Huawei MindSpore AI Development Framework

Foreword

- This chapter introduces the structure, design concept, and features of MindSpore based on the issues and difficulties facing by the AI computing framework, and describes the development and application process in MindSpore.

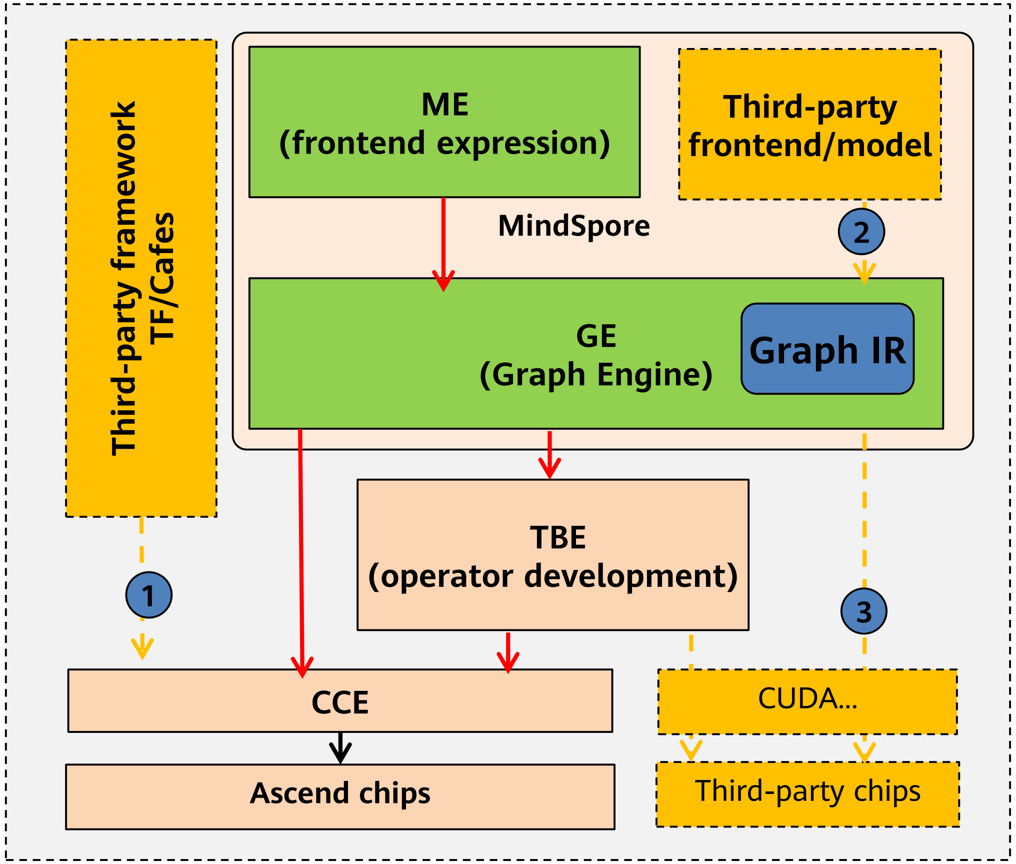
Objectives

Upon completion of this course, you will be able to:

- Learn what MindSpore is
- Understand the framework of MindSpore
- Understand the design concept of MindSpore
- Learn features of MindSpore
- Grasp the environment setup process and development cases

Contents

Architecture: Easy Development and Efficient Execution



ME (Mind Expression): interface layer (Python)

Usability: automatic differential programming and original mathematical expression

- Auto diff: operator-level automatic differential
- Auto parallel: automatic parallelism
- Auto tensor: automatic generation of operators
- Semi-auto labeling: semi-automatic data labeling

GE (Graph Engine): graph compilation and execution layer

High performance: software/hardware co-optimization, and full-scenario application

- Cross-layer memory overcommitment
- Deep graph optimization
- On-device execution
- Device-edge-cloud synergy (including online compilation)

① Equivalent to open-source frameworks in the industry, MindSpore preferentially serves self-developed chips and cloud services.

② It supports upward interconnection with third-party frameworks and can interconnect with third-party ecosystems through Graph IR, including training frontends and inference models. Developers can expand the capability of MindSpore.

③ It also supports interconnection with third-party chips and helps developers increase MindSpore application scenarios and expand the AI ecosystem.

Overall Solution: Core Architecture

MindSpore

- Unified APIs for all scenarios
- Auto differ | Auto parallelism | Auto tuning
- MindSpore intermediate representation (IR) for computational graph
- On-device execution | Pipeline parallelism | Deep graph optimization
- Device-edge-cloud co-distributed architecture (deployment, scheduling, communications, etc.)

Easy development:
AI Algorithm As Code

Efficient execution:
Optimized for Ascend
GPU support

Flexible deployment: on-demand cooperation across all scenarios

Processors: Ascend, GPU, and CPU

HUAWEI

MindSpore Design: Auto Differ

Technical path of automatic differential

Graph: TensorFlow

- Non-Python programming based on graphs
- Complex representation of control flows and higher-order derivatives

Operator overloading: PyTorch

- Runtime overhead
- Backward process performance is difficult to optimize.

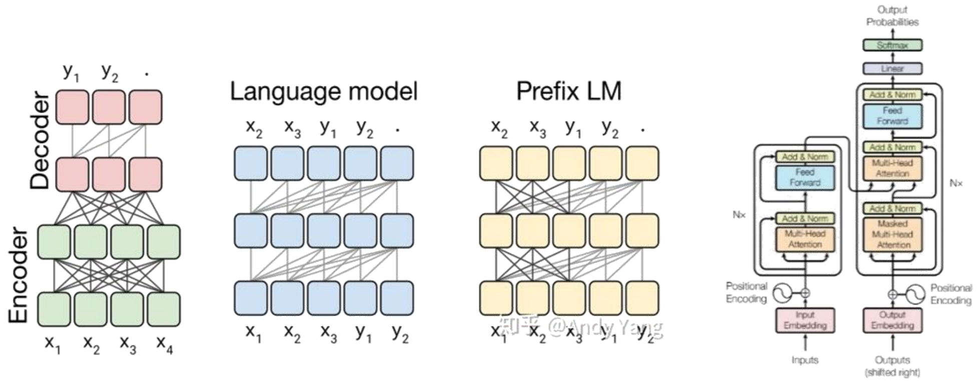
Source code transfer: MindSpore

- Python APIs for higher efficiency
- IR-based compilation optimization for better performance

Auto Parallelism

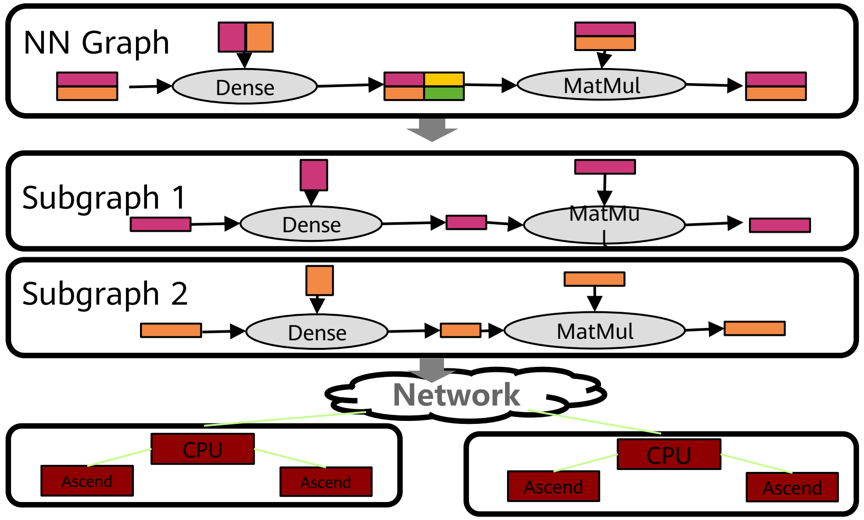
Challenges

Ultra-large models realize efficient distributed training: As NLP-domain models swell, the memory overhead for training ultra-large models such as Bert (340M)/GPT-2(1542M) has exceeded the capacity of a single card. Therefore, the models need to be split into multiple cards before execution. Manual model parallelism is **used currently**. Model segmentation needs to be designed and the cluster topology needs to be understood. The development is extremely challenging. The performance is lackluster and can be hardly optimized.



Key Technologies

Automatic graph segmentation: It can segment the entire graph based on the input and output data dimensions of the operator, and integrate the data and model parallelism. Cluster topology awareness scheduling: It can perceive the cluster topology, schedule subgraphs automatically, and minimize the communication overhead.



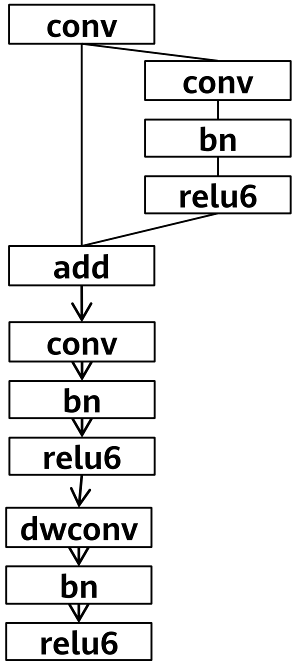
Effect: Realize model parallelism based on the existing single-node code logic, improving the development efficiency tenfold compared with manual parallelism.

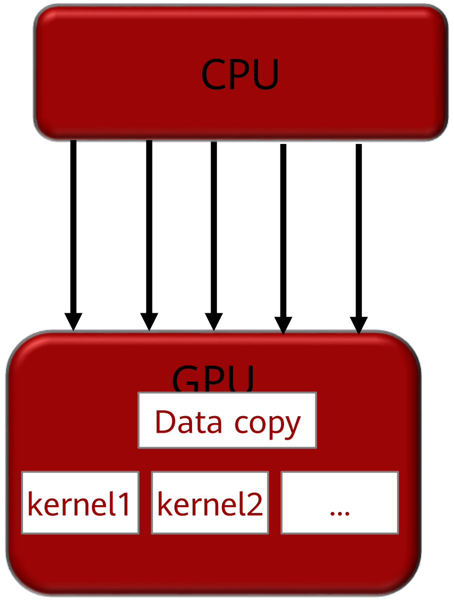
On-Device Execution (1)

Challenges

Challenges for model execution with supreme chip computing power:

Memory wall, high interaction overhead, and data supply difficulty. Partial operations are performed on the host, while the others are performed on the device. The interaction overhead is much greater than the execution overhead, resulting in the low accelerator usage.

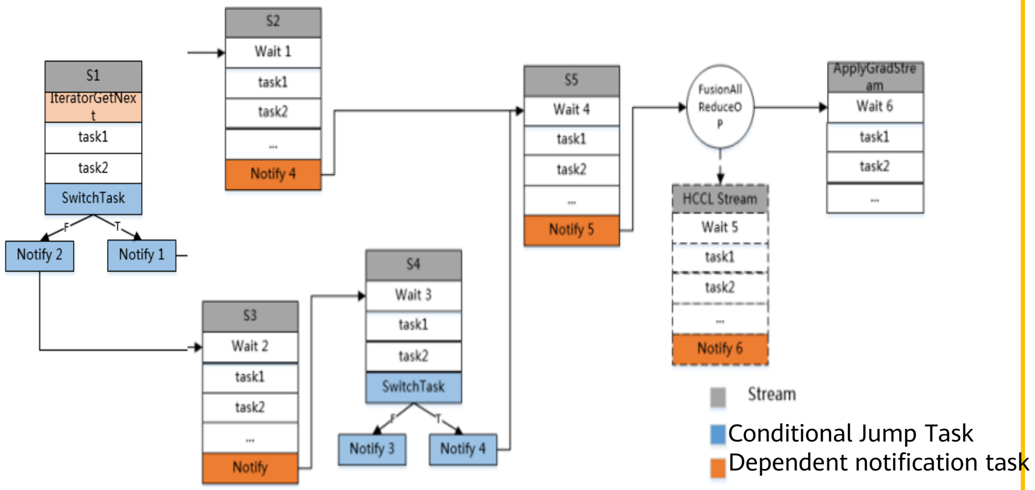




Large data interaction overhead and difficult data supply

Key Technologies

Chip-oriented deep graph optimization reduces the synchronization waiting time and maximizes the parallelism of data, computing, and communication. Data pre-processing and computation are integrated into the Ascend chip:

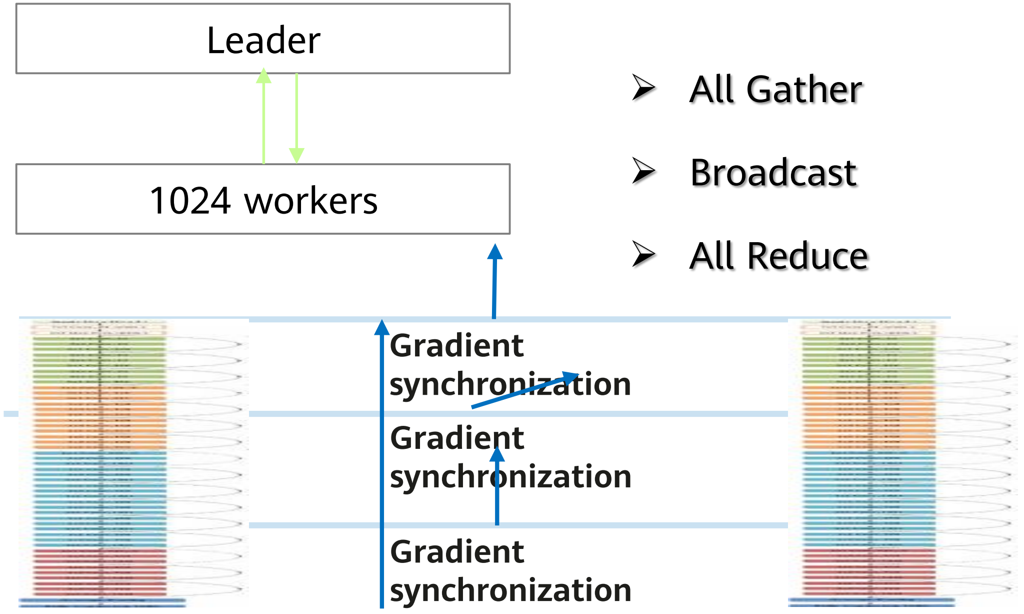


Effect: Elevate the training performance tenfold compared with the on-host graph scheduling.

On-Device Execution (2)

Challenges

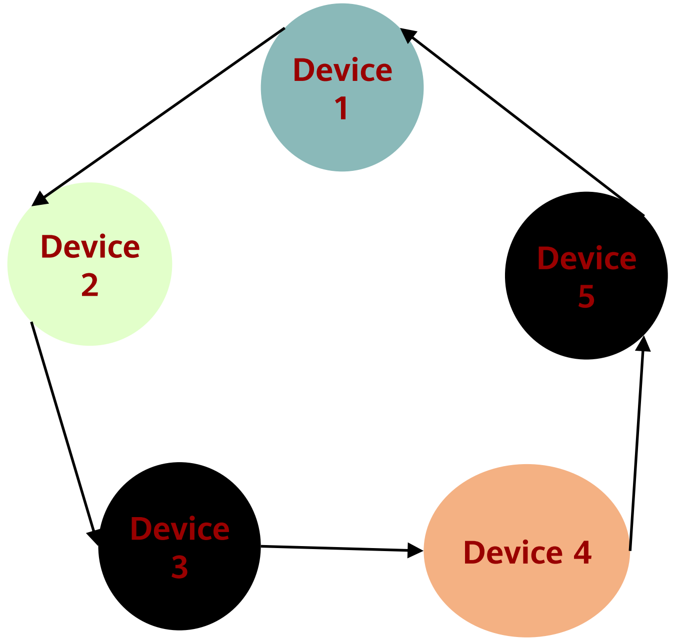
Challenges for distributed gradient aggregation with supreme chip computing power:
the synchronization overhead of central control and the communication overhead of frequent synchronization of ResNet50 under the single iteration of 20 ms; the traditional method can only complete All Reduce after three times of synchronization, while the data-driven method can autonomously perform All Reduce without causing control overhead.



- All Gather
- Broadcast
- All Reduce

Key Technologies

The optimization of the **adaptive graph segmentation driven by gradient data** can realize decentralized All Reduce and synchronize gradient aggregation, boosting computing and communication efficiency.



Effect: a smearing overhead of less than 2 ms

Distributed Device-Edge-Cloud Synergy Architecture

Challenges

The diversity of hardware architectures leads to full-scenario deployment differences and performance uncertainties. The separation of training and inference leads to isolation of models.

Key Technologies

- **Unified model IR** delivers a consistent deployment experience.
- **The graph optimization technology featuring software and hardware collaboration** bridges different scenarios.
- Device-cloud Synergy Federal Meta Learning breaks the device-cloud boundary and updates the multi-device collaboration model in real time.

Effect: consistent model deployment performance across all scenarios thanks to the unified architecture, and improved precision of personalized models

On-demand collaboration in all scenarios and consistent development experience

Device

Edge

Cloud

Contents

AI Computing Framework: Challenges

Industry Challenges

A huge gap between

industry research and all-scenario AI application

- High entry barriers
- High execution cost
- Long deployment duration

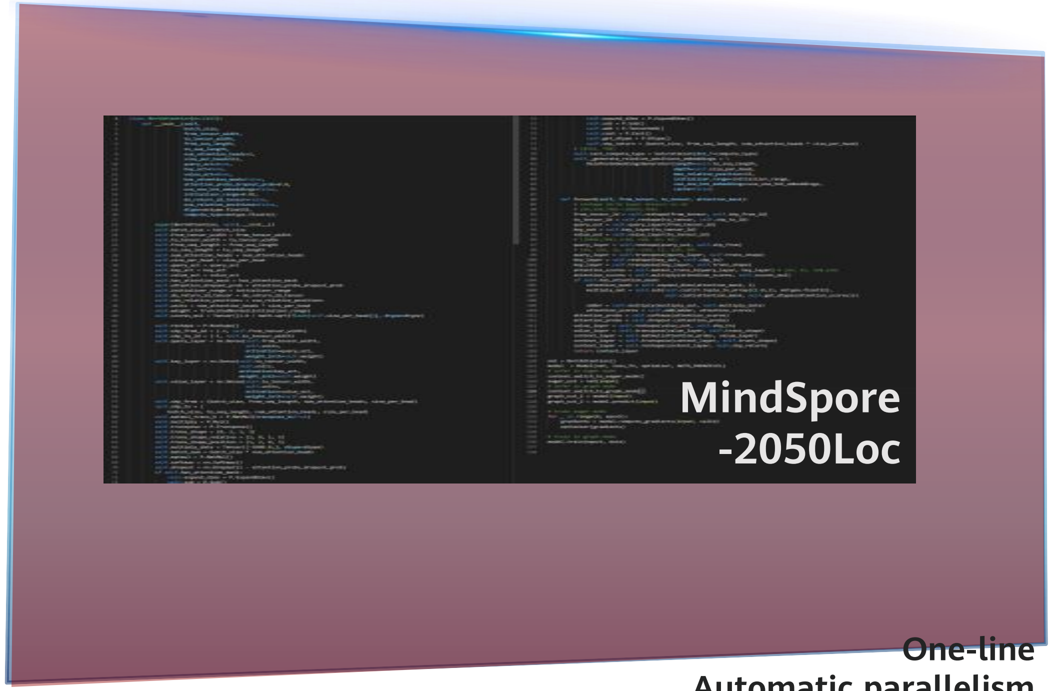
Technological Innovation

MindSpore facilitates inclusive AI across applications

- New programming mode
- New execution mode
- New collaboration mode

New Programming Paradigm

Algorithm scientist



Algorithm scientist
+
Experienced system developer

One-line Automatic parallelism

Efficient automatic differential

One-line debug-mode switch

NLP Model: Transformer

Code Example

TensorFlow code snippet: XX lines, manual parallelism

```
import tensorflow as tf

model() {
    with tf.device("/device:0")
        token_type_table = tf.get_variable(
            name=token_type_embedding_name,
        shape=[token_type_vocab_size, width],
        initializer=create_initializer(initializer_range))
        flat_token_type_ids = tf.reshape(token_type_ids, [-1])
        one_hot_ids = tf.one_hot(flat_token_type_ids, depth=token_type_vocab_size)
        token_type_embeddings = tf.matmul(one_hot_ids, token_type_table)

    with tf.device("/device:1")
        query_layer = tf.layers.dense(
            from_tensor_2d,
            num_attention_heads * size_per_head,
            activation=query_act,
            name="query",
            kernel_initializer=create_initializer(initializer_range))

    with tf.device("/device:2")
        key_layer = tf.layers.dense(
            to_tensor_2d,
            num_attention_heads * size_per_head,
            activation=key_act,
            name="key",
            kernel_initializer=create_initializer(initializer_range))
```

MindSpore code snippet: two lines, automatic parallelism

```
class DenseMatMulNet(nn.Cell):
    def __init__(self):
        super(DenseMutMulNet, self).__init__()
        self.matmul1 = ops.MatMul.set_strategy({[4, 1], [1, 1]})
        self.matmul2 = ops.MatMul.set_strategy({[1, 1], [1, 4]})
    def construct(self, x, w, v):
        y = self.matmul1(x, w)
        z = self.matmul2(y, v)
        return s
```

Typical scenarios: ReID

New Execution Mode (1)

Execution Challenges

Complex AI computing and diverse computing units

1. CPU cores, cubes, and vectors
2. Scalar, vector, and tensor computing
3. Mixed precision computing
4. Dense matrix and sparse matrix computing

Multi-device execution: High cost of parallel control

Performance cannot linearly increase as the node quantity increases.

On-device execution

Offloads graphs to devices, maximizing the computing power of Ascend

Framework optimization

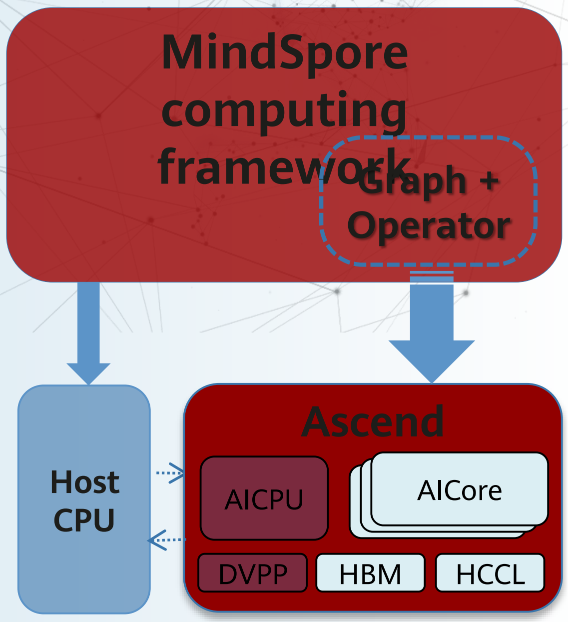
Pipeline parallelism

Cross-layer memory overcommitment

Soft/hardware co-optimization

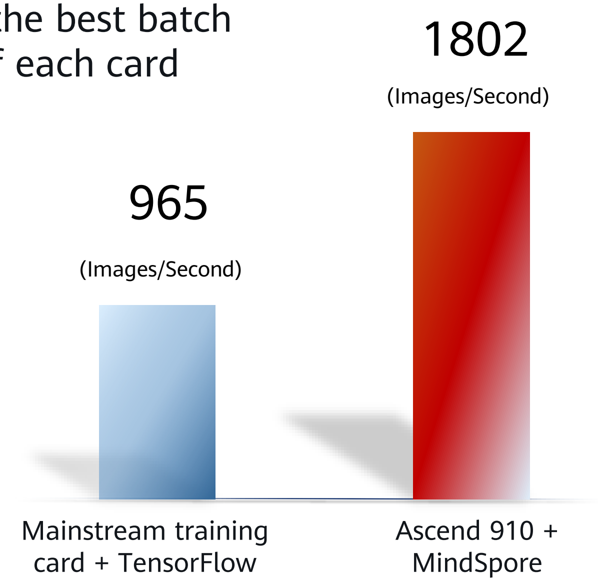
On-device execution

Deep graph optimization



New Execution Mode (2)

- ResNet 50 V1.5
- ImageNet 2012
- With the best batch size of each card



Performance of ResNet-50 is doubled.
Single iteration:
58 ms (other frameworks+V100) v.s. about **22 ms** (MindSpore) (ResNet50+ImageNet, single-server, eight-device, batch size=32)

Detecting objects in 60 ms

Tracking objects in 5 ms

Multi-object real-time recognition
MindSpore-based mobile deployment, a smooth experience of multi-object detection

New Collaboration Mode

Deployment Challenge

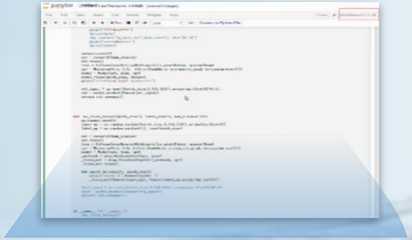
v.s.

- Varied requirements, objectives, and constraints for device, edge, and cloud application scenarios

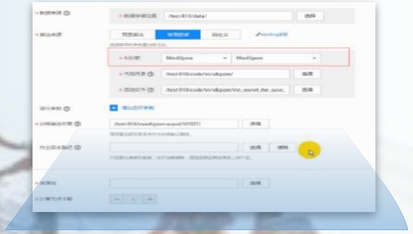
v.s.

- Different hardware precision and speed

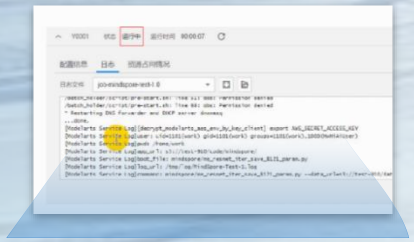
Unified development; flexible deployment; on-demand collaboration, and high security and reliability



Development



Deployment



Execution

Saving model

Unified development and flexible deployment

High Performance

AI computing challenges

Complex computing

Scalar, vector, and tensor computing

Mixed precision computing

Parallelism between gradient aggregation and mini-batch computing

Diverse computing units / processors

CPUs, GPUs, and Ascend processors

Scalar, vector, and tensor

MindSpore

Framework optimization

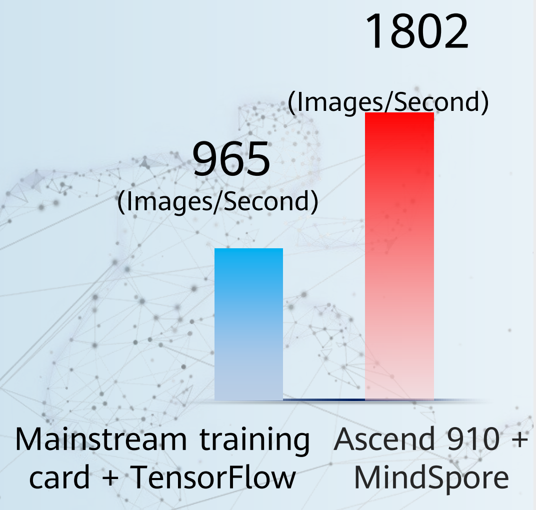
Pipeline parallelism

Cross-layer memory overcommitment

Software + hardware co-optimization

On-device execution

Deep graph optimization



- ResNet 50 V1.5
- ImageNet 2012
- Based on optimal batch sizes

HUAWEI

Vision and Value

Efficient development

Profound expertise required
Algorithms
Programming

MindSpore

Flexible deployment

Long deployment duration
Unified development
flexible deployment

Outstanding performance

Diverse computing units and models
CPU + NPU
Graph + Matrix

Contents

Installing MindSpore

Environment Requirements

System Requirements and Software Dependencies

Version	Operating System	Executable File Installation Dependencies	Source Code Compilation and Installation Dependencies
MindInsight 0.2.0-alpha	- Ubuntu 16.04 or later x86_64 - EulerOS 2.8 arrch64 - EulerOS 2.5 x86_64	- Python 3.7.5 - MindSpore 0.2.0-alpha - For details about other dependency items, see requirements.txt.	**Compilation dependencies:** - Python 3.7.5 - CMake >= 3.14.1 - GCC 7.3.0 - node.js >= 10.19.0 - wheel >= 0.32.0 - pybind11 >= 2.4.3 **Installation dependencies:** same as the executable file installation dependencies.

- When the network is connected, dependency items in the requirements.txt file are automatically downloaded during .whl package installation. In other cases, you need to manually install dependency items.

Installation Guide

Installing Using Executable Files

1. Download the .whl package from the MindSpore website. It is recommended to perform SHA-256 integrity verification first and run the following command to install MindInsight:

```
pip install mindinsight-{version}-cp37-cp37m-linux_{arch}.whl
```

2. Run the following command. If `web address: http://127.0.0.1:8080` is displayed, the installation is successful.

```
mindinsight start
```

Method 1: source code compilation and installation

Two installation environments: Ascend and CPU

```
adding 'mindspore/transforms/validators.py'
adding 'mindspore-0.1.0.dist-info/METADATA'
adding 'mindspore-0.1.0.dist-info/WHEEL'
adding 'mindspore-0.1.0.dist-info/top_level.txt'
adding 'mindspore-0.1.0.dist-info/RECORD'
removing build/bdist.linux-x86_64/wheel
------Successfully created mindspore package------
--------------- mindspore: build test end ---------------
```

Method 2: direct installation using the installation package

Two installation environments: Ascend and CPU

Installation commands:

1. pip install –y mindspore-cpu
2. pip install –y mindspore-d

Getting Started

- In MindSpore, data is stored in tensors. Common tensor operations:
  - asnumpy()
  - size()
  - dim()
  - dtype()
  - set_dtype()
  - tensor_add(other: Tensor)
  - tensor_mul(other: Tensor)
  - shape()
  - __Str__# (conversion into strings)

Module	Description
model_zoo	Defines common network models
communication	Data loading module, which defines the dataloader and dataset and processes data such as images and texts.
dataset	Dataset processing module, which reads and pro-processes data.
common	Defines tensor, parameter, dtype, and initializer.
context	Defines the context class and sets model running parameters, such as graph and PyNative switching modes.
akg	Automatic differential and custom operator library.
nn	Defines MindSpore cells (neural network units), loss functions, and optimizers.
ops	Defines basic operators and registers reverse operators.
train	Training model and summary function modules.
utils	Utilities, which verify parameters. This parameter is used in the framework.

Components of ME

Programming Concept: Operation

Softmax operator

```
class Softmax(PrimitiveWithInfer):
    r"""
    Returns A tensor of the same shape of input.

    This op will do softmax operation on the specified axis. Suppose a slice along the given
    aixs is :math:`x` then for each element :math:`x_i` softmax function is as follows:

    .. math::
        \text{output}(x_i) = \frac{exp(x_i)}{\sum_{j = 0}^{N-1}\exp(x_j)},

    where :math:`N` is the length of the Tensor.

    Args:
        axis (Union[int, tuple]): The axis to do softmax operation. Default: -1.
    """

    @prim_attr_register
    def __init__(self, axis=-1):
        """init softmax layer"""
        self.init_prim_io_names(inputs=['x'], outputs=['output'])
        validator.check_type("axis", axis, [int, tuple])
        if isinstance(axis, int):
            self.add_prim_attr('axis', (axis,))
        for item in self.axis:
            validator.check_type("item of axis", item, [int])

    def infer_shape(self, x_shape):
        return x_shape

    def infer_dtype(self, x_dtype):
        return x_dtype
```

1. Name
2. Base class
3. Comment
4. Attributes of the operator are initialized here
5. The shape of the output tensor can be derived based on the input parameter shape of the operator.
6. The data type of the output tensor can be derived based on the data type of the input parameters.

Common operations in MindSpore:

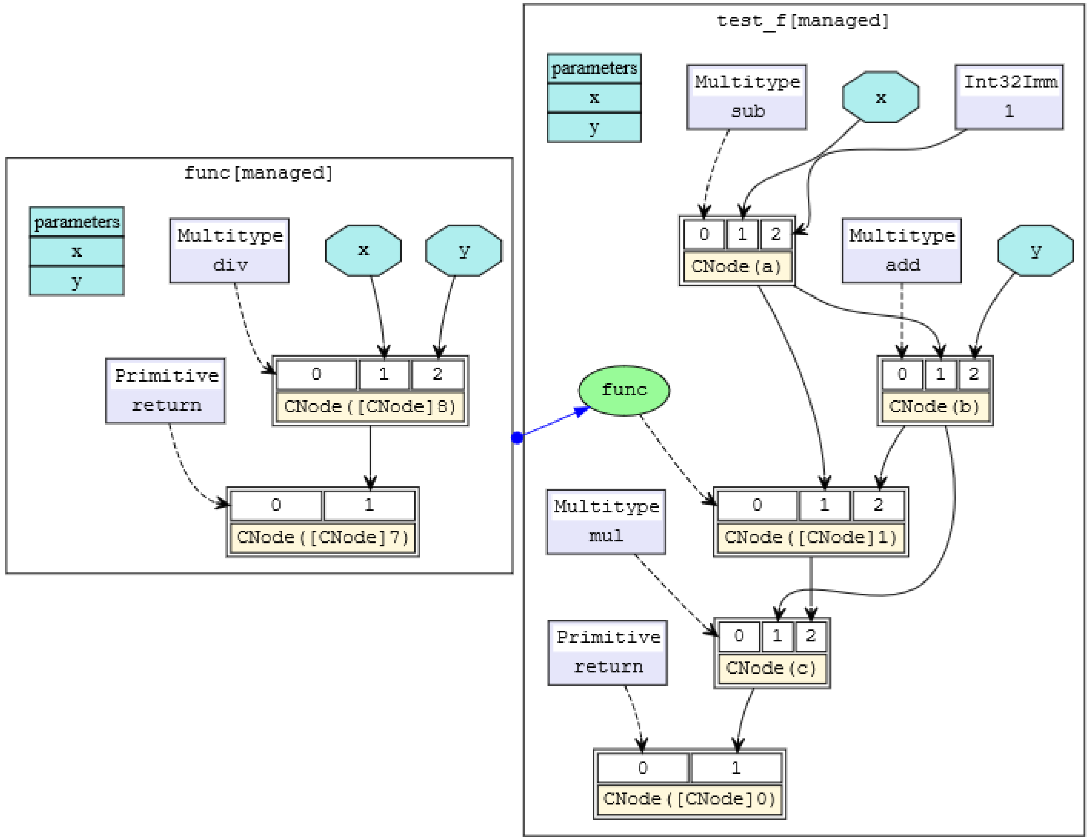
- **array: Array-related operators**
  - ExpandDims
  - Squeeze
  - Concat
  - OnesLike
  - Select
  - StridedSlice
  - ScatterNd…
- **math: Math-related operators**
  - AddN
  - Cos
  - Sub
  - Sin
  - Mul
  - LogicalAnd
  - MatMul
  - LogicalNot
  - RealDiv
  - Less
  - ReduceMean
  - Greater…
- **nn: Network operators**
  - Conv2D
  - MaxPool
  - Flatten
  - AvgPool
  - Softmax
  - TopK
  - ReLU
  - SoftmaxCrossEntropy
  - Sigmoid
  - SmoothL1Loss
  - Pooling
  - SGD
  - BatchNorm
  - SigmoidCrossEntropy…
- **control: Control operators**
  - ControlDepend
- **random: Random operators**

Programming Concept: Cell

- A cell defines the basic module for calculation. The objects of the cell can be directly executed.
  - __init__: It initializes and verifies modules such as parameters, cells, and primitives.
  - Construct: It defines the execution process. In graph mode, a graph is compiled for execution and is subject to specific syntax restrictions.
  - bprop (optional): It is the reverse direction of customized modules. If this function is undefined, automatic differential is used to calculate the reverse of the construct part.
- Cells predefined in MindSpore mainly include: common loss (Softmax Cross Entropy With Logits and MSELoss), common optimizers (Momentum, SGD, and Adam), and common network packaging functions, such as TrainOneStepCell network gradient calculation and update, and WithGradCell gradient calculation.

Programming Concept: MindSporeIR

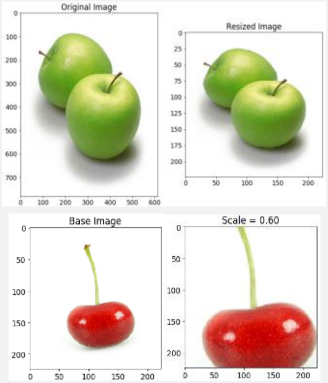
- MindSporeIR is a compact, efficient, and flexible graph-based functional IR that can represent functional semantics such as free variables, high-order functions, and recursion. It is a program carrier in the process of AD and compilation optimization.
- Each graph represents a function definition graph and consists of ParameterNode, ValueNode, and ComplexNode (CNode).
- The figure shows the def-use relationship.



Development Case

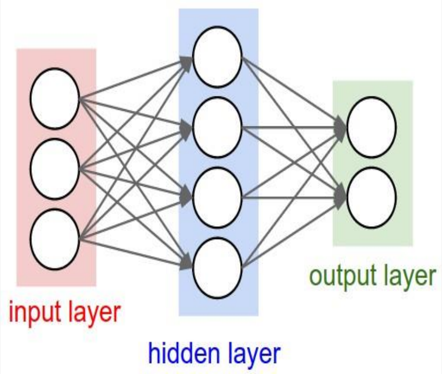
- Let’s take the recognition of MNIST handwritten digits as an example to demonstrate the modeling process in MindSpore.

Data



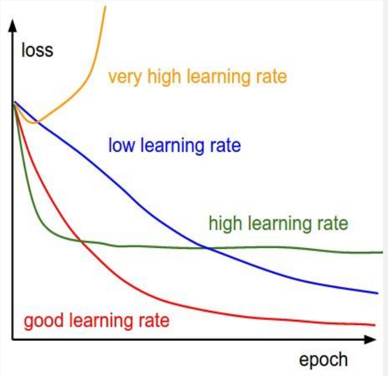
- **1. Data loading**
- **2. Data enhancement**

Network



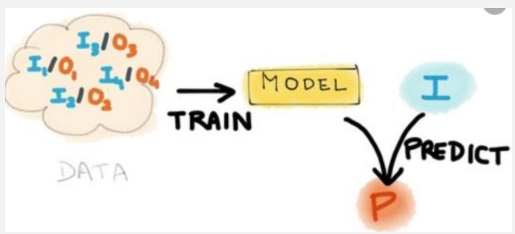
- **3. Network definition**
- **4. Weight initialization**
- **5. Network execution**

Model



- **6. Loss function**
- **7. Optimizer**
- **8. Training iteration**
- **9. Model evaluation**

Application



- **10. Model saving**
- **11. Load prediction**
- **12. Fine tuning**

Summary

- This chapter describes the framework, design, features, and the environment setup process and development procedure of MindSpore.