DNNs at the data-scarce edge," in *Proc. Int. Conf. Mach. Learn.*, 2024, pp. 25812–25843.

[92] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, vol. 54, A. Singh and J.Zhu., Eds., Fort Lauderdale, FL, USA, 2017, pp. 1273–1282.

[93] Z. Zhao et al., "Federated learning with non-IID data in wireless networks," *IEEE Trans. Wireless Commun.*, vol. 21, no. 3, pp. 1927–1942, Mar. 2022.

[94] K. Kopparapu, E. Lin, J. G. Breslin, and B. Sudharsan, "TinyFedTL: Federated transfer learning on ubiquitous tiny IoT devices," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops Affiliated Events (PerCom Workshops)*, Pisa, Italy, Mar. 2022, pp. 79–81.

[95] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, "FedHealth: A federated transfer learning framework for wearable healthcare," *IEEE Intell. Syst.*, vol. 35, no. 4, pp. 83–93, Jul. 2020.

[96] W. Guo, Y. Wang, X. Chen, and P. Jiang, "Federated transfer learning for auxiliary classifier generative adversarial networks: Framework and industrial application," *J. Intell. Manuf.*, vol. 35, no. 4, pp. 1439–1454, 2023.

[97] M. Ficco, A. Guerriero, E. Milite, F. Palmieri, R. Pietrantuono, and S. Russo, "Federated learning for IoT devices: Enhancing TinyML with on-board training," *Inf. Fusion*, vol. 104, Apr. 2024, Art. no. 102189.

[98] Y. Fan, Y. Li, M. Zhan, H. Cui, and Y. Zhang, "IoTDefender: A federated transfer learning intrusion detection framework for 5G IoT," in *Proc. IEEE 14th Int. Conf. Big Data Sci. Eng. (BigDataSE)*, Guangzhou, China, Dec. 2020, pp. 88–95.

[99] M. A. P. Putra, S. M. Rachmawati, M. Abisado, and G. A. Sampedro, "HFTL: Hierarchical federated transfer learning for secure and efficient fault classification in additive manufacturing," *IEEE Access*, vol. 11, pp. 54795–54807, 2023.

[100] X. Liu, Y. Deng, A. Nallanathan, and M. Bennis, "Federated learning and meta learning: Approaches, applications, and directions," *IEEE Commun. Surveys Tuts.*, vol. 26, no. 1, pp. 571–618, 1st Quart., 2024.

[101] H. Ren, D. Anicic, and T. A. Runkler, "TinyReptile: TinyML with federated meta-learning," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jun. 2023, pp. 1–9.

[102] H. Ren, X. Li, D. Anicic, and T. A. Runkler, "TinyMetaFed: Efficient federated meta-learning for TinyML," 2023, *arXiv:2307.06822*.

[103] H. Ren, D. Anicic, X. Li, and T. Runkler, "On-device online learning and semantic management of TinyML systems," *ACM Trans. Embedded Comput. Syst.*, vol. 23, no. 4, pp. 1–32, Jul. 2024.

[104] Z. Jia, T. Zhou, Z. Yan, J. Hu, and Y. Shi, "Personalized meta-federated learning for IoT-enabled health monitoring," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 10, pp. 3157–3170, Oct. 2024.

[105] Z. Gao, Y. Zhang, Z. Zhang, Y. Gong, and Y. Guo, "FedPT: Federated proxy-tuning of large language models on resource-constrained edge devices," 2024, *arXiv:2410.00362*.

[106] Y. J. Cho, L. Liu, Z. Xu, A. Fahrezi, and G. Joshi, "Heterogeneous LoRA for federated fine-tuning of on-device foundation models," 2024, *arXiv:2401.06432*.

[107] K. Pfeiffer, M. A. Ahmed, R. Khalili, and J. Henkel, "Efficient federated finetuning of tiny transformers with resource-constrained devices," 2024, *arXiv:2411.07826*.

[108] S. K. Atapour et al., "Leveraging foundation models for efficient federated learning in resource-restricted edge networks," 2024, *arXiv:2409.09273*.

[109] M. Xu, D. Cai, Y. Wu, X. Li, and S. Wang, "FwdLLM: Efficient federated finetuning of large language models with perturbed inferences," in *Proc. USENIX Annu. Tech. Conf.*, 2024, pp. 579–596.

[110] P. Wu, K. Li, T. Wang, Y. Dong, V. C. M. Leung, and F. Wang, "FedFMSL: Federated learning of foundation models with sparsely activated LoRA," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 15167–15181, Dec. 2024.

[111] H. Mei, D. Cai, A. Zhou, S. Wang, and M. Xu, "FedMoE: Personalized federated learning via heterogeneous mixture of experts," 2024, *arXiv:2408.11304*.

[112] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," 2018, *arXiv:1812.00564*.

[113] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2014, pp. 661–670.

[114] J. Lee, H. Lee, and W. Choi, "Wireless channel adaptive DNN split inference for resource-constrained edge devices," *IEEE Commun. Lett.*, vol. 27, no. 6, pp. 1520–1524, Jun. 2023.

[115] E. Samikwa, A. D. Maio, and T. Braun, "ARES: Adaptive resource-aware split learning for Internet of Things," *Comput. Netw.*, vol. 218, Dec. 2022, Art. no. 109380.

[116] Z. Li, W. Wu, S. Wu, and W. Wang, "Adaptive split learning over energy-constrained wireless edge networks," 2024, *arXiv:2403.05158*.

[117] S. Wang, X. Zhang, H. Uchiyama, and H. Matsuda, "HiveMind: Towards cellular native machine learning model splitting," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 2, pp. 626–640, Feb. 2022.

[118] J. Tirana, C. Pappas, D. Chatzopoulos, S. Lalis, and M. Vavalis, "The role of compute nodes in privacy-aware decentralized AI," in *Proc. 6th Int. Workshop Embedded Mobile Deep Learn.*, Jul. 2022, pp. 19–24.

[119] Y. Cao, S.-Y. Lien, C.-H. Yeh, D.-J. Deng, Y.-C. Liang, and D. Niyato, "Learning-based multitier split computing for efficient convergence of communication and computation," *IEEE Internet Things J.*, vol. 11, no. 20, pp. 33077–33096, Oct. 2024.

[120] S. Lyu, Z. Lin, G. Qu, X. Chen, X. Huang, and P. Li, "Optimal resource allocation for U-shaped parallel split learning," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Kuala Lumpur, Malaysia, Dec. 2023, pp. 197–202.

[121] M. Zhang, J. Cao, Y. Sahni, X. Chen, and S. Jiang, "Resource-efficient parallel split learning in heterogeneous edge computing," 2024, *arXiv:2403.15815*.

[122] Z. Lin et al., "Efficient parallel split learning over resource-constrained wireless edge networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 10, pp. 9224–9239, Oct. 2024.

[123] J. Jeon and J. Kim, "Privacy-sensitive parallel split learning," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2020, pp. 7–9.

[124] A. Chopra et al., "Adaptive split learning," in *Proc. Federated Learn. Syst. (FLSys) Workshop@ MLSys 2023*, Miami, FL, USA, Jun. 2023, pp. 1–13. [Online]. Available: https://openreview.net/forum?id=PKaQP7ywWtS

[125] M. Kim, A. DeRieux, and W. Saad, "A bargaining game for personalized, energy efficient split learning over wireless networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Glasgow, United Kingdom, Mar. 2023, pp. 1–6.

[126] J. Tirana, D. Tsigkari, G. Iosifidis, and D. Chatzopoulos, "Workflow optimization for parallel split learning," 2024, *arXiv:2402.10092*.

[127] J. Tirana, S. Lalis, and D. Chatzopoulos, "MP-SL: Multihop parallel split learning," 2024, *arXiv:2402.00208*.
[128] W. Wu et al., "Split learning over wireless networks: Parallel design and resource management," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 4, pp. 1051–1066, Apr. 2023.
[129] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "SplitFed: When federated learning meets split learning," in *Proc. AAAI Conf. Artif. Intell.*, Jun. 2022, vol. 36, no. 8, pp. 8485–8493.
[130] V. Turina, Z. Zhang, F. Esposito, and I. Matta, "Federated or split? A performance and privacy analysis of hybrid split and federated learning architectures," in *Proc. IEEE 14th Int. Conf. Cloud Comput. (CLOUD)*, Sep. 2021, pp. 250–260.
[131] C. Thapa, M. A. P. Chamikara, and S. Camtepe, "Advancements of federated learning towards privacy preservation: From federated learning to split learning," in *Proc. Federated Learn. Systems: Towards Next-Generation AI*, 2020, pp. 79–109.
[132] D.-J. Han et al., "Federated split learning with joint personalization-generalization for inference-stage optimization in wireless edge networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 6, pp. 7048–7065, Jun. 2024.
[133] G. Zhu, Y. Deng, X. Chen, H. Zhang, Y. Fang, and T. F. Wong, "ESFL: Efficient split federated learning over resource-constrained heterogeneous wireless devices," *IEEE Internet Things J.*, vol. 11, no. 16, pp. 27153–27166, Aug. 2024.
[134] Z. Lin, G. Qu, W. Wei, X. Chen, and K. K. Leung, "AdaptSFL: Adaptive split federated learning in resource-constrained edge networks," 2024, *arXiv:2403.13101*.
[135] W. Ni, H. Ao, H. Tian, Y. C. Eldar, and D. Niyato, "FedSL: Federated split learning for collaborative healthcare analytics on resource-constrained wearable IoMT devices," *IEEE Internet Things J.*, vol. 11, no. 10, pp. 18934–18935, May 2024.
[136] Z. Zhang, A. Pinto, V. Turina, F. Esposito, and I. Matta, "Privacy and efficiency of communications in federated split learning," *IEEE Trans. Big Data*, vol. 9, no. 5, pp. 1380–1391, Oct. 2023.
[137] L. U. Khan, M. Guizani, A. Al-Fuqaha, C. S. Hong, D. Niyato, and Z. Han, "A joint communication and learning framework for hierarchical split federated learning," *IEEE Internet Things J.*, vol. 11, no. 1, pp. 268–282, Jan. 2024.
[138] S. Li et al., "Synergizing foundation models and federated learning: A survey," 2024, *arXiv:2406.12844*.
[139] L. Cao, Y. Zhu, and W. Gong, "SFPrompt: Communication-efficient split federated fine-tuning for large pre-trained models over resource-limited devices," 2024, *arXiv:2407.17533*.
[140] J. Shin, J. Ahn, H. Kang, and J. Kang, "FedSplitX: Federated split learning for computationally-constrained heterogeneous clients," 2023, *arXiv:2310.14579*.
[141] Y. Shi et al., "Heterogeneous federated learning with splited language model," 2024, *arXiv:2403.16050*.
[142] S. Li, J. Hu, G. Min, and H. Huang, "Incentivizing multi-tenant split federated learning for foundation models at the network edge," 2025, *arXiv:2503.04971*.
[143] S. Yilmaz, E. Aydogan, and S. Sen, "A transfer learning approach for securing resource-constrained IoT devices," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 4405–4418, 2021.
[144] Z. Yang, W. Xu, L. Liang, Y. Cui, Z. Qin, and M. Debbah, "On privacy, security, and trustworthiness in distributed wireless large AI models (WLAM)," 2024, *arXiv:2412.02538*.
[145] W. Villegas-Ch, R. Gutierrez, A. M. Navarro, and A. Mera-Navarrete, "Optimizing federated learning on TinyML devices for privacy protection and energy efficiency in IoT networks," *IEEE Access*, vol. 12, pp. 174354–174370, 2024.
[146] H. Yang, H. He, W. Zhang, and X. Cao, "FedSteg: A federated transfer learning framework for secure image steganalysis," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 2, pp. 1084–1094, Apr. 2021.
[147] Y. Chen, "Efficient deep learning computing: From TinyML to LargeLM," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, Nov. 2023. [Online]. Available: https://hdl.handle.net/1721.1/153837
[148] S. Arcot, M. Masum, M. S. Kader, A. Saha, and M. Chowdhury, "TinyML for cybersecurity: Deploying optimized deep learning models for on-device threat detection on resource-constrained devices," in *Proc. IEEE Int. Conf. Big Data (BigData)*, Dec. 2024, pp. 5542–5550.
[149] R. Y. Patil, M. Bhamare, Y. H. Patil, and A. Bannore, "Securing TinyML in a connected world," in *TinyML for Edge Intelligence in IoT and LPWAN Networks*. Amsterdam, The Netherlands: Elsevier, 2024, pp. 311–330.
[150] J. Buban et al., "Encrypted large model inference: The equivariant encryption paradigm," 2025, *arXiv:2502.01013*.
[151] C. Benzaid and T. Taleb, "AI-driven zero touch network and service management in 5G and beyond: Challenges and research directions," *IEEE Netw.*, vol. 34, no. 2, pp. 186–194, Mar. 2020.
[152] Z. Chen, Z. Zhang, and Z. Yang, "Big AI models for 6G wireless networks: Opportunities, challenges, and research directions," *IEEE Wireless Commun.*, vol. 31, no. 5, pp. 164–172, Oct. 2024.
[153] S. Soro, "TinyML for ubiquitous edge AI," 2021, *arXiv:2102.01255*.
[154] O. Aouedi, V. A. Le, K. Piamrat, and Y. Ji, "Deep learning on network traffic prediction: Recent advances, analysis, and future directions," *ACM Comput. Surveys*, vol. 57, no. 6, pp. 1–37, Jun. 2025.
[155] I. Khan, "Edge Enhanced Network Monitoring Using TinyML," Master's thesis, Fac. Inf. Technol. Elect. Eng., Univ. Oulu, Oulu, Finland, Jun. 2024. [Online]. Available: https://oulurepo.oulu.fi/handle/10024/51084
[156] D. Bega, M. Gramaglia, A. Garcia-Saavedra, M. Fiore, A. Banchs, and X. Costa-Perez, "Network slicing meets artificial intelligence: An AI-based framework for slice management," *IEEE Commun. Mag.*, vol. 58, no. 6, pp. 32–38, Jun. 2020.
[157] M. Latah and L. Toker, "Artificial intelligence enabled software-defined networking: A comprehensive overview," *IET Netw.*, vol. 8, no. 2, pp. 79–99, Mar. 2019.
[158] J. Ali, H. Herbert Song, V. Sharma, and M. A. Al-Khasawneh, "DDoS intrusions detection in low power SD-IoT devices leveraging effective machine learning," *IEEE Trans. Consum. Electron.*, vol. 71, no. 1, pp. 343–351, Feb. 2025.
[159] M. Nekovee, S. Sharma, N. Uniyal, A. Nag, R. Nejabati, and D. Simeonidou, "Towards AI-enabled microservice architecture for network function virtualization," in *Proc. IEEE 8th Int. Conf. Commun. Netw. (ComNet)*, Oct. 2020, pp. 1–8.
[160] B. Cao, L. Zhang, Y. Li, D. Feng, and W. Cao, "Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework," *IEEE Commun. Mag.*, vol. 57, no. 3, pp. 56–62, Mar. 2019.
[161] A. Clemm, L. Ciavaglia, L. Z. Granville, and J. Tantsura, "RFC 9315: Intent-based networking-concepts and definitions," IETF, RFC 9315, Oct. 2022. [Online]. Available: https://datatracker.ietf.org/doc/rfc9315/
[162] Y. Njah, A. Leivadeas, and M. Falkner, "An AI-driven intent-based network architecture," *IEEE Commun. Mag.*, vol. 63, no. 4, pp. 146–153, Apr. 2025.
[163] L. Velasco et al., "End-to-end intent-based networking," *IEEE Commun. Mag.*, vol. 59, no. 10, pp. 106–112, Oct. 2021.
[164] D. M. Manias, A. Chouman, and A. Shami, "Towards intent-based network management: Large language models for intent extraction in 5G core networks," in *Proc. 20th Int. Conf. Design Reliable Commun. Netw. (DRCN)*, May 2024, pp. 1–6.
[165] Y. Ahn, "An intent-based management framework for on-device artificial intelligence in smart factory," in *Proc. KICS Winter Conf.*, 2025, pp. 1–3.
[166] S. O. Ajakwe and D.-S. Kim, "Time sensitive anti-infoswarm agnostic intelligence for safe UAV communication," in *Proc. 15th Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2024, pp. 1614–1619.
[167] M. Liyanage et al., "A survey on zero touch network and service management (ZSM) for 5G and beyond networks," *J. Netw. Comput. Appl.*, vol. 203, Jul. 2022, Art. no. 103362.
[168] W. B. Abbas, Q. Z. Ahmed, F. A. Khan, N. S. Mian, P. I. Lazaridis, and P. Sureephong, "Designing future wireless networks (FWN)s with net zero (NZ) and zero touch (ZT) perspective," *IEEE Access*, vol. 11, pp. 83301–83321, 2023.
[169] J. Shodamola, H. Qureshi, U. Masood, and A. Imran, "Towards addressing the spatial sparsity of MDT reports to enable zero touch network automation," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2021, pp. 1–6.
[170] G. Samaras, M. Mertiri, M.-E. Xezonaki, N. Psaromanolakis, V. Theodorou, and T. Bozios, "Unlocking the path towards automation of tiny machine learning for edge computing," in *Proc. Int. Conf. Smart Appl., Commun. Netw. (SmartNets)*, May 2024, pp. 1–6.
[171] T. Wang, J. Li, W. Wei, W. Wang, and K. Fang, "Deep-learning-based weak electromagnetic intrusion detection method for zero touch networks on industrial IoT," *IEEE Netw.*, vol. 36, no. 6, pp. 236–242, Nov. 2022.
[172] L. Yang, X. Lin, M. Reza, Y. Ren, J. Chen, and K. B. Letaief, "Towards zero touch networks: Cross-layer automated security solutions for 6G wireless networks," *IEEE Trans. Commun.*, vol. 73, no. 9, pp. 7650–7679, Sep. 2025.

[173] H. Y. Zhu, N. Q. Hieu, D. T. Hoang, D. N. Nguyen, and C.-T. Lin, "A human-centric metaverse enabled by brain–computer interface: A survey," *IEEE Commun. Surveys Tuts.*, vol. 26, no. 3, pp. 2120–2145, 3rd Quart., 2024.

[174] A. Kawala-Sterniuk et al., "Summary of over fifty years with brain–computer interfaces—A review," *Brain Sci.*, vol. 11, no. 1, p. 43, Jan. 2021.

[175] R. Das, F. Moradi, and H. Heidari, "Biointegrated and wirelessly powered implantable brain devices: A review," *IEEE Trans. Biomed. Circuits Syst.*, vol. 14, no. 2, pp. 343–358, Apr. 2020.

[176] J. Yoo and M. Shoaran, "Neural interface systems with on-device computing: Machine learning and neuromorphic architectures," *Current Opinion Biotechnol.*, vol. 72, pp. 95–101, Dec. 2021.

[177] D. Kudithipudi et al., "Neuromorphic computing at scale," *Nature*, vol. 637, no. 8047, pp. 801–812, 2025.

[178] A. Khajooei, M. Jamshidi, and S. B. Shokouhi, "A super-efficient TinyML processor for the edge metaverse," *Information*, vol. 14, no. 4, p. 235, Apr. 2023.

[179] W.-B. Jiang, L.-M. Zhao, and B.-L. Lu, "Large brain model for learning generic representations with tremendous EEG data in BCI," 2024, *arXiv:2405.18765*.

[180] D. Qiao, L. Qian, S. Guo, J. Zhao, and P. Zhou, "AMFL: Resource-efficient adaptive metaverse-based federated learning for the human-centric augmented reality applications," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 36, no. 5, pp. 7888–7902, May 2025.

[181] Y. Yang et al., "6G network AI architecture for everyone-centric customized services," *IEEE Netw.*, vol. 37, no. 5, pp. 71–80, Sep. 2023.

[182] Future Technology Trends of Terrestrial International Mobile Telecommunications Systems Towards 2030 and Beyond, Standard WP5D, International Telecommunication Union, 2022.

[183] *Study on Enhancement for Data Collection for NR and ENDC*, document TR-37.817, 2022. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/37_series/37.817/

[184] *Study on Enhancements for Artificial Intelligence (AI)/Machine Learning (ML) for NG-RAN*, document TR-38.743, 2024. [Online]. Available: https://www.3gpp.org/ftp/Specs/archive/38_series/38.743/

[185] X. Lin, L. Kundu, C. Dick, and S. Velayutham, "Embracing AI in 5G-advanced toward 6G: A joint 3GPP and O-RAN perspective," *IEEE Commun. Standards Mag.*, vol. 7, no. 4, pp. 76–83, Dec. 2023.

[186] S. Wang, M. A. Qureshi, L. Miralles-Pechuán, T. Huynh-The, T. R. Gadekallu, and M. Liyanage, "Applications of explainable AI for 6G: Technical aspects, use cases, and research challenges," 2021, *arXiv:2112.04698*.

[187] F. Rezazadeh et al., "Toward explainable reasoning in 6G: A proof of concept study on radio resource allocation," *IEEE Open J. Commun. Soc.*, vol. 5, pp. 6239–6260, 2024.

[188] S.-K. Chou, J. Hribar, M. Mohorčič, and C. Fortuna, "Towards the standardization of energy efficiency metrics of the AI lifecycle in 6G and beyond," in *Proc. IEEE Conf. Standards Commun. Netw.*, Belgrade, Serbia, Nov. 2024, pp. 187–190.

[189] S. Faye et al., "Integrating network digital twinning into future AI-based 6G systems: The 6G-TWIN vision," in *Proc. IEEE Joint Eur. Conf. Netw. Commun. 6G Summit*, Antwerp, Belgium, Jun. 2024, pp. 883–888.

[190] H. Du et al., "Enhancing deep reinforcement learning: A tutorial on generative diffusion models in network optimization," *IEEE Commun. Surveys Tuts.*, vol. 26, no. 4, pp. 2611–2646, 2024.

[191] Z. Wang, Q. Hu, R. Li, M. Xu, and Z. Xiong, "Incentive mechanism design for joint resource allocation in blockchain-based federated learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 5, pp. 1536–1547, May 2023.

[192] W. Tong and G. Y. Li, "Nine challenges in artificial intelligence and wireless communications for 6G," *IEEE Wireless Commun.*, vol. 29, no. 4, pp. 140–145, Aug. 2022.

[193] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. PMLR Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.

[194] X. Wei, B.-H. Juang, O. Wang, S. Zhou, and G. Y. Li, "Accretionary learning with deep neural networks with applications," *IEEE Trans. Cognit. Commun. Netw.*, vol. 10, no. 2, pp. 660–673, Apr. 2024.

[195] P. Li, Y. Xing, and W. Li, "Distributed AI-native architecture for 6G networks," in *Proc. Int. Conf. Inf. Process. Netw. Provisioning (ICIPNP)*, Beijing, China, Sep. 2022, pp. 57–62.

[196] J. Kühnel, T. Eißmann, C. Wiede, D. Schwung, and A. Grabmaier, "Semi-supervised anomaly detection in the TinyML domain through multi-target few-shot domain adaptation," in *Proc. IEEE 29th Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, vol. 133, Sep. 2024, pp. 1–8.

[197] S. Ullah et al., "Homomorphic encryption applications for IoT and light-weighted environments: A review," *IEEE Internet Things J.*, vol. 12, no. 2, pp. 1222–1246, Jan. 2025.

[198] O. Aouedi et al., "A survey on intelligent Internet of Things: Applications, security, privacy, and future directions," *IEEE Commun. Surveys Tuts.*, vol. 27, no. 2, pp. 1238–1292, Apr. 2025.

[199] S. A. Ajagbe, O. D. Adeniji, A. A. Olayiwola, and S. F. Abiona, "Advanced encryption standard (AES)-based text encryption for near field communication (NFC) using Huffman compression," *Social Netw. Comput. Sci.*, vol. 5, no. 1, p. 156, Jan. 2024.

[200] T.-H. Vu, S. K. Jagatheesaperumal, M.-D. Nguyen, N. Van Huynh, S. Kim, and Q.-V. Pham, "Applications of generative AI (GAI) for mobile and wireless networking: A survey," *IEEE Internet Things J.*, vol. 12, no. 2, pp. 1266–1290, Jan. 2025.

[201] T. L. Nguyen, M. T. de Oliveira, A. Braeken, A. Y. Ding, and Q.-V. Pham, "Towards verifiable federated unlearning: Framework, challenges, and the road ahead," *IEEE Internet Comput.*, early access, Jan. 23, 2026, doi: 10.1109/MIC.2026.3656638.

[202] W. Huang et al., "A plug-in tiny AI module for intelligent and selective sensor data transmission," 2024, *arXiv:2402.02043*.

[203] Y. Chen et al., "MAPO: Boosting large language model performance with model-adaptive prompt optimization," 2024, *arXiv:2407.04118*.

[204] X. Chen et al., "LightNER: A lightweight tuning paradigm for low-resource NER via pluggable prompting," 2021, *arXiv:2109.00720*.

[205] C. He et al., "UltraEval: A lightweight platform for flexible and comprehensive evaluation for LLMs," 2024, *arXiv:2404.07584*.

[206] N. C. Luong et al., "Advanced learning algorithms for integrated sensing and communication (ISAC) systems in 6G and beyond: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 28, pp. 2572–2611, 2026.

[207] J. Zhang et al., "Four steps toward 6G AI-enabled air interface: Wireless environmental information sensing, feature, semantics, and knowledge," *IEEE Commun. Mag.*, vol. 63, no. 8, pp. 56–62, Aug. 2025.

[208] Z. Feng, Z. Wei, X. Chen, H. Yang, Q. Zhang, and P. Zhang, "Joint communication, sensing, and computation enabled 6G intelligent machine system," *IEEE Netw.*, vol. 35, no. 6, pp. 34–42, Nov. 2021.

[209] Z. Liu et al., "Integrated sensing and edge AI: Realizing intelligent perception in 6G," *IEEE Commun. Surveys Tuts.*, vol. 28, pp. 2725–2770, 2026.

[210] M. Iovene et al., "Defining AI native: A key enabler for advanced intelligent telecom networks," Ericsson White Paper, Mar. 2023. [Online]. Available: https://www.ericsson.com/en/reports-and-papers/white-papers/ai-native

[211] L. Zheng et al., "MAgent: A many-agent reinforcement learning platform for artificial collective intelligence," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 8222–8223.

[212] M. Li, F. R. Yu, P. Si, Y. Zhang, and Y. Qian, "Intelligent resource optimization for blockchain-enabled IoT in 6G via collective reinforcement learning," *IEEE Netw.*, vol. 36, no. 6, pp. 175–182, Nov. 2022.

[213] L. Rosenberg, G. Willcox, H. Schumann, and G. Mani, "Towards collective superintelligence: Amplifying group IQ using conversational swarms," 2024, *arXiv:2401.15109*.

[214] S. Modi et al.. *The 6G Network is the Future of AI*. Boston Consulting Group. Accessed: Feb. 26, 2026. [Online]. Available: https://www.bcg.com/publications/how-6gnetworks-will-shape-the-next-era-of-ai