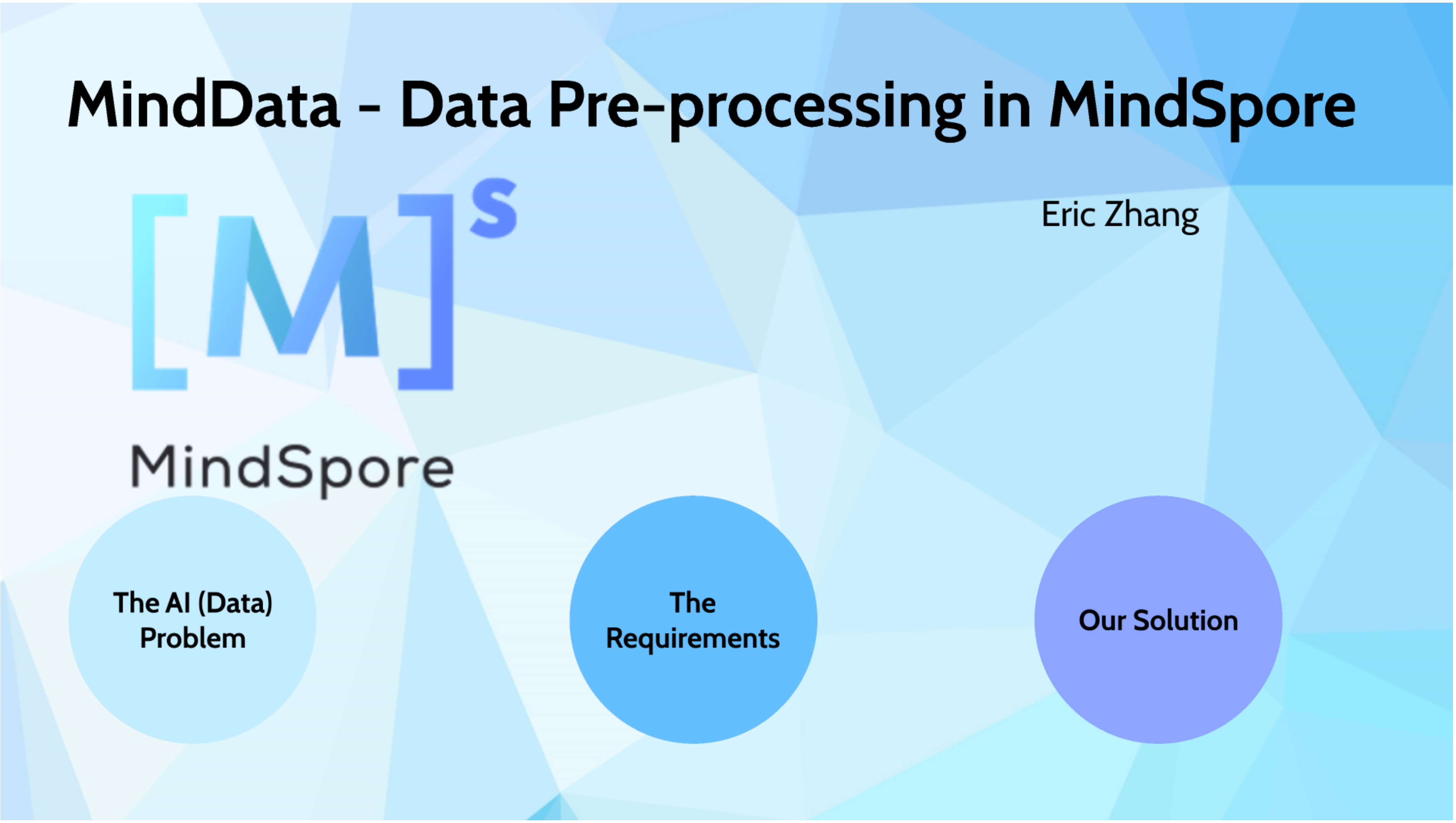

MindData - Data Pre-processing in MindSpore
Eric Zhang
MindSpore
The AI (Data) Problem
The Requirements
Our Solution

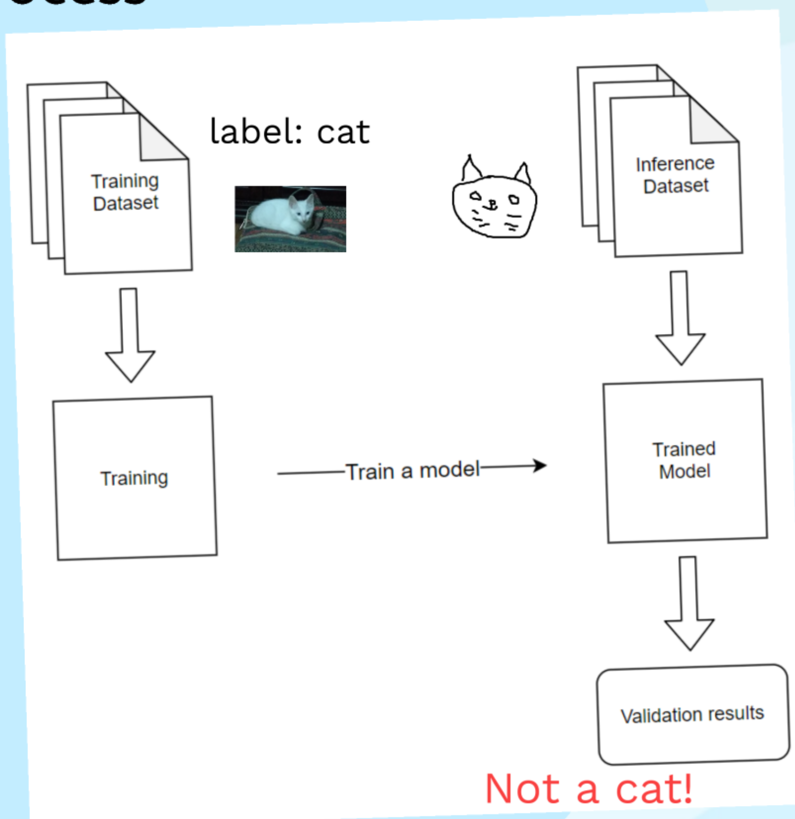
Typical training process

1. You take some data
2. Feed it to a model
3. Training adjusts the model
4. You end up with a model

You would have to continuously send data through the training process to get good training accuracy.

Training is computation intense
Training can take a long time

There are other pain points and challenges that aren't mentioned here...



Training too slow

Accuracy too low

API too complicated

Training can be very slow - Research today explores various ways of accelerating this process

Why is training slow?

-> Training happens over many **epochs**, one epoch represents traversal over the entire **dataset**

-> Data processing **operators** are computation intensive (image decoding and Gaussian calculations)

-> Model will have to be updated with results periodically, causing communication overhead

1. Hardware acceleration
2. Computation graph optimization
3. Minimize inter-device communication

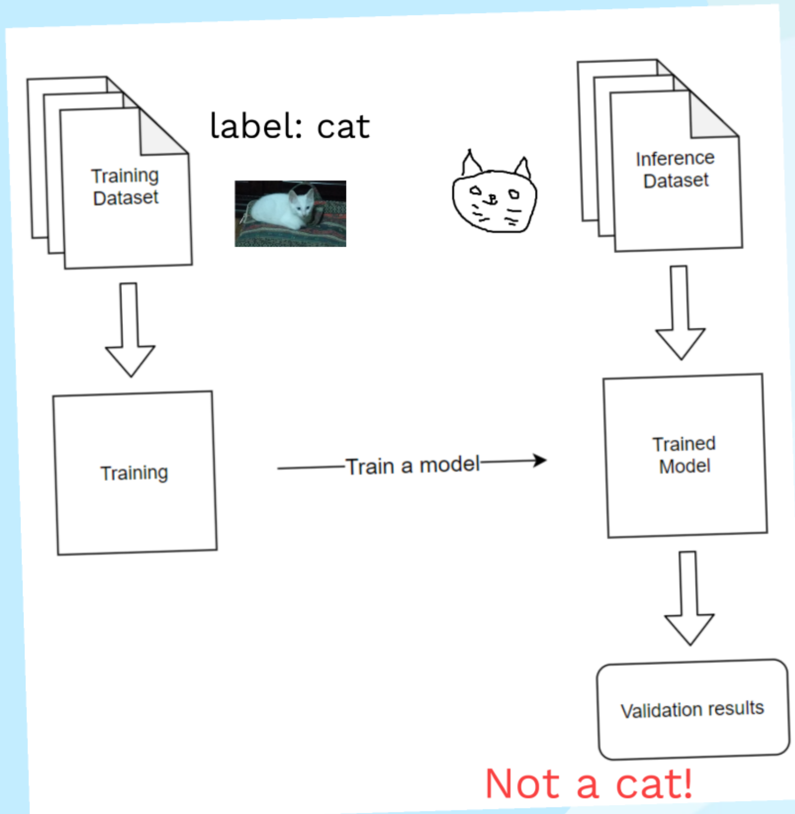
Typical training process

1. You take some data
2. Feed it to a model
3. Training adjusts the model
4. You end up with a model

You would have to continuously send data through the training process to get good training accuracy.

Training is computation intense
Training can take a long time

There are other pain points and challenges that aren't mentioned here...



Training too slow

Accuracy too low

API too complicated

Insufficient accuracy

Machine learning is an area where we still don't fully understand

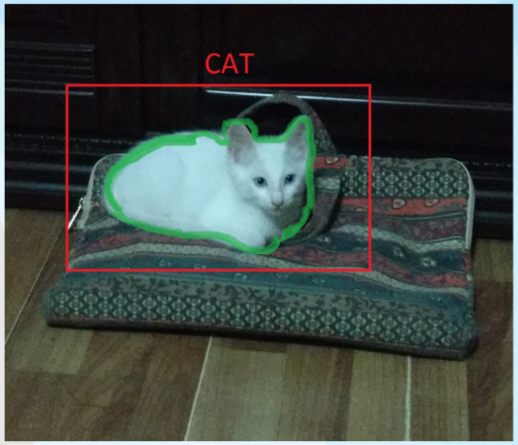
-> We don't know exactly what the machine is picking up, it could be picking out other details (edge artifacts for example)

-> As such the data goes through a series of transformations to allow machines to pick up the key features more easily (crop, re-size, normalize)

-> When tracking down the root reason of lost accuracy the list is endless

- bad interpolation mode (image pre-processing)
- bad model
- not enough epochs or bad training data set

...



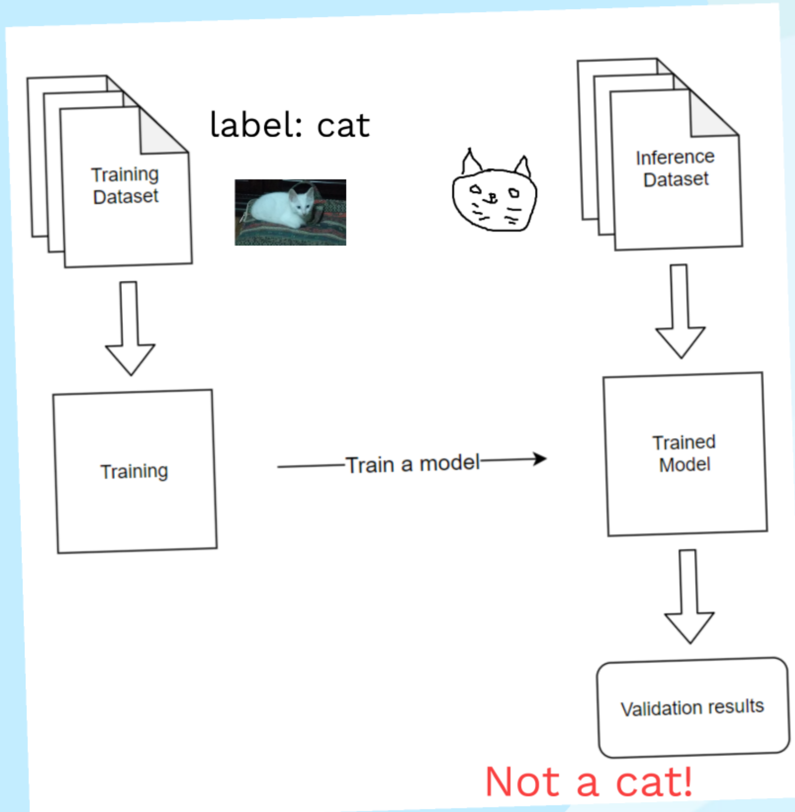
Typical training process

1. You take some data
2. Feed it to a model
3. Training adjusts the model
4. You end up with a model

You would have to continuously send data through the training process to get good training accuracy.

Training is computation intense
Training can take a long time

There are other pain points and challenges that aren't mentioned here...



Training too slow

Accuracy too low

API too complicated

API is too complicated...

There are many operators in the wild and many implementations. How many details does the user really need to control?

-> Users of machine learning comes from all sorts of backgrounds

-> Most of the time we don't want to give the user too many options even though we need different options for different work loads

```
MyResize(interpolation,
is_sparse,
preserve_ratio,
include_bounding_box,
fill_value,
is_random)
```

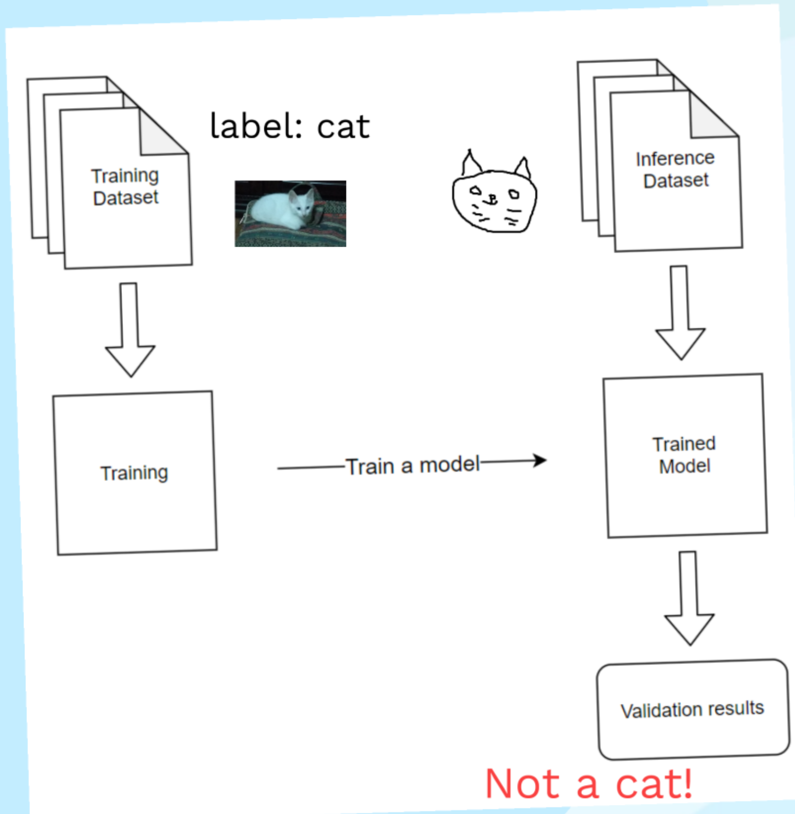
Typical training process

1. You take some data
2. Feed it to a model
3. Training adjusts the model
4. You end up with a model

You would have to continuously send data through the training process to get good training accuracy.

Training is computation intense
Training can take a long time

There are other pain points and challenges that aren't mentioned here...



Training too slow

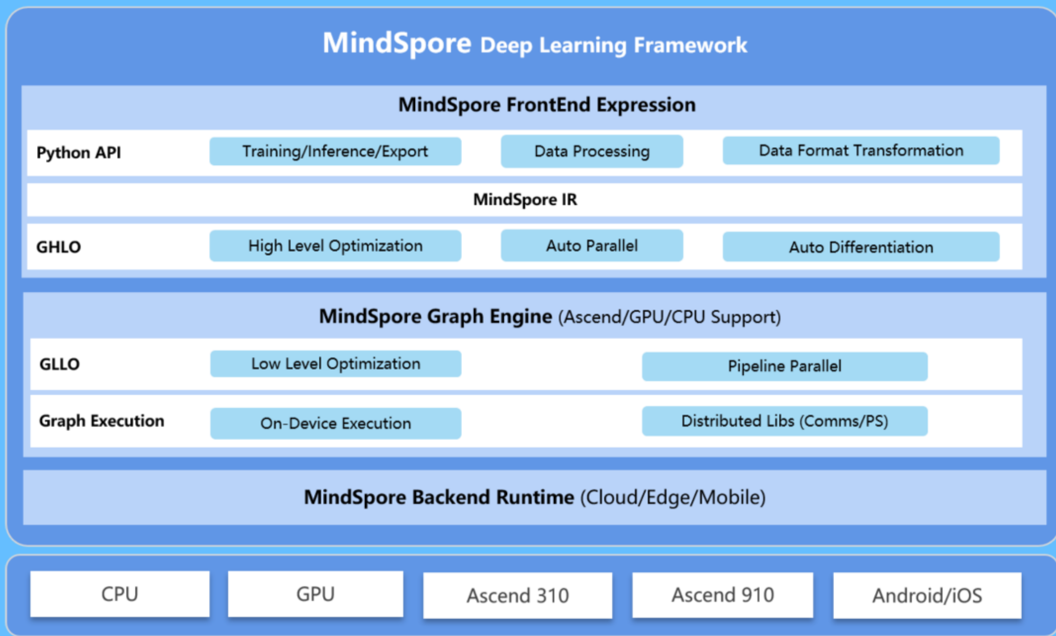
Accuracy too low

API too complicated

MindData - Data Pre-processing in MindSpore

Eric Zhang

- The AI (Data) Problem
- The Requirements
- Our Solution



In designing MindSpore the requirements are quite different between processing data and training. We will mostly focus on solving the problem of training being slow in this presentation

Two execution graphs

Sending data to different devices

Sharing dependencies

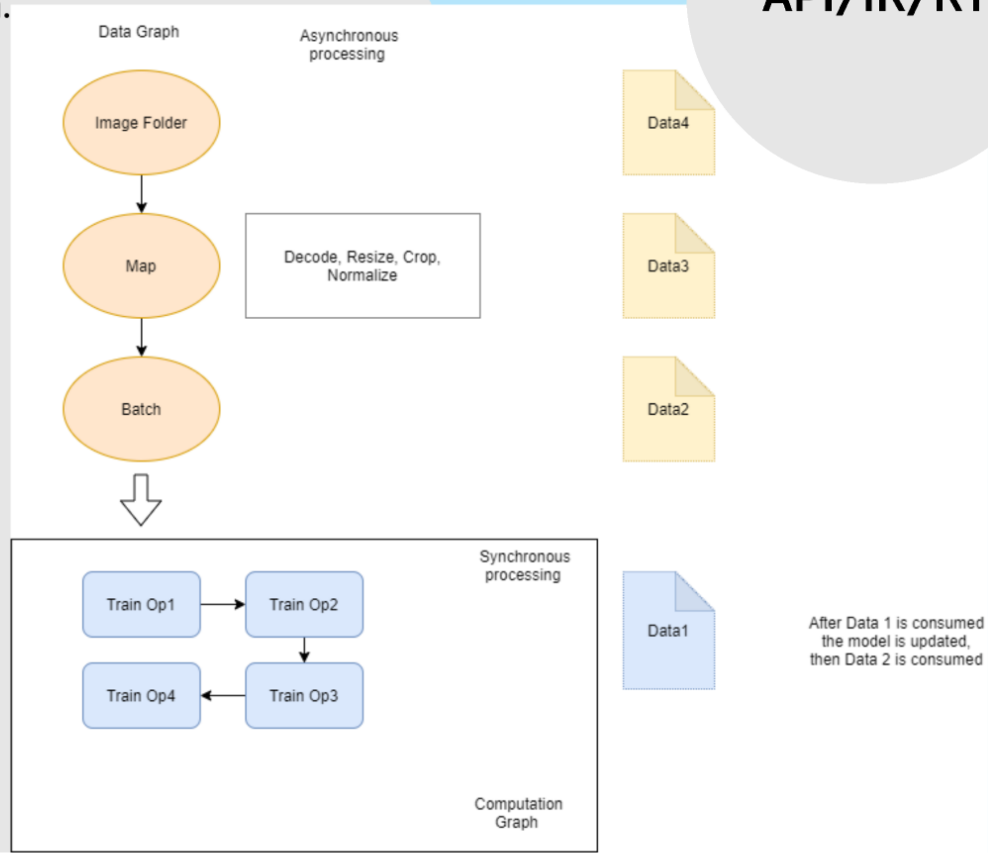
Within MindSpore

API/IR/RT

If we look at a sample training script we can break it down to **data graph** and **training graph**.

The data graph can execute independently of the training graph.

As such we are able to define two graphs, one for data pre-processing and one for training/inference. **MindData handles the building, optimization, and execution of the data graph.**



The different between API, IR and Runtime

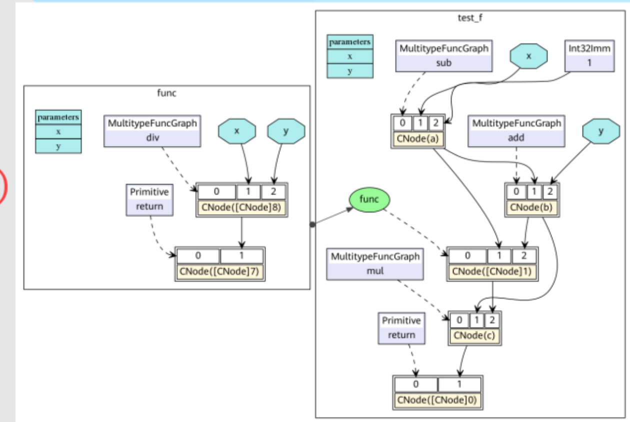
```
def func(x, y):
    return x / y

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

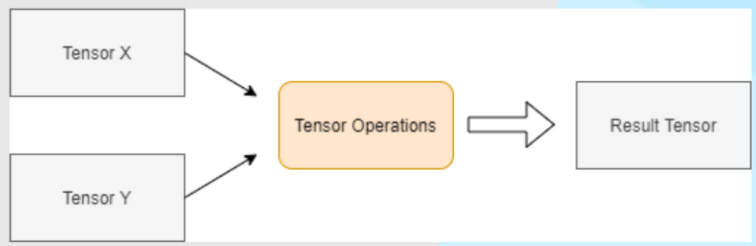
API

Taking a piece of user code and interpreting it as an executable graph can be very complicated, even more complicated when adding optimizations. **API** level logic is parsed into **intermediate representation(IR)**. IR is then optimized before being executed at **run-time (RT)**.

```
lambda (x, y)
    let a = x - 1 in
    let b = a + y in
    let func = lambda (x, y)
        let ret = x / y in
        ret end in
    let %1 = func(a, b) in
    let c = b * %1 in
    c end
```



IR



at run time the data actually gets computed, the data exists in the form of **tensors**

Runtime

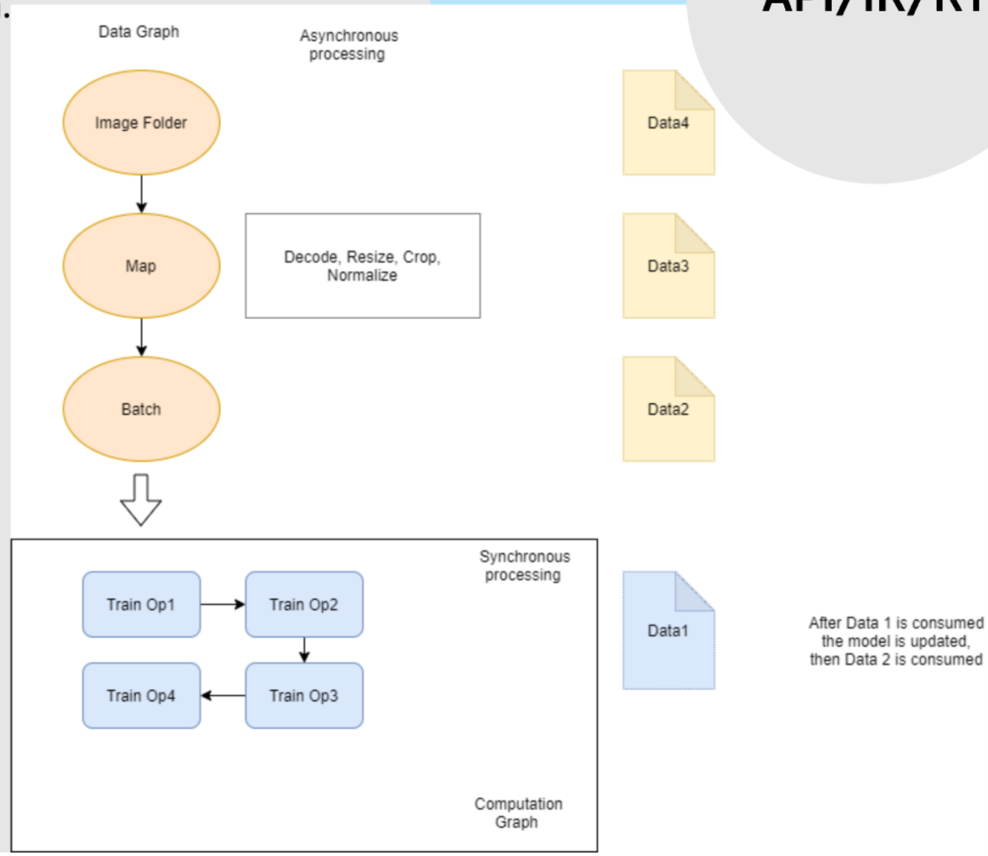
Within MindSpore

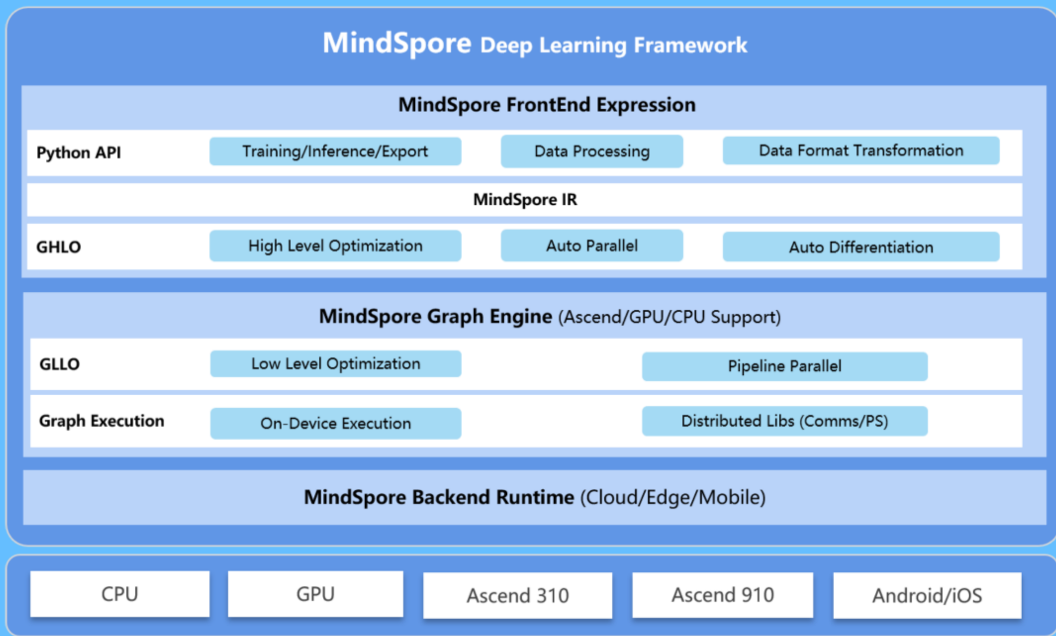
API/IR/RT

If we look at a sample training script we can break it down to **data graph** and **training graph**.

The data graph can execute independently of the training graph.

As such we are able to define two graphs, one for data pre-processing and one for training/inference. **MindData handles the building, optimization, and execution of the data graph.**





In designing MindSpore the requirements are quite different between processing data and training. We will mostly focus on solving the problem of training being slow in this presentation

Two execution graphs

Sending data to different devices

Sharing dependencies

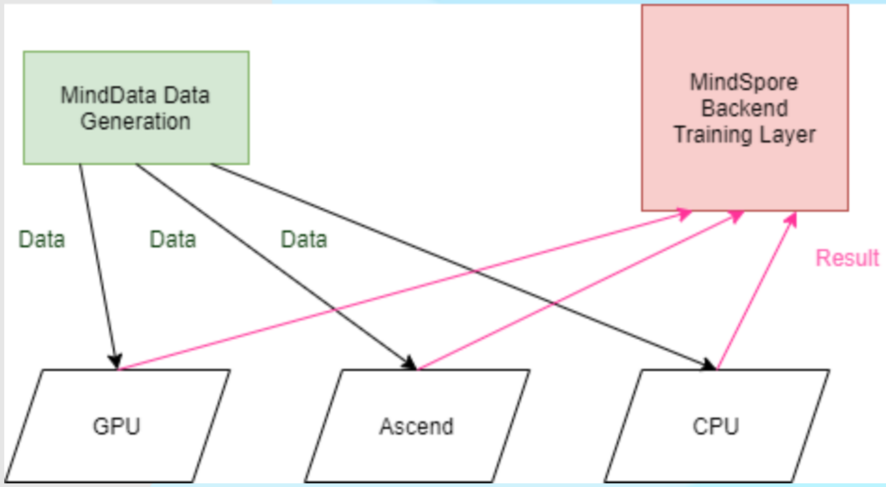
Sending data to various devices

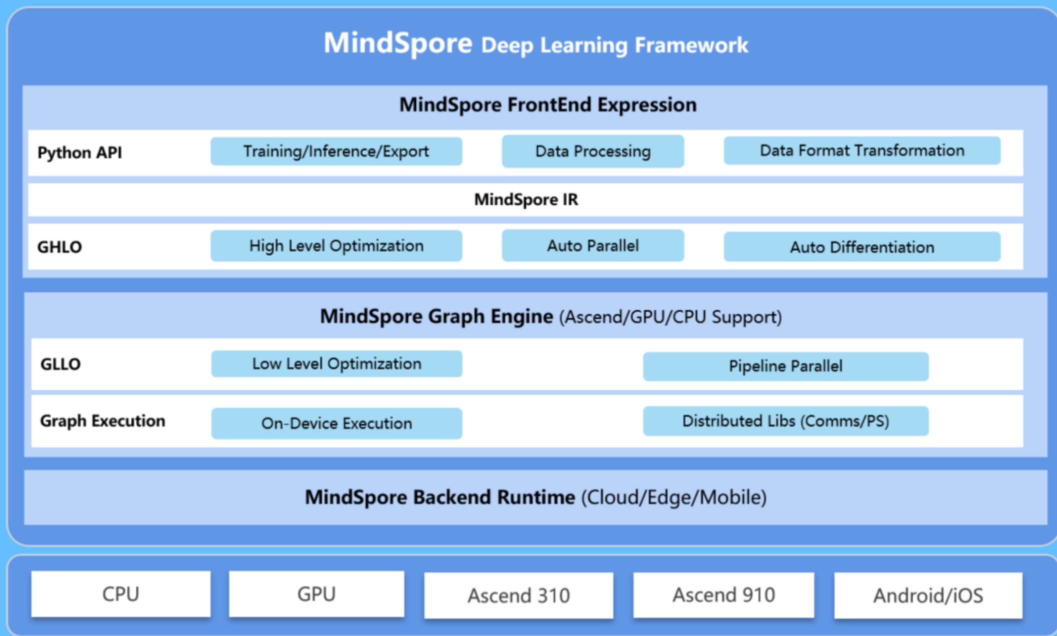
MindData has to send data to various edge devices after the data pre-processing process

For GPU and Ascend, we use something called a **device queue** - a tunnel which copies the data from host memory to edge device.

For CPU, data is passed directly from memory.

For Android, data is passed from memory*





In designing MindSpore the requirements are quite different between processing data and training. We will mostly focus on solving the problem of training being slow in this presentation

Two execution graphs

Sending data to different devices

Sharing dependencies

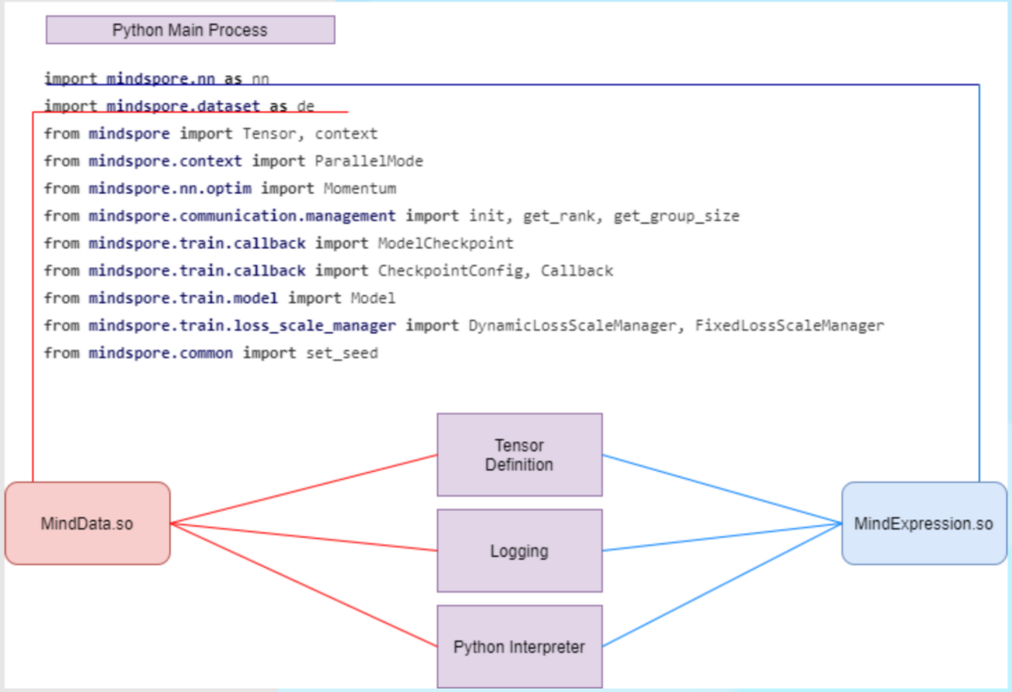
The data graph and training graph need to share dependencies

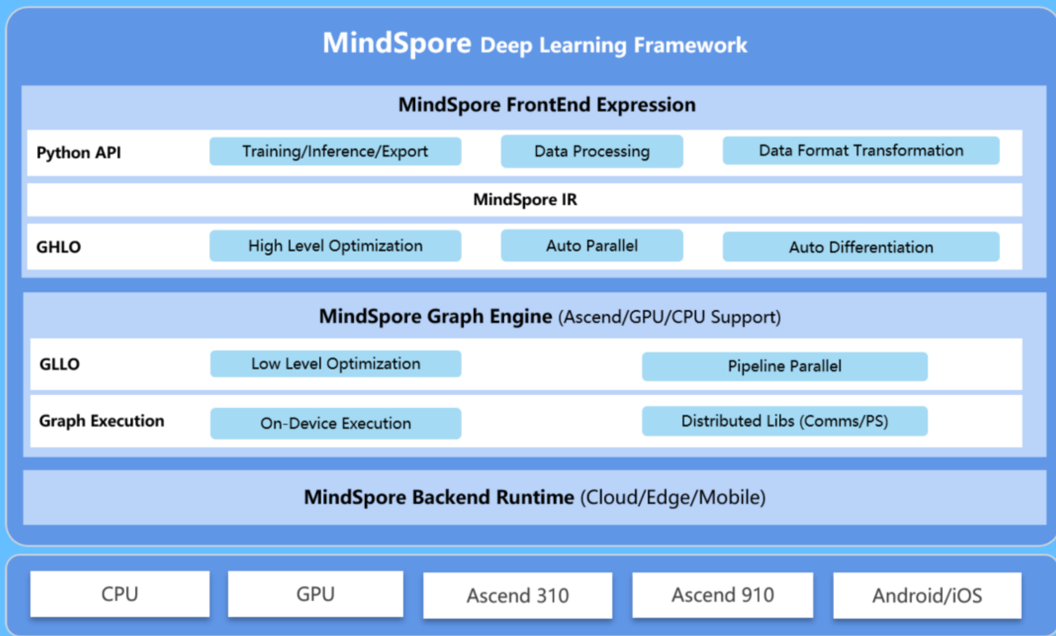
A less discussed portion of the code is sharing dependencies within the same project

An interesting constraint to the design of the data graph is that we have to share the many definitions and resources between the two different run-times.

The golden rule here to to define ownership of all the shared components to avoid any contention/race conditions.

As such, there is one main process owned by MindSpore, **all resource deallocation is handled by the owner of the resource**.





In designing MindSpore the requirements are quite different between processing data and training. We will mostly focus on solving the problem of training being slow in this presentation

Two execution graphs

Sending data to different devices

Sharing dependencies

MindData - Data Pre-processing in MindSpore

Eric Zhang

MindSpore

The AI (Data) Problem

The Requirements

Our Solution

The Data Problem

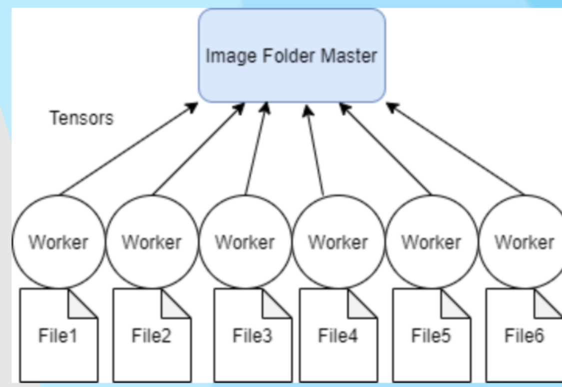
In order to continuously provision data for training, we want to utilize the resources as much as possible to achieve faster data pre-processing speeds

- The pipeline approach
- Layers of abstraction
- Supporting different API languages

The pipeline data processing approach

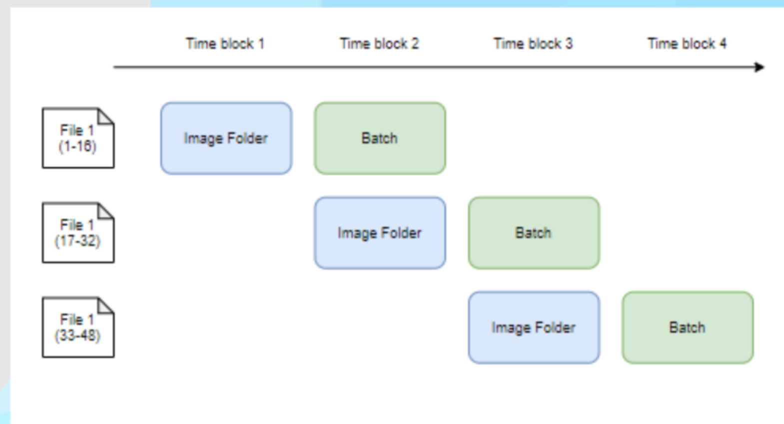
As mentioned before, the data graph is executed in asynchronous fashion

In the simple example, each file is
1. First read from disk into memory by Image Folder
2. Decoded into a 3 channel image
3. Batched into individual batchs



Having a pipeline allows us to utilize CPU resources to the max.

To parallelize reading from disk, MindData adopts a multi-worker approach to maximize reading speed



The Data Problem

In order to continuously provision data for training, we want to utilize the resources as much as possible to achieve faster data pre-processing speeds

- The pipeline approach
- Layers of abstraction
- Supporting different API languages

What does each layer of abstraction mean for MindData

MindData offers an API which is composed of various ops. The data pipeline has much simpler semantics compared to the training graph.

Therefore our IR representation is simpler

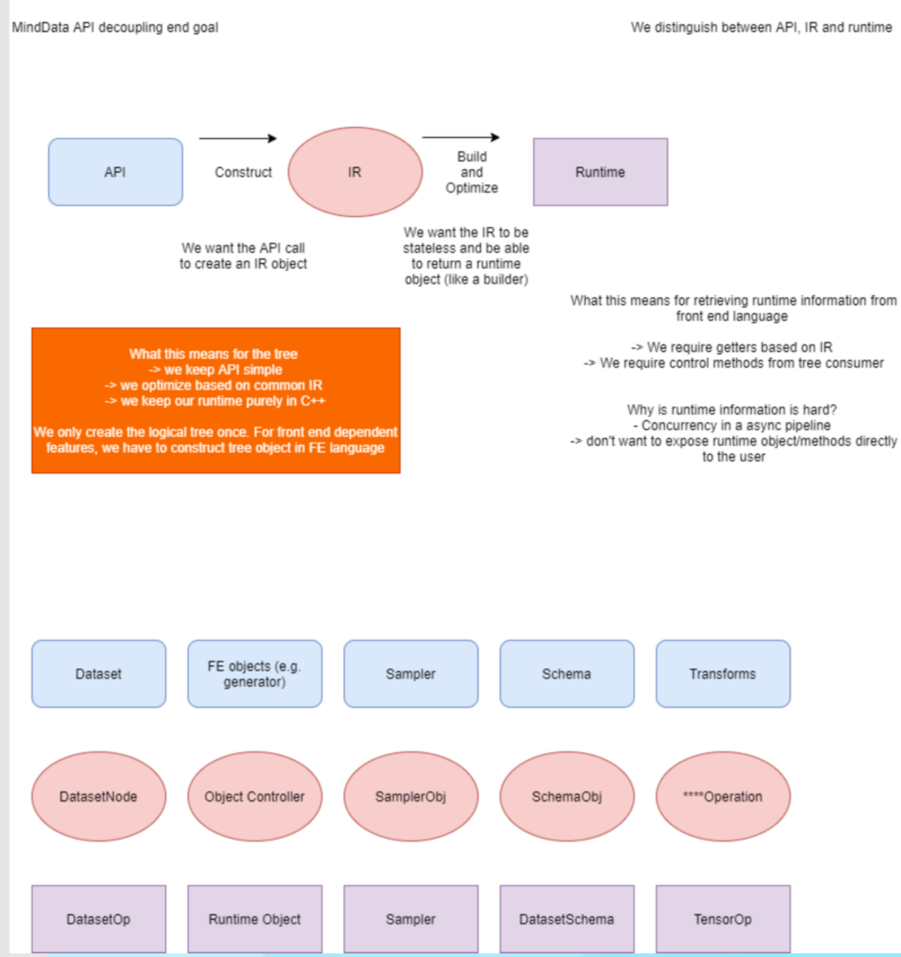
API: User facing classes, abstracting away the complexity of IR and underlying methods

IR: Logical representation of the object, knows how to serialize itself. Optimizer works on IR. IR is **stateless**

Sample
IR

RT: Stateful operators that handles data(tensor) input/output

The key take-away is that there is a fine balance between design complexity and features.
Respect objected oriented design and clearly define responsibility of each component



Data Graph


```
{
  "batch_size":2,
  "children":[
    {
      "callback":[

      ],
      "children":[
        {
          "callback":[

          ],
          "children":[
            {
              "children":[
                {
                  "children":[

                  ],
                  "class_indexing":{

                  },
                  "dataset_dir":"../data/dataset/testPK/data",
                  "decode":false,
                  "extensions":[

                  ],
                  "num_parallel_workers":8,
                  "op_type":"ImageFolderDataset",
                  "sampler":{
                    "child_sampler":[
                      {
                        "num_samples":0,
                        "sampler_name":"SequentialSampler",
                        "start_index":0
                      }
                    ],
                    "num_samples":11,
                    "replacement":true,
                    "sampler_name":"WeightedRandomSampler",
                    "weights":[
                      1.0,
                      0.1,
                      0.02,
                      0.3,
                      0.4,
                      0.05,
                      1.2,
                      0.13,
                      0.14,
                      0.015,
                      0.16,
                      1.1
                    ]
                  }
                }
              ],
              "count":1,
              "op_type":"Repeat"
            }
          ],
```


Training Graph

```
#IR entry      : @6_5_1_construct_wrapper.15
#attrs         :
check_set_strategy_valid_once_only : 1
#Total params  : 2

%para1_x : <Tensor[Float32]x[const vector][]>
%para2_y : <Tensor[Float32]x[const vector][]>

#Total subgraph : 1

subgraph attr:
check_set_strategy_valid_once_only : 1
subgraph @6_5_1_construct_wrapper.15() {
  %0([CNode]8) = Add(%para1_x, %para2_y) primitive_attrs: {output_names: [output], input_names: [x, y]}
      : (<Tensor[Float32]x[const vector][]>, <Tensor[Float32]x[const vector][]>) -> (<Tensor[Float32]x[const vector][]>)
      # In file /home/workspace/mindspore/mindspore/ops/composite/multitype_ops/add_impl.py(129)/    return F.add(x, y)/
      # In file demo.py(14)/        x = x + y/
  %1([CNode]10) = Mul(%0, %para2_y) primitive_attrs: {output_names: [output], input_names: [x, y]}
      : (<Tensor[Float32]x[const vector][]>, <Tensor[Float32]x[const vector][]>) -> (<Tensor[Float32]x[const vector][]>)
      # In file /home/workspace/mindspore/mindspore/ops/composite/multitype_ops/mul_impl.py(48)/    return F.tensor_mul(x, y)/
      # In file demo.py(15)/        x = x * y/
  return(%1)
      : (<Tensor[Float32]x[const vector][]>)
}
```

What does each layer of abstraction mean for MindData

MindData offers an API which is composed of various ops. The data pipeline has much simpler semantics compared to the training graph.

Therefore our IR representation is simpler

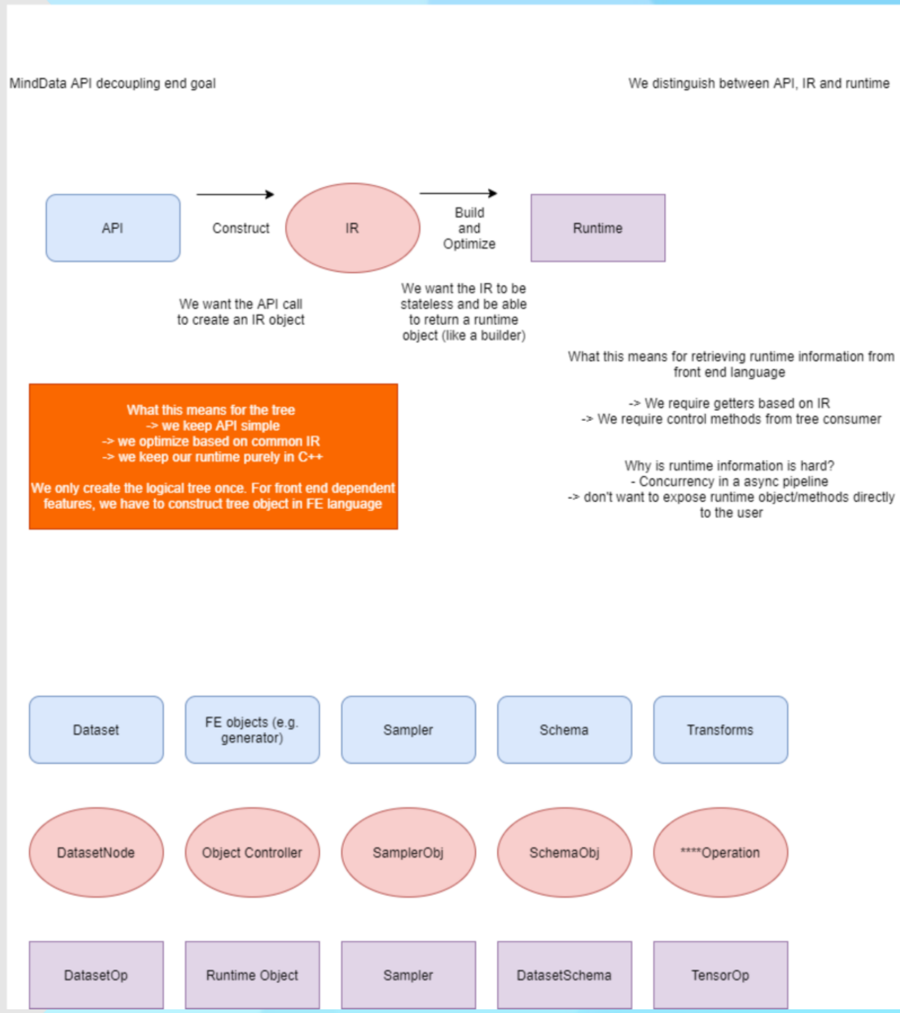
API: User facing classes, abstracting away the complexity of IR and underlying methods

IR: Logical representation of the object, knows how to serialize itself. Optimizer works on IR. IR is **stateless**

Sample IR

RT: Stateful operators that handles data(tensor) input/output

The key take-away is that there is a fine balance between design complexity and features.
Respect objected oriented design and clearly define responsibility of each component



The Data Problem

In order to continuously provision data for training, we want to utilize the resources as much as possible to achieve faster data pre-processing speeds

The pipeline approach

Layers of abstraction

Supporting different API languages

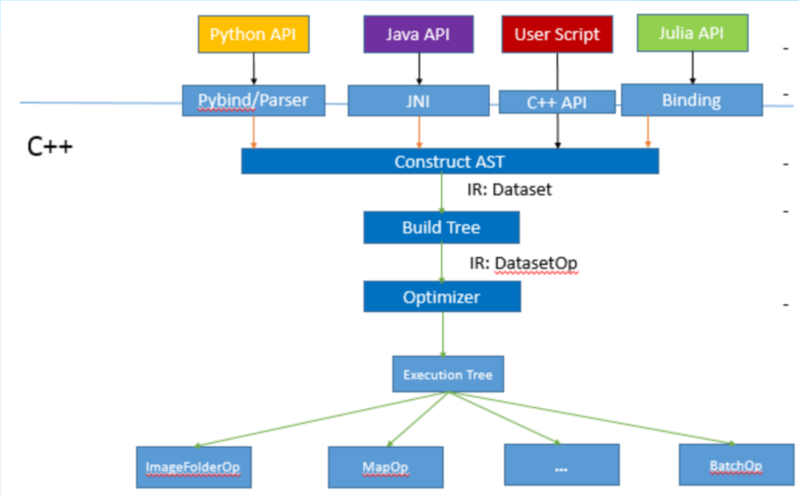
How MindData support different front end languages

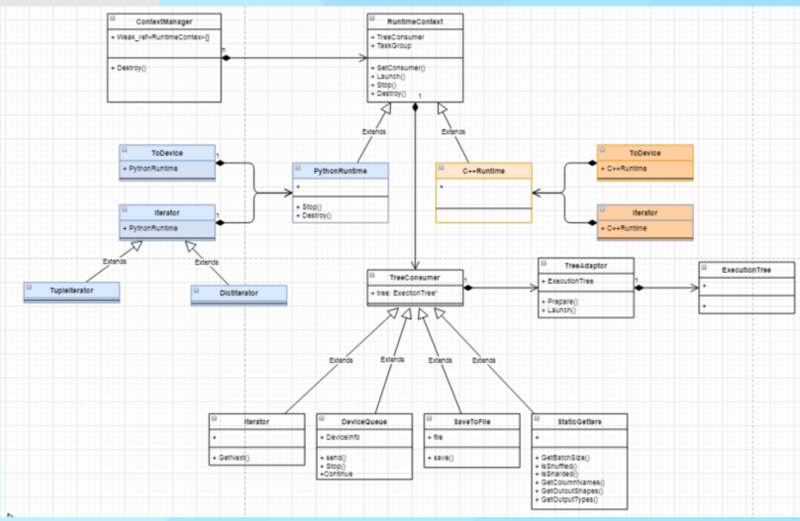
Once you get used to the responsibility separation of the respective classes...

We are now able to have API level classes build into the **same IR** for different front end languages.

By doing this we can easily add support for new front end languages **without changing the back end code**

This structure exists for all execution platforms and minimizes code maintenance efforts for mix/match execution environment and front end languages(You can run C++ front end on Linux/Android, or run python API for python/windows)





The Data Problem

In order to continuously provision data for training, we want to utilize the resources as much as possible to achieve faster data pre-processing speeds

The pipeline approach

Layers of abstraction

Supporting different API languages

MindData - Data Pre-processing in MindSpore

MindSpore

Eric Zhang

The AI (Data) Problem

The Requirements

Our Solution

Thank you!

MindSpore is opensource, for contribution, please refer to
https://gitee.com/mindspore/mindspore

Github mirror:
https://github.com/mindspore-ai

Additional information about MindSpore can be found at:
https://www.mindspore.cn/

MindData - Data Pre-processing in MindSpore

MindSpore

Eric Zhang

The AI (Data) Problem

The Requirements

Our Solution