International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Y.4475

(08/2020)

SERIES Y: GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS, NEXT-GENERATION NETWORKS, INTERNET OF THINGS AND SMART CITIES

Internet of things and smart cities and communities – Frameworks, architectures and protocols

Lightweight intelligent software framework for Internet of things devices

Recommendation ITU-T Y.4475

International Telecommunication Union

ITU-T Y-SERIES RECOMMENDATIONS

GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS, NEXT-GENERATION NETWORKS, INTERNET OF THINGS AND SMART CITIES

GLOBAL INFORMATION INFRASTRUCTURE	
General	Y.100–Y.199
Services, applications and middleware	Y.200–Y.299
Network aspects	Y.300–Y.399
Interfaces and protocols	Y.400–Y.499
Numbering, addressing and naming	Y.500–Y.599
Operation, administration and maintenance	Y.600–Y.699
Security	Y.700–Y.799
Performances	Y.800–Y.899
INTERNET PROTOCOL ASPECTS	
General	Y.1000–Y.1099
Services and applications	Y.1100–Y.1199
Architecture, access, network capabilities and resource management	Y.1200–Y.1299
Transport	Y.1300–Y.1399
Interworking	Y.1400–Y.1499
Quality of service and network performance	Y.1500–Y.1599
Signalling	Y.1600–Y.1699
Operation, administration and maintenance	Y.1700–Y.1799
Charging	Y.1800–Y.1899
IPTV over NGN	Y.1900–Y.1999
NEXT GENERATION NETWORKS	
Frameworks and functional architecture models	Y.2000–Y.2099
Quality of Service and performance	Y.2100–Y.2199
Service aspects: Service capabilities and service architecture	Y.2200–Y.2249
Service aspects: Interoperability of services and networks in NGN	Y.2250–Y.2299
Enhancements to NGN	Y.2300–Y.2399
Network management	Y.2400–Y.2499
Network control architectures and protocols	Y.2500–Y.2599
Packet-based Networks	Y.2600–Y.2699
Security	Y.2700–Y.2799
Generalized mobility	Y.2800–Y.2899
Carrier grade open environment	Y.2900–Y.2999
FUTURE NETWORKS	Y.3000–Y.3499
CLOUD COMPUTING	Y.3500–Y.3599
BIG DATA	Y.3600–Y.3799
QUANTUM KEY DISTRIBUTION NETWORKS	Y.3800–Y.3999
INTERNET OF THINGS AND SMART CITIES AND COMMUNITIES	
General	Y.4000–Y.4049
Definitions and terminologies	Y.4050–Y.4099
Requirements and use cases	Y.4100–Y.4249
Infrastructure, connectivity and networks	Y.4250–Y.4399
Frameworks, architectures and protocols	**Y.4400–Y.4549**
Services, applications, computation and data processing	Y.4550–Y.4699
Management, control and performance	Y.4700–Y.4799
Identification and security	Y.4800–Y.4899
Evaluation and assessment	Y.4900–Y.4999

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T Y.4475

Lightweight intelligent software framework for Internet of things devices

Summary

Recommendation ITU-T Y.4475 addresses the concept of the lightweight intelligent software framework (LISF) that supports Internet of things (IoT) applications requiring intelligent processing, and enables such processing to work on resource-limited IoT devices. Recommendation ITU-T Y.4475 identifies general requirements and provides a functional architecture for the LISF based on the IoT reference model established in Recommendation ITU-T Y.4000.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T Y.4475	2020-08-29	20	11.1002/1000/14377

Keywords

IoT device, intelligence software framework, lightweight.

* To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

Table of Contents

Recommendation ITU-T Y.4475

Lightweight intelligent software framework for Internet of things devices

1 Scope

This Recommendation specifies general requirements and functionalities of the lightweight intelligent software framework (LISF) working on resource-limited Internet of things (IoT) devices.

In particular, for the LISF, the scope of this Recommendation covers:

– the concept of the intelligent software framework (ISF) and necessity for a lightweight version;

– features and general requirements;

– functional architecture.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T Y.4453] Recommendation ITU-T Y.4453 (2016), *Adaptive software framework for Internet of things devices*.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 application [b-ITU-T Y.2091]: A structured set of capabilities, which provide value-added functionality supported by one or more services, which may be supported by an API interface.

3.1.2 Internet of things [b-ITU-T Y.4000]: A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies.

NOTE 1 – Through the exploitation of identification, data capture, processing and communication capabilities, the IoT makes full use of things to offer services to all kinds of applications, whilst ensuring that security and privacy requirements are fulfilled.

NOTE 2 – From a broad perspective, the IoT can be perceived as a vision with technological and societal implications.

3.1.3 capability [b-ITU-R M.1224-1]: The ability of an item to meet a service demand of given quantitative characteristics under given internal conditions.

3.1.4 service [b-ITU-T Y.2091]: A set of functions and facilities offered to a user by a provider.

3.1.5 functional entity [b-ITU-T Y.2012]: An entity that comprises an indivisible set of specific functions. Functional entities are logical concepts, while groupings of functional entities are used to describe practical, physical implementations.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms

3.2.1 lightweight intelligent software framework (LISF): Middleware used to enable intelligent capabilities for each Internet of things (IoT) application by using online partial learning and inferencing processing with the system resources of IoT devices.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

API Application Programming Interface

CNN Convolutional Neural Network

CPU Central Processing Unit

CUDA Compute Unified Device Architecture

FE Functional Entity

FPS Frames Per Second

GEMM General Matrix Multiply

GPGPU General Purpose Computing Graphics-Processing Unit

GPU Graphics-Processing Unit

HW Hardware

ICT Information and Communication Technology

ID Identifier

IoT Internet of Things

ISF Intelligent Software Framework

KPI Key Performance Indicator

LISF Lightweight Intelligent Software Framework

LMDB Lightning Memory-Mapped Database

OpenCL Open Computing Language

RAM Random Access Memory

SW Software

5 Conventions

The following conventions are used in this Recommendation:

– The keywords "is required to" indicate a requirement which must be strictly followed and from which no deviation is permitted, if conformance to this Recommendation is to be claimed.

– The keywords "is prohibited from" indicate a requirement which must be strictly followed and from which no deviation is permitted, if conformance to this Recommendation is to be claimed.

– The keywords "is recommended" indicate a requirement which is recommended but which is not absolutely required. Thus, this requirement need not be present to claim conformance.

– The keywords "is not recommended" indicate a requirement which is not recommended but which is not specifically prohibited. Thus, conformance with this Recommendation can still be claimed even if this requirement is present.
– The keywords "can optionally" indicate an optional requirement which is permissible, without implying any sense of being recommended. This term is not intended to imply that the vendor's implementation must provide the option and the feature can be optionally enabled by the network operator/service provider. Rather, it means the vendor may optionally provide the feature and still claim conformance with this Recommendation.
– The keyword "functions" is defined as a collection of functionalities.
– The keyword "functional block" is defined as a group of functionalities that have not been further subdivided at the level of detail described in this Recommendation.

6 Introduction

With the development of IoT technologies, intelligent technologies are rapidly developing and launching in various information and communication technology (ICT) service domains. In order to support intelligent capability on resource-limited IoT devices, an IoT software (SW) framework plays an important role. A relevant IoT SW framework is also published as [ITU-T Y.4453] to support adaptive application capability for IoT devices.

Generally, an ISF primarily performs in server-side cloud computing and requires high-performance computing environments with rich resources. A framework conceptually consists of pre-processing entity, learning entity, and inferencing entity for intelligence processing. Additionally, training data and real-data are needed. An ISF demands high-quality training data through a pre-processing entity, generates a learning model by using a learning entity and predicts inferencing for new real-data by an inferencing entity.

– The pre-processing entity filters data or transforms data into a different format.
– The learning entity searches for regularity and patterns in data.
– The inferencing entity classifies and estimates new real data.

Many kinds of current IoT devices have been embedded by a general-purpose computing graphics-processing unit (GPGPU) and multicore central processing unit (CPU) with limited processing capability. Nevertheless, with these IoT devices, new requirements for intelligent IoT services are demanded to provide near-real-time IoT data processing, privacy handling, and low latency. In order to provide intelligence capability in embedded systems with limited system resources, there are several issues when considering their hardware (HW) and SW.

– HW: Take into account clock speed, number of cores and efficient power management, regrading CPU and GPGPU in order to support one specified task or set of tasks.
– SW: Take into account acceleration technology using CPU- and GPGPU-based parallelism and lightweight technology through optimization.

Generally, embedded systems are dedicated to one specific task or set of tasks so that it is not easy for them to work complex programs with heavy workloads such as intelligent services, called CPU-intensive programs. When these heavy programs work on embedded systems, different types of overhead (e.g., high CPU or GPGPU utilization, CPU or GPGPU thermal and memory leak) can shut down or crash the entire system. Nevertheless, embedded systems tend to execute complex programs with heavy workloads in order to provide intelligent services, such as face recognition, vacuum cleaning and autonomous car or drone driving. Therefore, the LISF needs to consider a method for supporting intelligence on resource-limited IoT devices. With the LISF, IoT devices can run intelligent IoT applications handling in a resource-constrained environment and can support intelligent capability in a standardized way. The combination of IoT devices with the LISF enables intelligent services for users.

An intelligent job such as machine learning and deep learning require a lot of computation. Therefore, it is essential to support the best utilization of CPU and GPGPU capacities. The approach can be provided with accelerating technology based on SW such as Open Computing Language (OpenCL) and Compute Unified Device Architecture (CUDA). Also, a variety of intelligent applications working on embedded systems requires a high-performance computing environment, so that embedded systems need application of accelerating technology. Therefore, the LISF provides a method for operating an embedded system by accelerating processing capability. This method initializes, configures and processes in parallel. First, initializing and configuring entities perform mathematical operations using a parallel managing function entity. The parallel managing function entity allocates a device memory, copies data from a host to a device, sets a kernel and again copies results of an operation. Therefore, the instances of the kernel are executed in parallel; each of them processes a single work item and all are executed together as multiple work items as a part of a work group. Second, processing in parallel involves mathematical operations performed by an acceleration-managing function entity using the configured entities. In this situation, most entities have a trade-off between accuracy (e.g., object recognition rate) and real time (e.g., frames per second (FPS)). For example, if accuracy for object recognition rate is high, detection or inferencing time for objects can be slow. As an embedded system has overheads due to a lot of computation, applications and services may not work properly.

Figure 1 is a schematic diagram showing machine or deep learning from the system resources perspective. Figure 1 illustrates high-level features of LISF compared to ISF. ISF is designed to work on systems with rich resources (e.g., server computer, cloud computer), and has basically three entities, such as pre-processing, full learning and inferencing, and two data sources, such as a learning model and training data. An ISF focuses on providing faster intelligent services with high resource capability systems. So, an ISF can support full learning with big data for training. On the other hand, a LISF is designed to work on systems with limited resources, so that each entity and data must be optimized for these systems. Therefore, an LISF has three optimized functional entities (FEs), such as accelerated pre-processing, online partial learning and approximate inferencing, and two sources of input data, such as an optimal learning model and one-shot data for processing in resource-limited devices. These FEs constitute the core engine of an LISF.

An accelerated pre-processing entity accelerates data refinement, integration, reduction and transformation by making the best use of a GPGPU in order to maintain data consistency for learning. An online partial learning entity updates a part of the learning model without the assistance of a high-performance computer through real-time learning using one-shot training data, not batch learning. An initial learning model can be generated by a high-performance computer and the model is sent to embedded devices. An approximate inferencing entity predicts correspondence to new data when considering inference speed and accuracy. Additionally, an LISF is composed of two input data sources.

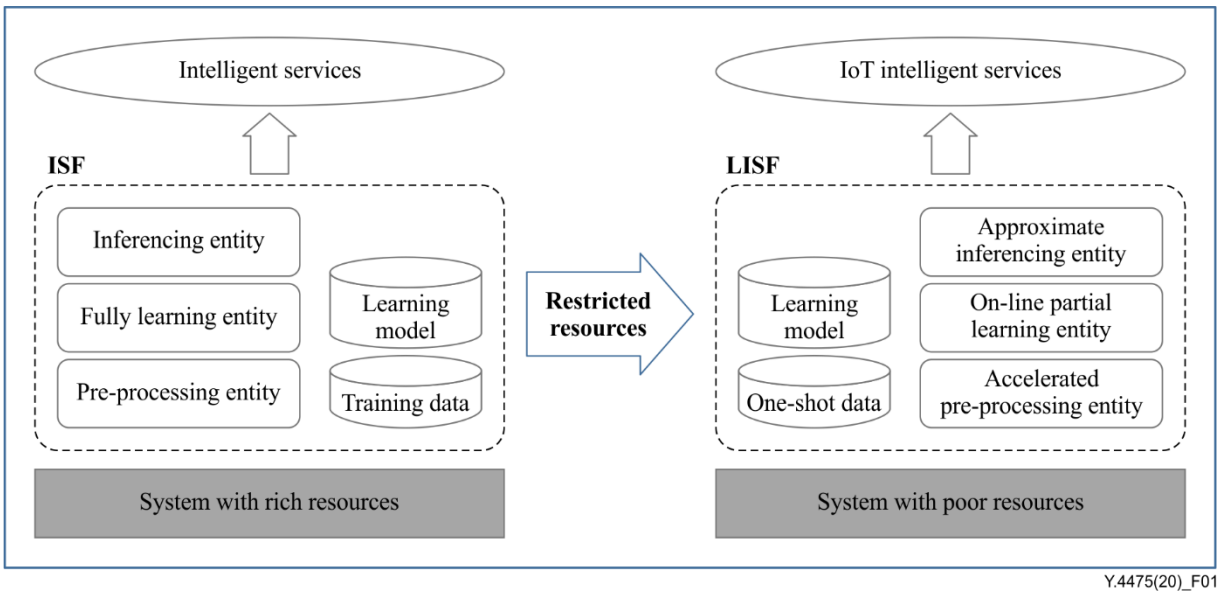

Figure 1 – Overview diagram of ISF and LISF in the aspect of system resources

7 LISF features and requirements

This clause describes LISF features and requirements. LISF features are explained by using a client and server mechanism.

7.1 LISF features

Performance, power and memory on embedded systems are aspects of an LISF when running. LISF-based artificial intelligence services support their real-time execution. However, the following three conditions should be avoided for providing the execution:

– slowing of service response time;

– stopping suddenly during execution;

– service termination due to abnormal execution.

These three conditions arise due to lack of resources in embedded systems. In order to resolve the issue, the LISF needs to support the following features:

– efficient use of embedded system resources, in which there are lightweight and accelerated techniques for CPU, GPGPU and memory;

– support for online partial learning due to resource-limited embedded systems – full learning generally works on server systems;

– generation of a personalized learning model that adapts to a system environment through online partial learning;

– collaboration with a client-server model, where the client is a poor-resource and the server a rich-resource system.

Figure 2 shows a client and server mechanism for the LISF. The server system and IoT device each have an intelligence framework. The system comprises of the device, platform, context, command queue and kernel. The device comprises actual processors for performing mathematical operations. The platform uses at least one CPU and one graphics-processing unit (GPU). The context comprises an entity for managing the resources in a device set. The command queue comprises an entity for executing a kernel and performing memory mapping or unmapping and synchronization. The kernel comprises a code running on the device. The artificial intelligence framework in the server system pre-processes and learns fully. Full learning can utilize a pruning method in order to generate an optimized learning model. The pruning method steadily changes superfluous weight values to zero

during learning training data. Pre-processing and full learning require a lot of computation, since they create an initial learning model using training data, so that the server system has a high-performance computing environment. The initial learning model transmits to the artificial intelligence framework of the IoT device.

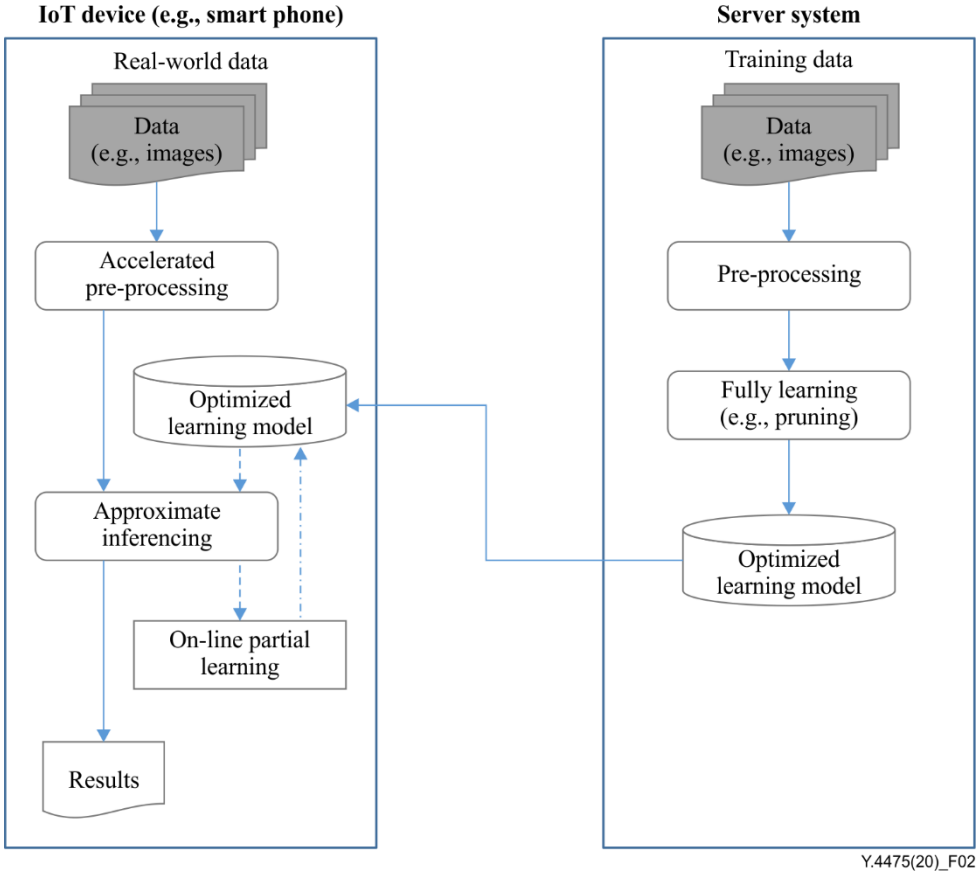

Figure 2 – Client and server mechanism of the LISF

The LISF consists of accelerated pre-processing, approximate inferencing, and online partial learning. Since the IoT device is a kind of embedded system, the LISF should be able to use system resources, such as CPU and GPGPU, as well as memory efficiently. Accelerated pre-processing, approximate inferencing and online partial learning should utilize accelerated techniques based on a GPGPU because embedded CPUs have complex computation process limitations. If real-world data input to an IoT device, the LISF uses the initial learning model to infer.

7.1.1 Accelerated pre-processing

Accelerated pre-processing takes real-world raw input data and converts it into a different form to generate optimized data for inferencing. Real-world input data may be somewhat inappropriate to use immediately for inference and learning. Since the LISF requires additional learning, called partial learning, for input data, it includes a function that generates a consistent learning database, such as the lightning memory-mapped database (LMDB) and LevelDB (key value-based storage library) in a pre-processing category. In order to support acceleration for methods of data conversion and database generation in LISF, LISF has three recipes: kernel-based data processing, internal memory management and data tiling.

There are many data conversion methods; LISF pre-processing supports four of them: normalization, transformation, reduction and discretization.

7.1.2 Online partial learning

Online partial learning processes one data source and updates the processed results in a learning model, which requires low computing power. In contrast, full learning initially uses a learning model transmitted from a server computer, which requires large amounts of training data for processing, called batch learning. The online partial learning mechanism is suitable for use in embedded systems such as IoT devices.

There are two types of online partial learning as follows.

– Fine-tuning based partial learning: When inferencing input data use a learning model, if an inference result for input data has low probability, fine-tuning based partial learning is used. Fine-tuning based partial learning upgrades an existing learning model through re-training regarding input data. It is possible to infer input data using an updated learning model.

– Scalable-tuning based partial learning: When inferencing input data use a learning model, if there is no inference result for input data, scalable-tuning based partial learning is used. Scalable-tuning based partial learning not only upgrades an existing learning model, but also adds a new class into the classification list (e.g., label.txt).

7.1.3 Approximate inferencing

Approximate inferencing predicts with pre-processed data. The predicted results can be used in a classification or estimation service, for example. For prediction, an approximate computing method is applied. Processing finds anr approximate rather than an exact answer. Approximate computing can achieve quicker response results than accurate ones.

7.1.4 Optimized learning model

The optimized learning model consists of a model architecture and model weighting. The model architecture determines a model configuration of stack layers from the perspective of machine learning or deep learning model. Model weights are initialized to arbitrary values, and are updated as learning proceeds with training data. Saving the learning model stores the model architecture and model weights. Three methods are used to generate an optimized learning model: sparse coding mechanism; learning-model compression mechanism; and data type lightweight mechanism.

7.1.5 Full learning

Full learning is a process of constructing a neural network from neural network nodes and then calculating a weighting of data input to each neural network node through a learning algorithm. Full learning can be achieved through various neural network-processing engines with learning algorithms. The neural network models can modify information directly described by the user or change the structure through editing tools to produce an optimized learning model.

7.2 LISF requirements

This clause describes the requirements for a LISF in accordance with the conceptual diagram in Figure 3. The LISF exists between the intelligent IoT application and the resource. The LISF supports online partial learning, approximate inference, performance and acceleration processing.

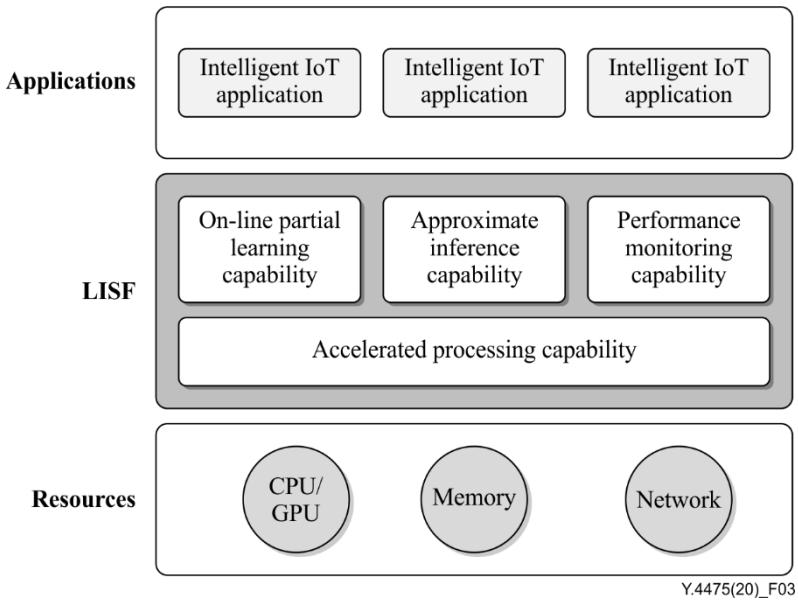

Figure 3 – A conceptual diagram of a LISF

7.2.1 Applications

Intelligent IoT applications utilize the capabilities exposed by the LISF and run on embedded devices. Intelligent IoT applications work with low latency and efficient energy power on embedded devices.

For development of intelligent IoT applications, based on the exposed capabilities of the LISF:

– it is required to perform intelligent computing (e.g., such as machine learning and deep learning) on a standalone IoT device;

– it is required to support an efficient, scalable intelligent computing mechanism in a resource-limited environment;

– it is recommended to support enhanced capabilities for handling low latency and accelerators;

– it is recommended to monitor the information about the resource usage status.

7.2.2 LISF

The LISF has capabilities for learning, inference, performance monitoring and accelerating processing. The LISF provides artificial intelligence optimized for embedded systems and handles processing in resource-constrained embedded systems that maximize low-power performance for real-time operation.

The LISF performs and exposes the capabilities as follows.

– Online partial learning capability performs online partial learning for intelligent IoT applications.

– Approximate inference capability performs real-time inferencing for intelligent IoT applications.

– Performance-monitoring capability performs monitoring for resource utilization.

– Accelerating processing capability supports accelerating technology for intelligent IoT applications.

For online partial learning capability:

– it is required to provide a mechanism for partial learning processing in a resource-limited environment;

– it is recommended to provide a lightweight learning model for an on-device environment;

– it is recommended to support high-density compression model and processing performance for quick learning;
– it is recommended to use parallel computing handling framework for accelerating the matrix operations of machine learning.

For approximate inference capability:

– it is recommended to support real-time processing for reasoning;
– it is recommended to support optimized graph network flow for inferencing;
– it is recommended to provide operation without network connection and simple configuration for artificial intelligence application.

For performance-monitoring capability:

– it is recommended to monitor performance for profiling function and data parsing analysis;
– it is recommended to provide information for GPGPU acceleration performance data collection;
– it is recommended to provide status information for resource execution.

For accelerating processing capability:

– it is recommended to provide GPU-based adaptively accelerating operations for embedded system performance on a number of IoT devices;
– it is recommended to provide operation in a binary format for parallel and acceleration processing;
– it is recommended to consider performance of IoT device processing to maximize parallelism regarding matrices with weights and bias values;
– it is recommended to share a memory between a server and IoT device to minimize the cost of the mathematical matrix multiply operation;
– it is recommended to provide an accelerator analysis unit with relevant IoT device information.

7.2.3 Resources

The resources utilize appropriate system-related capabilities, such as CPU/GPU, memory or network.

For the resource layer:

– it is required to collect and manage CPU/GPU to handle real-time processing of intelligent IoT application and LISF;
– it is recommended to support low-latency response and bandwidth efficiency for the network;
– it is recommended to provide a storage policy to handle private data.

8 Functional architecture of the LISF

The LISF provides intelligence to IoT applications that operate on resource-constrained IoT embedded devices.

The LISF is composed of four functions for:

1) online partial learning;
2) approximate inference;
3) performance monitoring;
4) accelerated processing.

The LISF starts operations through a service request to the intelligent IoT application, and executes the online partial learning function or the approximate inference function. It works in conjunction with accelerated processing capabilities to generate optimized learning models and reasoning output. The LISF provides the intelligent functionality for intelligent IoT applications consisting of generic application logic and intelligent processing logic.

The accelerated processing functions have the ability to configure computational resources and memory based on the processing requirements of the initial optimized learning model. This feature also enables fast and optimized work with online partial learning and approximate reasoning.

8.1 Online partial learning function

The online partial learning function, shown in Figure 4, is the foundation of the LISF. This function should run to update the initial learning model if some test dataset or reasoning capabilities of a new set of learning data are available. The function collects a training dataset from an intelligent IoT application and then performs machine learning to update the existing learning model. The training dataset consists of training data with labels in supervised modelling mode and delivers an online partial learning input. The format of training data with labels is required to comply with the specifications of the initial learning model. The online partial learning function supports its capabilities for intelligent IoT applications to allow them to train their learning models on it.

The online partial learning function consists of three FEs for learning set-up; model training; and training control.

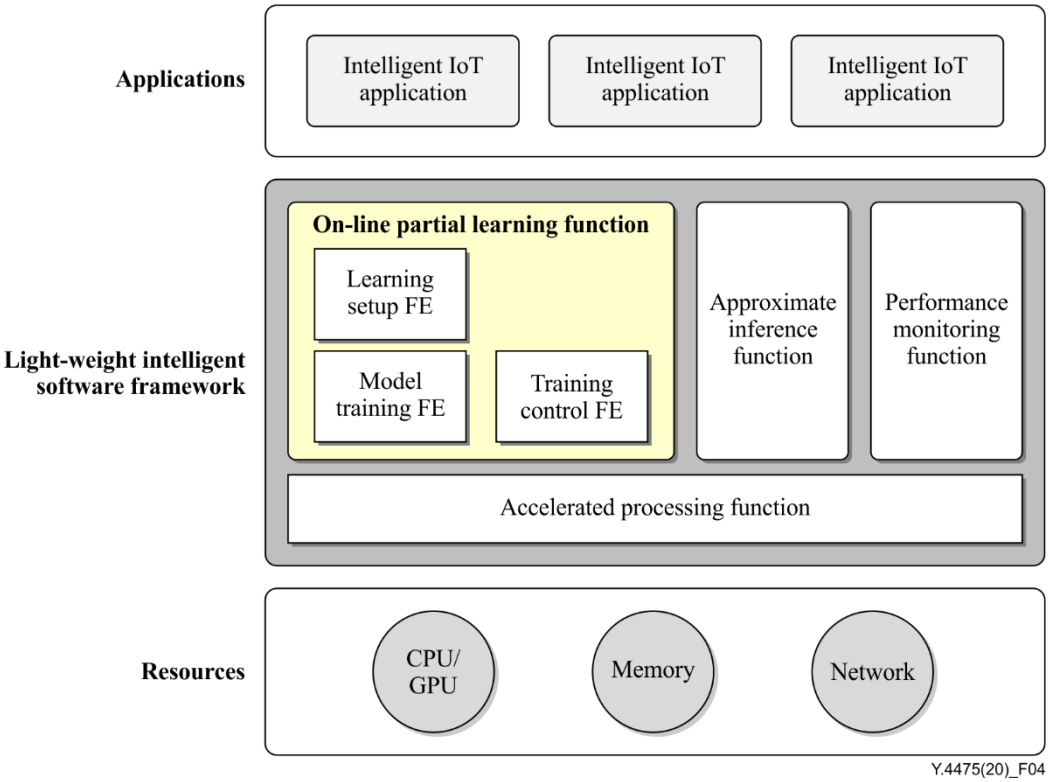

Figure 4 – Online partial leaning function in the LISF

8.1.1 Learning set-up FE

The learning set-up FE prepares initial model parameters using a trained and training dataset. It downloads training models from the server system through the network and imports a training dataset from the intelligent IoT application to update the initial learning model. The learning set-up FE operates the initial learning model for further modelling optimization. Model optimization can be

achieved by compression methods, such as parameter pruning and sharing, low-rank factorization, transferred and compact convolutional filters and knowledge distillation.

The learning set-up FE usually chooses a default model based on a specific learning model at the request of an intelligent service type or intelligent IoT application. The training dataset supports training data for the online partial learning function and the raw data or record format of the database containing the labels. The information listed in Table 8-1 is necessary for the learning set-up FE.

Table 8-1 –Information for the learning set-up FE

Set-up information	Description	Remarks
Identifier (ID) of intelligent service	ID for distinguishing intelligent service	Face recognition, character recognition, etc.
ID of learning model	The learning model to be used in training	Internal ID to distinguish learning model
Number of instances	The number of data instances in the training dataset	1 024, 2 048, or 4 096, etc.
Width	Width of each data contained in in the training dataset	64
Height	Height of each data contained in the training dataset	64
Number of channels	Number of channel of each data contained in the training dataset	3
Learning rate	Learning rate that controls the updating of gradient in backward pass	0.1, 0.01, etc.

8.1.2 Model training FE

The model training FE performs two major steps of the process for the learning model, namely the forward and backward passes. During a forward pass, training data is delivered through the neural network of the learning model; during a backward pass it returns through the neural network. The two passes are repeated until the cost of the loss and the loss in the iteration is less than or equal to a given threshold, to update the parameters (e.g., node weights and biases).

The model training FE has the same essential information as the learning set-up FE. The model training FE must generate the following information from all iterations:

– iteration number: the current iteration count in training;

– current cost: the cost calculated at current iteration n training.

The model training FE uses the generated information as input to the training control FE.

During the training process, the model training FE can request CPU/GPU and memory resources with accelerated processing function for rapid training. The following information is essential for the accelerated processing function:

– layer information: the types and quantities of constituent layers in the learning model.

8.1.3 Training control FE

The training control FE should be stopped if the cost is below a given level or the number of iterations exceeds a given threshold to prevent useless or unlimited training. The training control FE stops after monitoring the current cost of the learning model and the number of iterations of the model training FE. The following information is essential for training control:

– iteration threshold: the maximum number of iterations allowed for model training.

8.1.4 Operational procedure

The operation of the online partial learning function consists of three steps, as shown in Figure 5.

Step 1 is the model set-up phase for partial learning, which is executed in the learning set-up FE. An intelligent IoT application requests PartialLearnSetup() from the online partial learning function for initializing system resources, loading a model and converting the training data formatting with model, data and key performance indicator (KPI) information. Furthermore, the KPI received from the intelligent IoT application includes the minimum accuracy, maximum cost and maximum iteration, which are then used to set KPIs for the training control FE. After completing the work, PartialLearnSetupResult() notifies the set-up completion results to the intelligent IoT application.

Step 2 is the process for partial learning in the model training FE. The model training FE repeats the training iteration with the training model and dataset. The training control FE compares the initial target KPI with the measured KPI for each training iteration, and decides whether to perform additional training iterations based on the comparison result. If no additional training is required, the model training FE updates the training model. In addition, it notifies the result to the intelligent IoT application with the function PartialLearnResult().

Step 3 is executed in the learning set-up FE by PartialLearnComplete(), which is sent from the model training FE after completion of partial learning and training model update. Finally, this step releases system resources used for partial learning.

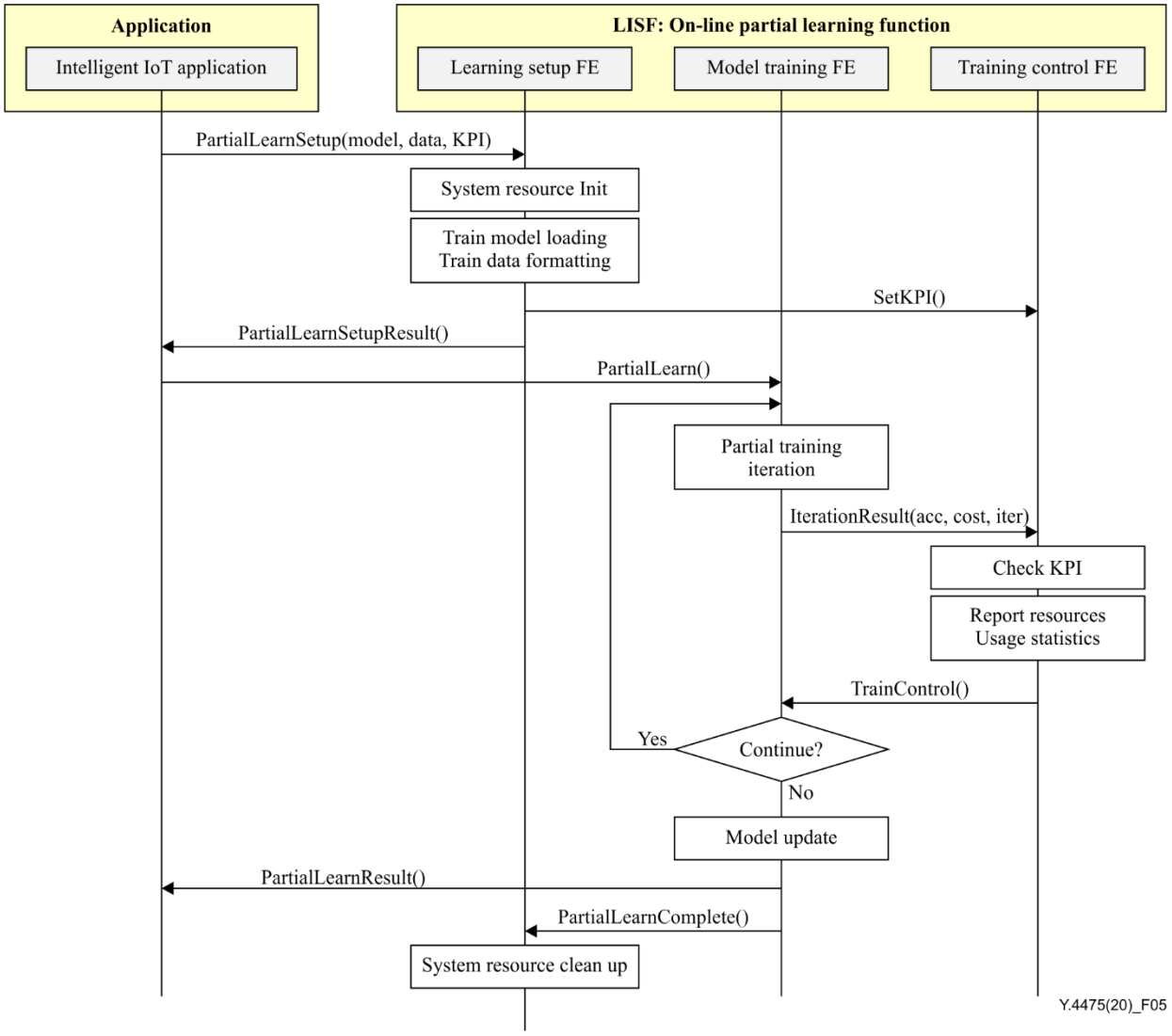

Figure 5 – Online partial learning procedure in the LISF

8.2 Approximate inference function

The approximate inference function, shown in Figure 6, manages and performs inference processes for intelligent IoT applications.

The approximate inference function consists of two FEs for: inference set-up and inference driving.

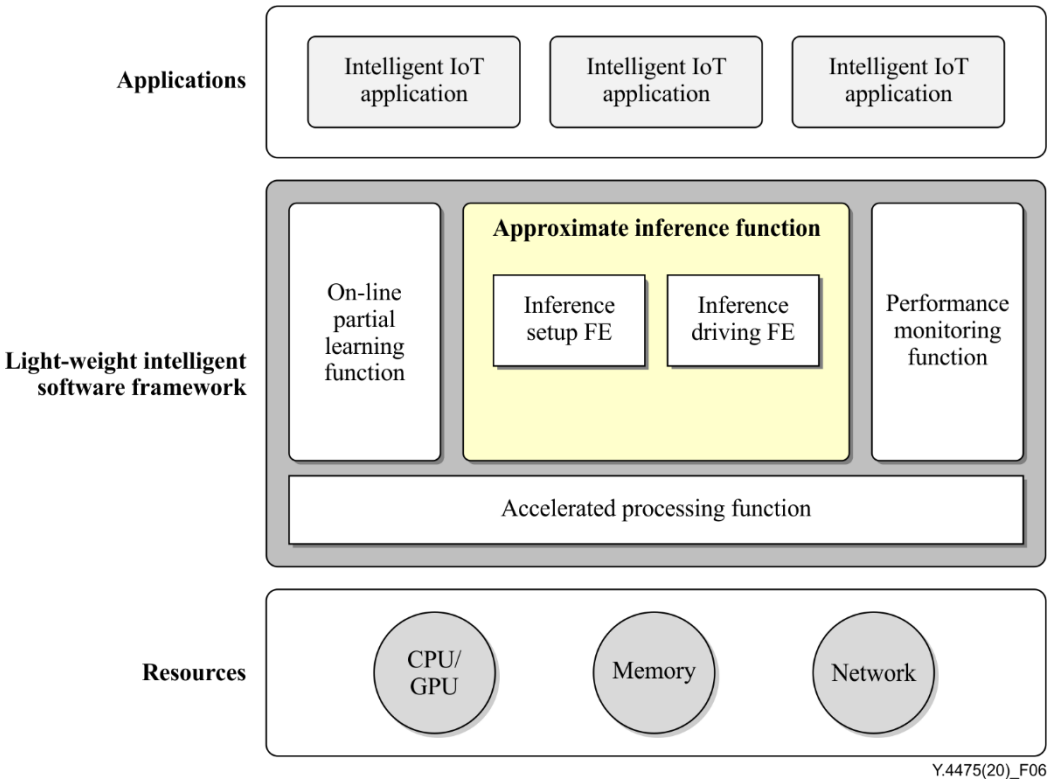

Figure 6 – Approximate inference function in the LISF

NOTE – The word approximate is used in this Recommendation to emphasize the inevitable loss of accuracy of the learning model due to the resource limitations of the IoT device, although the inference itself implies some approximation.

8.2.1 Inference set-up FE

The inference set-up FE prepares its learning model with a given test dataset. This FE installs the learning model from the server system via interprocess communication or network protocols, collects the test dataset from the intelligent IoT application, and verifies compliance with the learning model used in inference. The following information is necessary for inference set-up:

– ID for distinguishing intelligent service;

– ID of the learning model to be used in inference;

– the number of data instances in the test dataset;

– the width of each data item contained in the test dataset;

– the height of each data item contained in the test dataset;

– the number of channels of each data item contained in the test dataset.

8.2.2 Inference driving FE

The inference driving FE starts the inference process with information prepared by the inference set-up FE. This FE can request CPU/GPU and memory resources through the accelerated processing function for fast reasoning during inference. The following information should be assembled for use by the accelerated processing function:

– the ID of the learning model to be used in inference;
– learning model architectural information that describes how layers are interconnected;
– types and quantities of constituent layers in the learning model.

8.2.3 Operational procedure

The operation of the online approximate inference function consists of three steps as shown in Figure 7.

Step 1 is the model set-up phase performed in the inference set-up FE. It includes initializing the system resources necessary for the inference model and modelling process according to the model information by the intelligent IoT application.

Step 2 is the process to infer in the inference driving FE. The inference data requested from the intelligent IoT application is converted into a suitable format by the inference set-up FE, which requests inference from the inference driving FE. The inference driving FE sends the result to the inference set-up FE. The inference result also passes to the intelligent IoT application through the inference set-up FE to clean up the inference data. Step 2 repeats until completion for all inference data.

Step 3 is the process to release the resources used in the inference model. The inference set-up FE releases system resources to clean up the model.

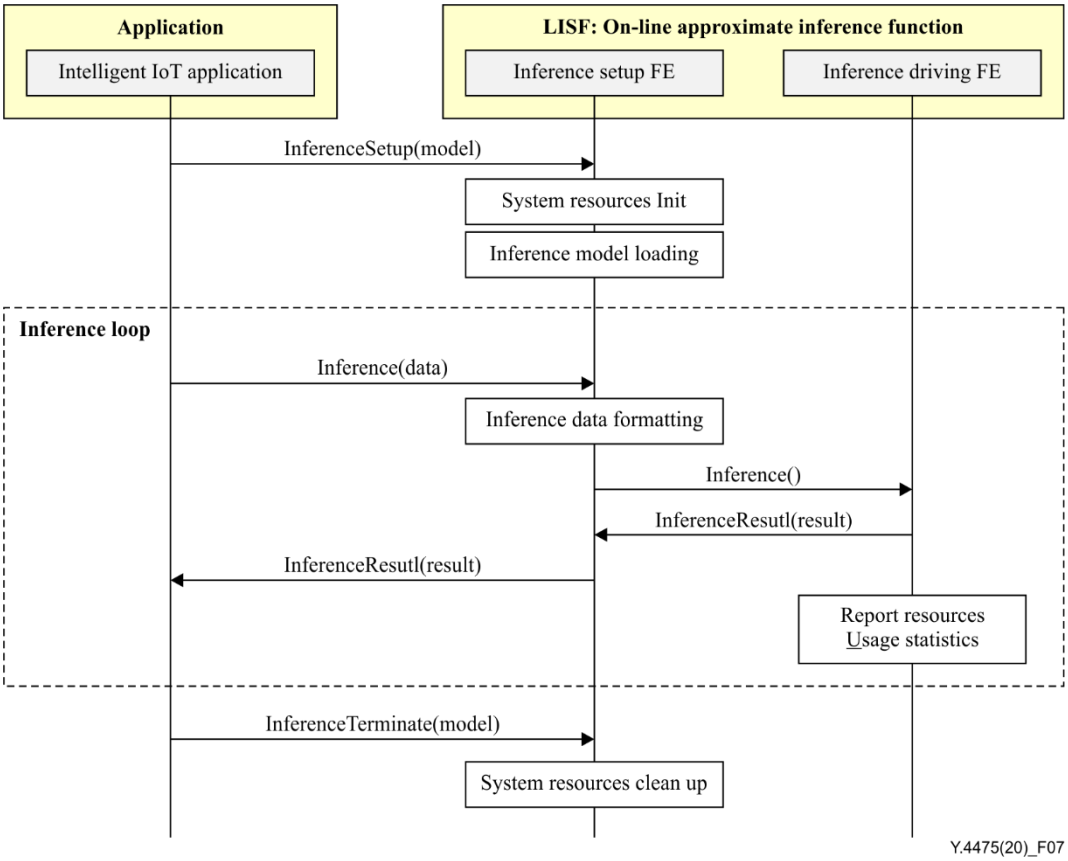

Figure 7 – Approximate inference procedure in the LISF

8.3 Performance-monitoring function

The performance-monitoring function, shown in Figure 8, gathers performance statistics about the CPU/GPU, memory and other system resource usage in the resource layer when the online partial learning function or approximate inference function operates. This function handles the accelerated processing function with collected statistics to investigate the possibility of accelerating the online

partial learning or approximate inference. The performance-monitoring function consist of two FEs for: learning monitoring and inference monitoring.

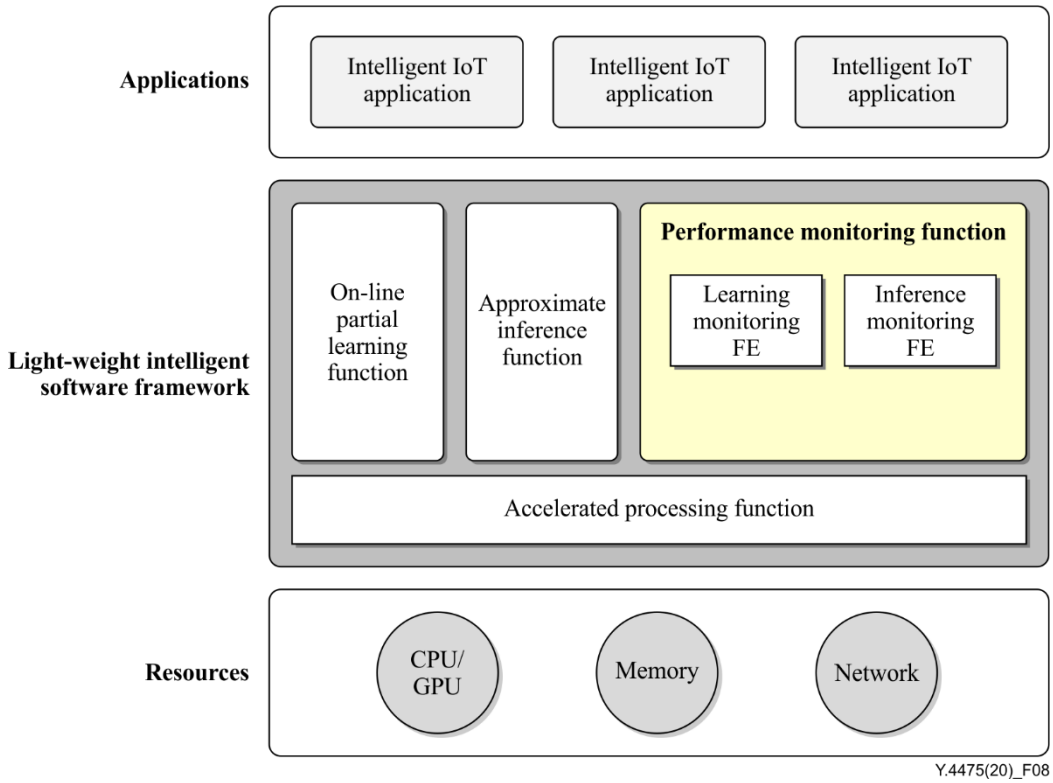

Figure 8 – Performance-monitoring function in the LISF

8.3.1 Learning-monitoring FE

The learning-monitoring FE keeps track of system resources, such as CPU/GPU usage and frequency, or the amount of random-access memory (RAM) available while the online partial learning function is running. When accelerated processing functions also use standard parallel computing application programming interfaces (APIs), the learning-monitoring FE collects context, queue and kernel usage information in real time, as well as information about the API trace and resource statistics.

8.3.2 Inference-monitoring FE

The inference-monitoring FE keeps tracks of system resources such as CPU/GPU usage and frequency, or the amount of RAM available, while the approximate inference function is running. The inference-monitoring FE also collects information about API traces and resource statistics in real time, as well as context, queue, and kernel usage when accelerated processing functions use the APIs specified in the standard parallel computing framework for accelerating performance.

8.3.3 Operational procedure

The operation of the performance-monitoring procedure is shown in Figure 9. The performance-monitoring function provides two sources of reporting resource usage statistics data on: the learning model; and the inference model.

The learning-monitoring FE collects performance metrics measured and reported by the training control FE in the online partial learning function in relation to the learning model during partial learning.

The inference-monitoring FE collects performance indicators measured and reported by the inference driving FE in the online approximate inference function.

The performance statistics data can be used for the parallelization-managing FE in the accelerated processing function, which optimizes the memory allocation, device allocation, and kernel configuration required for learning or inference models.

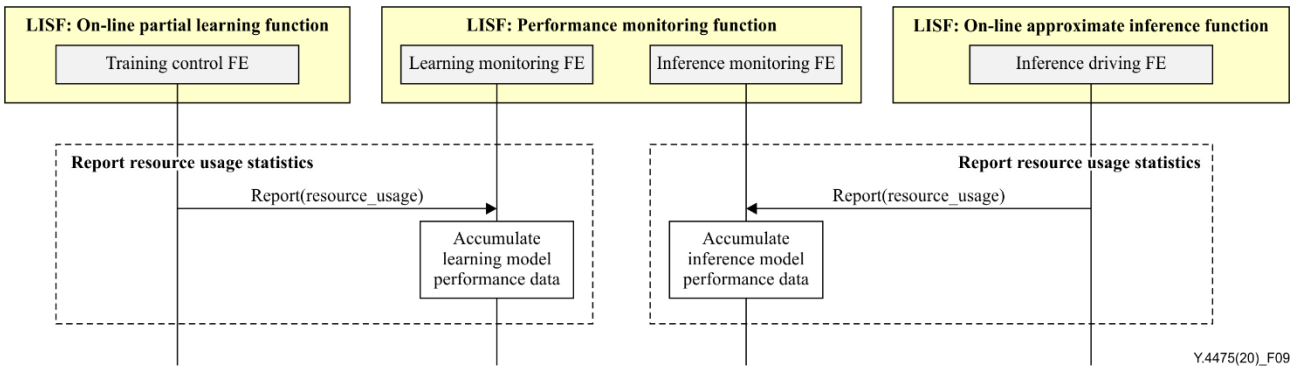

Figure 9 – Performance-monitoring procedure in the LISF

8.4 Accelerated processing function

The accelerated processing function, shown in Figure 10, provides optimized utilization of resources for a given learning model. Each layer of the learning model requires a variety of mathematical operations for learning and inference. The accelerated processing function is optimized for underlying HW resources and provides parallel mathematical operations, as well as examining the statistics provided by performance-monitoring function to identify bottlenecks. According to the results, the configuration of HW resource usage is updated to accelerate the online partial learning function and approximate inference function. The accelerated processing function is necessary for accelerating mathematical operations used in the constituent layers of the learning model. Mathematical operations are general linear algebra operations such as vector addition, scalar multiplication, linear combinations and matrix multiplication.

The accelerated processing function consist of two FEs for: parallelization management and acceleration management.

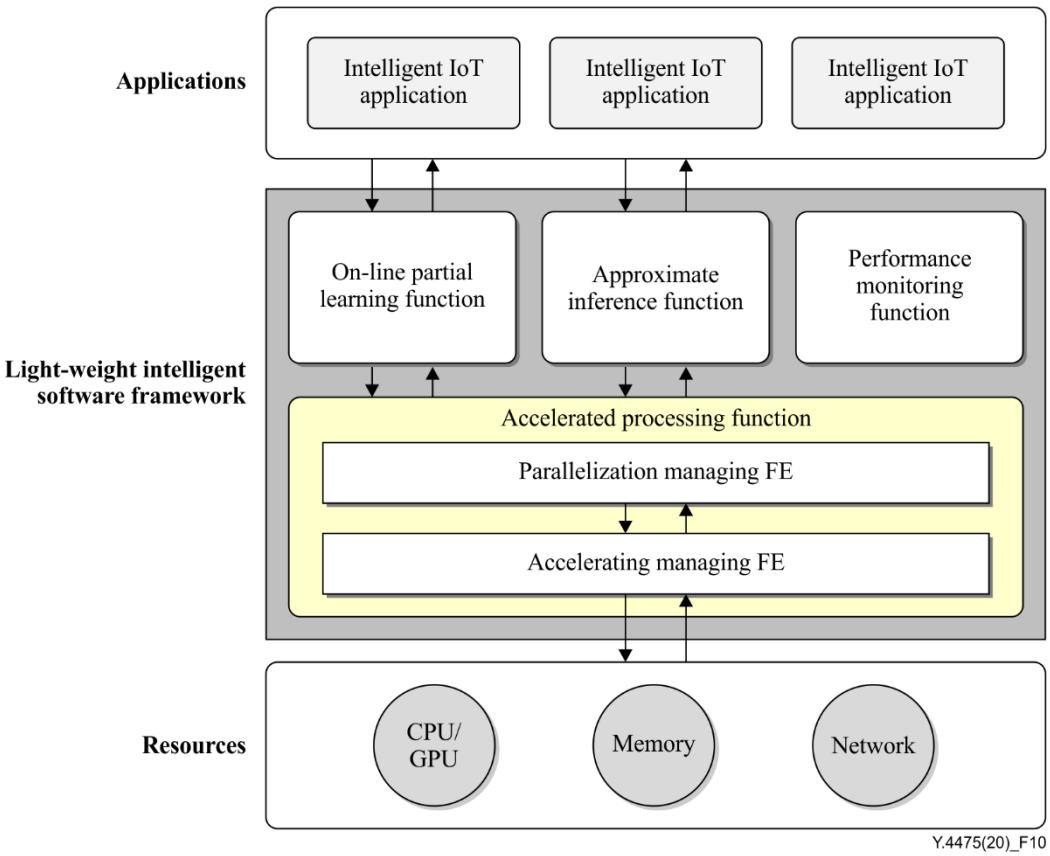

Figure 10 – Accelerating processing function in the LISF

8.4.1 Parallelization-managing FE

The parallelization-managing FE initializes and configures the following entities within the resource to run mathematical operations in parallel to speed up layer execution. Moreover, the parallelization-managing FE manages a parallel-processing queue for performing parallel processing depending on a number of devices in the embedded system, such as those with OpenCL. Furthermore, the parallelization-managing FE divides a matrix with weights and bias values taking the parallel processing performance of the device into consideration to maximize parallelism in multiple device environments. The parallel processing capability of the device is determined by the number of kernel instances that are executed at a time, a maximum work group size of the device or a maximum work item size:

– platform: specific targeted heterogeneous platform consisting of CPUs, GPUs, and other processors or HW accelerators;

– device: processors performing the calculation;

– context: an entity that manages the resources on a device set;

– command queue: an entity that executes the kernel and performs memory mapping or unmapping and synchronization;

– kernel: codes running on a device.

The parallelization-managing FE allocates device memory, copies data from the host to the device, sets up the kernel and copies the result again. It is necessary to design for parallel applications. The basic assumption is that many instances of the kernel run in parallel, each processing a single work item. Multiple work items run together as part of a work group. An instance of each kernel in the work group communicates with an additional instance.

8.4.2 Acceleration-managing FE

The acceleration-managing FE manages the framework for the kernel, which supports the set of mathematical operations. The set of mathematical operations includes those in neural networks, such as a convolutional neural network (CNN). The acceleration-managing FE supports mathematical routines that provide standard building blocks for performing basic vector and matrix operations. The acceleration-managing FE controls resources so that a device, such as a OpenCL device, performs a general matrix multiply (GEMM) operation on the divided matrix and input data depending on the divided matrix. In addition, the acceleration-managing FE groups the matrix into vectors to maximize a workload for each kernel, and determines on a size of a work group to allow each device to perform parallel processing. Furthermore, the acceleration-managing FE shares a memory between a host and devices to minimize the cost of the GEMM operation, each device performs mathematical routines without copying data between the host and the device by accessing the host's vector and a matrix using a memory address for operations:

– with scalar and vector or vector and vector;

– with matrix and vector; and

– between matrices.

8.4.3 Operational procedure

The operation of the accelerating processing procedure is shown in Figure 11. The accelerated processing function consists of a parallelization-managing FE and an accelerating managing FE.

The parallelization-managing FE performs tasks such as memory allocation, kernel configuration, and memory copy from host to device for partial learning model initialization and inference initialization requests. In the case of initial inference and partial learning, performance data allocates default memory and the kernel configuration. If there is performance data from the learning-monitoring FE or inference-monitoring FE, memory is allocated and the kernel configured using this information.

The accelerating managing FE provides the mathematical operations necessary for deep learning. It provides acceleration functions for mathematical operations between scalar, vector, and matrix quantities required for forward or backward pass in the learning or inference models.

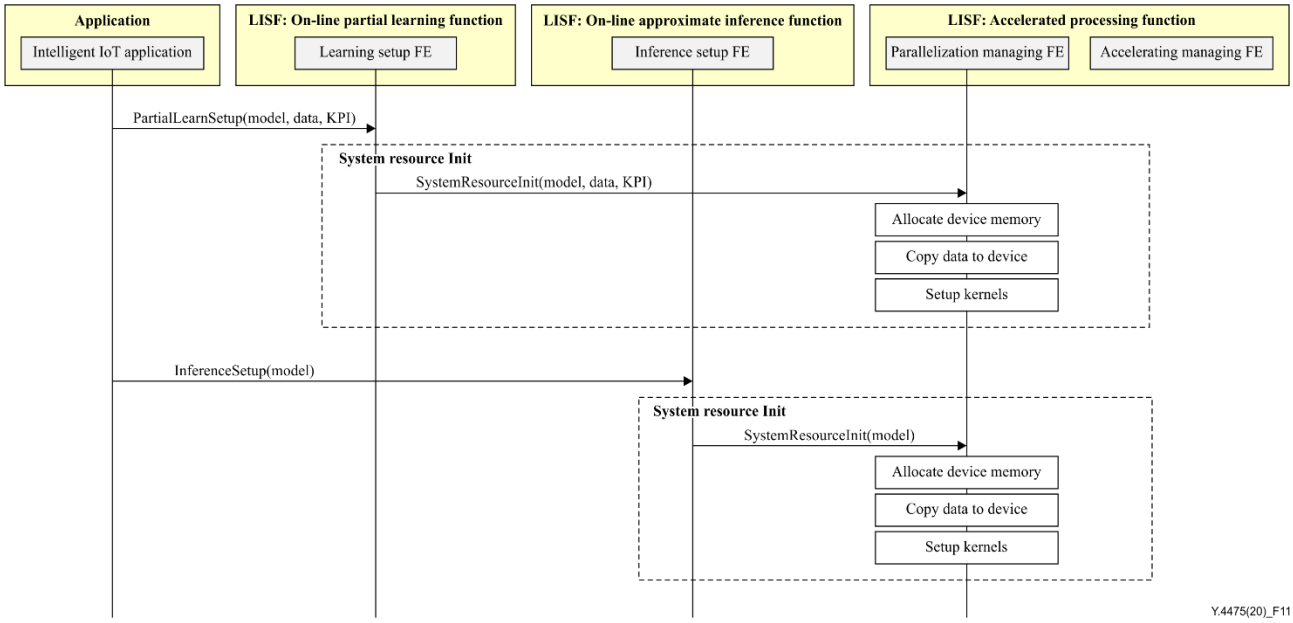

Figure 11 – Accelerating processing procedure in the LISF

Appendix I

Use case – Personal customization service with drone devices

(This appendix does not form an integral part of this Recommendation.)

This appendix gives an example of service for personal customization with drone devices for an LISF on embedded IoT devices. The LISF concept is gaining in importance in the current IoT service environment, because it enables resource-limited IoT devices to achieve intelligence over typical IoT platforms.

Figure I.1 shows an example of service for personal customization with drone devices. The service has four entities: control centre; drone stations; drones; and users.

– The control centre manages all drone stations with a data server and the collected data are used as training data to generate a new learning model.

– Drone stations have several drones waiting for user services from the control centre, and continuously communicate and forward collected data and a learning model from the drones to the control centre.

– Drones have embedded an initial inference engine and some functions: autonomous flight, real-time photographing, object (or face) detection, etc. A drone that receives user service from a drone station flies to the area where the user is located.

– Users request a service from the control centre using a smart phone and communicate with the drone.

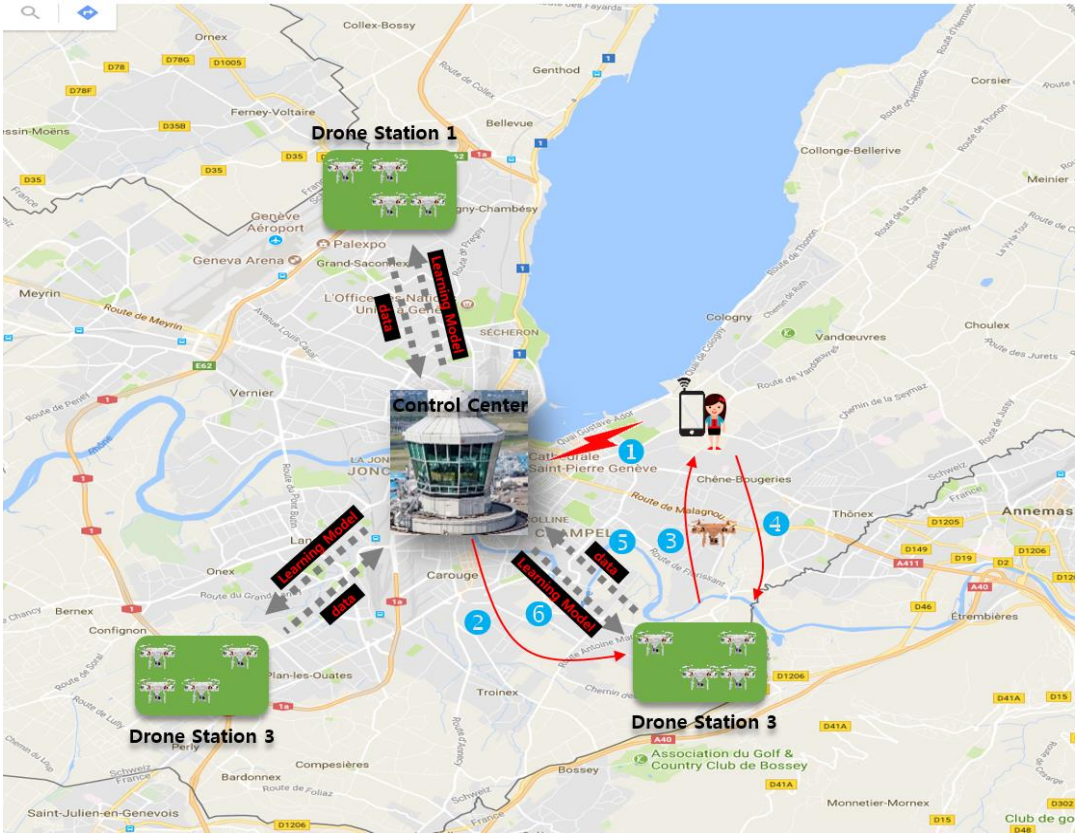

Figure I.1 – Service scenario for personal customization with drone devices

This procedure consists of three phases. The first is a configuration phase that transmits a user's request to an embedded IoT device, such as a drone. The second is an execution phase where an

embedded IoT device fulfils the user's request. The third is a learning phase that creates a new learning model.

The detailed procedure of the service is as follows.

1) Users request delivery service from a control centre with a batch learning server. When requesting the service, the following information is sent: user location; user identification; service type; etc. (sequence 1).
2) On receipt of the request, the control centre searches the nearest drone station to the user and sends delivery service to the one selected, e.g., drone station #3 (sequence 2).
3) The drone station, which manages embedded IoT devices, selects a drone that can be serviced and forwards information associated with delivery service to the selected drone (sequence 3).
4) The drone flies to the user to start the service. During the flight, the drone collects environmental image data using an embedded CAM device. When arriving at the destination from where the user has requested the service, the drone first identifies the user by using face detection and then provides the service corresponding to the user request. After finishing the service, the drone flies back to the drone station (sequence 4).
5) Drone stations send all data collected by drones to the control centre, which performs batch learning and generates a new inference engine for all drones (sequences 5 and 6).

The drone conducts intelligence processing, e.g., online partial learning, during the flight or at the drone station, so that the drone is updated with the new inference engine. Embedded IoT device such as drones can be applied to heterogeneous service domains or various data environments.

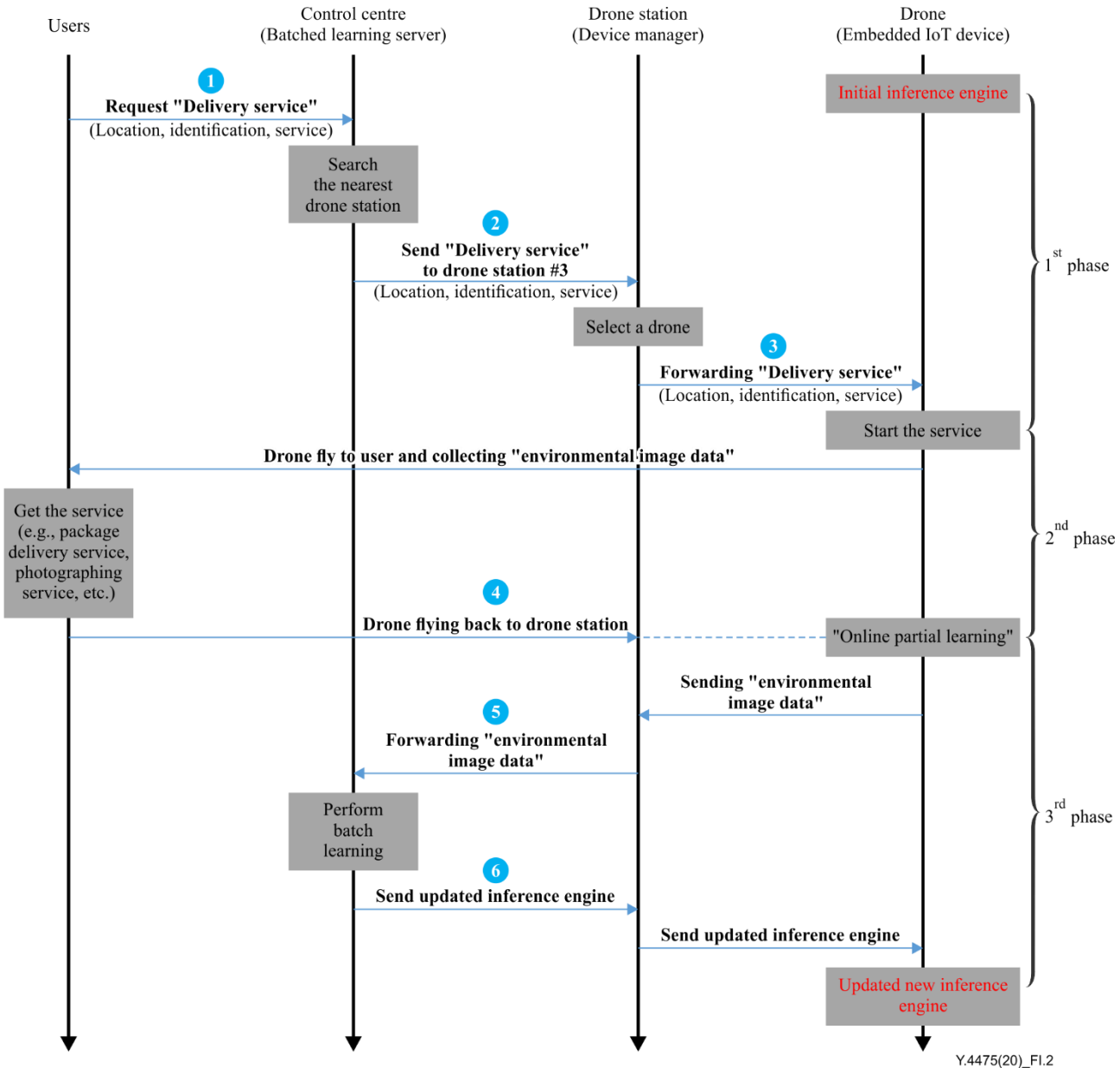

Figure I.2 – A procedural example of LISF service

Appendix II

Use case – Personal vision aids service

(This appendix does not form an integral part of this Recommendation)

This appendix gives an example of service for personal vision aids with an LISF on embedded IoT devices. The vision aids service is helpful for visually impaired people and can support personalized service in each individual's environment.

Figure II.1 shows an example of personal vision aids service with the LISF. The service has four entities: users; wearable device; LISF gateway; and service centre.

– Users use a personal and customized vision aids service.

– The wearable device detects the objects and transfers the image data to the LISF gateway. In addition, the wearable device can send object information to the user through sounds.

– The LISF gateway performs online partial learning on the IoT device with newly acquired data from the individual's environment. In addition, it transmits the acquired data to a service centre to perform batch learning. The data transferred are used as training data for regenerating the learning model.

  • The LISF gateway infers objects using pre-trained learning models and image data from wearable device (e.g., smart glasses or phone).

  • The LISF gateway transfers inference results to the wearable device.

– The service centre initially performs full (batch) learning with rich system resources and transfers a pre-trained learning model to the LISF gateway.

Figure II.1 – Service scenario for personal vision aids

This procedure consists of three phases. The first is a configuration phase in which users request the service from the service centre. The second is an execution phase in which embedded IoT devices detect and infer the objects when users move. The third is a learning phase in which an embedded IoT device performs online partial learning and sets up a personalized learning model.

The detailed procedure of the service is as follows.

1: Configuration phase (Sequence 1-4)

1) Users request vision aids service from a service centre via embedded IoT devices such as a wearable device and LISF gateway. When the service centre receives the service request, it transfers the pre-trained learning model (sequences 1, 2, and 3).

2) On receipt of the learning model, the LISF gateway sets up an initial inference and learning engine and reports a status service set-up to users via the wearable device using sound (sequence 4).

2: Inferencing phase (Sequence 5-8)

3) When users move, the wearable device detects the object and generates the image data for it. When data generation is completed, the wearable device transfers the generated data to the LISF gateway (sequences 5 and 6).

4) The LISF gateway infers the detected object using an inference engine. When finished, the LISF gateway provides the user with an inference result based on the pre-trained model (sequences 7 and 8).

3: Online partial and re-learning phase (Sequence 9)

5) Based on the data collected up to this point, the LISF gateway performs online partial learning and sets up a personalized learning model. The LISF gateway transfers the collected data to the service centre. The service centre performs batch learning and generates an updated learning model (sequence 9).

The LISF provides a lightweight learner, e.g., for online partial learning and predictor, optimized in embedded HW in a resource-limited environment.

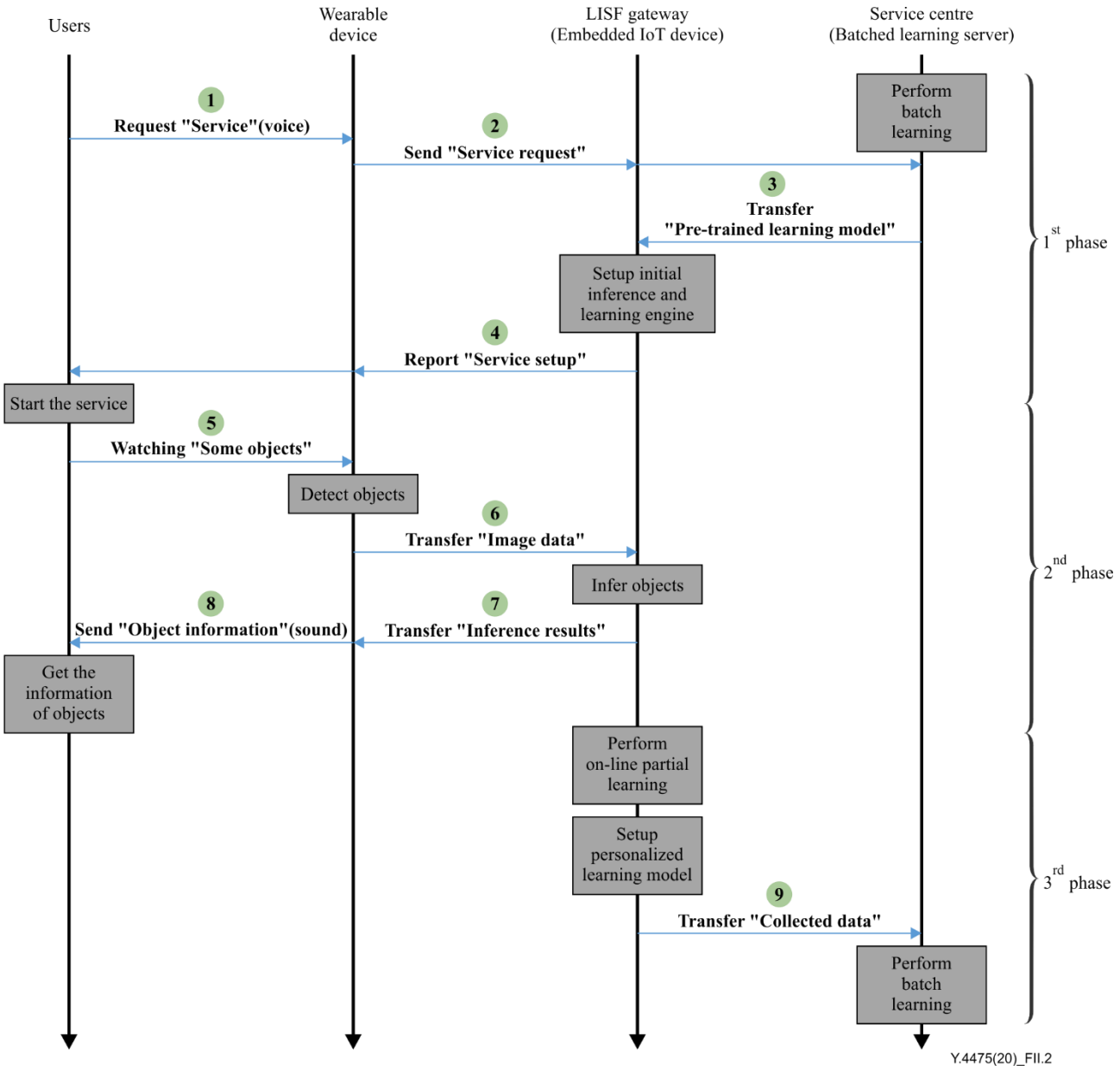

Figure II.2 – A LISF procedure example of a personal vision aids service

Bibliography

[b-ITU-T Y.2012] Recommendation ITU-T Y.2012 (2010), *Functional requirements and architecture of next generation networks*.

[b-ITU-T Y.2091] Recommendation ITU-T Y.2091 (2011), *Terms and definitions for next generation networks.*

[b-ITU-T Y.4000] Recommendation ITU-T Y.4000/Y.2060 (2012), *Overview of the Internet of things*.

[b-ITU-R M.1224-1] Recommendation ITU-R M.1224-1 (2012), *Vocabulary of terms for international mobile telecommunications (IMT).*

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	**Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities**
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2020