Technical Report

ISO/IEC TR 17903

Information technology — Artificial intelligence — Overview of machine learning computing devices

Technologies de l'information — Intelligence artificielle — Aperçu des dispositifs informatiques pour l'apprentissage automatique

First edition
2024-05

Reference number
ISO/IEC TR 17903:2024(en)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 42, *Artificial intelligence*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

As an important approach for realizing artificial intelligence (AI), machine learning (ML) has been applied to and has improved productivity in multiple domains (e.g. education, finance, environment protection). ML computing devices can be essential to the development and deployment of many types of AI systems.

An ML computing device can have a set of characteristics, including supported datatypes, ML operators, buffer settings, access and share mechanisms, memory access and addressing mechanisms, virtualization and sharing mechanisms, job scheduling mechanisms, topologies, data exchange mechanisms and memory interoperability mechanisms. Use and setting of these characteristics can affect the overall performance of an ML computing device. The performance of an ML computing device used to develop and deploy an AI system can be crucial to business effectiveness and efficiency.

This document surveys and provides information to AI stakeholders to assist them in understanding the representative characteristics of ML computing devices. In Clause 5, ML computing device-related concepts are discussed. Clause 6 summarizes ML computing device characteristics. In Clause 7, existing approaches for optimizing ML computing devices' performance are discussed.

Information technology — Artificial intelligence — Overview of machine learning computing devices

1 Scope

This document surveys machine learning (ML) computing devices, including the following:

- ML computing device terminology and characteristics;
- existing approaches to the setting and use of characteristics for optimizing ML computing device performance.

The information provided in this document is relevant for organizations of all types and sizes.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 22989, *Information technology — Artificial intelligence — Artificial intelligence concepts and terminology*

ISO/IEC 23053, *Framework for Artificial Intelligence (AI) Systems Using Machine Learning (ML)*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 22989 and ISO/IEC 23053 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at https://www.iso.org/obp
- IEC Electropedia: available at https://www.electropedia.org/

3.1
artificial intelligence computing
AI computing
processing that leverages infrastructures to enable the set of methods or automated entities that together build, optimize and apply an AI model

3.2
machine learning computing
ML computing
processing that leverages infrastructures to train and execute ML models

3.3
infrastructure
hardware and software environment to support computer system and software design, development, and modification

Note 1 to entry: Network facilities can be also infrastructure working with hardware and software.

[SOURCE: ISO/IEC/IEEE 12207:2017,[1] 3.1.25, modified: note 1 to entry is added.]

3.4
computing device
functional unit (3.6) that can perform substantial computations, including numerous arithmetic operations and logic operations with or without human intervention

Note 1 to entry: A computing device can consist of a stand-alone *unit* (3.7), or several interconnected units. It can also be a device that provides a specific set of functions, such as a phone or a personal organizer, or more general functions such as a laptop or desktop computer.

Note 2 to entry: A computing device contains at least one *unit* (3.7).

[SOURCE: ISO/IEC 19770-5:2015,[2] 3.6, modified: note 2 to entry is replaced.]

3.5
hardware
all or part of the physical components of an information processing system

EXAMPLE Computers, peripheral devices.

[SOURCE: ISO/IEC 2382:2015,[3] 2121277, modified: removed note to entry 2 and 3]

3.6
functional unit
entity of hardware or software, or both, capable of accomplishing a specified purpose

Note 1 to entry: In IEC 50 (191), the more general term item is used in place of functional unit. An item may sometimes include people.

[SOURCE: ISO/IEC 2382:2015, 2123022, modified: removed notes 2 and 3 to entry.]

3.7
unit
lowest level of hardware assembly for which acceptance and qualification tests are required

[SOURCE: ISO 24917:2020,[4] 3.4]

3.8
service
kind of application which encapsulates one or more computing modules and can be accessed through a specified interface

[SOURCE: ISO/IEC 19763-5:2015,[5] 3.1.18]

3.9
artificial intelligence computing device
AI computing device
computing device (3.4) that can be specifically used for accelerating some or all of artificial intelligence computing

Note 1 to entry: An artificial intelligence computing device often contains and works with specific enabling software.

3.10
machine learning computing device
ML computing device
computing device (3.4) that can be specifically used for accelerating machine learning computing

Note 1 to entry: A machine learning computing device often contains and works with specific enabling software.

Note 2 to entry: Machine learning computing device is a subset of *AI computing device* (3.9).

EXAMPLE A neural network process unit and its associated enabling software.

3.11
performance
measurable result

Note 1 to entry: Performance can relate either to quantitative or qualitative findings.

Note 2 to entry: Performance can relate to the management of activities, processes, products (including services), systems or organizations.

[SOURCE: ISO/IEC 27000:2018,[6] 3.52]

3.12
effectiveness
extent to which planned activities are realized and planned results are achieved

[SOURCE: ISO/IEC 33001:2015,[7] 3.1.3]

3.13
efficiency
resources expended in relation to the accuracy and completeness with which users achieve goals

[SOURCE: ISO/IEC 25063:2014,[8] 3.4]

3.14
datatype
set of distinct values, characterized by properties of those values and by operations on those values

[SOURCE: ISO/IEC 11404:2007,[9] 3.12]

3.15
operator
mathematical or logical symbol that represents an action to be performed in an operation, or a symbol representing the name of a function

[SOURCE: ISO/IEC/IEEE 24765:2017,[12] 3.2716, modified — 3.2716.3 and 3.2716.4 are combined.]

3.16
schedule
methods for controlling the timing of the execution of an activity within or represented by a managed object

[SOURCE: ISO/IEC 10164-15:2002,[33] 3.1.27]

3.17
topology
structure of the communication paths between the medium attachment points

[SOURCE: ISO/IEC 14543-2-1:2006,[48] 3.2.30]

3.18
stream
sequence of representations of jobs or parts of jobs to be performed, as submitted to an operating system

[SOURCE: ISO/IEC 2382:2015,[3] 2122886]

3.19
computing
actions performed or approaches implemented by a system, aiming at processing specific kinds of data or providing solutions to certain tasks

Note 1 to entry: Computing can also be a system providing functionalities or characteristics.

3.20
heterogeneous computing
system that uses more than one kind of processor or cores

[SOURCE: ISO/IEC 30145-3:2020, 3.1.5]

3.21
distributed computing
spreading of computation and data across a number of computers connected by a network

[SOURCE: ISO/IEC/IEEE 24765:2017,[12] 3.1240]

3.22
quantum computing
use of quantum phenomena for computational purposes

[SOURCE: ISO/TS 80004-12:2016, 6.5]

3.23
cloud computing
paradigm for enabling network access to a scalable and elastic pool of shareable physical or virtual resources with self-service provisioning and administration on-demand

Note 1 to entry: Examples of resources include servers, operating systems, networks, software, applications and storage equipment.

[SOURCE: ISO/IEC 22123-1:2021, 3.2.1]

3.24
inter-cloud computing
paradigm for enabling the interworking between two or more cloud service providers

[SOURCE: ISO/IEC 22123-1:2021, 3.12.1]

3.25
edge computing
distributed computing in which processing and storage takes place at or near the edge, where the nearness is defined by the system's requirements

[SOURCE: ISO/IEC TR 23188:2020, 3.1.3]

3.26
processing
run of an algorithm, treatment on data or a sequence of them performed by AI systems

3.27
buffer
device or storage area used to store data temporarily to compensate for differences in rates of data flow, time of occurrence of events, or amounts of data that can be handled by the devices or processes involved in the transfer or use of the data

[SOURCE: ISO/IEC/IEEE 24765:2017,[12] 3.430.1]

3.28
cache
temporary storage in computer memory, to improve operations by having frequently used data readily available for retrieval

[SOURCE: ISO/IEC/IEEE 24765:2017,[12] 3.452.1]

4 Abbreviated terms

AI	artificial intelligence
FIFO	first-in-first-out
GPU	graphics processing unit
IT	information technology
ML	machine learning
NPU	neural-network processing unit
PCIe	peripheral component interconnect express
SoC	system on chip
UML	unified modelling language

5 ML computing device concepts

5.1 Processing

Processing can refer to the following:

- an activity, or treatment, or a sequence of them to be conducted for a specific purpose;
- a run of an algorithm, which can provide certain treatment or trigger some activity for a specific purpose.

In comparison with computing, processing emphasizes specific actions, while computing can be a system that contains actions as well as approaches to be realized via actions. In the context of an ML computing device, process can refer to the action that stakeholders take for an ML task by using ML computing devices.

5.2 Computing

A prerequisite for deriving the meaning of AI computing device and ML computing device is to define AI computing and ML computing. In International Standards, a definition of computing does not exist. Based on the definitions in sub-areas of IT, the meaning of computing can be at minimum the following:

- a system having specific functionalities or characteristics aiming at processing a specific kind of data or providing solutions to certain tasks. This meaning of computing is used in definitions of heterogeneous computing and cognitive computing, where the term system explicitly appears;
- a set of approaches or actions that can be performed by a system aiming at processing a specific kind of data or providing solutions to certain tasks. This meaning of computing is used in definitions of semantic computing, quantum computing, cloud computing, inter-cloud computing, soft computing, edge computing and distributed computing.

AI contains a wide range of branches (e.g. ML, semantic computing and planning). Computing approaches and solutions are driven by the characteristics of data (e.g. tensor, vector and scalar). For ML tasks, when large number of vectors or tensors are involved, hardware modules and software components can be applied to compute elements in a vector or a tensor in parallel rather than sequentially.

5.3 Device

An ML computing device is a kind of device. Analysis of the meaning of device helps to define and understand the concept of ML computing device. Many definitions of device occur in ISO/IEC and IEEE standards. The meanings of a device can be the following:

- a physical appliance or apparatus or a combination of multiple of them that can provide functionalities for a specific purpose. This meaning of device is used in multiple areas (e.g. home electronic system,[64]-[65] health informatics,[68] nanotechnologies,[69] cloud computing[70] and industrial automation systems[72]);
- a combination of hardware and software components or an instance of software solely that can provide functionalities for a specific purpose. This meaning of device is used in multiple areas (e.g. operating system,[66] rich media user interface[67] and software engineering[12]).

A device provides functionalities for a specific purpose. Not all devices are for the purpose of computing. In the context of this document, a device can be a composition of collaborative devices that perform different computing tasks or different parts of one computing task. Each of these devices can contain hardware and associated enabling software.

EXAMPLE A server is a device that contains collaborative devices such as memory, disk and bus for generic computing, as well as those specifically designed for tensor multiplication for neural network processing. Memory, disk, bus and tensor computation components can work with associated enabling software. Some software is provided as drivers within an operating system, while other software (e.g. the enabling software library for tensor multiplication) needs to be additionally provided by a specific vendor.

From the perspective of an application, a device can be logical, which is a form of representation mapping to one, a part of one device, or a set of devices. The use of such a logical device triggers the functioning of mapped devices.

5.4 Infrastructure

In the information technology domain, an infrastructure can be software and hardware (including network facilities), even data and files, regardless of scale (see ISO/IEC 16350). In non-IT domain, the meaning of infrastructure can be wider. Infrastructures can be facilities, services, or supporting resources for the operation of an organization or a system (e.g. robotics domain,[75] space system domain[74] and management domain[76]).

Based on the meaning of infrastructure in IT domain, an infrastructure in ML domain can be anything that can be used by an ML application or its formation, including ML computing devices, datasets, ML software toolkits, or even ML models.

The main difference between device and infrastructure is that the latter can contain data, file, even services provided by a human or a group of humans. In the context of this document, a device can be a specific kind of infrastructure whose main purpose is not to provide data, files, or services and whose purpose does not include management or operation of an organization.

5.5 Service

By investigating the definitions of service specified in ISO/IEC standards, a service can be:

- a set of actions or values delivered from a provider (e.g. an organization) to a user or a customer. This meaning is used in multiple areas, including health informatics,[78] social management,[80] etc.;
- an application or software that encapsulates one or more computing modules and provides an interface to apply its functions to the input from users. This meaning is used in multiple areas, including information technology,[77],[84],[85] health informatics[79] and geomatics[81];
- a set of operations provided by an entity (e.g. web application server). This meaning is used in multiple areas, including software engineering,[82] web service[83] and ergonomics[86].

A service can be provided by utilizing IT or non-IT infrastructures, depending on the content of the service:

- When it takes use of certain IT infrastructures, it does not necessarily leverage ML techniques. When a service is an application that provides some function via an interface, its scope is larger than a computing device bound to specific business requirements. A service can also be a software component or an assembly of components at higher layer (e.g. operating system layer or application layer). When a service is provided that depends on an ML process, the service can take advantage of ML computing devices.
- A service can also takes use of non-IT infrastructures only, meaning some actions (e.g. transportation of goods) provided by a vendor using certain machines (e.g. a truck) to users or customers.

5.6 Performance

Characteristics of ML computing devices can affect the efficiency and use of ML. From this perspective, performance is an overall term that includes efficiency and effectiveness metrics. In some ISO/IEC standards (e.g. [6] and [87]), the term performance is defined as measurable results.

The meaning of ML computing devices' performance is different from the meaning of the performance of classification systems. Specified in ISO/IEC/TS 4213[100] the term performance emphasizes classification accuracy-related measurable results. The former puts more attention on the efficiency (e.g. running speed, throughput, power consumption and reliability) and effectiveness (e.g. compatibility to datatypes or protocols). An algorithm's performance in terms of running speed is largely determined by its execution environment. Speed of execution is not a characteristic of an algorithm, but subject to complexity (e.g. time complexity and space complexity) and execution environment.

An important target of study and manufacturing is to improve the efficiency of user applications. The performance of ML computing devices can be represented and measured by multiple metrics, defined in IEEE 2937[101] which covers training as well as inference processes. They provide options for users to choose in accordance with specific business requirements and restrictions. The absolute value of running speed (e.g. number of floating point operations per second) of an ML computing device does not systematically represent the actual outcome of computing on a specific task (e.g. natural language processing with an ML model BERT[11] or image classification with an ML model RESNET[41]). The actual throughput (e.g. number of images or sentences processed per second) can provide an overall evidence of the performance of ML computing device, meaning the magnitude of outcomes a user can obtain per a time unit.

5.7 Computing device

A computing device is a functional unit that can perform substantial computations. A functional unit can be hardware-only or a combination of hardware and software. A functional unit is a unit which is the lowest level of hardware assembly for which acceptance and qualification tests are required. Both hardware and software are derived from entity, as shown in Annex A.

An ML computing device is a kind of AI computing device since ML is a sub-area of AI and ML computing devices specifically accelerate ML computing. The concept AI computing device is a descendant of the concept computing device. This also holds for the concept ML computing device.

6 ML computing device characteristics

6.1 Datatypes

6.1.1 General

Datatype is an attribute of a piece of data that tells a computing system how to interpret its value. A computing device, based on the widths or the structures of its buffers, registers or calculators, consumes a piece of data in the form of sequences of bits.

The question of whether a datatype is acceptable or process-able by an ML system is focused on application and program aspects. An ML computing device only provides instructions (e.g. vector addition, matrix multiplication) for calculation over bit sequences. Advanced encoding or decoding and datatypes (e.g. string, integer, list, set) can be implemented by associated software libraries. These are useable by upper layer applications and programs but not directly applicable to devices.

In contrast, on the device side, it is important that an input bit sequence with certain width can be calculated in fewest clock signals. Specific hardware design for acceleration can be performed, driven by requirements. From this perspective, different parts of a computing device can have different designs to be competent to accelerate computations with intended widths of bit sequences.

Representative and frequently used widths of bit sequences are standardized by IEEE 754,[10] which specifies interchange and arithmetic formats and methods for binary and decimal floating-point arithmetic in computer programming environments. Depending on the requirements of the scenarios where ML computing devices are used, datatypes can be designed, implemented and applied for specific purposes. Typical precisions of data include 8-bit signed integer, 16-bit half-precision signed float point and 32-bit signed floating point.

In some cases, a specific ML computing device cannot directly process a piece of data in certain datatype defined in upper layer. For example, some ML computing devices whose maximum width of register is 16-bit cannot directly process data in 64-bit floating point. In such a case, if supported by the device's associated software libraries, additional operations are performed including but not limited to:

- casting data from higher precision to lower precision;
- filling unused bits with zeros;
- splitting data in higher precision into parts, processing one part at a time and composing the stepwise outcomes into the final result.

These operations can lead to performance loss, in terms of accuracy or overall time latency.

6.1.2 Effectiveness and efficiency

From the perspective of definitions, there are no advantages and disadvantages across datatypes, since the definitions of datatypes are driven by business requirements. However, from the perspective of ML computing device, micro architecture design, instruction sets and APIs can be prepared to support a certain datatype. If a type of ML computing device supports a relatively wide range of datatypes, the design of its registers, calculators, buffers and other components can be more flexible but can also be harder and less efficient. The risk of insufficient use of computing resources (e.g. unused bits in a 32-bit register when dealing with an 8-bit number) can also exist. In contrast, if a kind of ML computing device processes just one kind of datatype, the utilization can be higher and its design from this perspective can be simpler.

When datatypes are selected and applied for a specific ML task. Stakeholders can consider the trade-offs between efficiency and effectiveness in terms of the selection of datatypes. The target is to apply datatypes which allows to complete the task with as minimum resources as possible. For training, mixed-precision approaches (e.g. [89] and [88]) can be applied to decrease the resources (e.g. time) consumption and in the meanwhile guarantee the accuracy of the trained model with qualified precision. For inference, quantification techniques (e.g. [73]) can be conducted to achieve efficiency if the quantified model can still provide adequate accuracy that makes the AI system effective.

6.2 ML operators

6.2.1 General

An operator is a symbol representing a certain functionality. Operators can be implemented by software and wrapped as static and dynamic libraries. The software implementation can be regarded as a kind of abstraction that wraps those specific details about interactions with a certain hardware. In this way, manipulations on hardware (e.g. physically transmitting a signal to a port of hardware) can be done by invoking software functions.

For ML computing, specific operators[13] are designed to implement mathematical or logical calculations used in ML training (during forward propagation and backward propagation sub-processes) and inference. This kind of operator for ML computing is referred to as an ML operator. ML operators can be implemented by software and wrapped as libraries for programmable invocations for training and inference. ML operators can use fundamental mathematical operators,[14] which are not specifically for ML processes and can be used for another technical domain. ML operators include the ones for traditional ML process (e.g. regression and clustering) as well as deep learning[15]-[16].

In an ML system, ML operators are defined by AI computing devices and implemented by the associated enabling software libraries. Such implementation of operators exposes a set of supported functions that the ML computing device can execute.[17] A normative International Standard does not exist specifying a common set of operators, except the ones provided by ML computing device providers, such as the sets of operators defined in compute unified device architecture,[18] compute architecture for neural networks[19] and other libraries with similar functionalities. In Reference [20] an open, cross-industry standards-based initiative unified programming model is specified. It provides specifications and open-source projects. The sets of operators from ML computing device manufacturers can differ in terms of contents, semantics of similar functions, function signatures and approaches of use.

In an ML system, ML operators can be further wrapped and extended by the following upper layers:

- ML software toolkits can provide encapsulations over ML operators that are used by software internal computation and user procedures. ML software toolkits vendors also implement their own extensions on ML operators, which are the sets of functions executable by a specific toolkit, using the operators supported by the underlying hardware. In this layer, there is no formal standard specifying a complete and common set of operators that a considerable number of ML software toolkits (e.g. [21]-[23]) follow. Standards for operators in this layer are emerging (e.g. [24]), followed by a group of collaborative enterprises. Such standards can be associated with specific model compilers that can compile sequences of computation onto specific backend hardware.

- ML model can be in a specific representation formed by a set of specified ML operators that work as an intermediate form for ML model format exchange and transition to assist the portability of models. In this layer, a representative standard for model exchange is Reference [25]. It works as a bridge between models in different formats generated by different ML software toolkits. There does not exist a full consistency between the operators in an ML model intermediate representation with the encapsulated ones used by an ML software toolkit.

In addition, open standards (e.g. [26]) exist for general purpose parallel programming agnostic to the type of processors, giving software developers portable and efficient access to the power of these heterogeneous processing platforms. Reference [26] is aimed at supporting parallel computing, which works as a base for ML. Complex operators (e.g. K-mean and Conv), particularly in the scope of ML, have not been specified[26]. Reference [26] can benefit the following:

- To implement necessary sets of operators, such as linear algebra, to enable hardware-agnostic mathematic computations (e.g. MAGMA[27]).

- By ML computing device manufacturers and implemented with modifications in their accelerating libraries for enabling computation of mathematics via specific computing device.

A set of functionalities (e.g. communication, command and data exchange) exist which benefit calculations but do not directly carryout the calculation. They are the base of computing and can be used for multiple technical domains, not only for ML. In this area, operators can refer to the abstractions supported by static or dynamic libraries that execute on specific hardware.

6.2.2 Effectiveness and efficiency

The definitions of the operators of a specific platform (e.g. the ML library of an ML computing device, ML software framework and intermediate representation) provides advantages for improving applications' compliance to that platform. However, AI computing device and ML software toolkit manufacturers have already designed, implemented or extended their own operators. The following difficulties exist in establishing a unified and public set of ML operators for all kinds of AI computing devices and software:

- The soundness of the operator set is hard to guarantee if these operators come out of a merge of those existing operator sets. It is highly probable that a subset of operators exist that cannot be successfully executed by a subset of hardware and software. Conflicts and inconsistencies can exist among the operators, since the settings, requirements and use of parameters and contexts of similar operators across manufacturers can be different.
- The execution efficiency of a unified and public operator set cannot be guaranteed. The more abstract or general, the more adaptation or specification can be needed. This principle is true in ML computing devices, where multiple kinds of hardware and software can be involved by various users and application systems. To achieve compliance between operator sets to be merged, a loss of efficiency can happen in cases such as unused and redundant parameters as well as cascaded data.
- The workload for maintaining a set of unified and public ML operators can be massive, given that the underlying ML operator sets by different manufacturers can continuously and independently evolve.

6.3 Memory access and addressing mechanisms

6.3.1 General

ML computing needs memory for temporarily storing intermediate results of computations for further use. In modern settings, there exist at least two representative kinds of memory settings:

- ML computing devices can be equipped with on-chip memory[28]-[31] encapsulated with processor together.
- ML computing devices can be set on accelerating cards that work via PCIe or similar protocols. In this case, on-card memory or main memory can be used.

6.3.2 Effectiveness and efficiency

On-chip memory can be connected with a processor via channels where high-speed data transmission[31] is applied. In this case, memory addresses inside chips can vary. This indicates the difficulty when operations are requested across memories by different processors unless a specific mechanism is implemented for this purpose, such as the unified GPU memories of Reference [32].

Accelerating cards provide facilities for on-chip memory unified addressing mechanisms, which to some extent protect the content among memories from being inconsistent. However, this approach can limit the extension of the number of processors with on-chip memories, because of its connections between any pair of processors. In addition, on-chip memory can improve the efficiency of memory access, in comparison with the settings that use main memory via PCIe. PCIe's bandwidth is subject to not only the throughput of memory and processor, but also PCIe protocol versions. Different versions of PCIe protocols provide different theoretical bandwidths.

6.4 Scheduling

6.4.1 General

An ML task can be completed by an ML computing device in multiple steps by its different parts. These parts collaboratively work for that job, through a sequence of atomic computations, buffer access and other operations.[34]

EXAMPLE An execution of a convolution computation can contain five steps that form a sequence (e.g. fetching data from registers or buffer, fetching the instruction from instruction register, executing matrix multiplication and addition operations, accessing the output buffer and finally writing the result into an output buffer or register).

6.4.2 Effectiveness and efficiency

The scheduling logic for an ML job can be pre-set and executed by a processor or co-processor (e.g. [35]-[38]) within an ML computing device. Job scheduling can have different scales:

- Pipelines inside an ML computing device (e.g. pipeline for instructions' execution within a neural network processing unit system on chip).[39] At this scale, different parts of a SoC can execute different atomic computing in a sequence. Once a part finishes its work, it can advance to the same part of the next sequence, even though the current sequence has not been completely finished. Such a pipeline is helpful for improving performance. This improvement is achieved by proper physical design and arrangement of components inside an ML computing device (e.g. SoC), which design can require more registers or buffers and space. From this perspective, pipeline depth is important, as it determines the concurrency. However, the deeper a pipeline is, the more serious the pipeline flush problem can become.

- Pipelines between different ML computing devices[40] (e.g. the inference of a big deep learning model) in which one ML computing device does not have adequate memory to completely accommodate the data being processed. Such a model can be separated into several pieces and each piece can be set in an ML computing device (e.g. [41]-[45]). The outcome of the predecessor ML computing device can be consumed by the successor, in order to complete the whole inference. Similar to the pipeline inside an ML computing device, the condition control structure of a model can affect the effectiveness and efficiency of the entire process.

- Assignment of sub-jobs to intended ML computing devices and merging sub-results (e.g. data-parallelism training process of a deep learning model[46]). In such a process, parts of a training job are set onto different ML computing devices. Each responds to the training of a model based on local data subset. Via parameter synchronization and iterations, a final model can be achieved.

ML computing device-related job scheduling also includes ML job distribution and collaboration between computing centres. Due to the cost (e.g. energy) difference across regions, the flow of jobs and data are needed when large-scale datasets are used or large-scale models are trained. For this purpose, the computing ability of a certain computing centre can be described, measured, priced and exchanged. This holds for data generated everywhere, collected within and across regions and used with authorities. For a large-scale ML job, the collaboration between computing centres can be scheduled by an intermediate stand-alone scheduler, which, with adequate authority, collects the needs of computing ability quotas, requests and also related datasets. It can schedule the computing location for a specific part of a large-scale ML model. The advantages of this approach includes:

- It can observe, schedule and leverage less expensive computing resources for a proper job in terms of necessary computing volume and price.

- It can balance workloads across large-scale computing centres, such that the average latency of jobs can be decreased overall.

- It can re-use and unite relatively old computing centres by scheduling proper jobs and creating collaboration as a stronger computing centre. In this case, users do not need to invest and put resources to build a new one that fits the new requirement at hand.

Job distribution and data flow are related since the use or the processing of data (e.g. a training task) can happen either where the data exists or can be distributed. The aim of job distribution is to promote the degree of parallelism of the entire system, such that the overall utilization of computing devices can be promoted. The following representative parallelism mechanisms for training can be applied:

- Model parallelism: Model parallelism allows for splitting a model into parts and distributing them onto different ML computing devices, such that different parts of a model can be trained in parallel. Advantages of this mechanism include the competence of training large-scale models and reduction of communication overheads. However, when a model is trained in a pipelined model parallelism mechanism, staleness of weights can happen[47] in which old weights are used to compute gradient and lead to accuracy loss.

- Data parallelism: Data parallelism allows for splitting a dataset into parts and distributing each part on to a device where an entire model is trained based on the local piece of dataset. This mechanism enables the parallel training with a large dataset that cannot be settled in the memory of one device. Multiple trainer devices can communicate with a coordinator device that synchronizes parameters from different trainer devices. A disadvantage of this mechanism is the possible heavy communication overhead.

- Mixed parallelism: Mixed parallelism combines the principles of model parallelism and data parallelism. It allows for splitting both the dataset and model, distributing them onto different devices. It inherits advantages such as the ability to apply suitable parallelism mechanism to a specific part of a model (e.g. a part with sparse connections, a part with dense or full connections). However, this mechanism indicates a need for more complex scheduling. Furthermore, the scheduling can be influenced by model structure.

6.5 Topologies

6.5.1 General

In an ML system, topology is about the arrangement and communication relationships between useful components, such as accelerating processors, ML computing devices, memory and storage (e.g. [49] and [50]).

ML computing device-related topology can be in different scales:

- Inside an ML computing device, the arrangement and communication relationships between accelerating processors, memories, storage and other co-operated modules (e.g. the contributions in References [51]-[54]).
- In a system containing multiple ML computing devices, the arrangement and communication relationships between ML computing devices, memories, storage and other co-operated devices (e.g. [55]-[57]).

For both scales, multiple kinds of topologies can be selected and applied according to the actual behaviour of applications. Two types of topologies exist – regular topology and customized topology:

- Regular topology: Topologies of this kind include ring, star, N×N grid, N×N torus, N×N folded torus and tree topologies, where N stands for the number of nodes and each node can be an ML computing device, an accelerating processor, memory, storage module, etc.
- Customized topology: Topologies of this kind include those designed and implemented by an ML system provider. Such a topology can be an updated or modified version of any regular topology.

6.5.2 Effectiveness and efficiency

Advantages and disadvantages in terms of efficiency of regular topologies include:

- Ring: In this topology, a single wire connects each node. The degree of each node is 2. The main disadvantage of ring topology is that its diameter increases with each increase in the number of nodes. This can cause a scalability problem that the performance degrades when network size expands. Ring topology is also prone to a single point of failure due to the lack of path diversity.
- Star: In this topology, a central node connects to each non-central node and a non-central node only connects to the central node. The main advantage of Star is that it has a diameter of 2, regardless of its size, and the whole system can be tolerant to the failure of a non-central node. The main disadvantage is that its central node can become a performance bottleneck and the whole system is sensitive to the failure of the central node.
- N×N grid: In this topology, besides the nodes at the edges, each router is connected to a number of neighbouring nodes through communication channels. This topology offers a solution for path diversity and scalability. In addition, this topology provides flexibilities against link or node failures by providing alternative paths. The main disadvantage of this topology is that its diameter can increase with the number of nodes in the system.
- N×N torus: In this topology, extra horizontal or vertical wrap-around connections are established between edge nodes. These connections, to some extent, resolve the problem of unbalanced communication hops, or distance between source and receiver nodes, in an N×N grid. However, the diameter between those edge nodes without wrap-around connections is still subject to the size of the system.

- N×N folded torus: In this topology, extra horizontal and vertical wrap-around connections are established between edge nodes. The disadvantage of N×N torus topology is thereby mitigated.
- Tree: In this topology, nodes are arranged by layers. Nodes in both root and leaf layer only have connections to intermediate nodes. When the connections in this topology are sparse (e.g. a binary tree topology), performance bottlenecks can happen in root and intermediate nodes.

6.6 Streams

6.6.1 General

In information technology, stream describes a kind of job arrangement mechanism. A stream can be a sequence or a queue of device work. A host places work in the queue and a device schedules work from streams when resources are free.[58]-[60] Within a stream, the operators are FIFO and cannot overlap. Operations in different streams can overlap and unordered.

6.6.2 Effectiveness and efficiency

With stream mechanisms, different synchronous approaches can be implemented[60].

- When an ML application uses a single thread associated with one stream, the work delivered to that stream can be executed in a synchronous manner. The host can continue its work only after the single stream is completed. The completion of the stream can trigger an event that invokes the control flow of application.
- If multiple threads and streams are employed in an ML application, work can be arranged among streams in a parallel manner. The host can split its work into parts and when each is finished, the corresponding device work can be delivered to a stream. Multiple parts can be delivered to different streams and the latter can work in parallel. After all the work is arranged in streams and before the streams are finished, the host waits for results. Once these streams are completed, results are delivered to the host for further processing. This can achieve more efficient processing, up to x times greater than the one stream case, where the x is the number of streams.

In some ML computing devices, stream settings can be controlled in a detailed fashion. Events can be set and work associatively with streams. An event can be associated with a specific stream or multiple streams, indicating whether device work is finished. A host can wait for the notification of such events for sophisticated job scheduling and higher efficiency.

The stream mechanism can be implemented via hardware, associated software or both.[61][62][63] ML device manufacturers design and implement stream mechanisms based on their device architectures. The effectiveness of a specific stream mechanism can also be subject to the kinds of jobs.

6.7 Buffering mechanisms

6.7.1 General

A buffer makes use of a memory to temporarily store a selected piece of data when the data is moved. The main purpose of buffering is to adjust timing between entities (e.g. calculation units in a computing device, certain data generating process and data transmission component) to prevent data from being lost or ignored for processing. A buffer can be set within one or multiple ML computing devices, working with input and output facilities. Buffering mechanisms can include but are not limited to the following:

- FIFO: A FIFO queue data structure can be used to implement a buffer. The data that arrives earlier is processed with higher priority than data that arrives later.
- Circular: A circular buffer is a logical view of a buffer, in which the next piece of data is stored into the first position when the last position is just used. A circular buffer makes use a fixed-size and limited piece of memory.

6.7.2 Effectiveness and efficiency

The effectiveness and efficiency of a buffering mechanism can be subject to at least the following factors:

- Volume: A buffer can make use of a piece of memory. Wherever a buffer is set, its volume is important. Trade-offs between space restrictions and eliminating data can be present. When the restriction on the volume of a buffer is tight, the buffer can use a limited amount of storage for adjusting the time difference between inputs and outputs. Data loss can happen due to buffer overflow. A looser restriction on buffer volume can relieve this problem, but it also consumes more storage. When the magnitude of the transmitted data in this buffer is small, storage utilization can be a problem.

- Shape: A buffer can be designed to store a fixed size of data if the velocities (e.g. time sequence) of the buffer's input and output are fixed. For instance, a frame buffer is designed for accommodating a complete video frame. In an ML computing device, the result of a multiplication calculation between two matrices (i.e. by a multiplication component) can be stored in a buffer with fixed volume, since the dimensions of the result matrix can be deterministic.

- Multiplicity: Multiple buffering applies more than one buffer to hold a piece of data, such that the reader to the buffer can see a complete version of data. Video display can apply a double buffering mechanism, which stores a frame in a piece of memory (back buffer) the content of which is copied to the video random access memory for display. Double buffer mechanisms can relieve stutter and tearing for graphics display. For higher performance, triple buffering mechanisms can be applied that use two back buffers. Once the image has been sent to a monitor, the front buffer can copy data from a back buffer holding the most recent complete image.

6.8 Cache mechanisms

6.8.1 General

A cache differs from a buffer in that cache aims at providing storage and access for frequently accessed data, rather than movement of data. Data in a cache can change over time, but a cache itself can be permanent. In an ML computing device, a cache can be set on chip, with associated instructions or enabling software interfaces for access and management. Cache mechanisms include but are not limited to the following:

- Cache-aside: An application first looks into a cache for desired data before requesting from a data source access of which can be slower. In this mechanism, the application interacts with both the cache and data source. This mechanism is frequently applied in IT infrastructures and solutions (e.g. operating system, web application server and database).

- Read-through: This mechanism is similar to cache-aside. The difference is that the application does not need to directly interact with data source, instead delegating data source-cache synchronization task to a cache provider. The cache provider establishes a view of the data in the cache used by the application and synchronizes the data to the data source.

- Write-through: In contrast to read-through, write-through mechanisms support a cache facility for writing. The application changes data in a cache, which then synchronizes the change to the data source.

- Write-behind: This mechanism is similar to write-through, but it performs data source-cache synchronization for multiple buffer writes.

6.8.2 Effectiveness and efficiency

Cache effectiveness describes to what extent a cache mechanism selects the right data to store and decides a reasonable length of duration to keep it, such that the data can be found in the cache rather than being retrieved from the data source. The effectiveness and efficiency of a cache mechanism can be influenced by at least the following factors:

- Volume: The volume of a cache affects the rate of hits and misses. The smaller the volume, the larger the chance that potential cached data changes. In an ML computing device, the volume of cache can be limited when the cache is a size-fixed storage area provided by hardware.

- Coherence: Coherence defines the behaviour of reads and writes to a single address location. In an AI system with multiple ML computing devices and a shared memory, many copies of shared data can exist in different storage positions (e.g. one copy in main memory and one copy in the cache of an ML computing device). When a copy of the data is changed, the other copies are supposed to be changed. Cache coherence provides discipline to ensure that any change of a copy can be propagated through the system in a timely fashion.

6.9 Data exchange mechanisms

6.9.1 General

In the area of ML computing devices, the aim of data exchange is to facilitate computing. It refers to the mechanism for data transmission and processing between multiple components of one ML computing device or between multiple ML computing devices. This concept of data exchange differs from data exchange for data use. The latter covers data management rules including data transfer between roles (e.g. organizations providing training data and the organizations using data to train ML models) during the life cycle of data. In the latter area, guidance on data usage (e.g. [90]), metadata (e.g. [91] and [92]) and interoperability (e.g. [93] and [94]) is published or in development. In the area of AI, standards providing frameworks and approaches for the use of data in ML (e.g. [95] and [96]) emerge.

An ML computing device can implement data exchange mechanisms for different purposes (e.g. to enable computation over scalers, vectors, matrices or even tensors). A data exchange protocol can leverage buffers (e.g. queue), calculators (e.g. vector), data shifting components and other components under certain control protocols (e.g. token and scheduling). They are organized within and implemented via the specific architecture of an ML computing device. In an ML computing device, units (e.g. input and output, 3-dimensional tensor multiplication calculator and related buffers) can be organized into dies. Data exchange mechanisms are implemented between dies to realize design purposes (i.e. independent input-output die).

When an ML-based AI system contains multiple collaborative ML computing devices, data exchange can happen between them as well as between host processors and ML computing devices. A challenge is to handle the heterogeneity of computing devices. To resolve this, one can abstract and organize logical devices on top of ML computing devices. This provides a relatively united view over ML computing devices as well as their computing capacities. In this way, data exchange between heterogenous devices can be simplified. Specific designs such as pooling and switch can be applied to promote efficiency.

6.9.2 Effectiveness and efficiency

Key aspects that affect the effectiveness and efficiency of data exchange within an ML computing device or between multiple ML computing devices include but are not limited to the following:

- strategies for selecting data to exchange;
- occasions when data are exchanged;
- channels through which data are exchanged;
- processing capacity difference between parts of an exchange;
- buffering mechanisms.

7 Approaches and measures for performance optimization

7.1 Approaches

7.1.1 Overview

Optimizations of ML computing devices can improve the overall effectiveness and efficiency of an AI system. From the ML computing devices' perspective, optimizations at the computing resources level as well as enabling software level can be applied.

In an AI system, ML computing devices collaborate with other auxiliary computing resources (e.g. storage, bus, cooling and power supply) to execute ML computation. An optimization approach can involve settings on ML computing devices as well as auxiliary computing resources. This document summarizes optimization approaches related to ML computing device characteristics, rather than focusing on auxiliary components. The target is to maximize the computing capacity of ML computing devices and consequently improve the effectiveness and efficiency of an AI system.

7.1.2 Computing resource-level

A general approach for optimizing performance at the computing resource level is to identify and solve performance bottlenecks in an AI system. For this, the following can be considered:

- Observation on ML computing device working status: To identify performance bottlenecks in an ML-based AI system, it is important to check the status of ML computing devices, in particular when a decline in performance or loss of tasks due to being overloaded are detected or reported. ML computing device parameters can reflect running status, including utilization ratios of ML computing device or co-processor, memory utilization, memory bandwidth, power and temperature. Such information can be used for diagnostics in bottleneck analysis.
- Observation on system logs: System logs can be an alternative for checking system status with a more macroscopic view, compared to ML computing device working status observation. Depending on the logging configuration, multiple kinds of running information can be recorded. When an ML-based AI system records logs for its workflow (e.g. starting and ending timestamp of each phase in a workflow), abnormal performance or bottlenecks in relevant parts of an AI system can be identified.
- Time consumption analysis: An ML training or inference process contains several sub-processes and a delay in any sub-process can deteriorate the overall performance. It is helpful to stakeholders if the time consumption of key sub-processes in training or inference is analysed. An ML model inference can typically include pre-processing such as encoding or decoding of data (e.g. video, image or audio), execution of inference and post-processing. When supported by ML computing devices, specific sub-process such as image decoding input cache status can be checked.

7.1.3 Enabling software-level

In an AI system, computing resources can be limited and cannot be extended arbitrarily. Optimizations of enabling software for ML computing devices can improve computing capacity on ML tasks. This can be done with existing computing resources without extensions. Once optimized, the effective results can be the equivalent to the extension of computing resources.

Enabling software for ML computing devices includes but is not limited to ML operator implementations, software toolkits for training or inference, software implementing data processing for ML, ML model compiling tools and ML model compression tools. An AI system can implement one or more of these kinds of software.

Optimizations at the enabling software-level include but are not limited to the following:

- Computational graph optimization: An ML model can be represented as a computational graph. There exist at least the following kinds of approaches summarized as follows:
  - Graph division: Proper division of a graph and the distribution of sub-graphs to ML computing devices can provide opportunity for parallel processing. Computational graph optimization adjusts the policies for computing graph division, such that the divided sub-graphs can be executed in parallel. In an AI system, due to the diversity of ML model structures and configurations of ML computing devices (e.g. magnitude and topology), no general approach yields precise graph divisions for all possible cases. However, trial-based approaches can be considered, performing tests with different optimization policies on an ML-based AI system to determine and establish optimization strategy for similar cases. Such empirical information can be stored in an optimization-oriented knowledge base.
  - Sparsification: Sparsification[97] is used to set certain weights in less important parts in a computational graph to zero, for the purpose of saving the amount of computation spent on those parts. It can be applied as an approach for computational graph optimization. Sound selection of weights to be sparsified can improve inference efficiency at an acceptable loss of accuracy. Sparsification can be applied to computational graphs for various domains. Taking a computational graph serving computer vision domain as an instance, with structured sparsification, the approach is to set the weights of a convolution channel, a column of fully connected layer, or more parts to zero. Since it works based on the structure of a graph, precise selection of weights to be set cannot be possible. The higher degree of sparsification is, the more accuracy loss can exist. Arbitrary sparsification can set arbitrary weights to zero in a graph. It can reduce the space needed for storage but provide less strength on inference efficiency promotion. Due to the arbitrariness of weight selection, the number of weights to compute is not actually relieved and dense computations are still significant. Parallel computation mechanisms with ML computing devices are difficult to apply. In comparison with structured and arbitrary sparsifications, semi-structured sparsification is in between, which enables sparsification of a computational graph in block-wise or with proportional zeroing in interested structures (e.g. vector). To improve inference efficiency, these approaches are subject to the specific settings of certain ML computing devices (e.g. block reading and matrix multiplication as well as proportional sparsification-oriented acceleration by certain types of GPUs). Complementary sparsification makes the parts of a computational graph to be complementary between zero and non-zero weights, such that parts can be combined and computed with less operations. In a convolution computational graph, with offline indexing of non-zero weights and transpose-based matrix multiplication during online inference, efficiency can be significantly improved. Unlike the former block- or proportion-based approaches, complementary sparsification does not put specific requirements on ML computing devices.
- ML operator optimization: An ML model can contain multiple computations executed by a specific ML computing device backend. When a model is compiled, the contained ML operators are determined. Due to ML operator diversity, the computation magnitudes of different ML operators can vary. The execution of an ML operator triggers computation and memory access. Frequent memory access or data shifting can reduce the efficiency of the overall computation. Running numerous, small ML operators with light computation but memory access can exacerbate the performance deterioration. An approach dealing with this situation is ML operator fusion. Based on the dependencies between ML operators, ML operators are fused to decrease the chances of memory access.

7.2 Measures

Measures can be selected and applied to provide supporting evidence for optimizing the performance of ML computing devices. Typical measures include but are not limited to the following:

- Time consumption: Time consumption represents a family of measures. Each can be defined by stakeholders and corresponds to the execution or access time duration of a process relevant to optimization. For an AI system, time consumption can be applied to the overall efficiency of an entire workflow containing multiple ML processes (e.g. an ML-based image recognition workflow contains subsequent processes such as object detection and image classification, as well as necessary pre- and post-processing). For an ML-based AI component (e.g. using an ML computing device for inference), time consumption can be defined as the length of time used by a process (e.g. inference with an image input) performed by the component. Time consumption can also be measured for detailed processes (e.g. on-device memory access time latency, collective communication latency between ML computing devices).

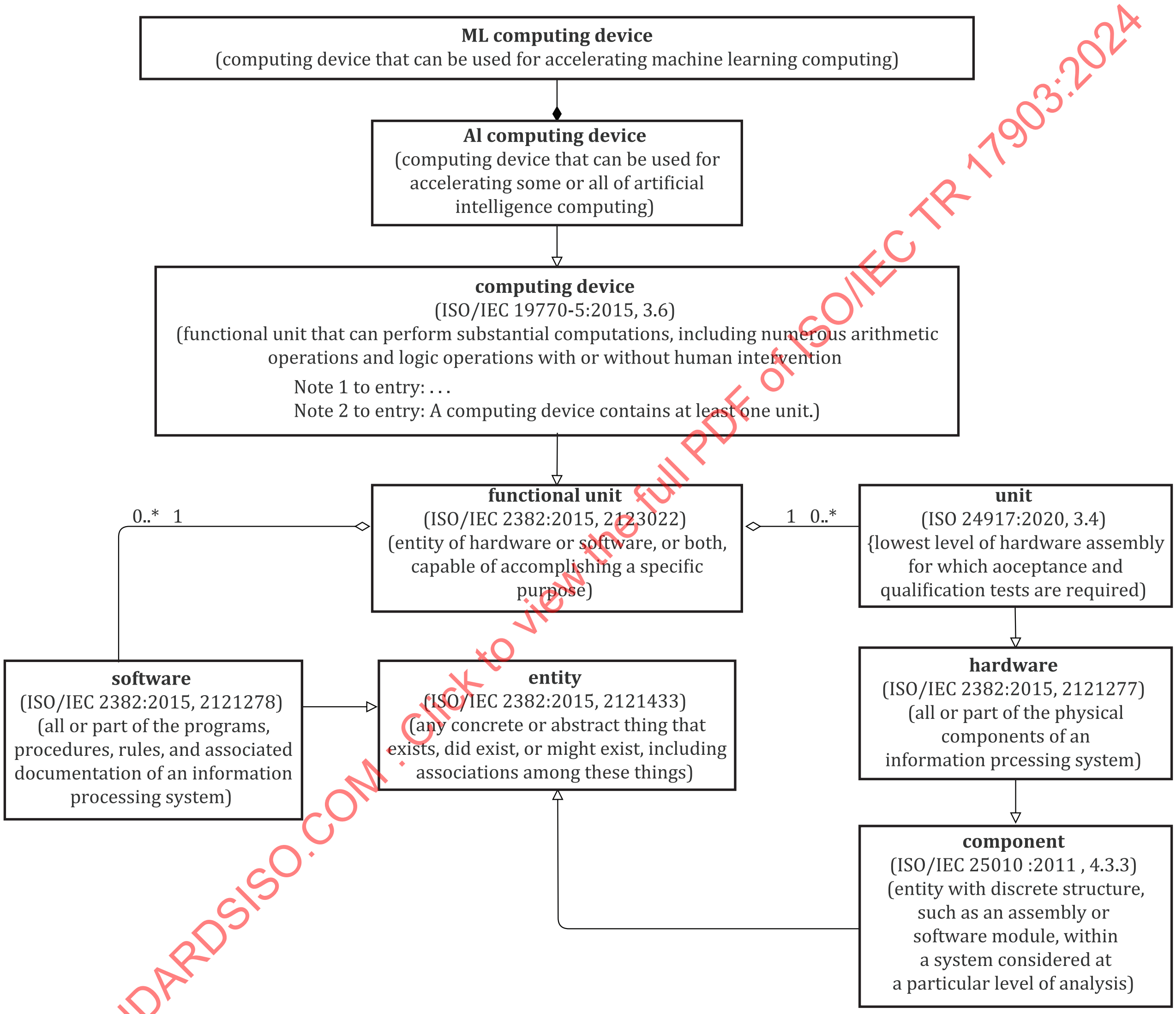
- Throughput: Throughput measures the amount of work that can be performed by an AI system or an AI component in a given period of time. In an ML-based AI system, training throughput refers to the amount of training data processed (e.g. training dataset) in a given time duration (e.g. time consumed by an epoch). Inference throughput is the number of samples inferenced in a given period of time, including necessary pre-processing and post-processing. In comparison with time consumption, throughput is more straightforward, representing the effective computing capacity of ML computing devices on a specific kind of ML task. An ML-based AI system can have different throughputs on different kinds of ML tasks (e.g. natural language processing and image classification). To measure the overall performance of the ML computing device in an AI system, a composite measure derived from throughput can be applied in the form of the weighted geometric mean[98] over relative throughputs.

- Power consumption: Considering sustainability aspects as well as the high power consumption of ML systems, it will be important to consider the power consumption of ML computing devices as one of the measures, together with other standard measures (e.g. power usage effectiveness in[99]) for sustainability.

Annex A
(informative)

Relationships between ML computing device-related definitions

The relationships between ML computing device-related definitions are presented in a UML class diagram in Figure A.1.


ML computing device
(computing device that can be used for accelerating machine learning computing)
AI computing device
(computing device that can be used for accelerating some or all of artificial intelligence computing)
computing device
(ISO/IEC 19770-5:2015, 3.6)
(functional unit that can perform substantial computations, including numerous arithmetic operations and logic operations with or without human intervention
Note 1 to entry: . . .
Note 2 to entry: A computing device contains at least one unit.)
functional unit
(ISO/IEC 2382:2015, 2123022)
(entity of hardware or software, or both, capable of accomplishing a specific purpose)
0..* 1
1 0..*
unit
(ISO 24917:2020, 3.4)
{lowest level of hardware assembly for which aoceptance and qualification tests are required)
software
(ISO/IEC 2382:2015, 2121278)
(all or part of the programs, procedures, rules, and associated documentation of an information processing system)
entity
(ISO/IEC 2382:2015, 2121433)
(any concrete or abstract thing that exists, did exist, or might exist, including associations among these things)
hardware
(ISO/IEC 2382:2015, 2121277)
(all or part of the physical components of an information prcessing system)
component
(ISO/IEC 25010 :2011 , 4.3.3)
(entity with discrete structure, such as an assembly or software module, within a system considered at a particular level of analysis)

NOTE See ISO/IEC 19505-1[71] for details on the notation in this diagram.

Figure A.1 — UML class diagram of relationships between ML computing device-related definitions

The term entity is the root of all the other definitions. A component is a sub-class of entity that has a restriction with discrete structure. Hardware is a kind of component which is physical. Unit is a sub-class of hardware with an additional restriction that is the lowest level of hardware assembly for which acceptance and qualification tests are required.

A functional unit can be hardware, software or both combined. It has aggregation relation (1 0...*) with software or hardware. Based on the definition, a functional unit can contain only software. This is not appropriate for a computing device, if computing device focuses on hardware. For this, the definition of