AIBench
User Manual

ICT, Chinese Academy of Sciences

Contacts (Email):

Prof. Jianfeng Zhan,
zhanjianfeng@ict.ac.cn

Table of Contents

1 Introduction 4
1.1 Context 4
1.2 Environment 5
1.3 Format Specification 6
2 Overview of Software Packages and Workloads 7
3 Installation and Configuration of Software 8
3.1 Setting up TensorFlow 8
3.2 Setting up PyTorch 8
4 Framework and Workloads 9
4.1 AIBench Framework 9
4.1.1 Framework Overview 9
4.1.2 Framework Deployment 9
4.1.2.1 Deploy Ansible 9
4.1.3 How to Use 9
4.2 Application Benchmarks 9
4.2.1 E-commerce Search 10
4.2.1.1 Introduction 10
4.2.1.2 The Design and Implementation of E-commerce Search 10
4.2.1.3 How to Use 11
4.2.2 DCMix 12
4.2.2.1 Introduction 12
4.2.2.2 DCMix Framework 12
4.2.2.3 How to use 13
4.3 Component Benchmarks 14
4.3.1 Image Classification 14
4.3.2 Image Generation 15
4.3.3 Text-to-Text Translation 16
4.3.4 Image to Text 17
4.3.5 Image to Image 19
4.3.6 Speech to Text 20
4.3.7 Face Embedding 23
4.3.8 3D Face Recognition 24
4.3.9 Object Detection 25
4.3.10 Recommendation 26
4.3.11 Video Prediction 27
4.3.12 Image compression 28
4.3.13 3D Object Reconstruction 29
4.3.14 Text Summarization 29
4.3.15 Spatial Transformer 30

4.3.16 Learning to Rank 30
4.4 Micro Benchmarks **31**
4.4.1 Convolution 31
4.4.2 Fully Connected 32
4.4.3 Relu, Sigmoid, Tanh 33
4.4.4 MaxPooling, AvgPooling 34
4.4.5 CosineNorm, BatchNorm 35
4.4.6 Dropout, Softmax 36

1 Introduction

1.1 Context

Today's Internet Services are undergoing fundamental changes and shifting to an intelligent computing era where AI is widely employed to augment services. In this context, many innovative AI algorithms, systems, and architectures are proposed, and thus the importance of benchmarking and evaluating them rises. However, modern Internet services adopt a micro-service based architecture and consist of various modules. The diversity of these modules and complexity of execution paths, the massive scale and complex hierarchy of datacenter infrastructure, the confidential issues of data sets and workloads pose great challenges to benchmarking.

First, the real-world data sets and workloads from Internet services are treated as first-class confidential issues by their providers, and they are isolated between academia and industry, or even among different providers. However, there are only a few publicly available performance model or observed insights about industry-scale Internet services that can be leveraged for further research. As there is no publicly available industry-scale Internet service benchmark, the state-of-the-art and state-of-the-practice are advanced only by the research staffs inside Internet service providers, which is not sustainable and poses a huge obstacle for our communities towards developing an open and mature research field.

Second, AI has infiltrated into almost all aspects of Internet services, ranging from offline analytics to online service. Thus, to cover the critical paths and characterize prominent characteristics of a realistic AI scenario, end-to-end application benchmarks should be provided [8, 9]. Meanwhile, there are many classes of Internet services. Modern Internet services workloads expand and change very fast, and it is not scalable or even impossible to create a new benchmark or proxy for every possible workload. Moreover, data sets have great impacts on system and microarchitectural characteristics, so diverse data inputs should be considered. So we need identify representative data sets, abstract the prominent AI problem domains (component benchmarks), and further understand what are the most intensive units of computation (micro benchmarks), on the basis of which, we can build a concise and comprehensive AI benchmark framework.

Finally but not least, from an architectural perspective, porting a full-scale AI applications to a new architecture at an earlier stage is difficult and even impossible, while using micro or component benchmarks alone are insufficient to discover the time breakdown of different modules and locate the bottleneck within a realistic AI application scenario at a later stage. Hence, a realistic AI benchmark suite should have the ability to run not only collectively as a whole end-to-end application to discover the time breakdown of different modules but also individually as a micro or component benchmark for fine tuning hot spot functions or kernels. So an industry standard Internet service AI benchmark suite consisting of a full spectrum of micro or component benchmarks and an end-to-end application benchmark is of great significance to bridge this huge gap.

We specify the common requirements of Big Data and AI workloads only algorithmically in a paper-and-pencil approach, reasonably divorced from individual implementations. We capture the differences and collaborations among IoT, edge, datacenter and HPC in handling Big Data and AI workloads. We consider each big data and AI workload as a pipeline of one or more classes of units of computation performed on initial or intermediate data inputs, each of which we call a data motif. For the first time, among a wide variety of big data and AI workloads, we identify eight data motifs (PACT'18 paper)— including Matrix, Sampling, Logic, Transform, Set, Graph, Sort and Statistic computation, each of which captures the common requirements of each class of unit of computation. Other than creating a new benchmark or proxy for every possible workload, we propose using data motif-based benchmarks—the combination of one or more data motifs—to represent diversity of big data and AI workloads.

AIBench is the first industry scale AI benchmark suite, joint with seventeen industry partners. First, we present a highly extensible, configurable, and flexible benchmark framework, containing multiple loosely coupled modules like data input, prominent AI problem domains, online inference, offline training and automatic deployment tool modules. We analyze typical AI application scenarios from three most important Internet services domains, including search engine, social network, and e-commerce, and then we abstract and identify sixteen prominent AI problem domains, including *classification, image generation, text-to-text translation, image-to-text, image-to- image, speech-to-text, face embedding, 3D face recognition, object detection, video prediction, image compression, recommendation, 3D object reconstruction, text summarization, spatial transformer, and learning to rank.* We implement sixteen component benchmarks for those AI problem domains, and further profile and implement twelve fundamental units of computation across different component benchmarks as the micro benchmarks. On the basis of the AIBench framework, we design and implement the first end-to-end Internet service AI benchmark with an underlying e-commerce searching business model. As a whole, it covers the major modules and critical paths of an industry scale e-commerce provider. The application benchmark reuses ten component benchmarks from the AIBench framework, receives the query requests and performs personalized searching, recommendation and advertising, integrated with AI inference and training. The data maintains the real-world data characteristics through anonymization. Data generators are also provided to generate specified data scale, using several configurable parameters.

1.2 Environment

This document presents user manual information on AIBench – including a brief introduction and the setting up guidelines of AI software stacks, and operating guide of all workloads in AIBench. The information and specifications contained are for researchers who are interested in AI benchmarking.

Note that the user manual information in the following passage are tested in the environment as follows.

Recommended browner: IE or Chrome.

Recommended OS : Centos 6.0 or later.

Libraries:

JDK 1.6 or later (Recommend version: jdk1.8.0_65)

C compiler, such as gcc, and C++ compiler, such as g++.

1.3 Format Specification

The following typographic conventions are used in this user manual:

Convention	Description
Bold	Bold for emphasis.
Italic	Italic for fold and file names.
$command	$command for command lines.
Contents	Contents for contents in configuration files.
Courier font	Courier font for screen output.
Footnote	Some exception explanations are put in footnote.

2 Overview of Software Packages and Workloads

Benchmark	Benchmark Type	Algorithm	Dataset	Software stacks
Convolution	Micro Benchmark	Convolution	Cifar, ImageNet	TensorFlow, Pthread, PyTorch
Fully Connected		Fully Connected	Cifar, ImageNet	TensorFlow, Pthread, PyTorch
Relu		Relu	Cifar, ImageNet	TensorFlow, Pthread, PyTorch
Sigmoid		Sigmoid	Cifar, ImageNet	TensorFlow, Pthread, PyTorch
Tanh		Tanh	Cifar, ImageNet	TensorFlow, Pthread, PyTorch
MaxPooling		MaxPooling	Cifar, ImageNet	TensorFlow, Pthread, PyTorch
AvgPooling		AvgPooling	Cifar, ImageNet	TensorFlow, Pthread, PyTorch
CosineNorm		CosineNorm	Cifar, ImageNet	TensorFlow, Pthread, PyTorch
BatchNorm		BatchNorm	Cifar, ImageNet	TensorFlow, Pthread, PyTorch
Dropout		Dropout	Cifar, ImageNet	TensorFlow, Pthread, PyTorch
Element-wise OP		Element-wise OP	Cifar, ImageNet	TensorFlow, Pthread, PyTorch
Softmax		Softmax	Cifar, ImageNet	TensorFlow, Pthread, PyTorch
Image Classification	Component Benchmark	ResNet50	ImageNet	TensorFlow, PyTorch
Image Generation		WGAN	LSUN	TensorFlow, PyTorch
Text-to-Text Translation		RNN	WMT English-German	TensorFlow, PyTorch
Image-to-Text		Neural image caption model	MS COCO dataset	TensorFlow, PyTorch
Image-to-Image		CycleGAN	Cityscapes	TensorFlow, PyTorch
Speech-to-Text		DeepSpeech2	Librispeech	TensorFlow, PyTorch
Face embedding		Facenet	Labeled faces in the wild	TensorFlow, PyTorch
3D Face Recognition		3D face models	77,715 samples from 253 face IDs	TensorFlow, PyTorch
Object detection		Faster R-CNN	Microsoft COCO	TensorFlow, PyTorch
Recommendation		Collaborative filtering	MovieLens	TensorFlow, PyTorch
Video Prediction		Motion-focused predictive models	Robot pushing dataset	TensorFlow, PyTorch
Image Compression		RNN	ImageNet	TensorFlow, PyTorch
3D Object Reconstruction		Convolutional encoder-decoder network	ShapeNet dataset	TensorFlow, PyTorch

Text summarization		Sequence-to-squence model	Gigaword dataset	TensorFlow, PyTorch
Spatial transformer		Spatial transformer network	MNIST	TensorFlow, PyTorch
Learning to rank		Ranking distillation	Gowalla	TensorFlow, PyTorch
DCMix	Application Benchmark	Datacenter	Mixed	Mixed
E-commerce Search		Internet Service	Alibaba	Alibaba framework

3 Installation and Configuration of Software

3.1 Setting up TensorFlow

1) Prerequisites

python, pip, numpy, scipy

2) Download and install TensorFlow

We recommend TensorFlow 1.1.0 version.

```
$ pip install --upgrade
https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-1.1.0-cp27-none-linux_x86_64.whl
```

3) Test TensorFlow

```
$ python
>>>import tensorflow
>>>
```

If command“import tensorflow” doesn’t return errors, then TensorFlow is successfully installed.

3.2 Setting up PyTorch

We recommend PyTorch 1.0.1

1) Prerequisites

python, pip, numpy, scipy

2) Download and install PyTorch

pip install with Python 2.x version:

```
pip3 install torch torchvision
```

pip install with Python 3.x version:

```
pip install torch torchvision
```

conda install:

```
conda install pytorch torchvision -c pytorch
```

conda install with specific cuda version:

```
conda install pytorch torchvision cudatoolkit=10.0 -c pytorch
```

install from source:

Follow instructions at this URL: https://github.com/pytorch/pytorch#from-source

3) Test PyTorch

```
$ python
>>>import torch
```

```
>>>
```

If command"import torch" doesn't return errors, then PyTorch is successfully installed.

4 Framework and Workloads

4.1 AIBench Framework

4.1.1 Framework Overview

AIBench framework can help you deploy AIBench to multiple nodes quickly and scalable. It supports Ansible(https://www.ansible.com/
) now and will support docker-compose and Kubernetes soon.

Ansible will install the required software and configure parameters in nodes automatically.

4.1.2 Framework Deployment

4.1.2.1 Deploy Ansible

1) Prerequisites
   We tested it at Centos 7, but Centos 6, RedHat 6/7, Ubuntu 14.04/16.04 should also be supported.
2) Install Ansible
   Please refer to https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html to install Ansible, and ensure that the controlled machines have installed python 2.
3) Ensure SSH login without password
   You can refer to https://www.thegeekstuff.com/2008/11/3-steps-to-perform-ssh-login-without-password-using-ssh-keygen-ssh-copy-id to configure SSH login without password from the control machine to other controlled machines.

4.1.3 How to Use

1) Replace the variables with correct hostname or IP according your needs in DeploymentToolModule/Ansible/group_vars/all file. The variables to be replaced have been surrounded by curly braces and begin with $.

In DeploymentToolModule/Ansible directory, run

```
$ ansible-playbook -i hosts site.yml
```

Wait for the command to complete, and the AIBench online module should have started running.

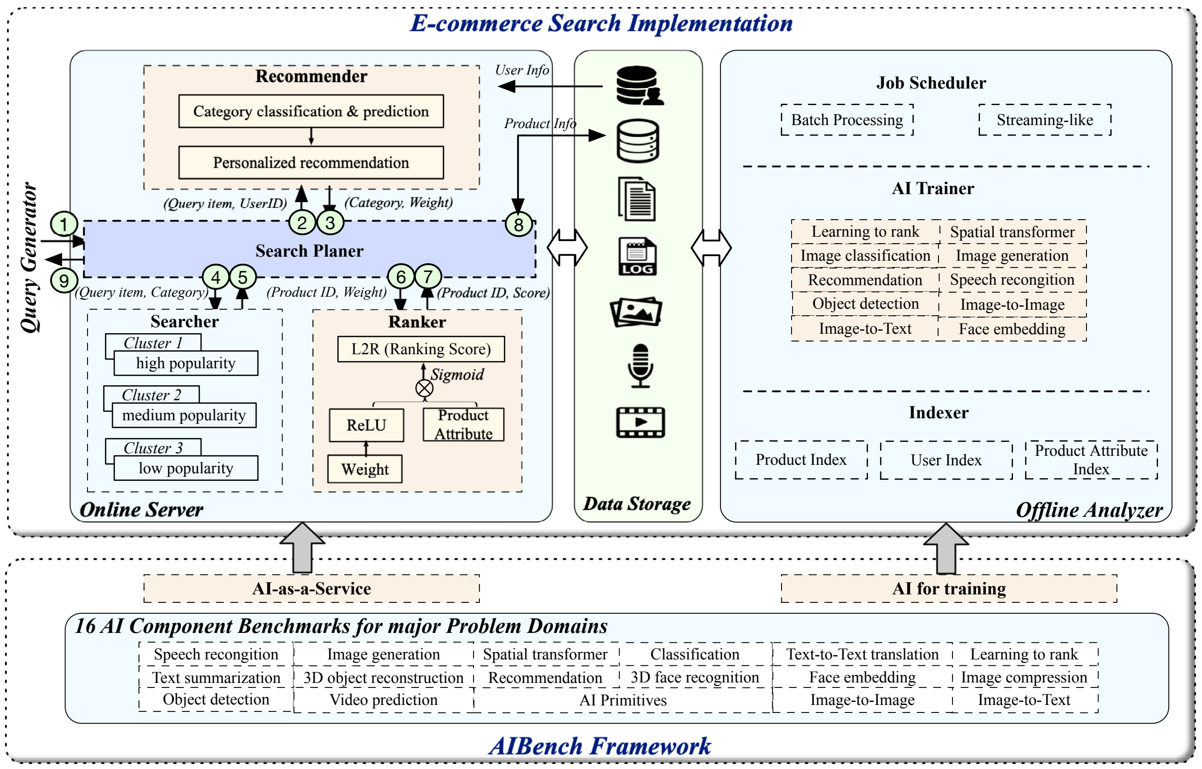
4.2 Application Benchmarks

4.2.1 E-commerce Search

4.2.1.1 Introduction

E-commerce Search is the first end-to-end AI application benchmark, modelling the complete use-cases of a realistic E-commerce search intelligence.

4.2.1.2 The Design and Implementation of E-commerce Search



The E-commerce Search benchmark consists of four modules: online server, offline analyzer, query generator, and data storage. Among them, online server receives the query requests and performs personalized searching and recommendation, integrated with AI inference.

Offline analyzer chooses the appropriate AI algorithm implementations and performs a training stage to generate a learning model. Also, the offline analyzer is responsible to build data indexes to accelerate data access.

Query generator is to simulate concurrent users and send query requests to online server based on a specific configuration. The configuration designates parameters like concurrency, query arriving rate, distribution, and user thinking time, to simulate different query characteristics and satisfy multiple generation strategies. We implement our query generator based on JMeter.

Data storage module stores all kinds of data, including the user database that saves all the attributes of user information, the product database that holds all the attributes of product information, logs that record the complete query histories, text data that contains the product description text or the user comments, image and video data that depict the appearance and usage of product vividly, and audio data that stores the voice search data and voice chat data. Overall, the data storage covers various data types including structured, unstructured, and semi-structured data, and diverse data sources, including table, text, image, audio and video.

To support scalable deployment on the clusters with different scales, each module is able to deploy on multiple nodes. Also, a series of data generators are provided to generate the e-commerce data with different scales, through setting several parameters, e.g., the number of products and product attribute fields, the number of users and user attribute fields.

4.2.1.3 How to Use

Online server provides personalized searching and recommendations combining traditional machine learning and deep learning technologies. Online server consists of four submodules including search planer, recommender, searcher, and ranker.

Search Planer is the entrance of online server. Recommender is to analyze the query item and pro- vides personalized recommendation, according to the user information obtained from the user database. Ranker uses the weight returned by Recommender as initial weight, and ranks the scores of products through a personalized L2R neural network. Search Planer returns the searched product information.

We use the Neo4j to store user data, and ElasticSearch to store the index of searcher and ranker, and product data. The workload-generator directory contains the Jmeter binary which is used as our workload generator. And the recommender contains two parts, one is a web service application built-in Flask framework, and the other one is a Tensorflow model serving which is for AI inference.

To get the e-commerce benchmark, you can download it from http://www.benchcouncil.org/AIBench/files/AIBench_Application_Benchmark.tar.gz.

1. Setup the user info database.
    a) Uncompress the neo4j-community-3.5.8-unix.tar.gz in the user-info directory.
    b) Start the neo4j using "neo4j-community-3.5.8/bin/neo4j console".
2. Setup the searcher, ranker, and product info database. There are three databases for storing the index of the searcher to mimic different popular distributions of products in the real world. However, for simplicity, we all use ElasticSearch to store the above data, and we can only set up an ElasticSearch instance to store all those data. To scale to multi nodes, you can set up one ElasticSearch cluster for each module.
    a) Uncompress the elasticsearch-oss-6.5.2.tar.gz in any directory of product-info, ranker, searcher.
    b) Start the ElasticSearch using "elasticsearch-6.5.2/bin/elasticsearch".
    c) Set up the index using the init_database.sh in all directory of product-info, ranker, searcher.
3. Setup the recommender-serving.
    a) You can use the native TensorFlow model serving and load the model named ranking_weights_model in the recommender-serving directory.
    b) Or you can use our provided Dockerfile to build a new docker image, and suppose the image name is tf-serving, you can start the recommender-serving module using "docker run -t --rm -p 8501:8501 -e MODEL_NAME=ranking_weights_model".
4. Setup the recommender-web.
    a) For simplicity, we use pipenv to manage the python virtual environment. You can initialize the python environment using "pipenv install" and then activate the environment using "pipenv shell".
    b) To start the recommender-web, just using "python recommender.py".
    c) Notice: the user info database must be initialized before the recommender-web, otherwise the recommender-web module won't start.
5. Setup the search-planer.
    a) We use the Spring framework to build the module and gradle to manage the module. You must set the correct host in the config file: application.properties and factorybean-spring-ctx.xml in the search-planer/src/main/resources/ directory.
    b) Simply run "./gradlew bootRun" in the search-planer directory to start the search-planer module.

c) Or you can also package the source code to a bootable jar using "./gradlew bootJar", and the bootable jar will be output in the build/libs directory. Then you can start the module using java -jar theBootableJarName.jar to start the module.

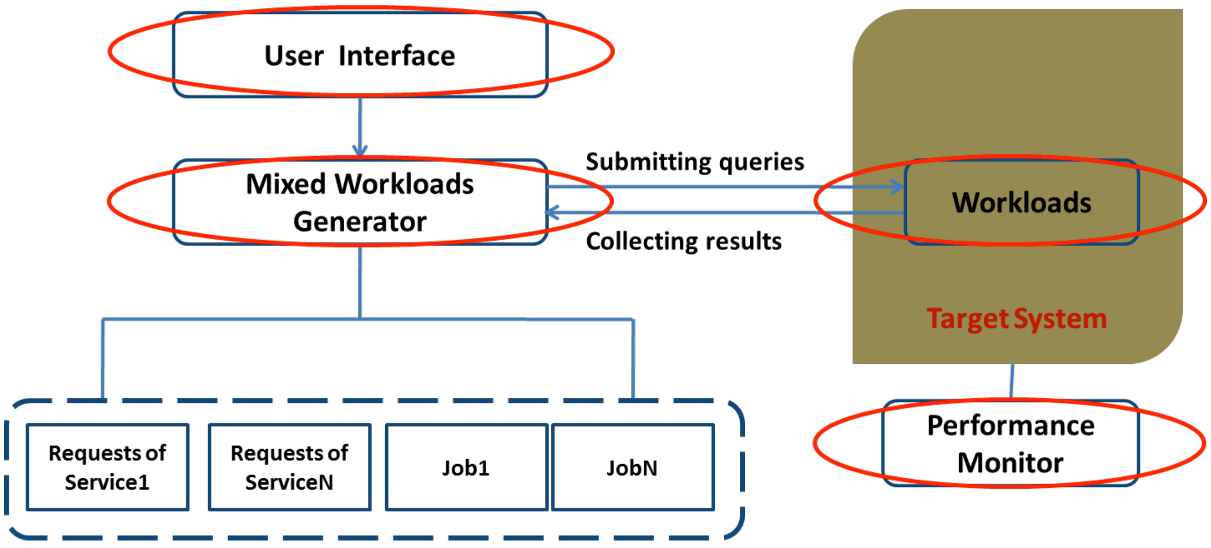
6. Populate the database.
   a) Using the benchmark-cli tool to populate the database. The tool can populate user-info, searcher, ranker, and product-info database.
7. Run the workload generator.
   b) We use JMeter as our workload generator to send the workload to the online AIBench.

4.2.2 DCMix

4.2.2.1 Introduction

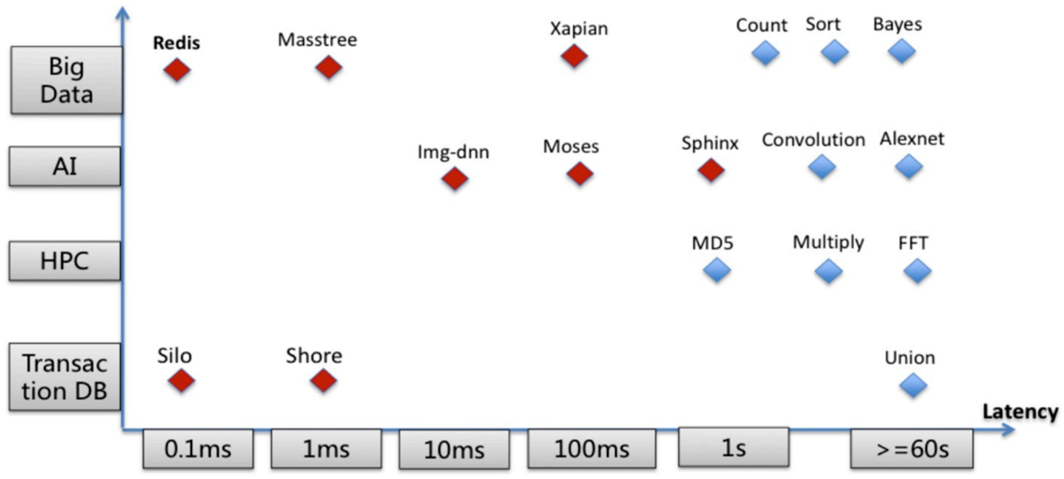
Modern datacenter computer systems are widely deployed with mixed workloads to improve system utilization and save cost. However, the throughput of latency-critical workloads is dominated by their worst-case performance-tail latency. To model this important application scenario, we propose an end-to-end application benchmark---DCMix to generate mixed workloads whose latencies range from microseconds to minutes with four mixed execution modes.

4.2.2.2 DCMix Framework



There are four main modules: Workloads, User interface, mixed workloads generator, and Performance monitor. DCMIX contains two types of workloads: online service and data analytic workloads and they are all deployed on the target system. User interface is the portal for user; users can specify their workload mix requirements, including workloads and mixture patterns. Mixed workloads generator can generate the mixed workloads through submitting queries (service requests queries and data analytics job submitting queries). Performance monitor can monitor the performance data of the target system, and the system entropy is calculated by these original monitor data.

1) Workload Overview



DCMIX contains two types of workloads: online service and data analytic workloads. These workloads have different application fields and different user experience (latency). DCMIX's application fields are big data, artificial intelligence, high-performance computing, transaction processing databases, et al. The latencies of DCMIX workloads range from microseconds to minutes.

2) Mixed Workload Generator

Mixed workloads generator can generate the mixed workloads through submitting queries (service requests queries and data analytics job submitting queries). Mixed workloads generator supports the mixture execution of serial execution and parallel execution. Serial execution means that the workload must start up after the previous workload complete. Parallel execution means that multiple workloads start up at the same time. Moreover, in the workload generator configuration file, users can set request configurations for each workload. For online-services, we provided request intensity, number of requests, number of warm-up requests, etc.; for offline-analytics, we provide path of the data set, threads number of jobs, etc.

4.2.2.3 How to use

Step 1. Required software

Python, gcc, gcc-c++, make, automake, autoconf, epel-release, libtool, libuuid,e2fsprogs, opencv, bison, swig, boost-devel, realine-devel,libdb-cxx-devel,numactl-devel,libaio-devel

Step 2. Download DCMix

You can download DCMIX via http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/DCMIX.tar.gz

Step 3. Install DCMix

1. To install tailbench workloads (i.e, online services):

```
$ cd tailbench-v0.9
$ bash ./build.sh
```

2. To install dwarf workloads (i.e, offline applications):

```
$ cd dwarf-set
$ bash ./build.sh
```

Step 4. Prepare the input data

1. To get tailbench dataset:

```
$ mkdir -p tailbench-data
$ wget -c http://tailbench.csail.mit.edu/tailbench.inputs.tgz
```

2. To generate dwarf dataset:

```
$ g++ -std=c++11 gen-data.cpp -o gen-data
$ ./gen-data
```

Step 5. Run DCMix Workload

1. To run tailbench workloads, let's take xapian as an example:

```
$ cd tailbenchv-0.9/xapian
$ ./run_xapian_server.sh
$ ./run_xapian_client.sh
```

You can set the request parameters in the above 2 script files.

2. To run dwarf workloads:

```
$ ./run_all.sh
```

4.3 Component Benchmarks

Considering the benchmarking scalability, we use the motif combinations to compose original complex workloads with a DAG-like structure considering the data pipeline. The DAG-like structure is to use a node representing original or intermediate data set being processed, and an edge representing a data motif.

4.3.1 Image Classification

1) TensorFlow based

Step 1. Required Software Stacks

1. tensorflow-gpu 1.12 or tensorflow 1.12

   ```
   $pip install tensorflow-gpu==1.12 or pip install tensorflow==1.12
   ```

   if you want to build tensorflow from source, see
   https://www.tensorflow.org/install/source
2. Cuda 9.0

   Downloads cuda9.0: https://developer.nvidia.com/cuda-90-download-archive
   Installation guide for linux: https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html
3. Cudnn 7.4.2

   Downloads cudnn 7.4.2: https://developer.nvidia.com/cudnn
   Installation guide for linux: https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd TensorFlow/Image_classification
```

Step 3. Prepare the input

1. Download ImageNet ILSVRC2012 Dataset from http://www.image-net.org/
2. Convet these raw images to TFRecords by using build_imagenet_data.py script.

Step 4. Run the workloads

```
$python imagenet_main.py –data_dir=/path/to/imagenet
```

Both the training dataset and the validation dataset are in the same directory. The model will begin training and will automatically evaluate itself on the validation data roughly once per epoch.

Some running options:

--model_dir: to choose where to store the model

--resnet_size: to choose the model size (options include ResNet-18 through ResNet-200)

--num-gpus: to choose computing device

0: Use OneDeviceStrategy and train on CPU

1: Use OneDeviceStrategy and train on GPU

2+: Use Mirroredstrategy (data parallelism) to distribute a batch between devices

Full list of options, see resnet_run_loop.py

Step 5. Collect the running results

When the workload run is complete, it will display the output.

2) PyTorch based

Step 1. Required Software Stacks

1. Python 2.7
2. Anaconda 5.3.0

```
curl –O https://repo.anaconda.com/archive/Anaconda2-5.3.0-Linux-x86_64.sh
sh Anaconda2-5.3.0-Linux-x86_64.sh
```

3. Pytorch 1.0

```
conda install pytorch torchvision cudatoolkit=9.0 -c pytorch
```

(https://pytorch.org/get-started/locally/)

Step 2. Get workloads from AIBench

Download the Benchmark from this link:

http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd PyTorch/Image_classification
```

Step 3. Prepare the input

1. Download the ImageNet dataset
2. Move validation images to labeled subfolders, you can use the following script: https://raw.githubusercontent.com/soumith/imagenetloader.torch/master/valprep.sh

Step 4. Run the workload

```
bash run_image_classify ${batchSize} ${dataDir}
```

Step 5. Collect the running results

When the workload run is complete, it will display the output.

4.3.2 Image Generation

1) TensorFlow based

Step 1. Required Software Stacks

python 2.7

tensorflow >= 1.2 (verified on 1.2 and 1.3)
tqdm
Step 2. Get workloads from AIBench
Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd TensorFlow/Image_generation
```

Step 3. Prepare the input

```
$ ./prepareData.sh
```

Step 4. Run the workloads;

```
$ python main.py --dataset mnist --gan_type WGAN --epoch 5 --batch_size 64
```

Step 5. Collect the running results
When the workload run is complete, it will display the output.

2) PyTorch based
Step 1. Required Software Stacks
PyTorch
PyTorchvision
Step 2. Get workloads from AIBench
Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd PyTorch/Image_generation
```

Step 3. Prepare the input
The dataset we use is LSUN-bedroom. http://www.yf.io/p/lsun .
You can download the dataset by:

```
$ python3 lsun/download.py -o <data_dir> -c bedroom
```

Or you can also download the lsun dataset from http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_DataSet
Step 4. Run the workload

```
$ cd WGAN
$ python main.py --mlp_G --ngf 512 --dataset lsun --dataroot <lsun-train-folder>
--cuda
```

Step 5. Collect the running results
When the workload run is complete, it will display the output.

4.3.3 Text-to-Text Translation

1) TensorFlow based
Step 1. Required Software Stacks
tensorflow-gpu
tensorflow
Step 2. Get workloads from AIBench
Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd TensorFlow/Text_to_Text
```

Step 3. Prepare the input

Download tensor2tensor from https://github.com/tensorflow/tensor2tensor. And make sure you can access to the Internet. For compatibility you need to change the io file of python.

```
vim /usr/local/lib/python3.6/dist-packages/tensorflow/python/lib/io/file_io.py
```

Change:

```
def rename(oldname, newname, overwrite=False):
def rename_v2(src, dst, overwrite=False):
```

To:

```
def rename(oldname, newname, overwrite=True):
def rename_v2(src, dst, overwrite=True):
```

Step 4. Run the workloads;

```
$./run.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the output.

2) PyTorch based

Step 1. Required Software Stacks

PyTorch 1.0.1

Step 2. Get workloads from AIBench

Download the Benchmark from the link:

http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd PyTorch/Text_to_Text
```

Step 3. Prepare the input

```
$ sh download.sh
```

Or you can also find the dataset from:

http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_DataSet

Step 4. Run the workload

```
$ sh run.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the output.

```
[Info] Finished.
```

4.3.4 Image to Text

1) TensorFlow based

Step 1. Required Software Stacks

Bazel

Natural Language Toolkit (NLTK)

Unzip

Numpy

Tensorflow 1.0 or greater

Step 2. Get workloads from AIBench

Download the Benchmark from the link:

http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd TensorFlow/Image_to_Text
```

Step 3. Prepare the input

The dataset we use is COCO 2014. http://cocodataset.org/#download .
You can download the dataset via COCO webpage, or http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_DataSet.
After downloading the dataset, please execute the following command.

```
$ IM2TXT_HOME=/path/to/your/coco2014-dataset
$ # Directory containing preprocessed MSCOCO data.
$ MSCOCO_DIR="${IM2TXT_HOME}/im2txt/data/mscoco"
$ # Inception v3 checkpoint file.
$  INCEPTION_CHECKPOINT="${IM2TXT_HOME}/im2txt/data/inception_v3.ckpt"
$ # Directory to save the model.
$ MODEL_DIR="${IM2TXT_HOME}/im2txt/model"
$ # Build the model.
$ cd research/im2txt
$ bazel build -c opt //im2txt/..
```

Step 4. Run the workloads;

```
$ bazel-bin/im2txt/train \
  --input_file_pattern="${MSCOCO_DIR}/train-?????-of-00256" \
  --inception_checkpoint_file="${INCEPTION_CHECKPOINT}" \
  --train_dir="${MODEL_DIR}/train" \
  --train_inception=false \
  --number_of_steps=1000000
```

Step 5. Collect the running results

When the workload run is complete, it will display the output.

2) PyTorch based

Step 1. Required Software Stacks

torch
torchvision
matplotlib
nltk
numpy
Pillow
argparse
Cython
Scipy

```
$ pip install softwareName
```

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd PyTorch/Image_to_Text
```

Step 3. Prepare the input

This can be done by running `$ ./prepareData.sh` or following the steps bellow:

Download the dataset from BigDataBench or website

1. Download from BigDataBench at the folder of 'DataSet/coco2014'. Rename 'coco2014' as 'data' and then put 'data' in the folder of 'Image_to_Text'.
2. Download from the website by running:

```
$ ./download.sh
```

Preprocessing

```
$ python build_vocab.py
$ python resize.py
```

Step 4. Run the workload

This can be done by running `$ ./run_imageTotext.sh` or following the steps bellow:

Train the model

```
$ python train.py
```

Test the model

```
$ python sample.py --image='png/example.png'
```

Step 5. Collect the running results

When training the model, it will display the output:

```
Epoch [0/1], Step [0/3236], Loss: 9.2094, Perplexity: 9990.5262
Epoch [0/1], Step [10/3236], Loss: 5.8074, Perplexity: 332.7434
…
```

At each '--save_step', it will save the training model in the folder './models' with the name of 'encoder-{epoch}-{step}.ckpt' and 'decoder-{epoch}-{step}.ckpt'. The default '--num_epochs' is 5, '--save_step' is 1000.

Test the model, the output is something like the following sentence.

```
<start> a man is sitting on a tennis court . <end>
```

The default training model used is 'encoder-2-1000.ckpt' and 'decoder-2-1000.ckpt', which can be changed by '--encoder_path' and '--encoder_path'.

4.3.5 Image to Image

1) TensorFlow based

Step 1. Required Software Stacks

Pyton3

Tensorflow1.2

click (pip install click)

unzip

Step 2. Get workloads from AIBench

Download the Benchmark from the link:

http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd TensorFlow/Image_to_Image/CycleGAN
```

Step 3. Prepare the input

```
$ ./download_datasets.sh cityscapes
```

Step 4. Run the workloads;

Add export LC_ALL=C.UTF-8 export LANG=C.UTF-8 to /etc/profile

```
$./run.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the output.

2) PyTorch based

Step 1. Required Software Stacks

PyTorch

PyTorchvision

PyDominate

PyVisdom

Step 2. Get workloads from AIBench

Download the Benchmark from the link:

http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd PyTorch/Image_to_Image
```

Step 3. Prepare the input

The dataset we use is cityscapes.

You can download the dataset by:

```
$ bash ./datasets/download_cyclegan_dataset.sh cityscapes
```

Or you can also download the cityscapes dataset from http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_DataSet

Step 4. Run the workload

```
$ python train.py --dataroot ./datasets/ cityscapes --name cityscapes_cyclegan --model cycle_gan
```

Step 5. Collect the running results

When the workload run is complete, it will display the output.

4.3.6 Speech to Text

1) TensorFlow based

Step 1. Required Software Stacks

Python 2/3

Tensorflow>=1.1

```
$ pip/pip3 install -r requirements.txt
```

Step 2. Get workloads from AIBench

Download the Benchmark from the link:

http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd TensorFlow/Speech_to_Text
```

Step 3. Prepare the input

```
$ python data/download.py
```

Arguments:

--data_dir: Directory where to download and save the preprocessed data. By default, it is /tmp/librispeech_data.

Use the --help or -h flag to get a full list of possible arguments.

Step 4. Run the workloads;

```
$python deep_speech.py
```

Arguments:

--model_dir: Directory to save model training checkpoints. By default, it is /tmp/deep_speech_model/.

--train_data_dir: Directory of the training dataset.

--eval_data_dir: Directory of the evaluation dataset.

--num_gpus: Number of GPUs to use (specify -1 if you want to use all available GPUs).

There are other arguments about DeepSpeech2 model and training/evaluation process. Use the --help or -h flag to get a full list of possible arguments with detailed descriptions.

Step 5. Collect the running results

A shell script run_deep_speech.sh is provided to run the whole pipeline with default parameters. Issue the following command to run the benchmark:

```
sh run_deep_speech.sh
```

Note by default, the training dataset in the benchmark include train-clean-100, train-clean-360 and train-other-500, and the evaluation dataset include dev-clean and dev-other.

When the workload run is complete, it will display the output.

2) PyTorch based

Step 1. Required Software Stacks

PyTorch 1.0.1, Torchaudio, apex, warp-ctc bindings, flac, sox, tqdm, librosa, levenshtein.

Cuda 10.0+

Since the torchaudio has high compatibility requirements, we suggest using conda to create a new environment and install the speech_to_text workload. Otherwise, “import torchaudio” would report segmentation fault (core dumped). The installation processes are as follows:

i) create a new environment named imageText, note that you can change the name.

```
$ conda create –n imageText python
```

ii) activate the new environment

```
$ source activate imageText
```

iii) install PyTorch in the new environment

```
$ conda install pytorch torchvision -c pytorch
```

iv) install torchaudio

```
$ git clone https://github.com/pytorch/audio.git
$ cd audio && python setup.py install
```

v) install apex

```
$ git clone --recursive https://github.com/NVIDIA/apex.git
$ cd apex && pip install .
```

vi) install warp-ctc binding

```
$ git clone https://github.com/SeanNaren/warp-ctc.git
$ cd warp-ctc
$ mkdir build
$ cd build
$ cmake ..
$ make
```

Note that the conda environment may install the gcc 6+ version, while warp-ctc doesn't support gcc 6+, so you need to edit the CMakeLists.txt file to use old gcc version. Insert the following two lines in CMakeLists.txt (you need to change the path of old gcc version according to your environment) and then repeat the upper commands.

```
SET(CMAKE_C_COMPILER "/usr/bin/gcc4.8")
SET(CMAKE_CXX_COMPILER "/usr/bin/g++4.8")
```

vii) install pytorch_binding

```
$ cd warp-ctc/pytorch_binding
$ python setup.py install
```

viii) install flac

```
$ wget https://ftp.osuosl.org/pub/xiph/releases/flac/flac-1.2.1.tar.gz
$ tar –xf flac-1.2.1.tar.gz
$ cd flac-1.2.1
$ ./configure && make && make install
```

Note that if you encounter the error “main.cpp:75:27: error: 'memcmp' was not declared in this scope”, you need to insert “#include <string.h> ” in the file “examples/cpp/encode/file/main.cpp”.

ix) install sox

Download sox-14.4.2.tar.gz from
https://sourceforge.net/projects/sox/files/sox/14.4.2/sox-14.4.2.tar.gz/download

```
$ ./configure --with-lame --with-flac --with-libvorbis
$ make -s
$ make install
```

x) install tqdm, librosa, and levenshtein

```
$ pip install tqdm
$ pip install librosa
$ pip install python-levenshtein
```

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd PyTorch/Speech_to_Text
```

Step 3. Prepare the input

The workload use LibriSpeech dataset. Preprocess the dataset:

```
$ cd deepspeech.pytorch/data
$ mkdir LibriSpeech_dataset
$ mkdir LibriSpeech_dataset/test_clean
$ mkdir LibriSpeech_dataset/test_other
$ mkdir LibriSpeech_dataset/train
$ mkdir LibriSpeech_dataset/val
$ python librispeech.py
```

you can use the parameter --files-to-use to specify the dataset.

The command will generate four files under the data directory:

libri_test_clean_manifest.csv,
libri_test_other_manifest.csv,
libri_train_manifest.csv,
libri_val_manifest.csv

Step 4. Run the workload

Training use CPU:

```
$ python train.py --train-manifest data/libri_train_manifest.csv --val-manifest data/libri_val_manifest.csv
```

Training use GPU:

```
$ python train.py --train-manifest data/libri_train_manifest.csv --val-manifest data/libri_val_manifest.csv –cuda
```

Testing use CPU:

```
$ python test.py --model-path models/deepspeech_final.pth --test-manifest data/libri_test_clean_manifest.csv
```

Testing use GPU:

```
$ python test.py --model-path models/deepspeech_final.pth --test-manifest data/libri_test_clean_manifest.csv --cuda
```

Step 5. Collect the running results

When the workload run is complete, it will display the output.

4.3.7 Face Embedding

1) TensorFlow based

Step 1. Required Software Stacks

Tensorflow
Scipy
Scikit-learn
Opencv-python
H5py
Matplotlib
Pillow
Requests
Psutil

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd TensorFlow/Face_embedding
```

Step 3. Prepare the input

The dataset we use is VGGFace2. http://zeus.robots.ox.ac.uk/vgg_face2/login/ .

You can download dataset from VGGFace2 webpage, or you can also download the cityscapes dataset from:

http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_DataSet .

After downloading the dataset, you also need to perform the image alignment, which might take several hours.

```
$ ../scripts/face-align-VGGface2.sh
```

Step 4. Run the workloads;

```
$ ../scripts/cls_training_triplet_webface.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the output.

2) PyTorch based

Step 1. Required Software Stacks

Pytorch 1.0.1

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd PyTorch/Face_embedding/facenet
```

Step 3. Prepare the input

Rewrite `datasets/write_csv_for_making_dataset.py`， you need to change `which_dataset` and `root_dir`。

Step 4. Run the workload

```
$ python train.py
```

Step 5. Collect the running results

When the workload run is complete, it will display the output.

4.3.8 3D Face Recognition

1) TensorFlow based

Step 1. Required Software Stacks

Python 3.3+;
TensorFlow>1.6;

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd TensorFlow/3D_face_recognition
```

Step 3. Prepare the input

To prepare the data:

```
./align.sh
python preprocess/get_dataset_csv.py
```

Step 4. Run the workloads;

i) To train on a single machine:

```
./train.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the output.

2) PyTorch based

Step 1. Required Software Stacks

PyTorch

imageio
pandas
pillow
Scikit-image
scipy
tqdm
TensorboardX

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd PyTorch/3D_face_recognition
```

Step 3. Prepare the input

```
./align.sh
python preprocess/get_dataset_csv.py
```

Step 4. Run the workload

```
$ ./scripts/train.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the output.

4.3.9 Object Detection

1) TensorFlow based

Step 1. Required Software Stacks

Python 3.3+;
OpenCV;
TensorFlow>1.6;

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd TensorFlow/Object_detection
```

Step 3. Prepare the input

To prepare the data:
./prepareData.sh

Step 4. Run the workloads;

i) To train on a single machine:
./run_objectDetect.sh
ii) To run distributed training:
Set TRAINER=horovod in the config.py file
./run_objectDetect.sh

Step 5. Collect the running results

When the workload run is complete, it will display the output.

2) PyTorch based

Step 1. Required Software Stacks

PyTorch
CyThon
Cffi
Opencv-python
Scipy
Msgpack
Easydict
Matplotlib
Pyyarml
TensorboardX

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd PyTorch/Object_detection
```

Step 3. Prepare the input

The dataset we use is COCO 2014. http://cocodataset.org/#download .
You can download the dataset via COCO webpage, or http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_DataSet.
After downloading the dataset, please execute the following command.

```
$ # set your own coco dataset path
$ COCO_PATH=/your/path/to/coco2014
$ mkdir -p data data/pretrained_model
$ set -x
$ if [[ ! -d data/coco ]]; then
$ cd data
$ git clone https://github.com/pdollar/coco.git && cd coco/PythonAPI
$ make -j32 && cd ../../
$ cd ../
$ fi
$ if [[ ! -f data/coco/annotations || ! -h data/coco/annotations ]]; then
$    ln -sv $COCO_PATH/annotations data/coco/annotations
$ fi
$ if [[ ! -f data/coco/images  || ! -h data/coco/images ]]; then
$    ln -sv $COCO_PATH data/coco/images
$ fi
```

Step 4. Run the workload

```
$ ./scripts/train.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the output.

4.3.10 Recommendation

1) TensorFlow based

Step 1. Required Software Stacks

TensorFlow1.12.0

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd TensorFlow/Recommendation
```

Step 3. Prepare the input

```
$ cd official/recommendation/
$ ./run_pre_data.sh
```

Step 4. Run the workloads;

```
$  ./run.sh
```

Step 5. Collect the running results

When the workload run complete, it will display the running information and generate output file: /tmp/ncf/

2) Pytorch based

Step 1. Required Software Stacks

PyTorch
Unzip
Curl

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd PyTorch/Recommendation
```

Step 3. Prepare the input

```
$ cd ./training/recommendation/
$ ./download_dataset.sh
$ unzip ml-20m.zip
$ cd ../data_generation/fractal_graph_expansions/
$ pip install -r requirements.txt
$ ./data_gen.sh
```

Step 4. Run the workload

```
$ cd ../../recommendation/pytorch/
$ pip install -r requirements.txt
$ python convert.py ../ml-20mx1x1 --seed 0
$ ./run.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the running information.

4.3.11 Video Prediction

1) TensorFlow based

Step 1. Required Software

TensorFlow1.12.0

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd TensorFlow/Video_prediction
```
Step 3. Prepare the input & Step 4. Run the workload
```
$ ./download_data.sh
$ ./train.sh
```
Step 5. Collect the running results
When the workload run is complete, it will display the running information.

2) Pytorch based
Step 1. Required Software Stacks
PyTorch
Step 2. Get workloads from AIBench
Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component
```
$ cd DC_AIBench_Component
$ cd PyTorch/Video_prediction
```
Step 3. Prepare the input & Step 4. Run the workload
```
$ ./run.sh
```
Step 5. Collect the running results
When the workload run is complete, it will display the running information.

4.3.12 Image compression
1) TensorFlow based
Step 1. Required Software
TensorFlow1.12.0
Step 2. Get workloads from AIBench
Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component
```
$ cd DC_AIBench_Component
$ cd TensorFlow/Image_compression
```
Step 3. Prepare the input & Step 4. Run the workload
```
$ ./run.sh
```
Step 5. Collect the running results
When the workload run is complete, it will display the running information.

2) Pytorch based
Step 1. Required Software Stacks
PyTorch
Step 2. Get workloads from AIBench
Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component
```
$ cd DC_AIBench_Component
$ cd PyTorch/Image_compression
```
Step 3. Prepare the input & Step 4. Run the workload
```
$ ./run.sh
```
Step 5. Collect the running results

When the workload run is complete, it will display the running information.

4.3.13 3D Object Reconstruction

1) TensorFlow based

Step 1. Required Software

TensorFlow1.12.0

Step 2. Get workloads from AIBench

Download the Benchmark from the link:

http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd TensorFlow/3D_object_reconstruction
```

Step 3. Prepare the input & Step 4. Run the workload

```
$ ./train.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the running information.

2) Pytorch based

Step 1. Required Software Stacks

PyTorch

Step 2. Get workloads from AIBench

Download the Benchmark from the link:

http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd PyTorch/3D_object_reconstruction
```

Step 3. Prepare the input & Step 4. Run the workload

```
$ ./train.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the running information.

4.3.14 Text Summarization

1) TensorFlow based

Step 1. Required Software

TensorFlow1.12.0

Step 2. Get workloads from AIBench

Download the Benchmark from the link:

http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd TensorFlow/Text_summarization
```

Step 3. Prepare the input & Step 4. Run the workload

```
$ ./train.sh
$ ./test.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the running information.

2) Pytorch based

Step 1. Required Software Stacks

PyTorch

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd PyTorch/Text_summarization
```

Step 3. Prepare the input & Step 4. Run the workload

```
$ ./run.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the running information.

4.3.15 Spatial Transformer

1) TensorFlow based

Step 1. Required Software

TensorFlow1.12.0

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd TensorFlow/SpatialTransformer
```

Step 3. Prepare the input & Step 4. Run the workload

```
$ ./run.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the running information.

2) Pytorch based

Step 1. Required Software Stacks

PyTorch

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd PyTorch/SpatialTransformer
```

Step 3. Prepare the input & Step 4. Run the workload

```
$ ./run.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the running information.

4.3.16 Learning to Rank

1) TensorFlow based

Step 1. Required Software Stacks

TensorFlow1.12

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd TensorFlow/LearningRank
```

Step 3. Prepare the input & Step 4. Run the workload

```
$ ./run.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the running information.

2) Pytorch based

Step 1. Required Software Stacks

PyTorch

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Component

```
$ cd DC_AIBench_Component
$ cd PyTorch/LearningRank
```

Step 3. Prepare the input & Step 4. Run the workload

```
$ ./run.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the running information.

4.4 Micro Benchmarks

Data motifs are fundamental concepts and units of computation among a majority of big data and AI workloads. We design a suite of micro benchmarks, each of which is a single data motif implementation.

4.4.1 Convolution

1) TensorFlow based

Step 1. Required Software Stacks

Python 2 or 3
Scipy and Numpy
TensorFlow 1.0+

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Micro

Step 3. Prepare the input

The TensorFlow micro benchmarks use random generate data, you need to specify the ***batch size***, ***image size***, ***channel***, and ***filter size***.

Step 4. Run the workloads;

```
$ cd DC_AIBench_Micro/TensorFlow
$ python conv2d.py <batch_size> <img_size> <channel> <filter_size>
```

Running conv2d with scripts:

```
$ ./run-tensorflow.sh conv <datasize>
```

Parameter “datasize” can be large/medium/small.

large: 224*224*64 (means length, width and channel respectively)
medium: 112*112*128
small: 56*56*256

Step 5. Collect the running results

When the workload run complete, it will display the running time information.

2) Pthreads based

Step 1. Required Software Stacks

g++ compiler
OpenCV, recommend 3.2 version
Dependences: libopencv_core.so.3.2 and libopencv_imgproc.so.3.2

Step 2. Get workloads from AIBench

Download the Benchmark from this link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Micro

Step 3. Prepare the input

```
$ cd DC_AIBench_Micro/Pthread
```

The Pthread micro benchmarks use ImageNet as data input, *ImageData* directory contains the image data with three sizes: Image_1000, Image_10000 and Image_100000.

Step 4. Compile the workload

```
$ make
```

This command will produce an executable file named conv2d.

Step 5. Run the workload

```
$ ./conv2d ../ImageData/image_$imgsize/img$imgsize/ NCHW 12 227 227 100
```

Here $imgsize can be 1000, 10000, or 100000.

Step 6. Collect the running results

When the workload run is complete, it will display the output.

4.4.2 Fully Connected

1) TensorFlow based

Step 1. Required Software Stacks

Python 2 or 3
Scipy and Numpy
TensorFlow 1.0+

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Micro

Step 3. Prepare the input

The TensorFlow micro benchmarks use random generate data, you need to specify the ***batch size***, ***image size***, and ***channel***.

Step 4. Run the workloads;

```
$ cd DC_AIBench_Micro/TensorFlow
$ python matmul.py <batch_size> <img_size> <channel>
```

Running fully connected with scripts:

```
$ ./run-tensorflow.sh matmul <datasize>
```

Parameter “datasize” can be large/medium/small.
large: 224*224*64 (means length, width and channel respectively)
medium: 112*112*128
small: 56*56*256
Step 5. Collect the running results
When the workload run complete, it will display the running time information.

2) Pthreads based
Step 1. Required Software Stacks
g++ compiler
OpenCV, recommend 3.2 version
Dependences: libopencv_core.so.3.2 and libopencv_imgproc.so.3.2
Step 2. Get workloads from AIBench
Download the Benchmark from this link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Micro
Step 3. Prepare the input

```
$ cd DC_AIBench_Micro/Pthread
```

The Pthread micro benchmarks use ImageNet as data input, *ImageData* directory contains the image data with three sizes: Image_1000, Image_10000 and Image_100000.
Step 4. Compile the workload

```
$ make
```

This command will produce an executable file named matmul.
Step 5. Run the workload

```
$ ./matmul ../ImageData/image_$imgsize/img$imgsize/ 12 227 227 100
```

Here $imgsize can be 1000, 10000, or 100000.
Step 6. Collect the running results
When the workload run is complete, it will display the output.

4.4.3 Relu, Sigmoid, Tanh

We use the Relu as an example, the running processes of Sigmoid and Tanh are the same, you can just change the relu in the command to sigmoid or tanh.
1) TensorFlow based
Step 1. Required Software Stacks
Python 2 or 3
Scipy and Numpy
TensorFlow 1.0+
Step 2. Get workloads from AIBench
Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Micro
Step 3. Prepare the input
The TensorFlow micro benchmarks use random generate data, you need to specify the ***batch size***, ***image size***, and ***channel***.
Step 4. Run the workloads;

```
$ cd DC_AIBench_Micro/TensorFlow
$ python relu.py <batch_size> <img_size> <channel>
```

Running relu/sigmoid/tanh with scripts:

```
$ ./run-tensorflow.sh <workload> <datasize>
```

Parameter “workload” can be relu/sigmoid/tanh.
Parameter “datasize” can be large/medium/small.
large: 224*224*64 (means length, width and channel respectively)
medium: 112*112*128
small: 56*56*256

Step 5. Collect the running results

When the workload run complete, it will display the running time information.

2) Pthreads based

Step 1. Required Software Stacks

g++ compiler
OpenCV, recommend 3.2 version
Dependences: libopencv_core.so.3.2 and libopencv_imgproc.so.3.2

Step 2. Get workloads from AIBench

Download the Benchmark from this link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Micro

Step 3. Prepare the input

```
$ cd DC_AIBench_Micro/Pthread
```

The Pthread micro benchmarks use ImageNet as data input, *ImageData* directory contains the image data with three sizes: Image_1000, Image_10000 and Image_100000.

Step 4. Compile the workload

```
$ make
```

This command will produce an executable file named relu, sigmoid or tanh.

Step 5. Run the workload

```
$ ./relu ../ImageData/image_$imgsize/img$imgsize/ 12 227 227 100
```

Here $imgsize can be 1000, 10000, or 100000.

Step 6. Collect the running results

When the workload run is complete, it will display the output.

4.4.4 MaxPooling, AvgPooling

We use the MaxPooling as an example, the running process of AvgPooling is the same, you can just change the max_pool in the command to avg_pool.

1) TensorFlow based

Step 1. Required Software Stacks

Python 2 or 3
Scipy and Numpy
TensorFlow 1.0+

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Micro

Step 3. Prepare the input

The TensorFlow micro benchmarks use random generate data, you need to specify the ***batch size***, ***image size***, and ***channel***.

Step 4. Run the workloads;

```
$ cd DC_AIBench_Micro/TensorFlow
$ python max_pool.py <batch_size> <img_size> <channel>
```

Running MaxPooling and AvgPooling with scripts:

```
$ ./run-tensorflow.sh <workload> <datasize>
```

Parameter "workload" can be maxpool or avgpool.

Parameter "datasize" can be large/medium/small.

large: 224*224*64 (means length, width and channel respectively)

medium: 112*112*128

small: 56*56*256

Step 5. Collect the running results

When the workload run complete, it will display the running time information.

2) Pthreads based

Step 1. Required Software Stacks

g++ compiler

OpenCV, recommend 3.2 version

Dependences: libopencv_core.so.3.2 and libopencv_imgproc.so.3.2

Step 2. Get workloads from AIBench

Download the Benchmark from this link:

http://159.226.41.254:8090/AIBench/DC_AIBench_Micro

Step 3. Prepare the input

```
$ cd DC_AIBench_Micro/Pthread
```

The Pthread micro benchmarks use ImageNet as data input, *ImageData* directory contains the image data with three sizes: Image_1000, Image_10000 and Image_100000.

Step 4. Compile the workload

```
$ make
```

This command will produce an executable file named max_pool or avg_pool.

Step 5. Run the workload

```
$ ./max_pool ../ImageData/image_$imgsize/img$imgsize/ 12 227 227 100
```

Here $imgsize can be 1000, 10000, or 100000.

Step 6. Collect the running results

When the workload run is complete, it will display the output.

4.4.5 CosineNorm, BatchNorm

We use the BatchNorm as an example, the running process of CosineNorm is the same, you can just change the batch_norm in the command to cosine_norm.

1) TensorFlow based

Step 1. Required Software Stacks

Python 2 or 3

Scipy and Numpy

TensorFlow 1.0+

Step 2. Get workloads from AIBench

Download the Benchmark from the link:

http://159.226.41.254:8090/AIBench/DC_AIBench_Micro

Step 3. Prepare the input

The TensorFlow micro benchmarks use random generate data, you need to specify the ***batch size***, ***image size***, and ***channel***.

Step 4. Run the workloads;

```
$ cd DC_AIBench_Micro/TensorFlow
$ python batch_normalization.py <batch_size> <img_size> <channel>
```

Running batchNorm with scripts:

```
$ ./run-tensorflow.sh batchNorm <datasize>
```

Parameter “datasize” can be large/medium/small.

large: 224*224*64 (means length, width and channel respectively)

medium: 112*112*128

small: 56*56*256

Step 5. Collect the running results

When the workload run complete, it will display the running time information.

2) Pthreads based

Step 1. Required Software Stacks

g++ compiler

OpenCV, recommend 3.2 version

Dependences: libopencv_core.so.3.2 and libopencv_imgproc.so.3.2

Step 2. Get workloads from AIBench

Download the Benchmark from this link:

http://159.226.41.254:8090/AIBench/DC_AIBench_Micro

Step 3. Prepare the input

```
$ cd DC_AIBench_Micro/Pthread
```

The Pthread micro benchmarks use ImageNet as data input, *ImageData* directory contains the image data with three sizes: Image_1000, Image_10000 and Image_100000.

Step 4. Compile the workload

```
$ make
```

This command will produce an executable file named batch_norm.

Step 5. Run the workload

```
$ ./batch_norm ../ImageData/image_$imgsize/img$imgsize/ 12 227 227 100
```

Here $imgsize can be 1000, 10000, or 100000.

Step 6. Collect the running results

When the workload run is complete, it will display the output.

4.4.6 Dropout, Softmax

1) TensorFlow based

Step 1. Required Software Stacks

Python 2 or 3

Scipy and Numpy

TensorFlow 1.0+

Step 2. Get workloads from AIBench

Download the Benchmark from the link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Micro

Step 3. Prepare the input

The TensorFlow micro benchmarks use random generate data, you need to specify the ***batch size***, ***image size***, and ***channel***.

Step 4. Run the workloads;

```
$ cd DC_AIBench_Micro/TensorFlow
$ python dropout.py <batch_size> <img_size> <channel>
```

Running fully connected with scripts:

```
$ ./run-tensorflow.sh dropout <datasize>
```

Parameter "datasize" can be large/medium/small.
large: 224*224*64 (means length, width and channel respectively)
medium: 112*112*128
small: 56*56*256

Step 5. Collect the running results

When the workload run complete, it will display the running time information.

2) Pthreads based

Step 1. Required Software Stacks

g++ compiler
OpenCV, recommend 3.2 version
Dependences: libopencv_core.so.3.2 and libopencv_imgproc.so.3.2

Step 2. Get workloads from AIBench

Download the Benchmark from this link:
http://159.226.41.254:8090/AIBench/DC_AIBench_Micro

Step 3. Prepare the input

```
$ cd DC_AIBench_Micro/Pthread
```

The Pthread micro benchmarks use ImageNet as data input, *ImageData* directory contains the image data with three sizes: Image_1000, Image_10000 and Image_100000.

Step 4. Compile the workload

```
$ make
```

This command will produce an executable file named dropout.

Step 5. Run the workload

```
$ ./dropout ../ImageData/image_$imgsize/img$imgsize/ 12 227 227 100
```

Here $imgsize can be 1000, 10000, or 100000.

Step 6. Collect the running results

When the workload run is complete, it will display the output.