

Welcome to Algorithm Components's documentation!

Introduction

This library was designed for high performance pipeline of feature transform and model prediction as a consistent solution for both online and offline scenes.

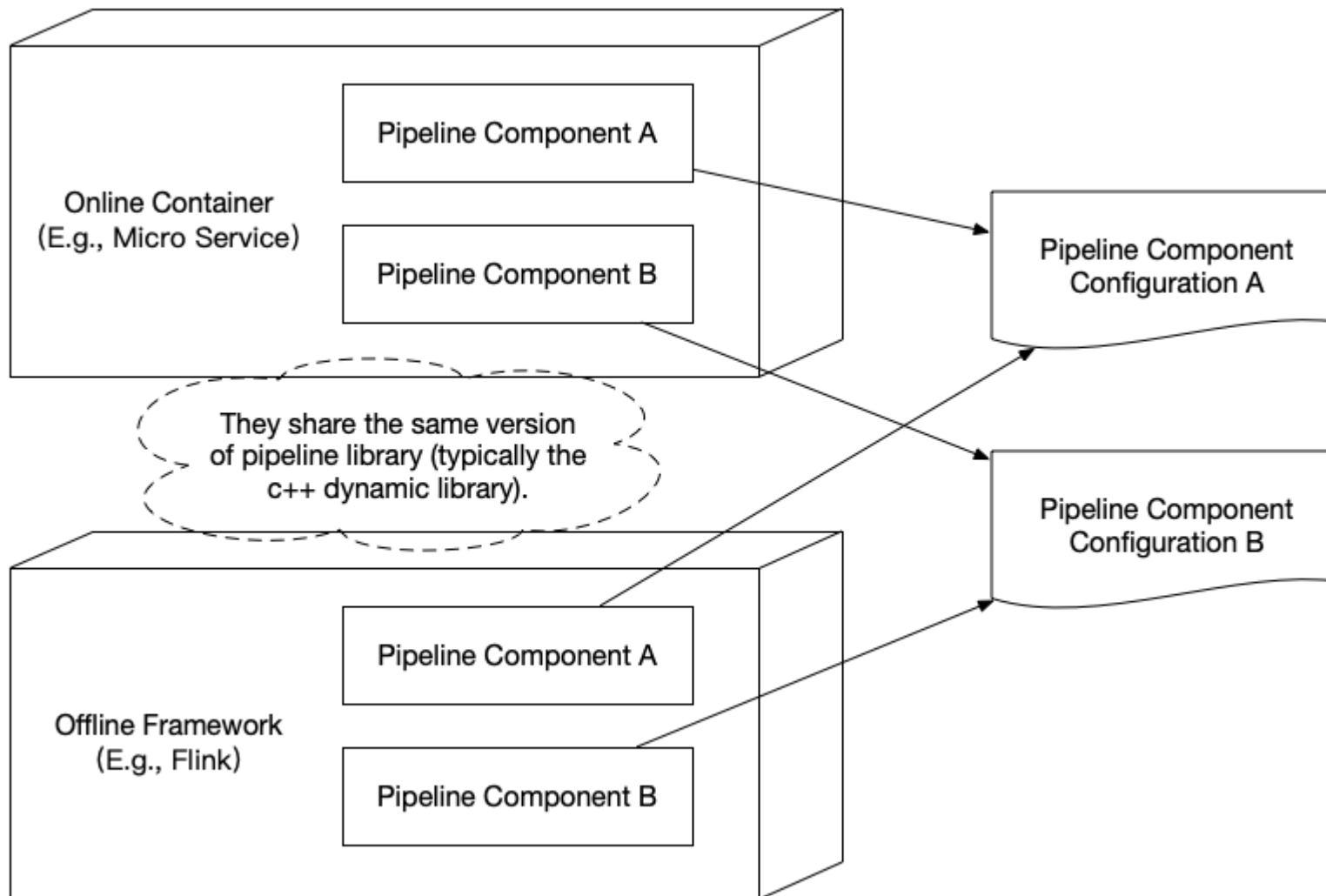
The advantages are as follows.

(1) The users can deploy the same library and configuration files in both online and offline environments which guarantees the computation consistency.

(2) To implement a large number of complex feature transforms and model predictions, the users only need to write a few configuration files without hard code anything. We have predefined a lot of commonly useful attribute transforms, feature transforms and model predictions (e.g., linear model and xgb model). In most of cases they will be enough. The users can also define their own transforms if necessary.

(3) The library was implemented based on C++17 with thread pool support and efficient external libraries (e.g., google's SparseHash and Abseil) which implies the high performance of RT and QPS.

(4) We wrapped our library using Swig. This indicates that the users can deploy it in a large variety of containers (e.g., Docker, Hadoop and Flink) with other programming languages (e.g., Java, Scala and Python) as invokers.



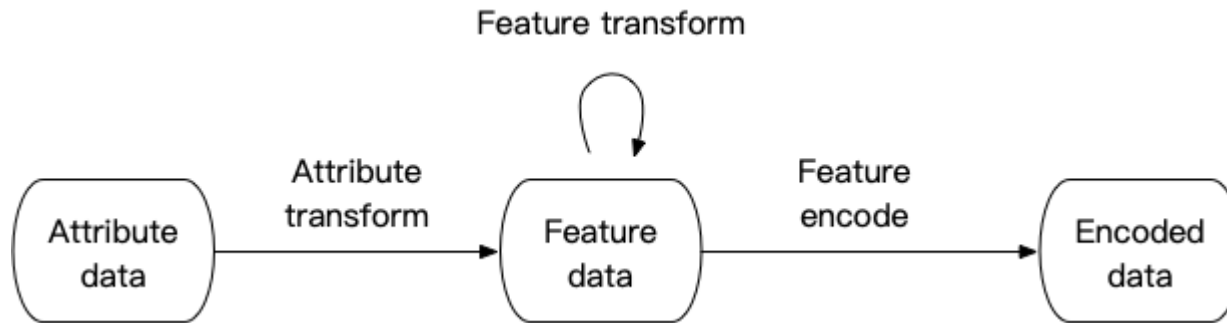
User Manual

The users only need 3 steps to adopt Algorithm Components library in their own applications.

1. Format input data into AUCImprAttrs structure (protobuf format).
2. Define a few configuration files with attribute transforms, feature transforms and model predictions.
3. Write a few lines of code to load the configurations and apply the predefined efficient transforms.

Step 1: Format input data

In our designed system, the data has 3 statuses throughout lifetime: attribute data, feature data and encoded data.



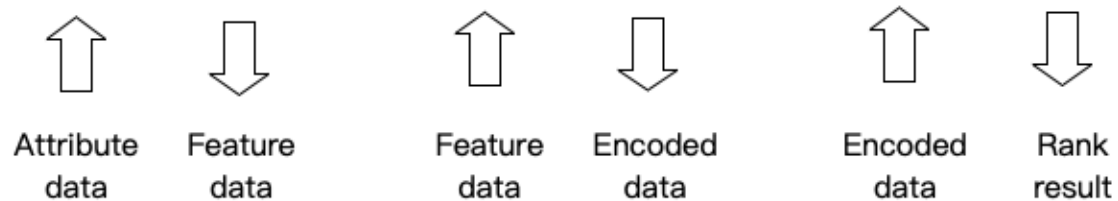
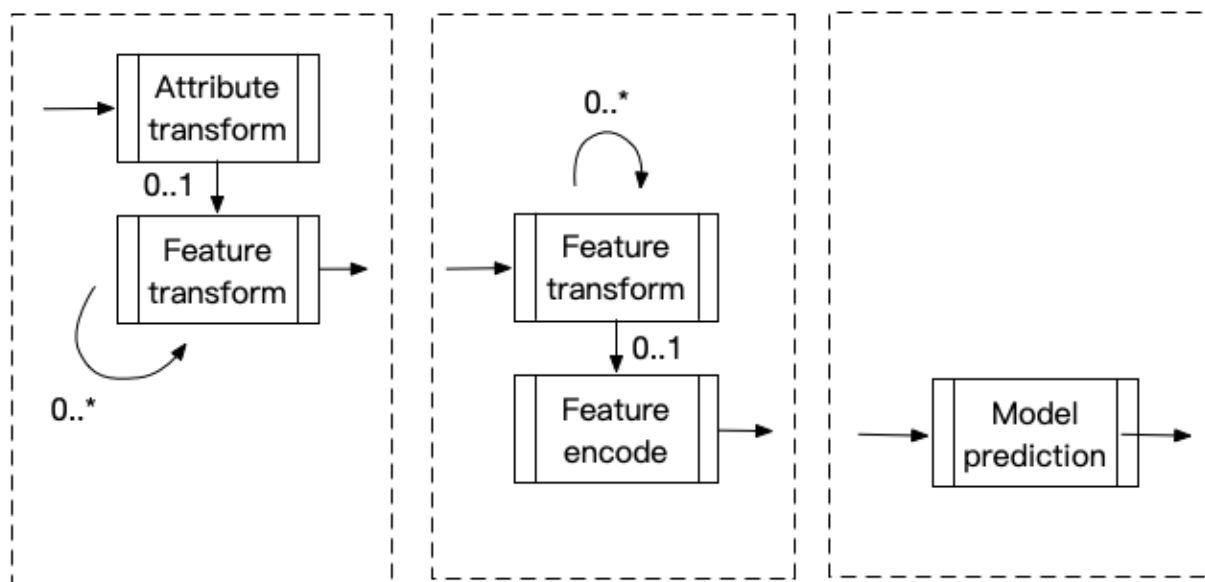
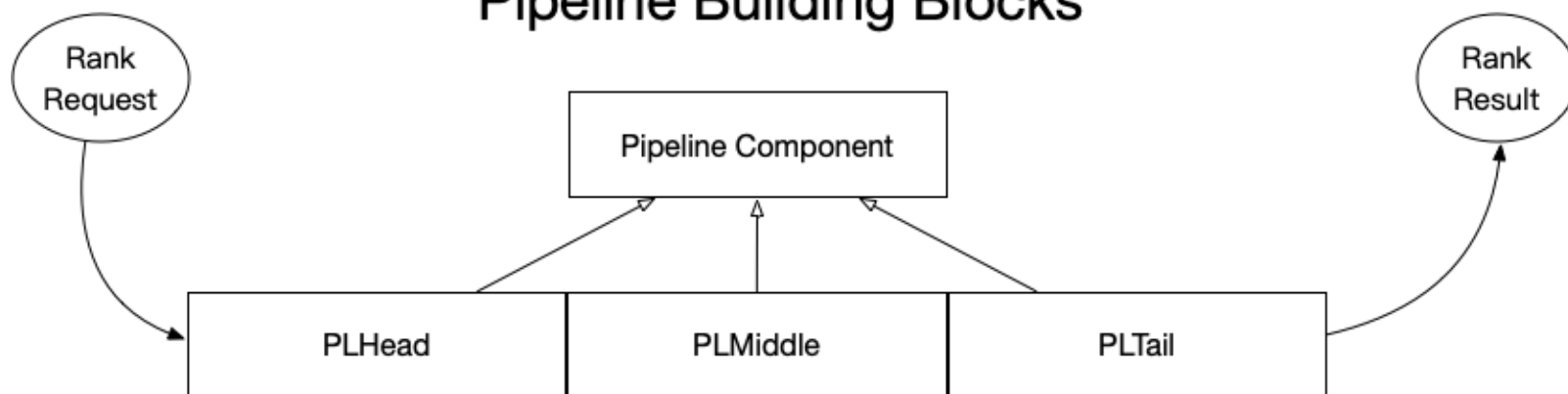
We represent the attribute data and feature data with 2 protobuf format structures: AUCImprAttrs and AUCImprFeats. We define these structures in a separate library (i.e., data-model-protobuf). Hence the user's first step is to format the application data into attribute data (i.e., AUCImprAttrs). And the library will take over the rest of work. Here is a detailed description of [Data Model](#).

Step 2: Define transform configurations

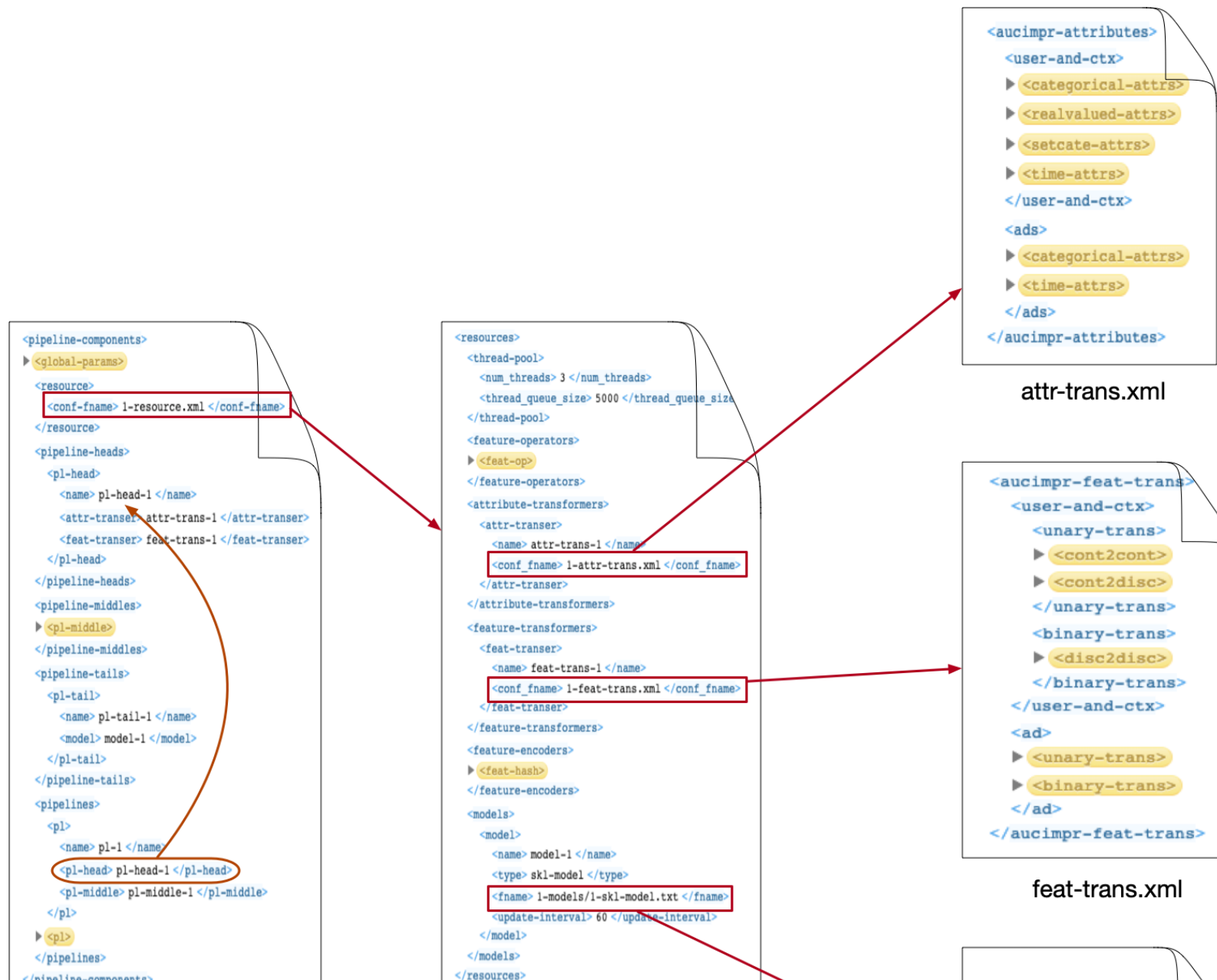
The next and also one of the most important things is to define a few configuration files. These configurations tell the Algorithm Components library in which way to organize the pipeline.

In our design we breakdown a complete pipeline into 3 components: PLHead, PLMiddle and PLTail. Each one has its own skill which is demonstrated as follows.

Pipeline Building Blocks



The configuration files define the pipelines in top-down mode. The lower things (e.g., thread pool, attribute transforms and feature transforms) can be reused many times. Here is a detailed example (the files are located in test/test-data directory).





Step 3: Write a few lines of code

In the last step, the users only need to write a few lines of code to load the configurations and apply the predefined pipelines to data. The following is an example which is adopted from `app/src/algocomp_eval_time.cpp`.

```
// Initialize logger
string logger_name = "algocomp_app_eval_time_logger";
string logger_dir_name = "algocomp_logs/app_eval_time";
string logger_file_name = logger_dir_name + "/log";
auto logger_dir = fs::path(logger_dir_name);
if (fs::exists(logger_dir)) fs::remove_all(logger_dir);
fs::create_directories(logger_dir);
ACLog::create_loggers(logger_name, logger_file_name);

// Initializer pipeline component pool (load configurations)
auto pool = PLCompPool("test/test-data/1-pipeline.xml");
auto pl = pool.get_pipeline("pl-1");

// Read input data
std::ifstream in("test/test-data/encoded_attributes.txt");
```

```

string line;
std::getline(in, line);
string attrs_pb;
Base64::Decode(line, &attrs_pb);
in.close();

// Apply predefined pipeline (transforms) to data
auto encoded = pl->head_to_middle(attrs_pb.c_str(), attrs_pb.length());

```

Here is another example with Java as invoker language (can from Hadoop streaming or Flink) since we wrap the Algorithm Components library with Swig.

```

// Load dynamic library
final String dir = System.getProperty("user.dir");
System.load(dir + "/bin/libalgocomp_swig.so");

// Initialize logger
String exampleLoggerName = "java_wrap_logger";
String exampleLoggerFile = "algocomp_logs/java_wrap/log";
ACLog.create_loggers(exampleLoggerName, exampleLoggerFile);

// Read input data
String line = "";
BufferedReader reader;
try {
    reader = new BufferedReader(new FileReader(dir + "/conf/encoded_attributes.txt"))
    line = reader.readLine();
    reader.close();
} catch (IOException e) {
    e.printStackTrace();
}

// Parse input data to protobuf string
byte[] attrStr = Base64.getDecoder().decode(line);
AUCImprAttrs attrs = AUCImprAttrs.parseFrom(attrStr);
byte[] attrsPB = attrs.toByteArray();

```

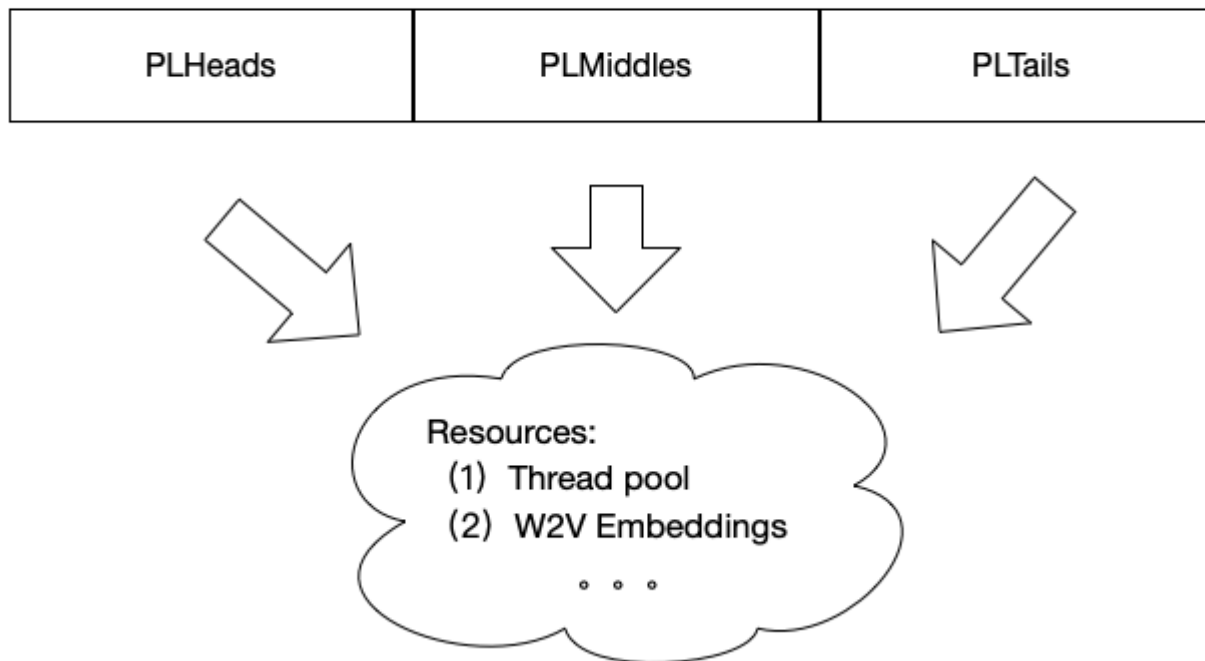
```
// Initializer pipeline component pool (load configurations)
PLCompPool plPool = new PLCompPool(dir + "/conf/pipeline.xml");
Pipeline pl = plPool.get_pipeline("pl-1");

// Apply predefined pipeline (transforms) to data
StrVec encoded = pl.head_to_middle(attrsPB, attrsPB.length);
```

Additional information: Resource Sharing

All the bottom things are basic resources like thread pool, w2v embeddings, etc. They will be shared across the upper things.

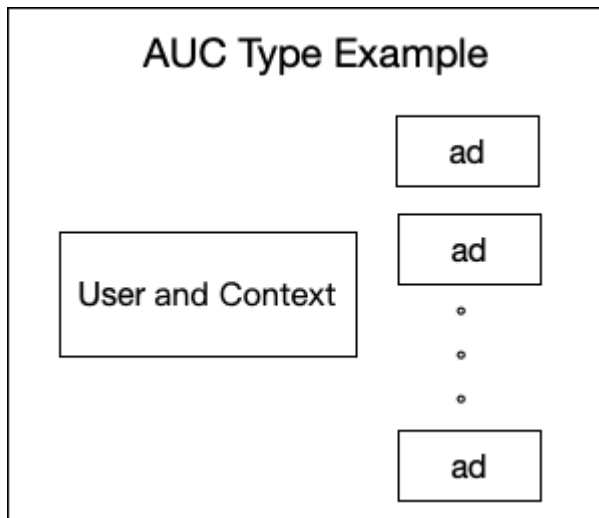
Pipeline Resource Sharing



Data Model

As a very typical kind of applications we can apply our pipelines of complex transforms in either computational advertising scenario or recommendation system scenario.

In these domain applications the data is composed of a user with his/her real time context followed by a list of candidate advertises. Usually the number of candidate advertises varies from hundreds to thousands depending on system's computational capability and ability of upper recall systems.



Due to the frequency usages in above mentioned application scenarios and the consideration of implementation efficiency, we define our data model in this way.

In our designed system the data has 3 statuses: attribute data, feature data and encoded data. Here is the definition of attribute data with some commonly used types (e.g., categorical, set categorical, real valued and time).

```
message AUCImprAttrs {  
  repeated float labels = 1;  
  AttributeBunch user_and_ctx = 4;  
  repeated AttributeBunch ads = 5;
```

```

}

message AttributeBunch {
    map<string, AttributeData> attrs = 1;
}

message AttributeData {

    enum AttributeType {
        UNKNOWN = 0;
        CATEGORICAL = 1; // string, int_val
        SET_CATEGORICAL = 2; // SetCateValue --> [<string, float>]
        REAL_VALUED = 3; // float
        UNIX_TIMESTAMP = 4; // int_val (1546235518)
    }

    AttributeType attr_type = 1;

    oneof oneof_attr_val {
        string str_val = 2;
        int32 int_val = 3;
        int64 long_val = 4;
        float float_val = 5;
        SetCateValue scate_val = 6;
    }

    message SetCateValue {
        map<string, float> vals = 1;
    }
}

```

And the following is the definition of feature data with discrete type and continuous type.

```

message AUCImprFeats {
    repeated float labels = 1;
    FeatureBunch user_and_ctx = 4;
    repeated FeatureBunch ads = 5;
}

```

```

message FeatureBunch {
    map<string, DiscFeatGrp> disc_feats = 1;
    map<string, ContFeatGrp> cont_feats = 2;
}

message DiscFeatGrp {
    map<string, DiscFeatData> feats = 1;
}

message ContFeatGrp {
    map<string, ContFeatData> feats = 1;
}

message DiscFeatData {
    string gname = 1;
    string fname = 2;
}

message ContFeatData {
    string gname = 1;
    string fname = 2;
    float fval = 3;
}

```

The definitions are implemented with Protobuf format which is very suitable for data passed across different kinds of systems with different implemented containers and programming languages. The users can check the data-model-protobuf library for real definitions.

Configurations

This section describes the configurations of attribute transform and feature transform.

Attribute Transform

The method of attribute transform supports the four types of attribute data as follows:

1. Categorical Attribute.
2. Realvalued Attribute.
3. Time Attribute.
4. Set-Categorical Attribute.

The next and also important thing is how to define attribute transform above four types of attribute data.

```
<attribute in_aname="input_attribute_name"  
          out_gname="output_group_name"  
          out_fname="output_feature_name"  
          procs="func1;func2"  
          params="func1_param1,func1_param2,...;func2_param1,func2_param2,..."/>
```

As shown above example of xml format, we need to configure five elements that are “**in_aname**”, “**out_gname**”, “**out_fname**”, “**procs**” and “**params**” regardless of categorical attributes, realvalued attribute, time attributes or set-categorical attribute.

There are some brief explanations about above five elements as follows:

in_aname:

Attribute name of input data.

out_gname:

Group name of output which after attribute data transferring.

out_fname:

Feature name of output which after attribute data transferring. It just need to configure symbol “%” for categorical attribute, set-categorical attribute and time attribute and symbol “=” for realvalued at-

tribute since the program can generate the feature name after attribute data transferring by parsing corresponding symbol automatically.

procs:

The functions of process attribute data which are different for processing different types attribute data. They are separated with symbol “;” like “proc_func1;proc_func2” if there are multiple different functions.

params:

Parameters corresponding to multiple different functions which also are separated with symbol “;” like “func1_params;func2_params”. Parameters of a function are separated with symbol “,” like “func1_param1,func1_param2,...”. If a function has no parameters, it just need to configure symbol “-”.

(1) Categorical Attribute

The following is an example of xml format which is used for configuring categorical attribute.

```
<category-attrs>
  <attribute in_aname="userSex"
             out_gname="user_sex"
             out_fname="%out_gname/user_sex%male:1.0"
             procs="prune_max_len"
             params="skip,10"/>
</category-attrs>
```

out_fname:

Just configure symbol “%” or string “%your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

procs:

Only one function:

prune_max_len: Pruning maximum length.

params

The parameters of the function **prune_max_len**:

- (1) “**skip, max_len**” such as “skip,10”. The **max_len** is maximum length of attribute value. The **skip** represents that the program will skip the attribute if the length of attribute value exceed **max_len**.
- (2) “**cut_head, max_len**” such as “cut_head,10”. The **cut_head** represents that the program will cut the attribute value from head to **max_len**.
- (3) “**cut_tail, max_len**” such as “cut_tail,10”. The **cut_tail** represents that the program will cut the attribute value to tail with **max_len**.

There is another one example of xml format about **params** as follow:

```
<categorical-attrs>
  <attribute in_aname="userSex"
             out_gname="user_sex"
             out_fname="%out_gname/user_sex%male:1.0"
             procs="prune_max_len"
             params="cut_head,10"/>
</categorical-attrs>
```

(2) Realvalued Attribute

The following is an example of xml format which is used for configuring realvalued attribute.

```
<realvalued-attrs>
  <attribute in_aname="userAge"
             out_gname="user_age"
             out_fname="=out_gname/user_age:26"
             procs="prune_range;scale_min_max"
```

```
params="skip,5,70;5,70"/>  
</realvalued-attrs>
```

out_fname:

Just configure symbol “=” or string “=your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

procs:

The functions of process realvalued attribute value as follows:

1. **prune_range**: Pruning over range.
2. **log**: Log computation.
3. **scale_min_max**: Min-max scale.
4. **scale_z_score**: Standard scale.

params:

The parameters of the function **prune_range**:

1. “**skip, min_value, max_value**” such as “skip,0,1e6”.
2. “**replace, min_value, max_value**” such as “replace,0,1e6”.

The parameters of the function **log**:

1. “**log10, min_neg_val**” such as “log10,-10”.
2. “**log, min_neg_val**” such as “log,-5”.
3. “**log2, min_neg_val**” such as “log2,-1”.

The parameters of the function **scale_min_max**:

“**min_value, max_value**” such as “0,200”.


The parameters of the function **scale_z_score**:

“mean_value, std_value” such as “0,1”.

(3) Time Attribute

The following is an example of xml format which is used for configuring time attribute.

```
<time-attrs>
  <attribute in_aname="happen_time"
    out_gname="=in_aname/happen_time"
    out_fname="%out_gname/happend_time%weekend:1.0,happend_time%raw:15342"
    procs="all"
    params="" />
</time-attrs>
```



out_gname:

Just configure symbol “=” or string “=your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

out_fname:

Just configure symbol “%” or string “%your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

procs:

The functions of process time attribute value as follows:

1. **all**: Transform with all time level.
2. **partial**: Transform with partial time level (e.g., without hour level).
3. **time_str**: Time string format if input data type is string.

params:

The parameters of the function **all**:

using symbol “-” since it is parameterless.

The parameters of the function **partial**:

a combination of one or more of “**hour, day, week, month**” such as “week”, “hour,week”, “hour,day,week” and so on.

The parameters of the function **time_str**:

time and date formatting symbols such as “%Y-%m-%d %H:%M:%S”, “%Y-%m-%d” and so on.

(4) Set-Categorical Attribute.

The following is an example of xml format which is used for configuring set-categorical attribute.

```
<setcate-attrs>
  <attribute in_aname="poi_cpt"
             out_gname="=in_aname/poi_cpt"
             out_fname="%out_gname/poi_cpt%ent:0.3,poi_cpt%sport:0.7"
             procs="prune_range;norm_sum"
             params="skip,0,10;-"/>
</setcate-attrs>
```

out_gname:

Just configure symbol “=” or string “=your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

out_fname:

Just configure symbol “%” or string “%your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

procs:

The functions of process realvalued attribute value as follows:

1. **prune_max_len**: Pruning maximum length.
2. **prune_range**: Pruning over range.
3. **norm_sum**: Norm sum scale.

params:

The parameters of the function **prune_max_len**:

- (1) “**skip, max_len**” such as “skip,100”. The **max_len** is maximum length of attribute value. The **skip** represents that the program will skip the attribute if the length of attribute value exceed **max_len**.
- (2) “**cut_head, max_len**” such as “cut_head,100”. The **cut_head** represents that the program will cut the attribute value from head to **max_len**.
- (3) “**cut_tail, max_len**” such as “cut_tail,100”. The **cut_tail** represents that the program will cut the attribute value to tail with **max_len**.

The parameters of the function **prune_range**:

1. “**skip, min_value, max_value**” such as “skip,0,1e6”.
2. “**replace, min_value, max_value**” such as “replace,0,1e6”.

The parameters of the function **norm_sum**:

using symbol “-” since it is parameterless.

Finally, there is a complete attribute transform example of xml format as follow:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<aucimpr-attributes>
```

```
  <user-and-ctx>
```

```
    <categorical-attrs>
```

```
      <attribute in_aname="userSex" out_gname="user_sex" out_fname="%"  
                procs="prune_max_len" params="skip,10"/>
```

```
    </categorical-attrs>
```

```
    <realvalued-attrs>
```

```
      <attribute in_aname="userAge" out_gname="user_age" out_fname="=out_gname  
                procs="prune_range;scale_min_max" params="skip,5,70;5,70"/>
```

```
      <attribute in_aname="userCost" out_gname="user_cost" out_fname="=out_gname  
                procs="prune_range;scale_z_score" params="replace,0,1e6;150,5
```

```
                <attribute in_aname="userCost2" out_gname="user_cost2" out_fname="=out_gname  
                procs="prune_range;scale_z_score;log" params="replace,-2000,1
```

```
    </realvalued-attrs>
```

```
    <setcate-attrs>
```

```
      <attribute in_aname="poi_cpt" out_gname="=in_aname/poi_cpt"  
                out_fname="%out_gname/poi_cpt%ent:0.3,poi_cpt%sport:0.7" proc  
                params="skip,0,10;-"/>
```

```
    </setcate-attrs>
```

```
    <time-attrs>
```

```
      <attribute in_aname="happen_time" out_gname="=in_aname/happen_time"  
                out_fname="%out_gname/happend_time%weekend:1.0,happend_time%r  
                params="" />
```

```
    </time-attrs>
```

```
  </user-and-ctx>
```

```
</ads>
```

```

<categoryal-attrs>
  <attribute in_aname="productId" out_gname="product_id" out_fname="%out_c
    procs="prune_max_len" params="cut_tail,150"/>
</categoryal-attrs>

<time-attrs>
  <attribute in_aname="happen_time2" out_gname="=in_aname/happen_time2"
    out_fname="%out_gname/happend_time%weekend:1.0,happend_time%r
    procs="partial;time_str"
    params="week,month,day;%Y-%m-%d"/>
  <attribute in_aname="happen_time3" out_gname="=in_aname/happen_time3"
    out_fname="%out_gname/happend_time%weekend:1.0,happend_time%r
    procs="all;time_str"
    params="-;%Y-%m-%d %H:%M:%S"/>
</time-attrs>
</ads>

</aucimpr-attributes>

```

Feature Transform

The method of feature transform supports the two types of feature transform as follows:

1. Unary Feature.
 - A. cont2cont
 - B. cont2disc
2. Binary Feature.
 - disc2disc

The next and also important thing is how to define above two types of feature transform.

```
<op-ctx in_gname="user_cost"  
        in_fname="in_gname/user_cost"  
        out_gname="#in_fname/user_cost#log"  
        out_fname="out_gname/user_cost#log"  
        op="log"  
        scope="fname"  
        procs="log10"  
        params="-"/>
```

As shown above example of xml format, we need to configure eight elements which are “in_gname”, “in_fname”, “out_gname”, “out_fname”, “op”, “scope”, “procs” and “params”.

There are some brief explanations about above eight elements as follows:

in_gname:

Group name of input corresponding to “out_gname” in Attribute Transform.

in_fname:

Feature name of input corresponding to “out_fname” in Attribute Transform.

out_gname:

Group name of output which after feature transferring.

out_fname:

Feature name of output which after feature transferring.

op:

Feature transform operation.

scope:

Scope of feature transform operation.

procs:

The functions of process attribute data which are different for processing different types attribute data. They are separated with symbol “;” like “proc_func1;proc_func2” if there are multiple different functions.

params:

Parameters corresponding to multiple different functions which also are separated with symbol “;” like “func1_params;func2_params”. Parameters of a function are separated with symbol “,” like “func1_param1,func1_param2,...”. If a function has no parameters, it just need to configure symbol “-”.

(1) Unary Feature

A. cont2cont

The following is an example of xml format which is used for configuring cont2cont feature transform.

```
<cont2cont>
  <op-ctx in_gname="ad_cost"
    in_fname="=in_gname/ad_cost"
    out_gname="#in_fname/ad_cost#log"
    out_fname="=out_gname/ad_cost#log"
    op="log" scope="fname"
    procs="log" params="-"/>
</cont2cont>
```

in_fname:

Just configure symbol “=” or string “=your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

out_gname:

Just configure symbol “#” or string “#your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

out_fname:

Just configure symbol “=” or string “=your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

scope:

1. **gname**
2. **fname**

op:

log

procs:

1. **log10**
2. **log**
3. **log2**

params:

using symbol “-” since it is parameterless

B. cont2disc

The following is an example of xml format which is used for configuring cont2disc feature transform.

```
<cont2disc>
  <op-ctx in_gname="ad_revenue"
    in_fname="=in_gname/ad_revenue"
    out_gname="#in_fname/ad_revenue#bkt"
```

```
out_fname="@out_gname/ad_revenue#bkt@1"  
op="bucketize" scope="fname"  
procs="predefined"  
params="ten-equal-width-zscore"/>  
</cont2disc>
```

in_fname:

Just configure symbol “=” or string “=your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

out_gname:

Just configure symbol “#” or string “#your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

out_fname:

Just configure symbol “@” or string “@your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

scope:

1. **gname**
2. **fname**

op:

bucketize

procs:

1. **customized**
2. **predefined**

params:

1. customized

va1, val2, ..., valn such as “1,2,3,4,5,6,7,8,9,10”

2. predefined

a. **ten-equal-width-zscore** such as “ten-equal-width-zscore”

b. **ten-equal-width** such as “ten-equal-width”

(2) Binary Feature

A. disc2disc

The following is an example of xml format which is used for configuring disc2disc feature transform.

```
<disc2disc>
  <op-ctx in_gname1="all" in_fname1="%in_game1"
    in_gname2="all" in_fname2="%in_game2"
    out_gname="!cross^in_gname1^in_gname2"
    out_fname="!cross^in_fname1^in_fname2"
    op="crossall" scope="gname" procs="ad-inner"/>
</disc2disc>
```

in_fname:

Just configure symbol “=” or string “=your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

out_gname:

Just configure symbol “!” or string “!your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

out_fname:

Just configure symbol “!” or string “!your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

scope:

1. **gname**
2. **fname**

op:

1. **cross**
2. **crossall**

procs:

1. The procs of op **crossall**:

1. **ad-inner**
2. **uc-inner**
3. **ad2uc**
4. **ignore_uc_gnames**
5. **ignore_ad_gnames**
6. **ignore_uc_fnames**
7. **ignore_ad_fnames**
8. **ignore_double_cross**

2. The procs of **cross**:

using symbol “-”

params:

The procs of op **crossall**:

1. **ad-inner**

using symbol “-”

2. **uc-inner**

using symbol “-”

3. **ad2uc**

using symbol “-”

4. **ignore_uc_gnames**

name1, name2, ..., namen such as “sex,age”

5. **ignore_ad_gnames**

name1, name2, ..., namen such as “sex,age”

6. **ignore_uc_fnames**

name1, name2, ..., namen such as “sex,age”

7. **ignore_ad_fnames**

name1, name2, ..., namen such as “sex,age”

8. **ignore_double_cross**

using symbol “-”

B. **cont2cont**

The following is an example of xml format which is used for configuring cont2cont feature transform.

```
<cont2cont>
  <op-ctx in_gname1="user_cost#log" in_fname1=="in_game1/user_cost#log"
    in_gname2="ad_cost#log" in_fname2=="in_game2/ad_cost#log"
    out_gname="!cross^in_gname1^in_gname2/!comp^user_cost#log^ad_cost#log"
    out_fname="!cross^out_gname/!comp$avg_dot^user_cost#log^ad_cost#log"
    op="comp" scope="fname"
```

```
procs="partial" params="avg_min, avg_max, avg_dot"/>
</cont2cont>
```

in_fname:

Just configure symbol “=” or string “=your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

out_gname:

Just configure symbol “!” or string “!your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

out_fname:

Just configure symbol “!” or string “!your_comments” since the program just parse first char of the string and the left chars of the string are ignored.

scope:

1. **gname**
2. **fname**

op:

comp

procs:

1. **partial**
2. **all**

params:

1. **partial**

a combination of one or more of (**avg_min**, **avg_max**, **avg_dot**) such as “avg_min,avg_max”, “avg_min, avg_max, avg_dot” and so on.

2. all

no parameter

Finally, there is a complete feature transform example of xml format as follow:

```
<?xml version="1.0" encoding="utf-8"?>

<aucimpr-feat-trans>

  <user-and-ctx>
    <unary-trans>
      <cont2cont>
        <op-ctx in_gname="user_cost"
          in_fname="=in_gname/user_cost"
          out_gname="#in_fname/user_cost#log"
          out_fname="=out_gname/user_cost#log"
          op="log" scope="fname" procs="log10"/>
      </cont2cont>

      <cont2disc>
        <op-ctx in_gname="poi_cpt"
          in_fname="%in_gname/poi_cpt%ent"
          out_gname="#in_fname/poi_cpt%ent#bkt"
          out_fname="@out_gname/poi_cpt%ent#bkt@1"
          op="bucketize" scope="gname" procs="predefined"
          params="ten-equal-width"/>
        <op-ctx in_gname="user_age"
          in_fname="=in_gname/user_age"
          out_gname="#in_fname/user_age#bkt"
          out_fname="@out_gname/user_age#bkt@1"
          op="bucketize" scope="fname" procs="customized"
          params="0,18,25,30,40,45,50"/>
      </cont2disc>
    </unary-trans>
```

```

<binary-trans>
  <disc2disc>
    <op-ctx in_gname1="user_sex"
      in_fname1="%in_game1/user_sex%male"
      in_gname2="user_age#bkt"
      in_fname2="%in_game2/user_age#bkt@1"
      out_gname="!cross^in_gname1^in_gname2/!cross^user_sex^user_a
      out_fname="!cross^in_fname1^in_fname2/!cross^user_sex%male^u
      op="cross" vscope="gname"/>

    <op-ctx in_gname1="all"
      in_fname1="%in_game1"
      in_gname2="all"
      in_fname2="%in_game2"
      out_gname="!cross^in_gname1^in_gname2"
      out_fname="!cross^in_fname1^in_fname2"
      op="crossall" scope="gname" procs="uc-inner"/>
    </disc2disc>
  </binary-trans>
</user-and-ctx>

<ad>
  <unary-trans>
    <cont2cont>
      <op-ctx in_gname="ad_cost"
        in_fname="=in_gname/ad_cost"
        out_gname="#in_fname/ad_cost#log"
        out_fname="=out_gname/ad_cost#log"
        op="log" scope="fname" procs="log"/>
    </cont2cont>

    <cont2disc>
      <op-ctx in_gname="ad_revenue"
        in_fname="=in_gname/ad_revenue"
        out_gname="#in_fname/ad_revenue#bkt"
        out_fname="@out_gname/ad_revenue#bkt@1"

```

```

        op="bucketize" scope="fname" procs="predefined"
        params="ten-equal-width-zscore"/>
    </cont2disc>

</unary-trans>

<binary-trans>

    <cont2cont>
        <op-ctx in_gname1="user_cost#log"
            in_fname1="=in_game1/user_cost#log"
            in_gname2="ad_cost#log"
            in_fname2="=in_game2/ad_cost#log"
            out_gname="!cross^in_gname1^in_gname2/!comp^user_cost#log^ac
            out_fname="!cross^out_gname/!comp$avg_dot^user_cost#log^ad_c
            scope="fname"
            procs="partial" params="avg_min, avg_max, avg_dot"/>
    </cont2cont>

    <disc2disc>
        <op-ctx in_gname1="all" in_fname1="%in_game1" in_gname2="all"
            in_fname2="%in_game2"
            out_gname="!cross^in_gname1^in_gname2"
            out_fname="!cross^in_fname1^in_fname2"
            op="crossall" scope="gname" procs="ad-inner"/>

        <op-ctx in_gname1="all" in_fname1="%in_game1" in_gname2="all"
            in_fname2="%in_game2"
            out_gname="!cross^in_gname1^in_gname2"
            out_fname="!cross^in_fname1^in_fname2"
            op="crossall" scope="gname"
            procs="ad2uc;ignore_uc_gnames;ignore_double_cross"
            params="-;feat1_grp>true"/>
    </disc2disc>

</binary-trans>

```

</ad>

</aucimpr-feat-trans>

Build Library

It is simple to build the library.

1. Copy ExternallibConfig.cmake.in.example to ExternallibConfig.cmake.in and modify the path of dependent libraries.
2. Run the following build script.

```
build_type=Debug # Debug/Release/Profile  
bash shell/build ${build_type}
```

We use the following versions.

- gcc-7
- g++-7
- googletest release-1.8.1 (Set INSTALL_GTEST in googletest to OFF)
- spdlog v1.2.1
- boost v1.65.1
- eigen v3.3.7
- abseil master 7b46e1d (HEAD has bug as of 2019.01.07)
- swig v3.0.12
- java 1.8