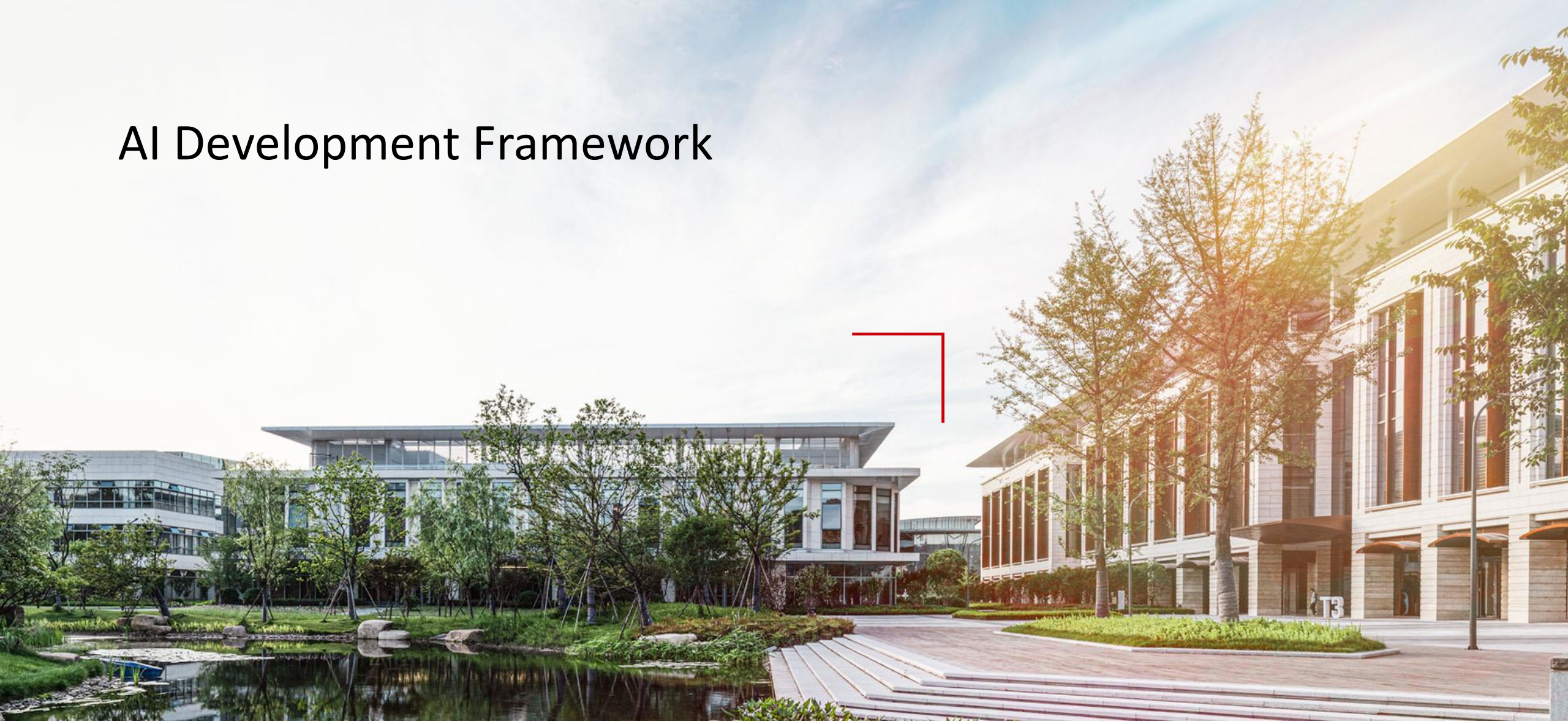
AI Development Framework



HUAWEI

Foreword

- Over the past few decades, machine learning has been the subject of extensive research and application. In fields such as image recognition, speech recognition and synthesis, autonomous driving, and machine vision, deep learning has witnessed significant achievements. This also poses higher requirements on the algorithm application and dependent frameworks.
- With the continuous development of the deep learning framework, a large number of computing resources can be conveniently used when a neural network model is trained on a large dataset. This course describes how to use the AI framework and develop AI applications.

Objectives

- On completion of this course, you will be able to:
  - Understand the current mainstream AI development framework.
  - Master the AI application development process.
  - Understand the structure and features of MindSpore.
  - Understand MindSpore development components.

Contents

AI Development Framework

- The deep learning framework lowers the entry barriers for AI development. We can use the existing model to configure parameters as required and train the model automatically, instead of compiling the code through the complex neural network and backpropagation algorithm. We also can add the user-defined network layer based on the existing model, or select the required classifier and optimization algorithm.

TensorFlow

PyTorch

PaddlePaddle

[M]S 昇思 MindSpore

Current Market Environment

- Similar AI framework:
  - Outside China: TensorFlow (Google), PyTorch (Meta), MXNet (Amazon)...
  - China: Paddle (Baidu), MindSpore (Huawei)...
- Mainstream computing processors:
  - GPU
  - CPU
  - NPU

PyTorch

- PyTorch is a machine learning computing framework released by Meta, formerly known as Torch. Torch is a scientific computing framework supported by a large number of machine learning algorithms and a tensor library similar to Numpy. Torch has a flexible programming environment but is not commonly used because its programming language is Lua. Therefore, PyTorch is created based on Python.
- PyTorch is a tensor library optimized for deep learning using GPUs and CPUs, as well as a mainstream deep learning framework. Enterprises such as Twitter, GMU, and Salesforce all adopt PyTorch.

PyTorch Features

- **Python preferred**: PyTorch supports fine-grained access for Python instead of adopting Python based on C++ framework. We can use PyTorch as easily as using Numpy and Scipy.
- **Dynamic neural network**: A static computation graph is required before running the TensorFlow 1.x and then repeatedly executing the graph through feed and run operations. But PyTorch programs can dynamically build or modify computational graphs during their execution.
- **Easy to debug**: PyTorch can generate dynamic diagrams during running. Developers can stop the interpreter in the debugger and view the output of a node.
- PyTorch provides tensors that can run on CPU and GPU and greatly accelerate the computation.

TensorFlow

- TensorFlow (TF) is an end-to-end open-source machine learning platform designed by Google. Currently, the main version is TensorFlow 2.x.
  - TensorFlow has three iterative versions: TensorFlow 0.1, TensorFlow 1.0, and TensorFlow 2.0. Currently, TensorFlow 2.x is mainly used.
  - TensorFlow 2.X includes TensorFlow.js, JavaScript, TensorFlow Lite, and TensorFlow Extended. They build a complete ecosystem for TensorFlow.
  - AlphaGo Zero is trained based on TensorFlow.

TensorFlow 2.x VS TensorFlow 1.x

- Disadvantages of TensorFlow V1.0:
  - In TensorFlow 1.0, the result generated for creating a tensor cannot be directly returned. Instead, a session mechanism needs to be created and the graph is included. Then the session needs to be run. This style is similar to the hardware programming language VHDL.
  - TensorFlow 1.0 is difficult to debug, APIs are disordered, and it is difficult for beginners. As a result, many researchers change to use PyTorch.

TensorFlow 2.x VS TensorFlow 1.x

- TensorFlow 2 features
  - Advanced Keras API

    Easy to use (the most important feature): The graph and session mechanisms are removed.

    Main improvements:

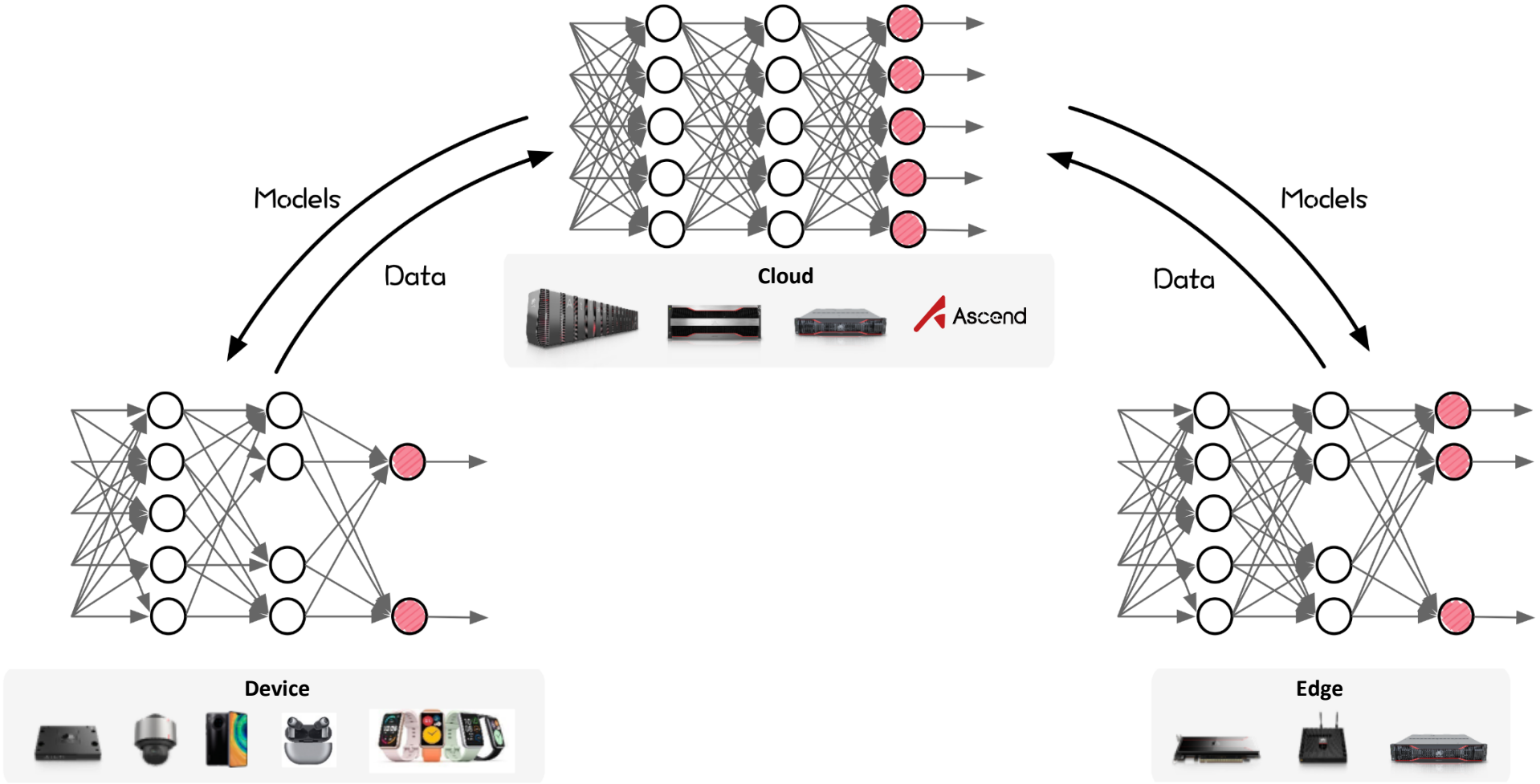
    - The core function of TensorFlow 2 is the dynamic graph mechanism **Eager Execution**. This allows users to compile programs and debug models and make developers easier to learn and use TensorFlow.
    - Supports more platforms and languages, and improves the compatibility between components by standardizing the API exchange format and providing base lines.
    - Delete discarded and duplicate APIs to prevent confusion.
    - Although tf.contrib is no longer used, the valuable modules will be applied in other applications, and the remaining will be deleted.

MindSpore

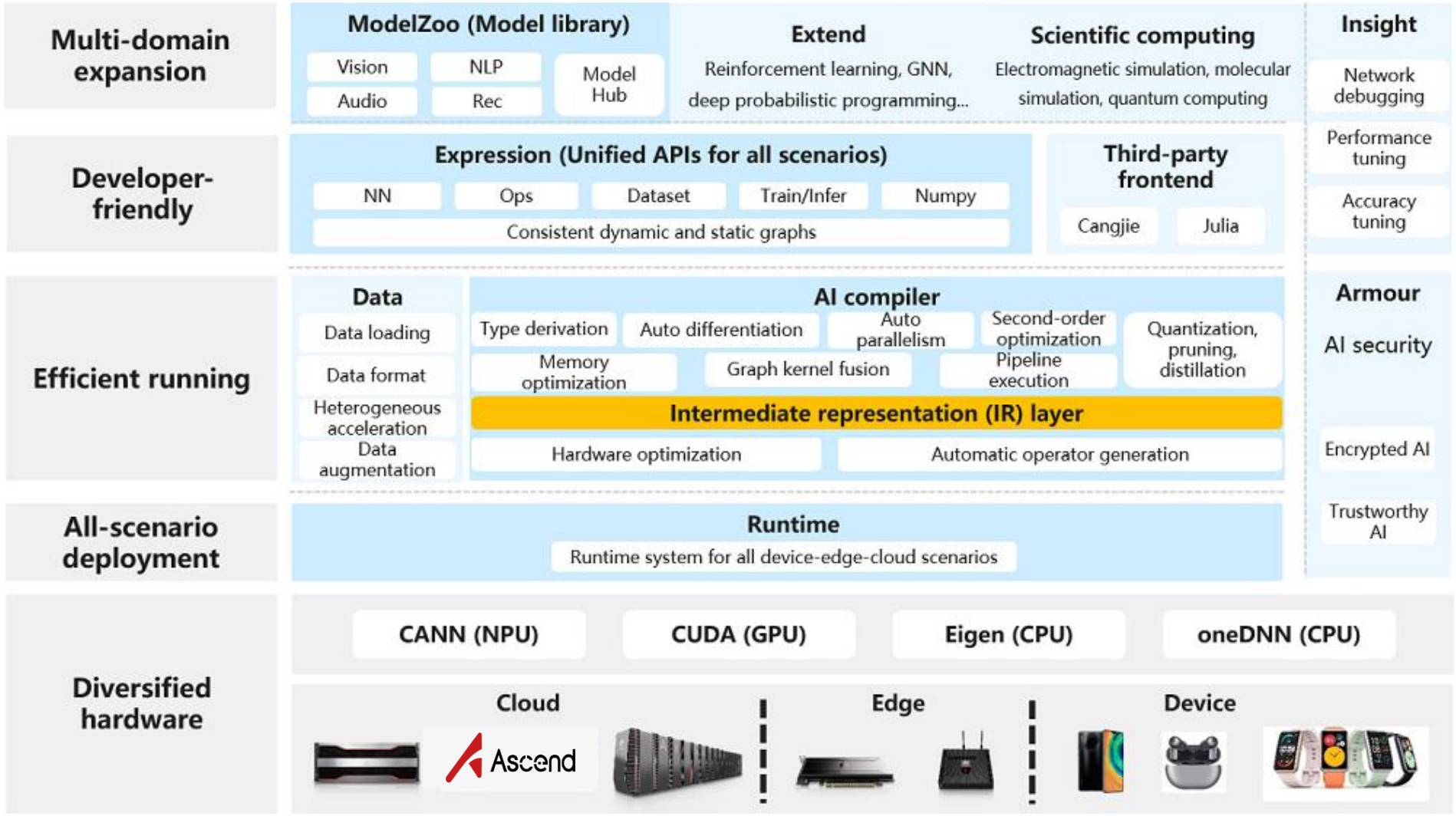
- MindSpore is Huawei's next-generation deep learning framework. **By leveraging the computing power of Huawei Ascend processors**, enabling flexible deployment in device, edge, and cloud scenarios, and integrating the industry's best practices, MindSpore defines a new paradigm in AI programming and lowers the threshold for AI development.
  - MindSpore aims to **achieve three goals**: easy development, efficient execution, and all-scenario coverage.
  - MindSpore is **highly flexible and supports data parallel**, model parallel, and hybrid parallel training.
  - MindSpore has the **automatic parallelism capability** which efficiently searches for a fast parallel strategy in a large strategy space.

Contents

All-Scenario AI Framework

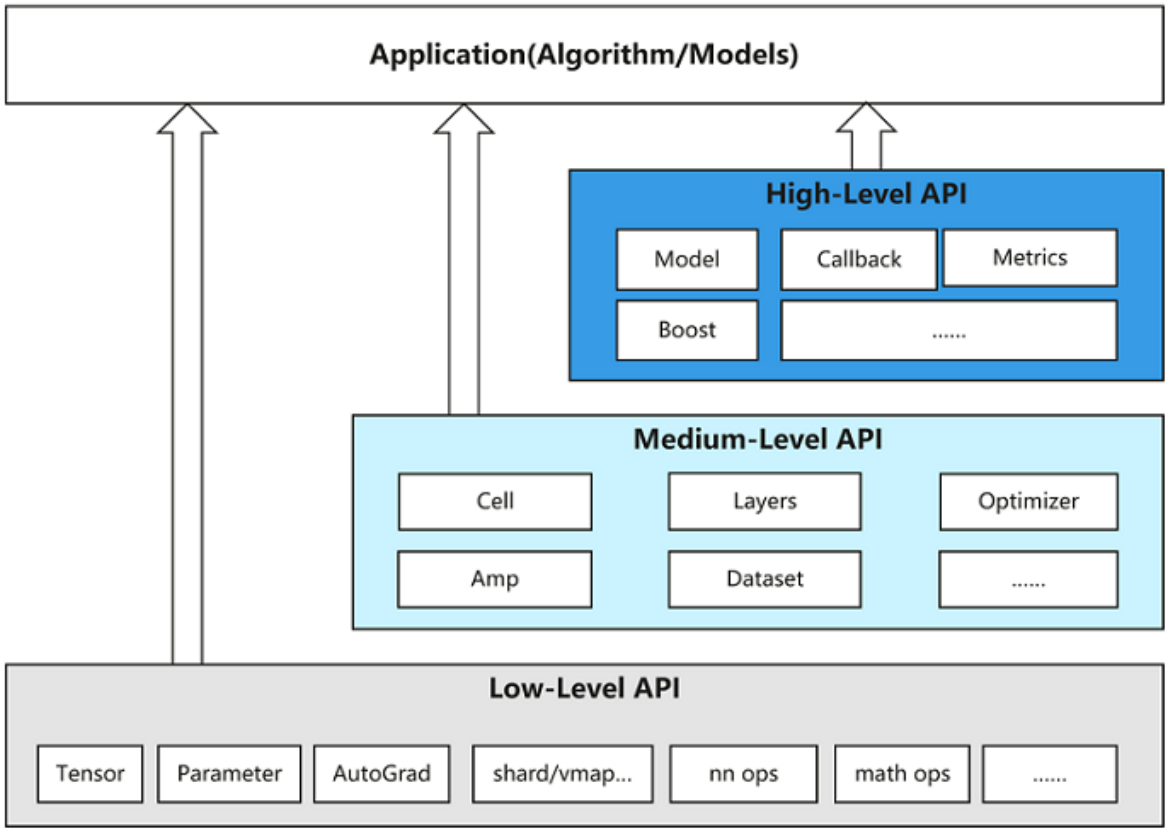


MindSpore Architecture



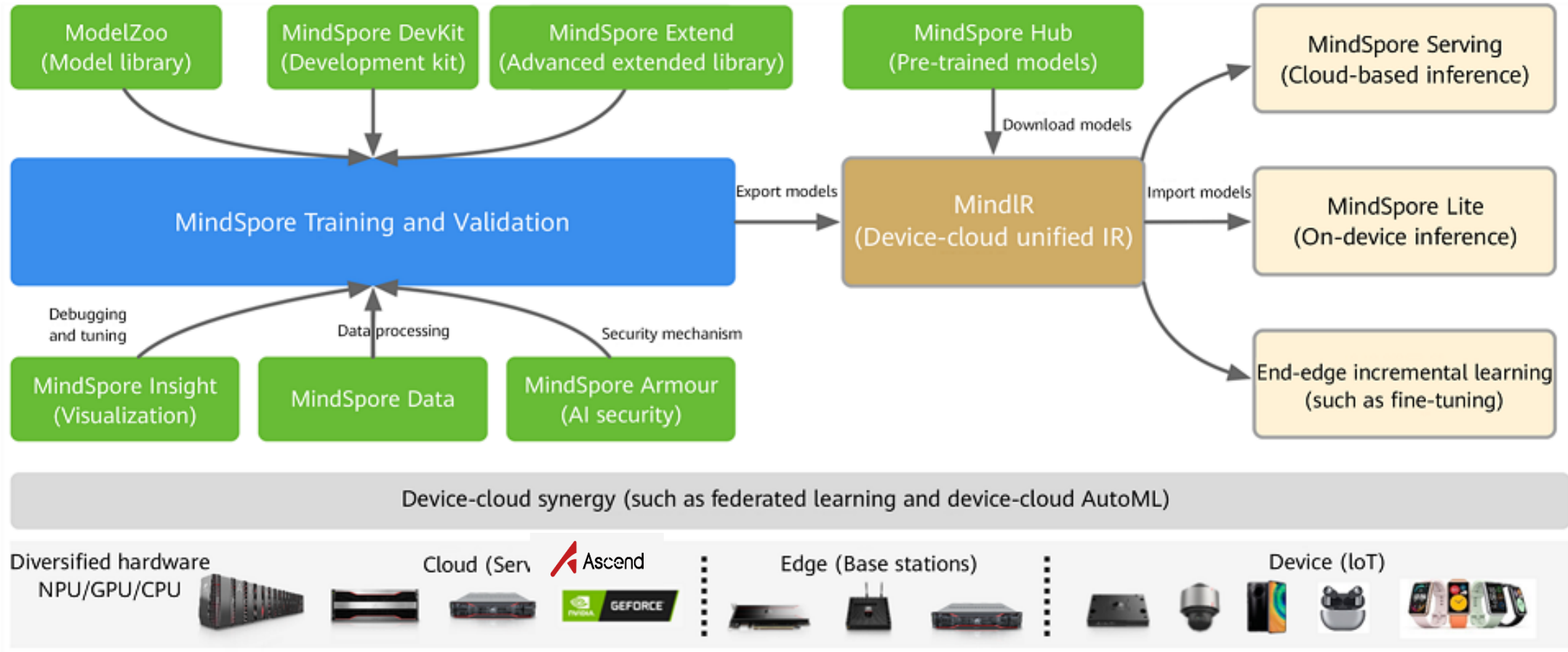
API Level Structure

- To develop AI applications, algorithms, and models, MindSpore provides users with three levels of APIs: low-level Python APIs, medium-level Python APIs, and high-level Python APIs.
- **High-level** Python APIs have **better encapsulation**, **low-level APIs** have better **flexibility**, and **medium-level APIs have both flexibility and encapsulation**, meeting developers' requirements in different fields and at different levels.



HUAWEI

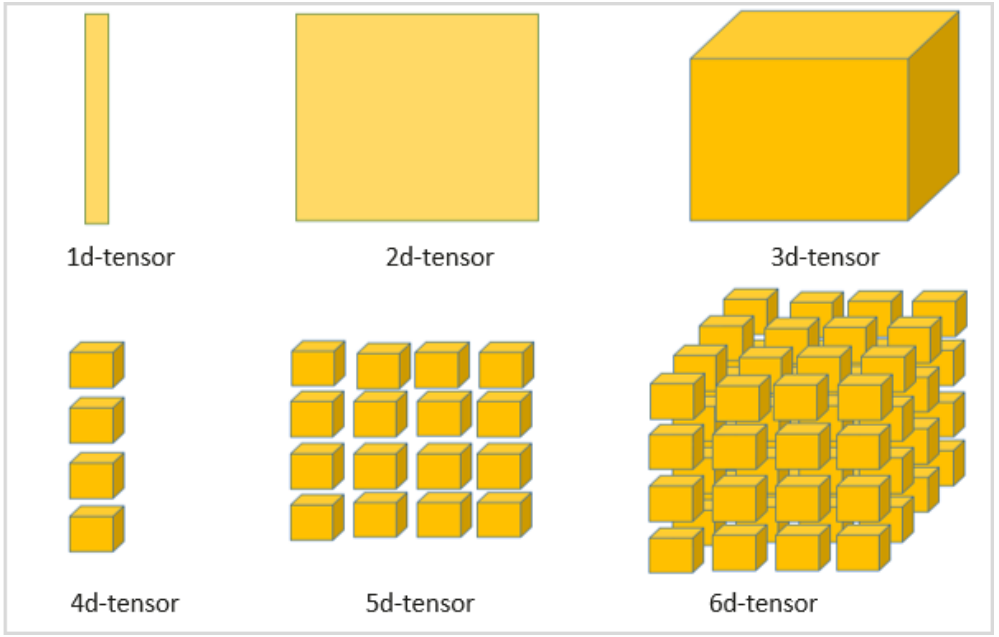
MindSpore All-Scenario Support Mode



Contents

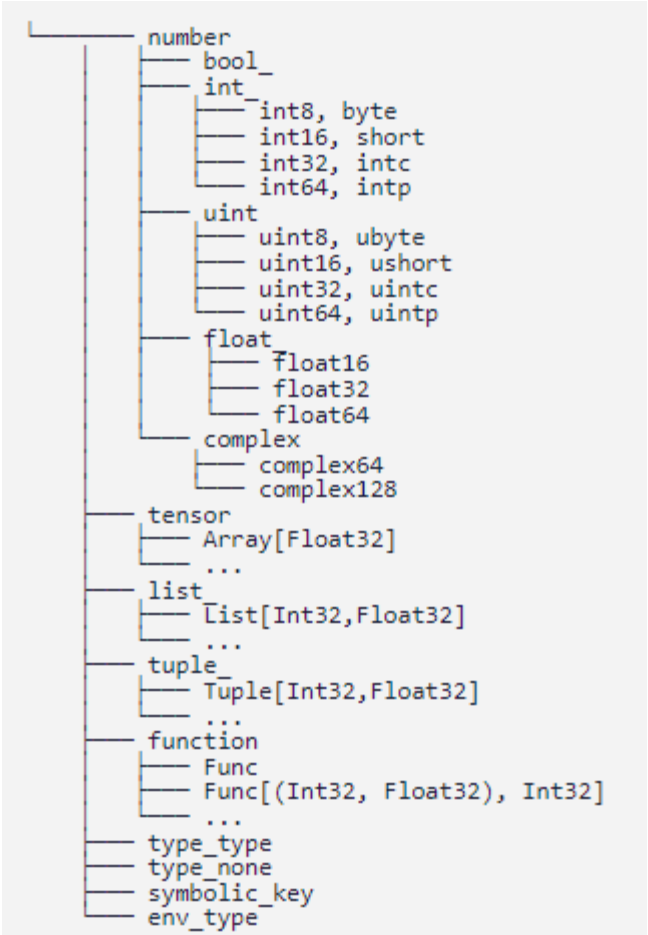
Tensor

- The most basic data structure in MindSpore is tensor. All data is encapsulated in tensors.
  - Tensor: a multi-dimensional array.
  - Zero-order tensor: scalar
  - First-order tensor: vector
  - Second-order tensor: matrix
- In MindSpore, you can use the following methods to quickly create a tensor:
  - mindspore.Tensor.
    - Collect a set of data to create a tensor. **Dimension can be specified during creation.**



Data Types in MindSpore

- MindSpore supports data types such as int, uint, and float.
  - For details, see the figure on the right.
- In MindSpore, you can use **mindspore.dtype** to create a data type object.
- MindSpore is **compatible with data types in NumPy** and Python, and can **convert data types** through **mindspore.dtype_to_nptyp** and **mindspore.dtype_to_pytype**.



Running Environment Setup

- When you run MIndSpore, you need to specify **environment parameters**, including the **graph mode**, **device**, and **memory size**. You can use related APIs to obtain detailed information about the current running environment.
- APIs related to the **running environment**:
  - **mindspore. context.set_context** is used to set the context of the running environment.
  - **mindspore. context. get_context** is used to obtain the attribute value in the context according to the input key.
  - **mindspore. context. ParallelMode** is used to enable the parallel mode.
  - **mindspore. context. set_ps_context** is used to set the context of the training mode in the parameter server
  - ……

context

- The context of MindSpore can be used to **set up the current operating environment**, including the **execution mode**, **execution backend**, and **feature switching**.
  - mindspore.context.set_context(**kwargs)

Parameter	Description	Value
device_id	Target device ID	Value range: [0,4096-1]. The default value is **0**.
device_target	Target device to be run	Value range: Ascend, GPU, and CPU
mode	Selects the running mode	The default value is **GRAPH_MODE**, **GRAPH_MODE/0** or **PYNATIVE_MODE/1**.
enable_sparse	Whether to enable the sparse feature	The default value is **False**.
save_graphs	Whether to save the computation graph	The default value is **False**.
runtime_num_threads	Controls the number of threads in the thread pool during running	The default value is **30**.

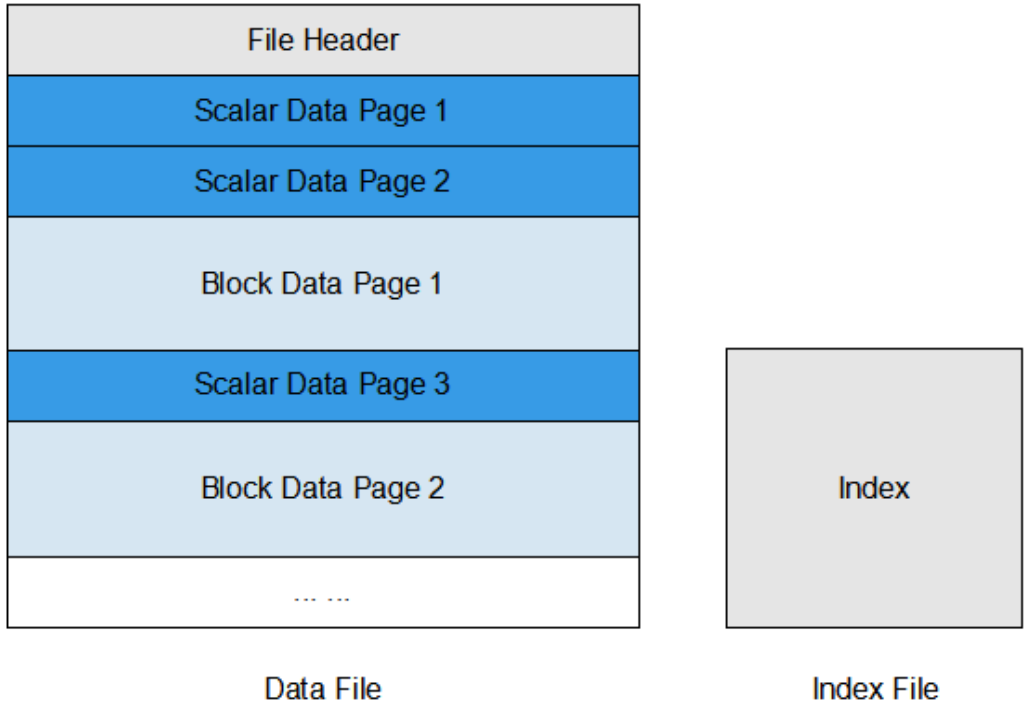
MindSpore Data Processing Module

- MindSpore provides the dataset submodule for processing datasets.
  - The **mindspore.dataset** module provides APIs for **loading and processing various public datasets**, such as **MNIST**, **CIFAR-10**, **COCO**, **ImageNet**, and **CLUE**. It can also load datasets in **standard formats** in the industry, such as MindRecord, TFRecord, and Manifest. You can also **use this module to define and load your own datasets**.
  - This module also provides the **data enhancement function** for data such as **voice, text, and image**.

MindRecord

- To read data efficiently, serialize the data and store it in a **set of files that can be read linearly**. Each file ranges from **100 MB to 200 MB**.

- MindRecord is an efficient data format developed by MindSpore.
  - The mindspore.mindrecord module converts different datasets into the MindRecord format and provides some methods to read, write, and retrieve MindRecord data files.
  - MindRecord data format can **reduce the disk I/O and network I/O overhead**.



Neural Network Module in MindSpore

- The mindspore.nn module provides predefined building blocks or computing units in a neural network.
  - The module contains the following components for building the neural network:
    - Network structures such as **RNN, CNN, and LSTM**.
    - **Loss functions** such as MSELoss and SoftmaxCrossEntropyWithLogits.
    - **Optimizers** such as Momentum and Adam.
    - **Model evaluation indicators** such as F1 Score and AUC.
- **mindspore.Model** is the high-level API for model training and inference.

Callback Function

- The callback function in MindSpore **is not a function but a class**. You can use the callback function to **observe the internal status and related information of the network** during training or to perform specific actions in a specific period. For example, monitor loss functions, save model parameters, dynamically adjust parameters, and terminate training tasks in advance.
- MindSpore allows users to insert **user-defined operations** in a specific phase of training or inference through the following callback capabilities:
  - Callback classes provided by **MindSpore framework**, such as ModelCheckpoint, LossMonitor, and SummaryCollector.
  - User-defined callback.

Model File for Inference

MindSpore can **execute inference** tasks on different hardware platforms based on trained models.

Two types of data are supported: training parameters and network models.

1. Training parameters are stored in the checkpoint format.
2. Network models are stored in the MindIR, AIR, or ONNX format.

File Format	Basic Concept	Application Scenario
Checkpoint	Uses the Protocol Buffers format and stores all network parameter values	Used to resume training after a training task is interrupted or execute a fine-tuning task after training.
MindIR	A graph-based function-like IR of MindSpore which defines scalable graph structures and operator IRs.	Eliminates model differences between different backends and is generally used to perform inference tasks across hardware platforms.
ONNX	An open format built to represent machine learning models.	Used for model migration between different frameworks or used on the inference engine TensorRT.
AIR	An open file format defined by Huawei for machine learning.	Adapts to Huawei AI processors well and is generally used to execute inference tasks on Ascend 310.

Contents

Dynamic and Static Graphs

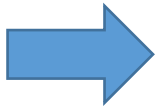
- Currently, there are **two execution modes** of a mainstream deep learning framework: a static graph mode (**GRAPH_MODE**) and a dynamic graph mode (**PYNATIVE_MODE**).
- In **static graph mode**, when the program is built and executed, the graph structure of the neural network is generated first, and then the computation operations involved in the graph are performed. Therefore, in the static graph mode, the **compiler can achieve better execution performance by using technologies** such as graph optimization, which **facilitates large-scale deployment** and **cross-platform running**.
- In **the dynamic graph mode**, the program is executed line by line according to the code writing sequence. In the forward execution process, the backward execution graph is dynamically generated according to the backward propagation principle. In this mode, the compiler delivers the operators in the neural network to the device **one by one for computing**, enabling users to **build and debug the neural network model**.

MindSpore Static Graph

- In MindSpore, the static graph mode is also called the **Graph mode**. You can set the static graph mode by calling the set_context API.
  - **set_context (mode=GRAPH_MODE)** is the default mode.
- In Graph mode, MindSpore **converts the Python source code into an IR** (MindIR), optimizes related graphs based on the IR, and executes the optimized graphs on the hardware device.
  - In Graph mode, you need to use the **nn.Cell class and write execution code in the construct function**, or use the @ms_function modifier.
  - **The Graph mode is compiled and optimized based on MindIR**.

MindSpore Static Graph Usage

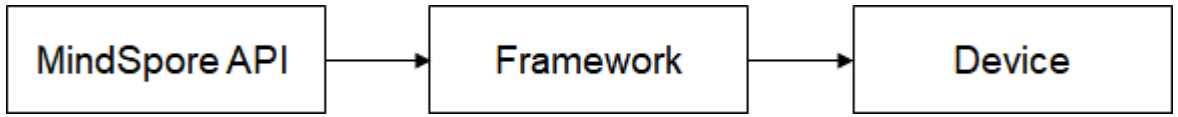
```
import numpy as np
import mindspore.nn as nn
import mindspore.ops as ops
import mindspore as ms

ms.set_context(mode=ms.GRAPH_MODE, device_target="CPU")

class Net(nn.Cell):
    def __init__(self):
        super(Net, self).__init__()
        self.mul = ops.Mul()
    def construct(self, x, y):
        return self.mul(x, y)

x = ms.Tensor(np.array([1.0, 2.0, 3.0]).astype(np.float32))
y = ms.Tensor(np.array([4.0, 5.0, 6.0]).astype(np.float32))

net = Net()
print(net(x, y))
```



```
[ 4. 10. 18.]
```

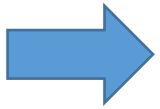
MindSpore Dynamic Graph

- In MindSpore, the dynamic graph mode is also called the **PyNative mode**. You can set the dynamic graph mode by calling the set_context API.
  - **set_context(mode=PYNATIVE_MODE)**
- In PyNative mode, you can use complete Python APIs. In addition, when APIs provided by MindSpore are used, the framework executes operator API operations on the selected hardware platform (Ascend/GPU/CPU) and returns the corresponding result.



MindSpore Dynamic Graph Usage

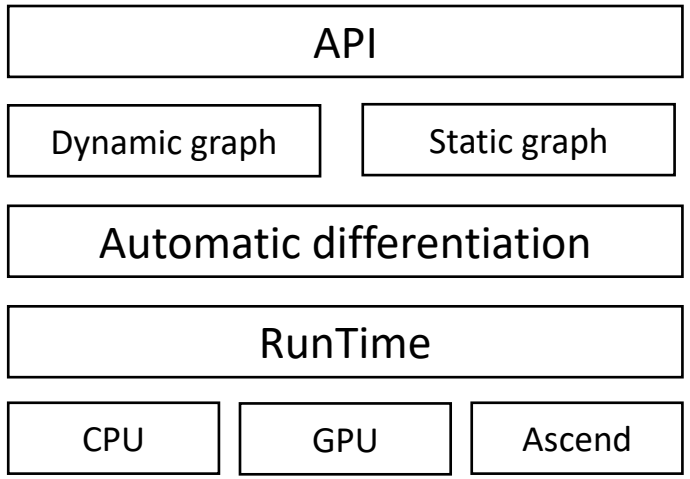
```
import numpy as np
import mindspore.nn as nn
import mindspore as ms
import mindspore.ops as ops

ms.set_context(mode=ms.PYNATIVE_MODE, device_target="CPU")
x = ms.Tensor(np.ones([1, 3, 3, 4]).astype(np.float32)) # Create a tensor.
y = ms.Tensor(np.ones([1, 3, 3, 4]).astype(np.float32))
output = ops.add (x, y) # Add
print(output.asnumpy())
```



```
[[[[2. 2. 2. 2.]
   [2. 2. 2. 2.]
   [2. 2. 2. 2.]]

  [[2. 2. 2. 2.]
   [2. 2. 2. 2.]
   [2. 2. 2. 2.]]

  [[2. 2. 2. 2.]
   [2. 2. 2. 2.]
   [2. 2. 2. 2.]]]]
```

Unification of Dynamic and Static Graphs

- Currently, dynamic and static graphs are supported in the industry. Dynamic graphs are executed through explanation, with dynamic syntax affinity and flexible expression. Static graphs are executed through just in time (JIT) build, which focuses on static syntax and has many syntax constraints. The build process of the dynamic graph is different from that of the static graph. As a result, the syntax constraints are also different.
- For dynamic and static graph modes, MindSpore first unifies the API expression and uses the same APIs in the two modes. Then, it unifies the underlying differentiation mechanism of dynamic and static graphs.



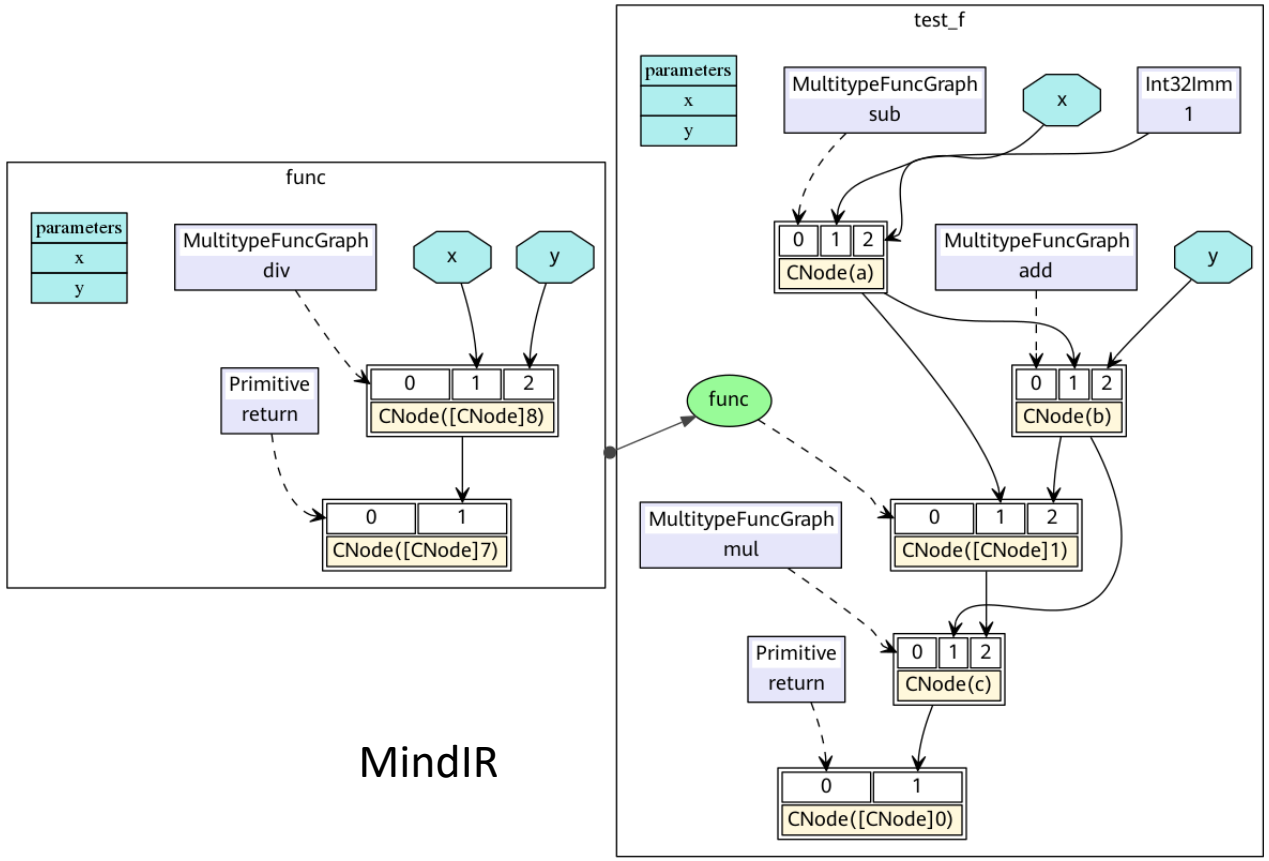
MindIR

- An **intermediate representation (IR)** is a representation of a program between the source and target languages, which facilitates program analysis and optimization for the compiler.
- MindSpore IR (MindIR) is a function-style IR based on graph representation. Its core purpose is to serve automatic differential transformation, which uses semantics close to ANF functional IR.
  - MindIR can help you implement multiple deployments in one training session and implement device-cloud interconnection.

Example for MindIR

```
def func(x, y):
    return x / y

@ms_function
def test_f(x, y):
    a = x - 1
    b = a + y
    c = b * func(a, b)
    return c
```

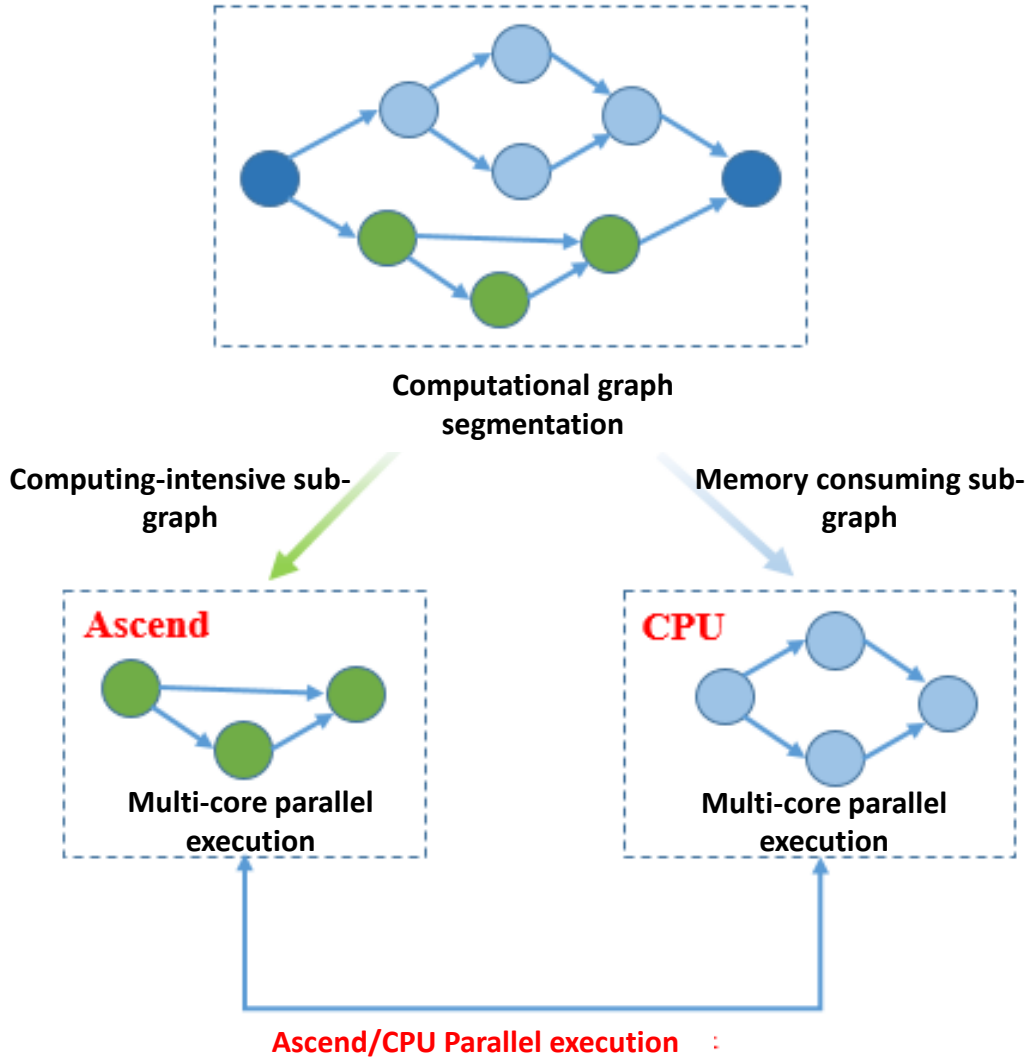
Python code



MindIR

Heterogeneous Parallel Training

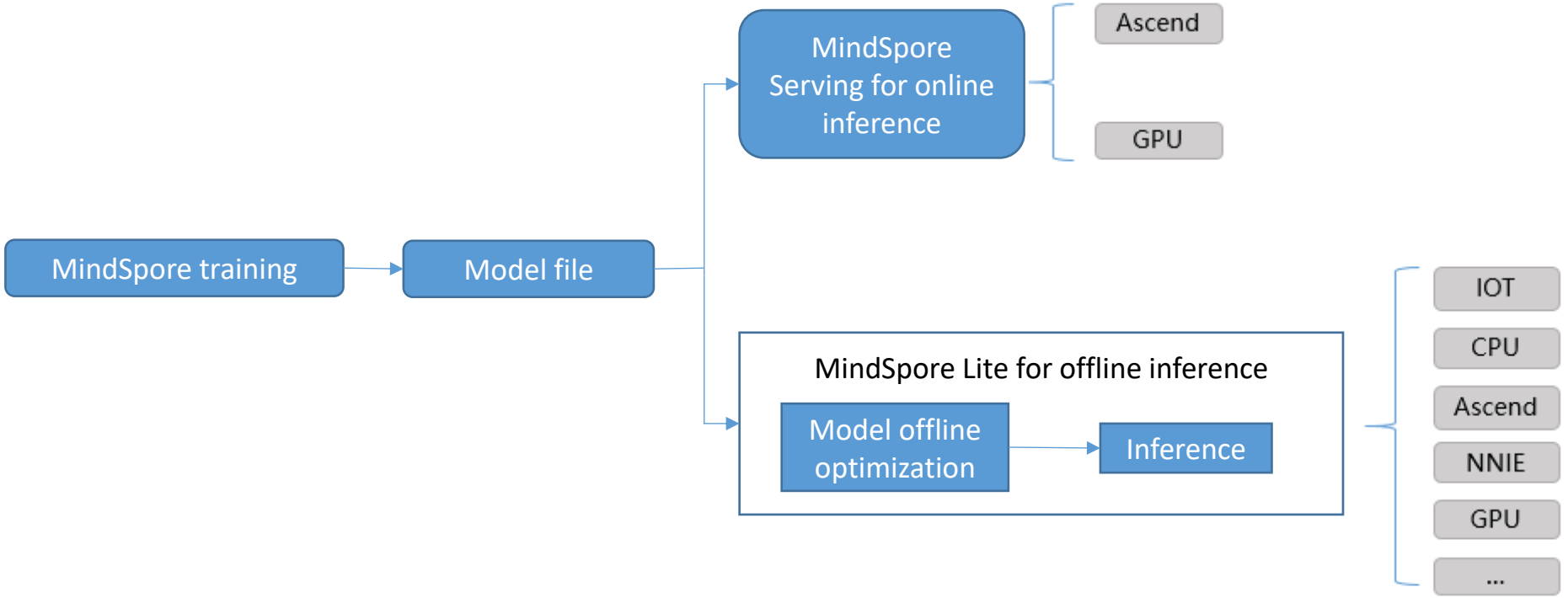
- Through the heterogeneous parallel training method, the operators that have high memory consumption or are suitable for CPU logical processing, are segmented to the CPU sub-graph. Operators that have low memory consumption are segmented to the hardware accelerator sub-graph. Different sub-graphs perform network training under the framework, and this method allows independent sub-graphs in different hardware to be executed in parallel.



Contents

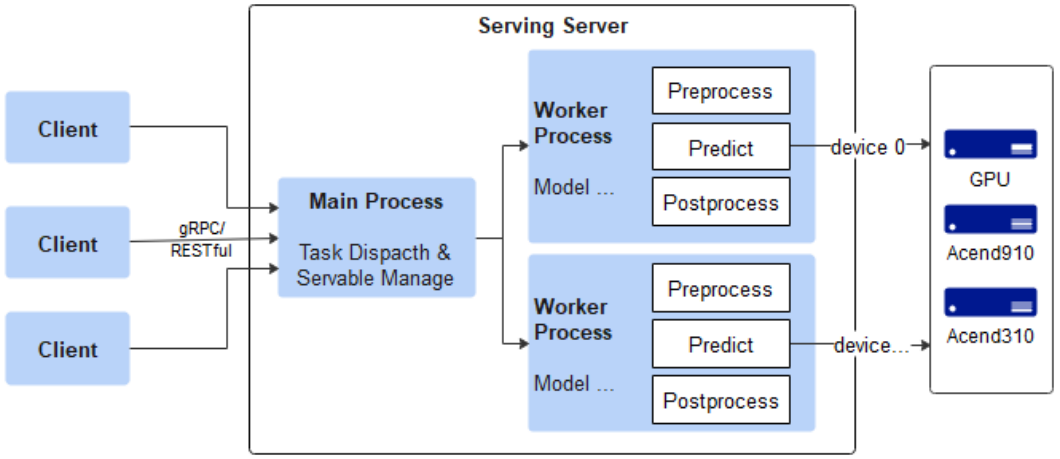
MindSpore Inference Deployment Process

- The model file trained by MindSpore can be executed in cloud services through MindSpore Serving and executed on servers and devices through MindSpore Lite. Besides, MindSpore Lite supports offline model optimization by using the convert tool to build a lightweight inference framework and achieve high-performance model execution.



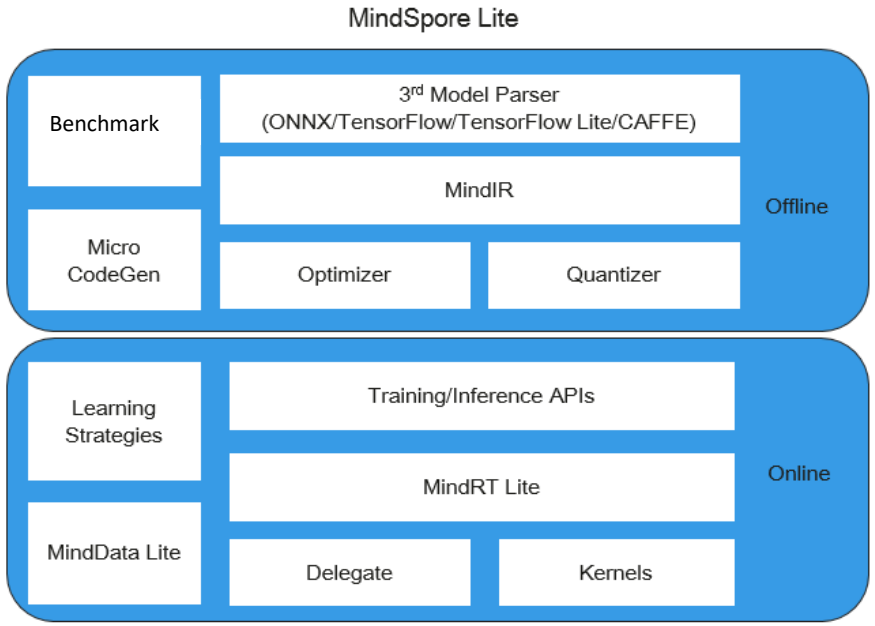
MindSpore Serving

- MindSpore Serving is a lightweight and high-performance service module that helps MindSpore developers efficiently deploy online inference services in the production environment.
- Three ways are available to access the MindSpore Serving services:
  - Call the gRPC API to access MindSpore Serving services.
  - Call the RESTful API to access MindSpore Serving services.
  - The Servable of MindSpore Serving provides the inference services:
    - From a single model.
    - From the combination of multiple models.



MindSpore Lite

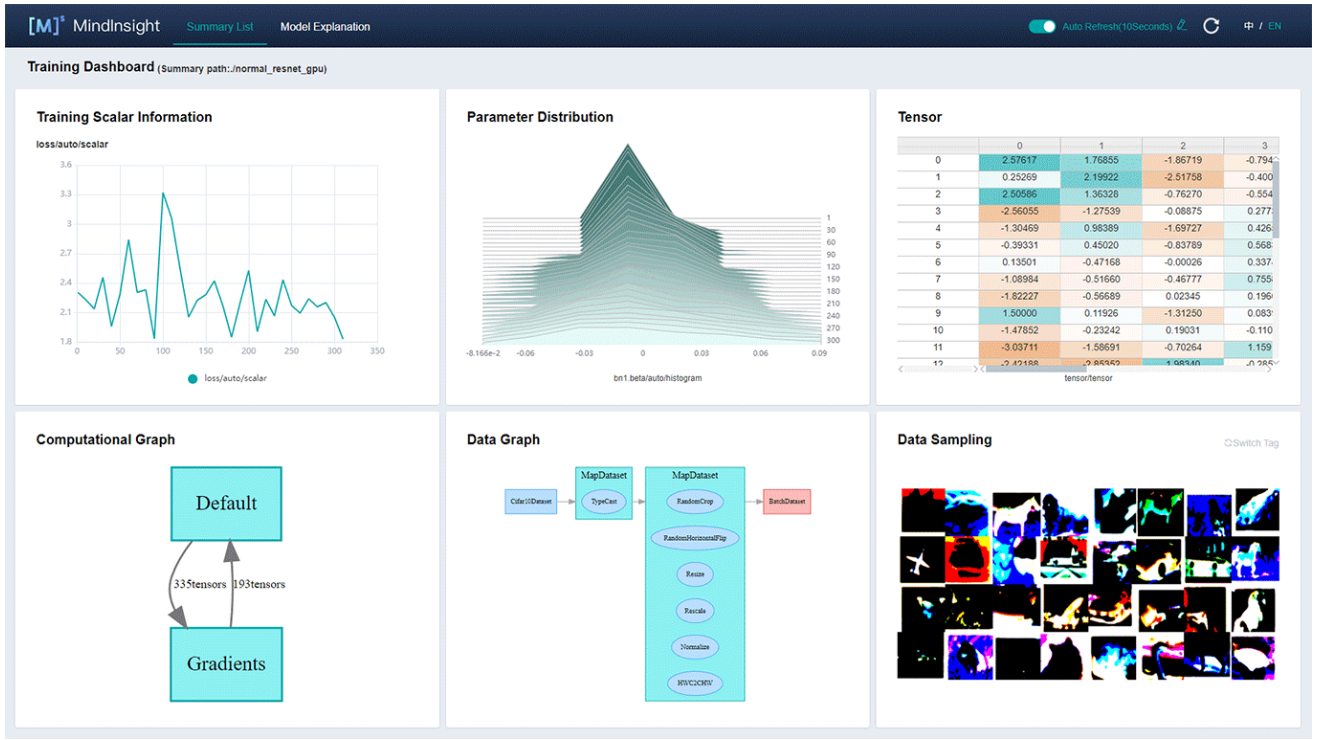
- MindSpore Lite is an **ultra-fast**, **intelligent, and simplified AI engine** that enables intelligent applications in all scenarios, provides **E2E solutions for users**, and helps users enable AI capabilities.
- MindSpore Lite consists of two modules: **online and offline**.



HUAWEI

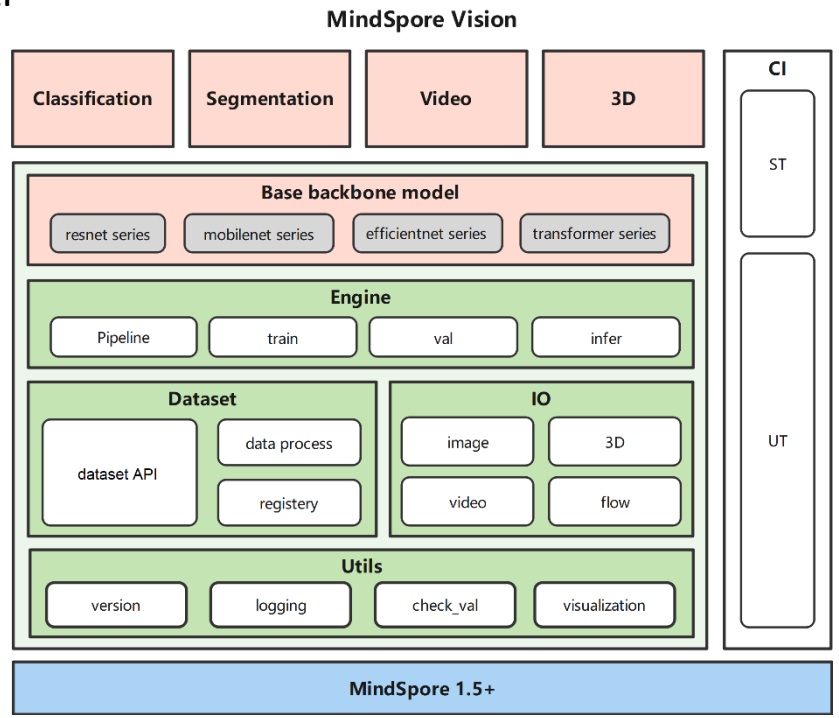
MindInsight

- MindInsight is a **visualized debugging and optimization tool** of MindSpore. It can visualize the training process, optimize model performance, and improve debugging accuracy. Besides, MindInsight also provides a command line for users to easily search for hyperparameters and migrate models.



MindSpore Vision

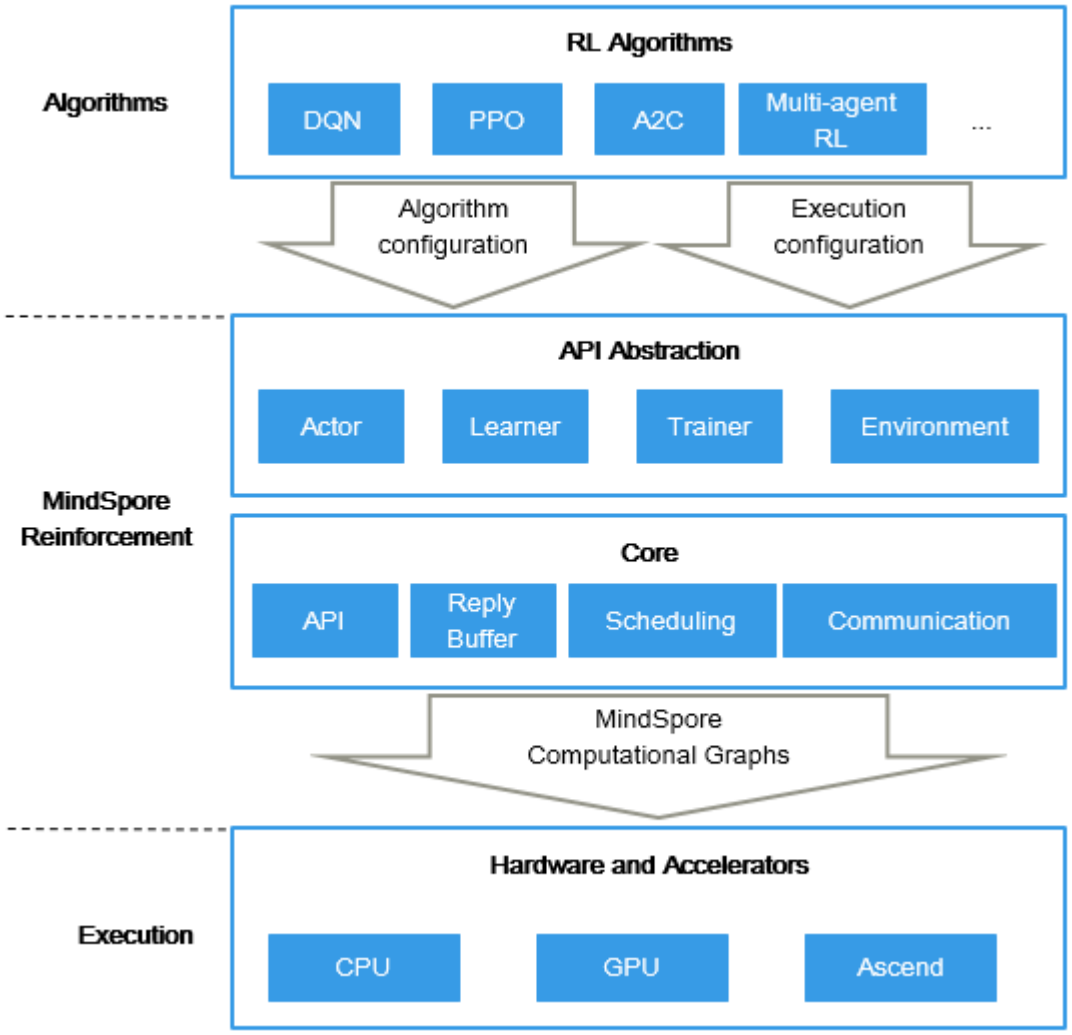
- MindSpore Vision is an **open source computer vision research tool library** based on the MindSpore framework. The tasks executed through the tool library include **classification**, **segmentation** (under development), **video** (under development), and **3D** (under development). MindSpore Vision aims to provide easy-to-use APIs to help users redevelop existi [illegible] models.



HUAWEI

MindSpore Reinforcement

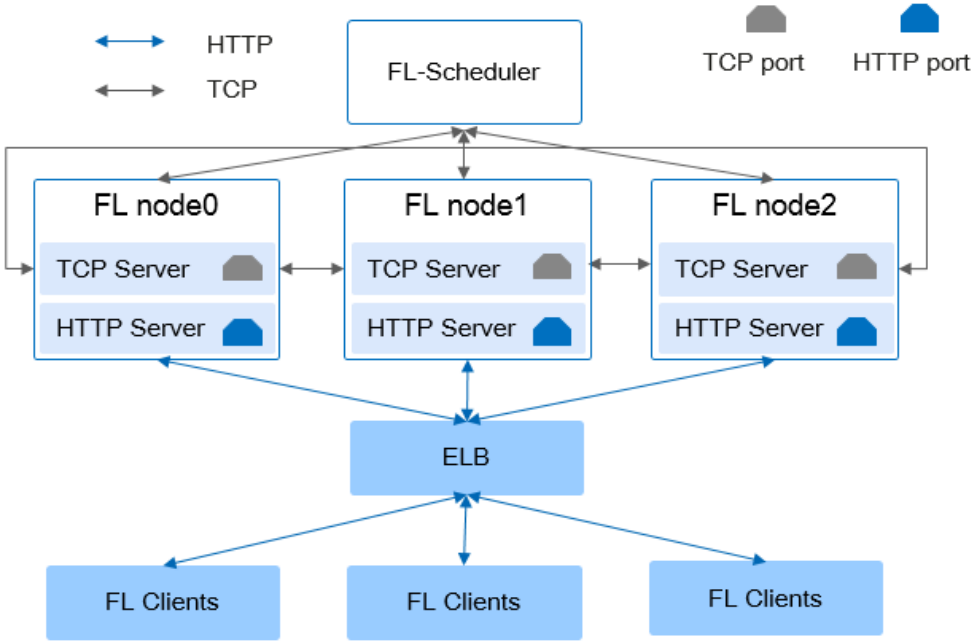
- MindSpore Reinforcement is an **open-source reinforcement learning framework** that supports **distributed training of agents** using reinforcement learning algorithms.
  - MindSpore Reinforcement provides simple and abstract APIs for writing reinforcement learning algorithms. It decouples algorithms from specific deployment and execution processes, including accelerator usage, parallelism degree, and cross-node computing scheduling.



HUAWEI

MindSpore Federated

- Federated learning is an **encrypted distributed machine learning technology** that allows different participants to build AI models **without sharing local data**. MindSpore Federated is an open-source federated learning framework that supports the commercial deployment of tens of millions of stateless devices.



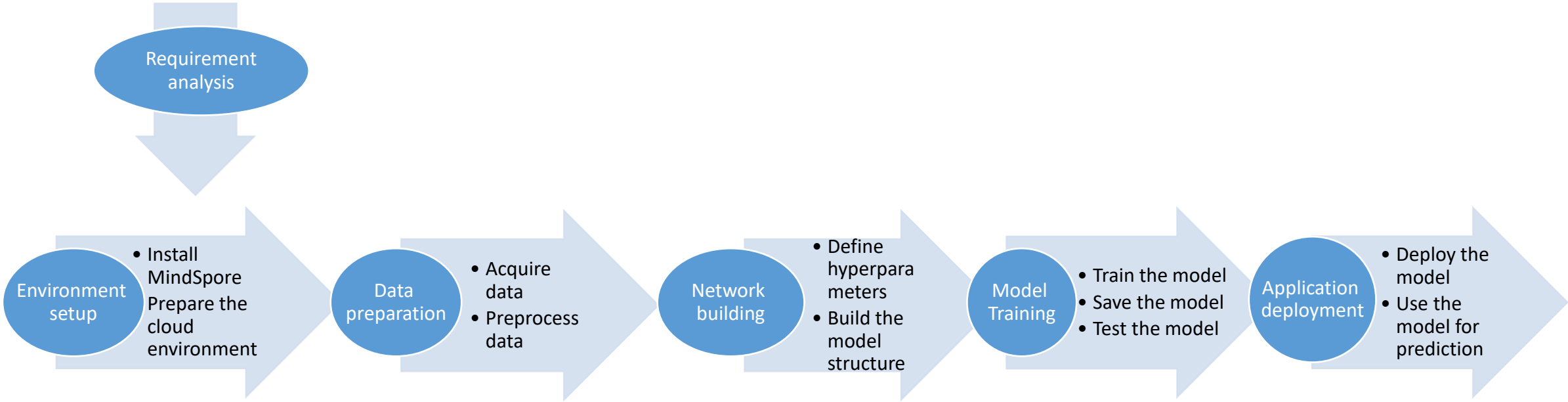
HUAWEI

Other components

- **MindQuantum**: An open-source deep learning framework of MindSpore and a common quantum computing framework developed by HiQ. It supports training and inference of multiple quantum neural networks.
- **MindSpore Probability**: MindSpore probabilistic programming provides a framework that seamlessly integrates Bayesian learning and deep learning. It aims to provide users with a complete probability learning library for establishing probabilistic models and applying Bayesian inference.
- **MindSpore Golden Stick**: model compression algorithm set
- ……

Contents

AI Application Development Process



HUAWEI

ResNet50 Image Classification Application

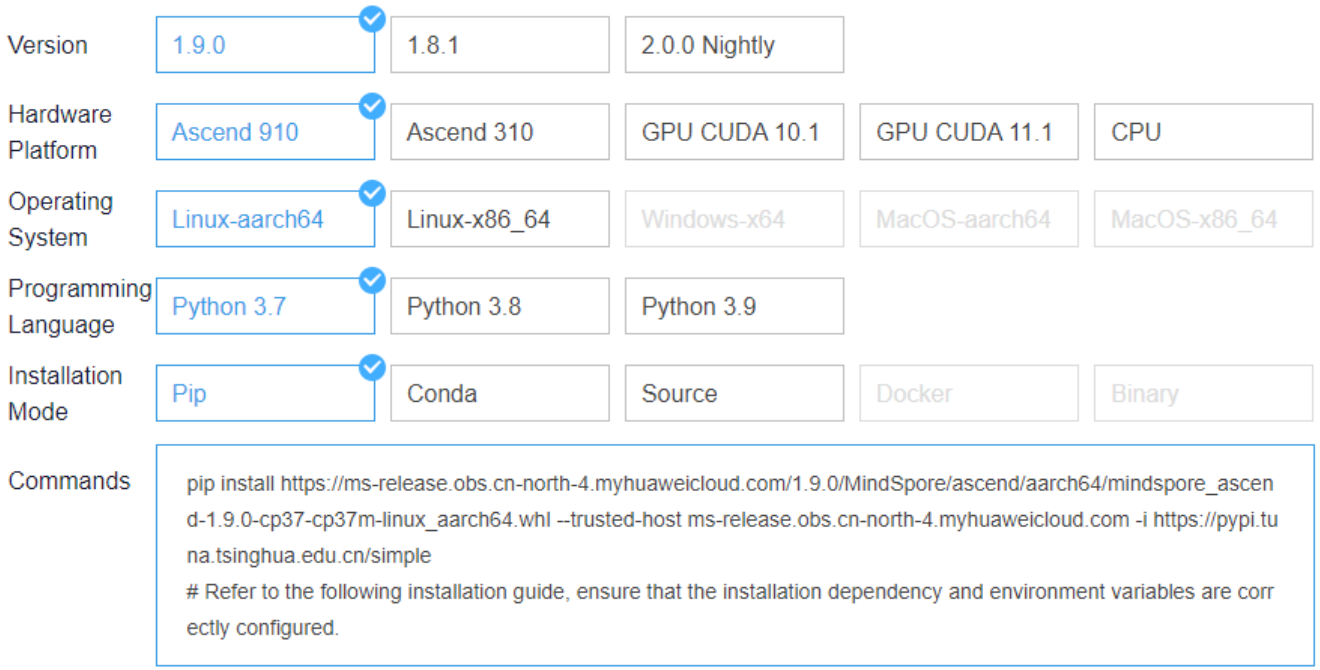
- Image classification is the most basic computer vision application and belongs to the supervised learning category. For example, we can determine the category to which an image (such as an image of a cat, a dog, an airplane, or a car) belongs. This application describes how to use the ResNet50 network to classify flower datasets.

What kind of flower is this?



Environment Setup

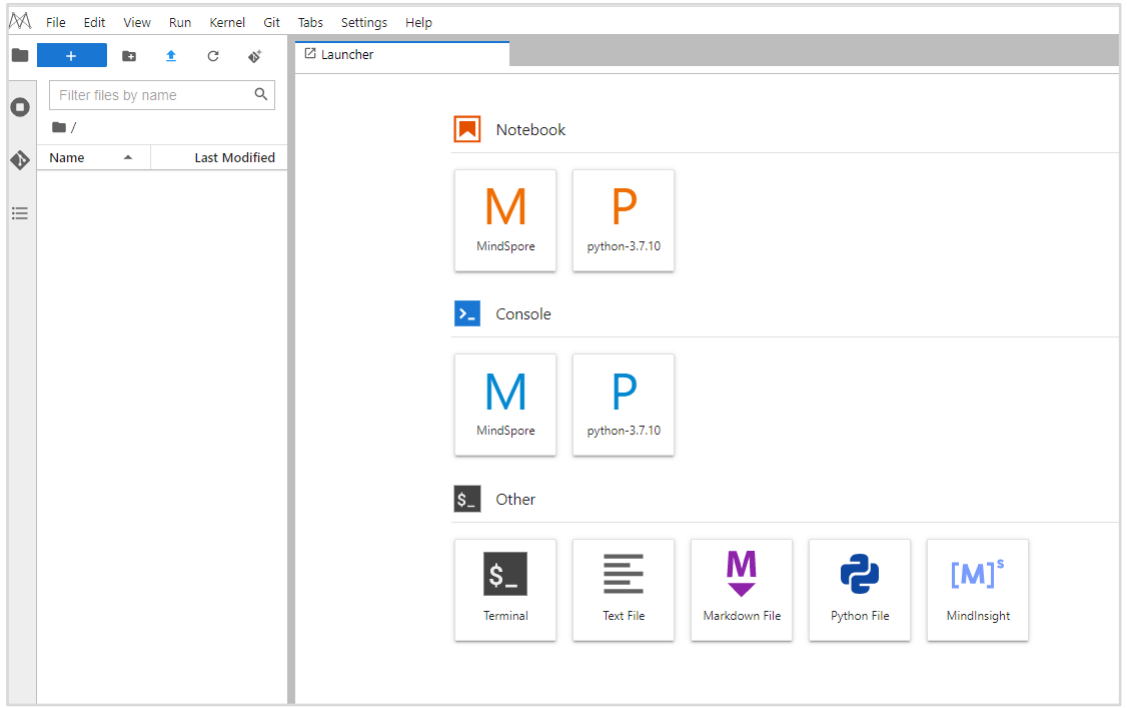
- Select the specifications of your operating environment to obtain the installation commands.
  - For example, install MindSpore 1.7.1 in Linux or Python 3.7 in pip mode:



https://www.mindspore.cn/install

Cloud Environment (optional)

- If no environment is available, you can use Huawei Cloud ModelArts for development, model training, and deployment.
  - ModelArts **helps you quickly create and deploy models and manage the AI development lifecycle**.



Data Preparation

- Photos of five kinds of flowers (open-source data) is used for training: daisy (633 photos), dandelion (898 photos), roses (641 photos), sunflowers (699 photos), tulips (799 photos). These 3670 photos are respectively saved in 5 folders. The file structure of the photos contains two parts: flower_photos_train and flower_photos_test.

```
flower_photos_train
    ├── daisy
    ├── dandelion
    ├── roses
    ├── sunflowers
    ├── tulips
    └── LICENSE.txt
flower_photos_test
    ├── daisy
    ├── dandelion
    ├── roses
    ├── sunflowers
    ├── tulips
    └── LICENSE.txt
```

File structure

Data Preprocessing

- To improve the model accuracy and ensure the model generalization capability, operations such as data enhancement and standardization are performed before the data is used to train the model.
- You can load data sets by defining the read_dataset function, which has the following functions:
  - Read the dataset.
  - Define parameters required for data augmentation and processing.
  - Generate corresponding data augmentation operations according to the parameters.
  - Use the map function to apply data operations to the dataset.
  - Process the generated dataset.
  - Display the processed data as an example.

APIs For Dataset Processing

- **mindspore.dataset** is used to load and process various common datasets.
- **mindspore.dataset.vision** is used to enhance image data.

```
from mindspore.dataset.vision import c_transforms as vision
vision.Normalize(mean, std) # Normalize the input image with the mean and standard deviation.
# std/mean The list and tuple of image composed of the mean and standard deviation. The mean and standard deviation should range from 0.0 to 255.0.
vision.HWC2CHW() # Convert the shape of the input image from <H, W, C> to <C, H, W>. The channel of input image should be 3.
vision.CenterCrop(size) # Crop the central area of the input image. If the size of the input image is less than that of the output image, the edge of the input image is padded with 0 pixel before cropping.
```

- **mindspore.dataset.text** is used to enhance text data.
- **mindspore.dataset.audio** is used to enhance audio data.
- **mindspore.dataset.transforms** is the general data enhancement module.

Code Implementation for Data Preprocessing

```
def read_data(path,config,usage="train"):
    dataset = ds.ImageFolderDataset(path,class_indexing={'daisy':0,'dandelion':1,'roses':2,'sunflowers':3,'tulips':4})
    decode_op = vision.Decode() # Operator for image decoding
    normalize_op = vision.Normalize(mean=[cfg._R_MEAN, cfg._G_MEAN, cfg._B_MEAN], std=[cfg._R_STD, cfg._G_STD, cfg._B_STD]) # Operator for
image normalization
    resize_op = vision.Resize(cfg._RESIZE_SIDE_MIN) # Operator for image resizing
    center_crop_op = vision.CenterCrop((cfg.HEIGHT, cfg.WIDTH)) # Operator for image cropping
    horizontal_flip_op = vision.RandomHorizontalFlip() # Operator for image random horizontal flipping
    channelswap_op = vision.HWC2CHW() # Operator for image channel quantity conversion
    # Operator for random image cropping, decoding, encoding, and resizing
    random_crop_decode_resize_op = vision.RandomCropDecodeResize((cfg.HEIGHT, cfg.WIDTH), (0.5, 1.0), (1.0, 1.0), max_attempts=100)
    ......
    ......
dataset = dataset.repeat(1) # Data enhancement
    dataset.map_model = 4
    return dataset
```

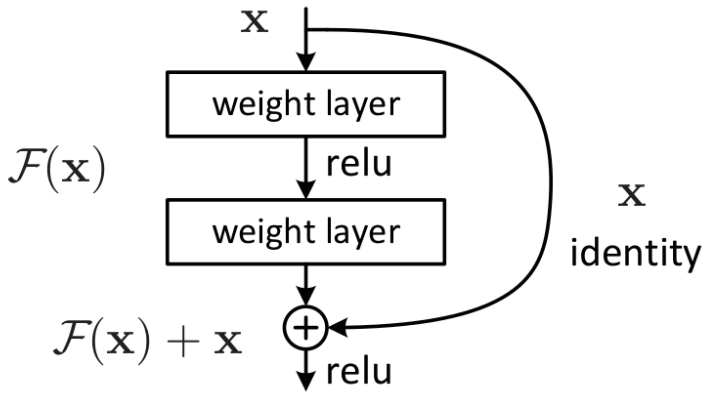
Hyperparameters Definition

- Models contain both parameters and hyperparameters. Hyperparameters enable the model to learn the optimal parameters.

```
cfg = edict({
    '_R_MEAN': 123.68, # Average value of CIFAR10
    '_G_MEAN': 116.78,
    '_B_MEAN': 103.94,
    '_R_STD': 1, # Customized standard deviation
    '_G_STD': 1,
    '_B_STD':1,
    '_RESIZE_SIDE_MIN': 256, # Minimum resize value for image enhancement
    '_RESIZE_SIDE_MAX': 512,

    'batch_size': 32, # Batch size
    'num_class': 5,     # Number of classes
    'epoch_size': 5,  # Number of training times
    'loss_scale_num':1024,
})
```

Introduction to ResNet

- ResNet-50 was proposed by He Kaiming of Microsoft Research in 2015 and won the 2015 ILSVRC.
- The residual network is a main tag of ResNet with which the degradation problem can be effectively alleviated and a deeper network can be designed.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Introduction to Network Structure APIs (1)

- **mindspore.nn**: the cell of neural network which provides predefined building blocks or compute units in a neural network.
  - **mindspore.nn.Cell**: the Cell class of MindSpore is the base class for building all networks and the basic unit of a network.
  - **mindspore.nn.LossBase**: specifies the base class of the loss function.
- Neural network structure APIs:
  - mindspore.nn.Dense(in_channels, out_channels, weight_init= 'normal', bias_init='zeros', has_bias=True, activation=None): specifies the fully connected layer.
    - in_channels (int): specifies the spatial dimension of the tensor input to the Dense layer.
    - out_channels (int): specifies the spatial dimension of the tensor output from the Dense layer.
    - weight_init: specifies the weight parameter initialization method.

Introduction to Network Structure APIs (2)

- mindspore.nn.Conv2d: (in_channels, out_channels, kernel_size, stride=1, pad_mode= "same", padding=0, dilation=1, group=1, has_bias=False, weight_init= "normal", bias_init="zeros", data_format="NCHW"): two-dimensional convolutional layer.
  - kernel_size: specifies the height and width of the two-dimensional convolution kernel.
  - stride: specifies the stride of the two-dimensional convolution kernel.
  - pad_mode: specifies the padding mode. The value can be **same**, **valid**, or **pad**. The default value is **same**.
- **mindspore.nn.MaxPool2d** (kernel_size=1, stride=1, pad_mode= 'valid', data_format='NCHW'): specifies the two-dimensional maximum pooling layer.

Introduction to Network Structure APIs (3)

- mindspore.nn.RNN(*args, **kwargs): specifies the recurrent neural network (RNN) layer, whose activation function is tanh and relu.
  - input_size (int): specifies the feature vector dimension input to the input layer.
  - hidden_size (int): specifies the feature vector dimension output from the hidden layer.
  - num_layers (int): specifies the number of stacking RNN layers. The default value is 1.
- mindspore.nn.Dropout(keep_prob=0.5, dtype=mstype.float32): automatically set some neurons output to 0 during training based on the dropout probability 1 – keep_prob.
  - keep_prob (float):specifies the input neuron retention rate, ranging from 0 to 1.
- mindspore.nn.ReLU: specifies the activation function of the rectified linear unit.
- mindspore.nn.Softmax (axis=-1): specifies the Softmax activation function.
  - axis: specifies the axis of the Softmax operation.

MindSpore Network Building

- When a neural network is required, you need to inherit the Cell class and overwrite the __init__ and construct methods.

```
# For details about ResNet code implementation, see the experiment guide.
class ResNet(nn.Cell):
    def __init__(self, *args, **kwargs):
        super(ResNet, self).__init__()
        ......
        Layers in the network
        ......
    def construct(self, x):
        ......
        Network structure
        ......
```

Model Training

- Start model training after data preprocessing and network building.
- There are two ways to train the model:
  - **Start training from scratch based on your own dataset**. This method applies to scenarios where the data volume is large and the training resources are robust.
  - **Start training based on the trained model and perform fine-tuning training on own dataset**. This method applies to scenarios where the data volume is small (applied in the current experiment).
    - Pre-trained model: ResNet model file trained on the ImageNet dataset.
    - Modify the parameters at the last layer of the pre-trained model parameters. (The model is pre-trained on the ImageNet dataset to classify 1001 types. However, the current experiment is to classify the five types of flowers.)
    - Training is performed based on its self-owned data set.
- Parameter tuning: During training, you can tune the hyperparameter combinations.
- For details about the code, see the experiment guide.

Introduction to Training APIs

- mindspore.Model(network,loss_fn=None, optimizer=None, metrics=None, eval_network=None, eval_indexes=None, amp_level= "O0", boost_level="O0", **kwargs): The model encapsulates instances that can be trained or inferred based on the parameters input by users.
  - network (Cell): used for training and inference of the neural network.
  - loss_fn (Cell): specifies the loss function.
  - optimizer (Cell): an optimizer used to update network weights.
  - metrics (Union[dict, set]): a set of evaluation functions used for model evaluation.

Model Saving and Loading

- After the network training is complete, save the network model as a file. There are two types of APIs for saving models:
  - One is to simply save the network model before and after training.

```
import mindspore as ms
Use the save_checkpoint provided by MindSpore to save the model, pass it to the network, and save the path.
# net indicates a defined network model, which is used before or after training.
ms.save_checkpoint(network, "./MyNet.ckpt")
```

  - You can also save the interface during network model training. MindSpore automatically saves the number of epochs and number of steps set during training. That is, the intermediate weight parameters generated during the model training process are also saved to facilitate network fine-tuning and stop training.

```
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig
# Set the value of epoch_num.
epoch_num = 5
# Set model saving parameters.
config_ck = CheckpointConfig(save_checkpoint_steps=1875, keep_checkpoint_max=10)
# Apply the model saving policy.
ckpoint = ModelCheckpoint(prefix="lenet", directory="./lenet", config=config_ck) model.train(epoch_num, dataset_train, callbacks=[ckpoint])
```

Model Loading and Prediction (1)

- Use the trained model weights to call model.predict() to test the test data.
  - After the training is complete, call model.predict for prediction.
  - Perform prediction after the weight file is loaded.
    - To load the model weight, you need to create an instance of the same model and then use the load_checkpoint and load_param_into_net methods to load parameters.

Model Loading and Prediction (2)

- The load_checkpoint method loads the network parameters in the parameter file to the param_dict dictionary.
- The load_param_into_net method loads the parameters in the param_dict dictionary to the network or optimizer. After the loading, parameters in the network are stored by the checkpoint.

```
from mindspore import load_checkpoint, load_param_into_net
# Save the model parameters to the parameter dictionary. The model parameters saved during the training are loaded.
 param_dict = load_checkpoint("./ResNet_1875.ckpt")
# Redefine a ResNet neural network.
net = ResNet (num_classes=5, pretrained=False)
# Load parameters to the network. load_param_into_net(net, param_dict)
# Redefine the optimizer function.
net_opt = nn.Momentum(net.trainable_params(), learning_rate=0.01, momentum=0.9)
model = Model(net, loss_fn=net_loss, optimizer=net_opt, metrics={"accuracy"})
```

Model Deployment

- Deployment on device (using mobile phones as an example):
  - Convert the CKPT file to the MindIR file format, and then convert MindIR file format to the MindSpore Lite recognizable file on the Android phone (MS model file).
  - Deploy the app APK on the mobile phone, that is, download a MindSpore Vision suite Android APK.
  - Import the MS model file to the mobile phone.
- Deployment on cloud:
  - Deploy the model in cloud services through MindSpore Serving.
  - Quickly deploy model on the cloud using ModelArts.
- Deployment on edge:
  - Deploy the model based on the AscendCL and Atlas computing platforms.

Summary

- This course describes the knowledge related to the AI development framework, including the basic knowledge of the common AI framework, MindSpore framework basics, and AI application development process.

Recommendations

- Official MindSpore website
  - https://www.mindspore.cn/

Thank you.

Bring digital to every person, home, and organization for a fully connected, intelligent world.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.