HUGECTR - 端到端点击率预估训练解决方案介绍（一）

15 Nov 2019

NVIDIA

AGENDA

Click-Through Rate Prediction

Challenges in CTR Training

HugeCTR Introduction

CLICK-THROUGH RATE PREDICTION

WHAT IS CTR

Wikipedia:

“**Click-through rate** (**CTR**) is the ratio of users who click on a specific link to the number of total users who view a page, email, or advertisement.”

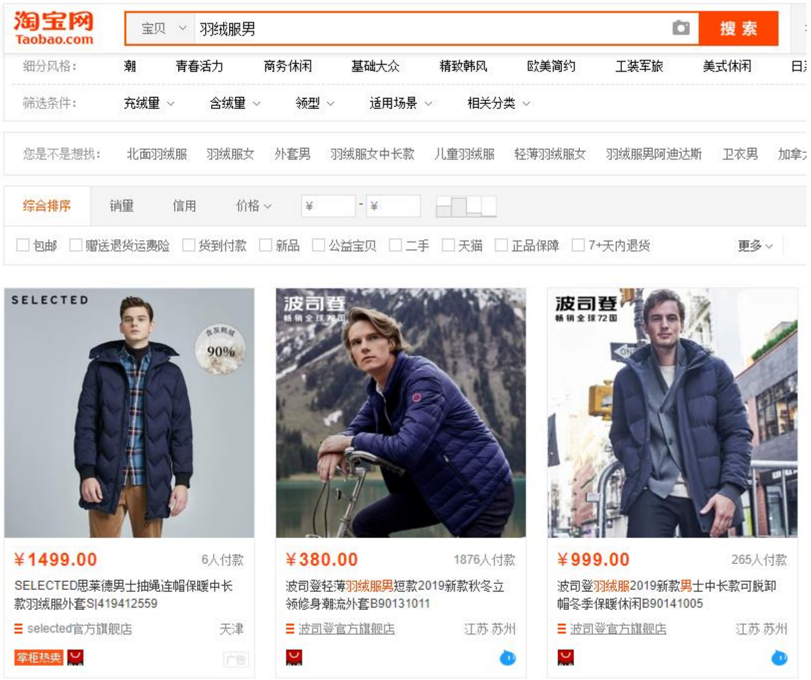
Relatives:

Data Mining, Learning to Rank, NLP, CV

APPLICATIONS

Search Advertising

Recommend based on input query && advs && user information



NVIDIA

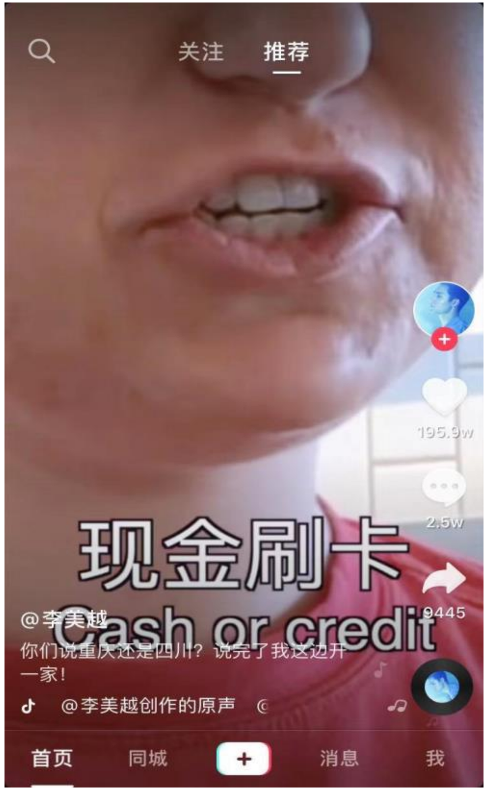
APPLICATIONS

Recommended Ads

Recommend based on advs && user information

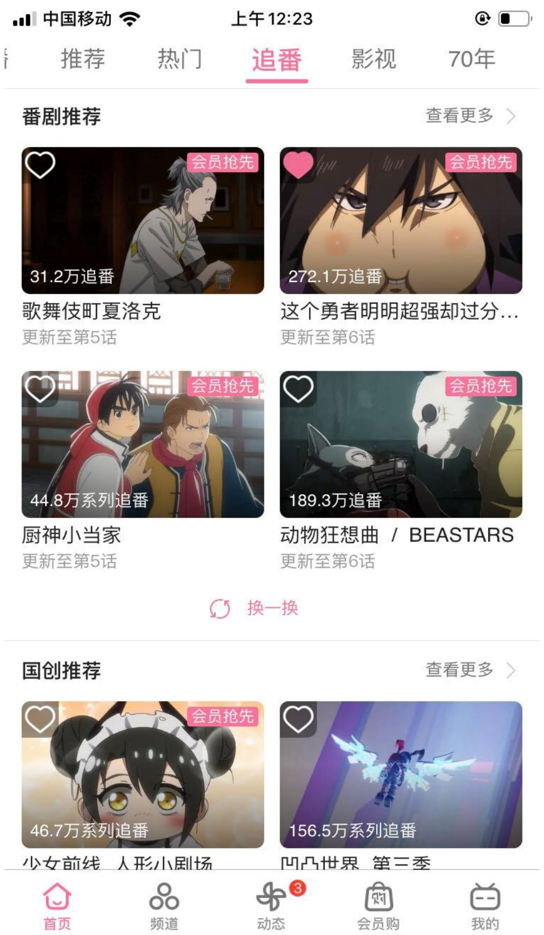
APPLICATIONS

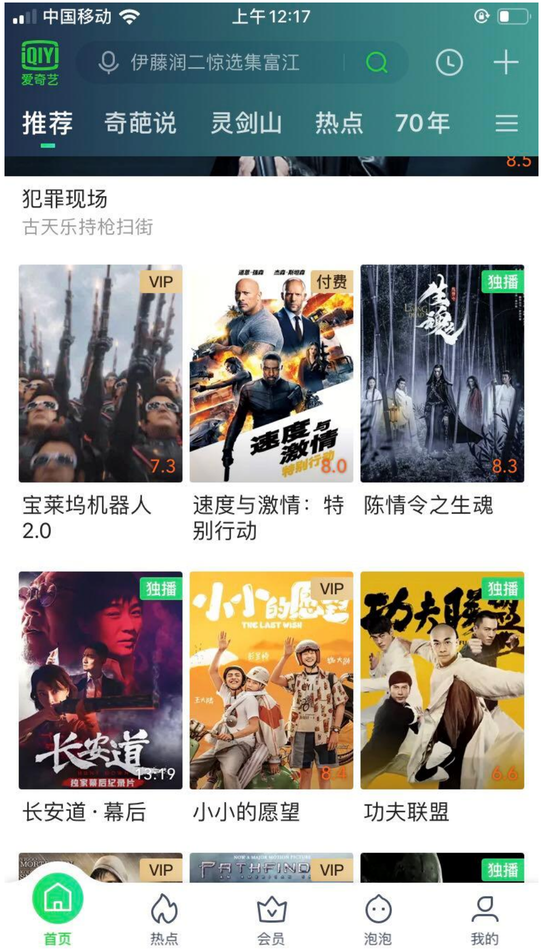
Content Recommendation： UGC



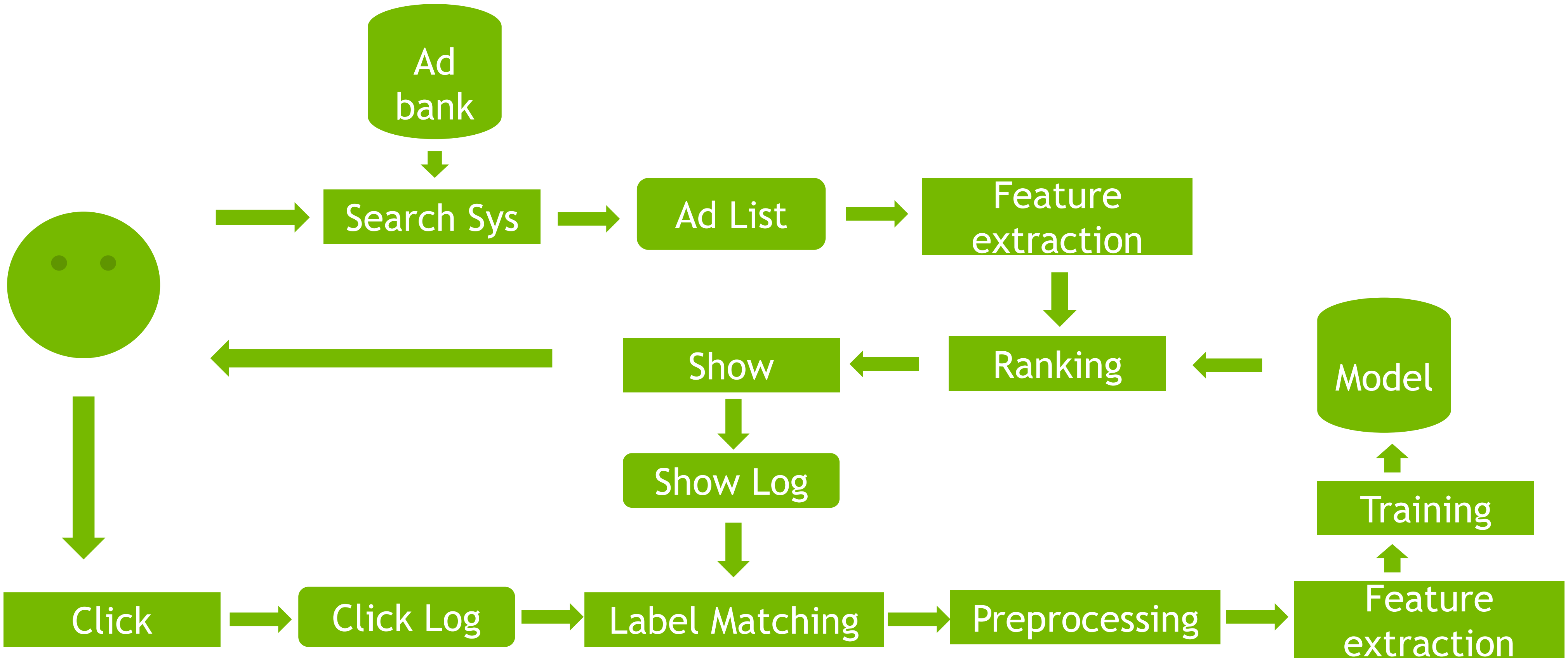
APPLICATIONS

Content Recommendation： PGC



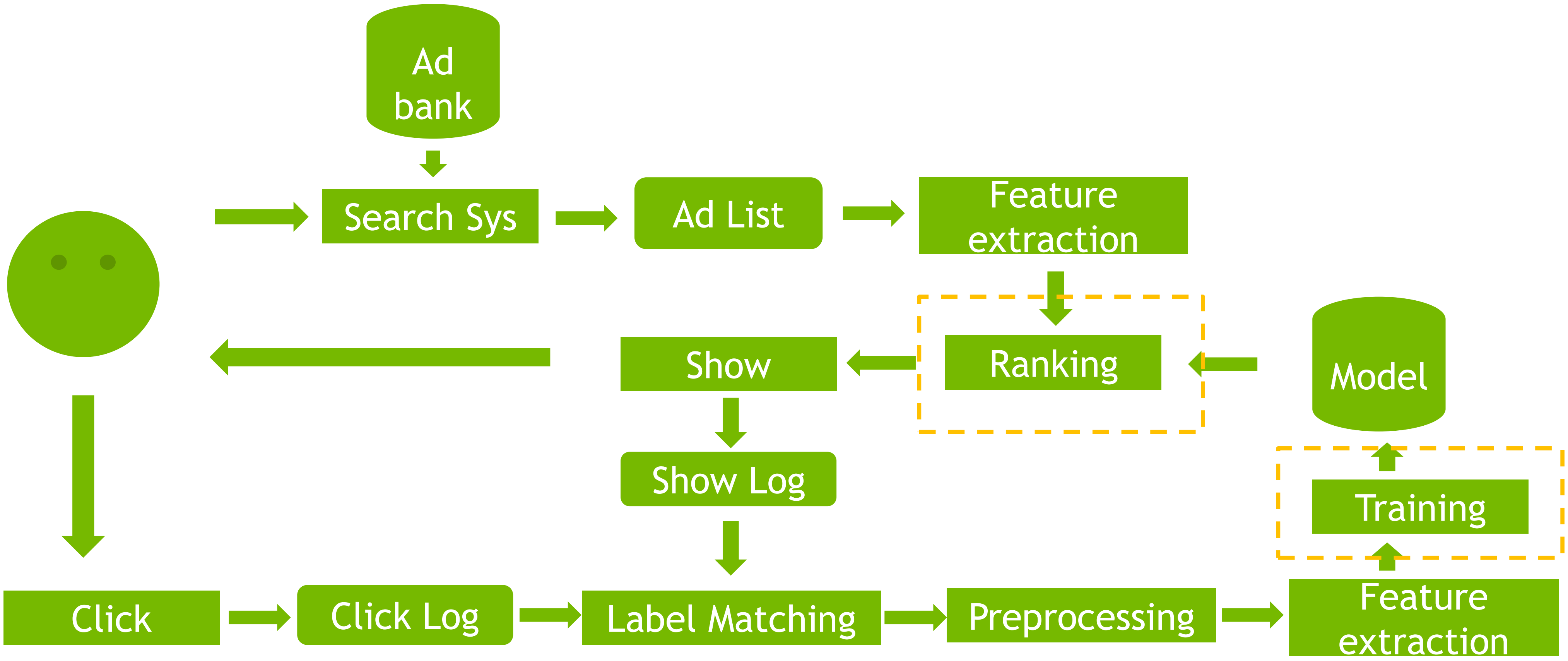


SEARCH ADVERTISING DISTRIBUTION SYSTEM



https://www.cnblogs.com/futurehau/p/6181008.html

SEARCH ADVERTISING DISTRIBUTION SYSTEM

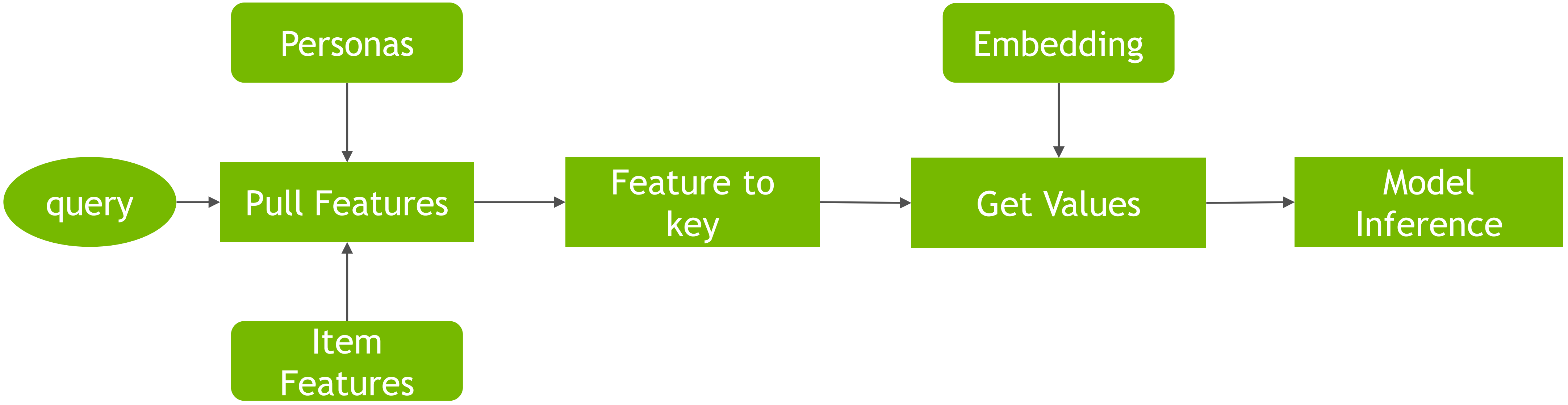


TWO STAGES RANKING

- Collaborative Filtering: user/item based
- Topic Model: LSA / LDA ..
- Content Model

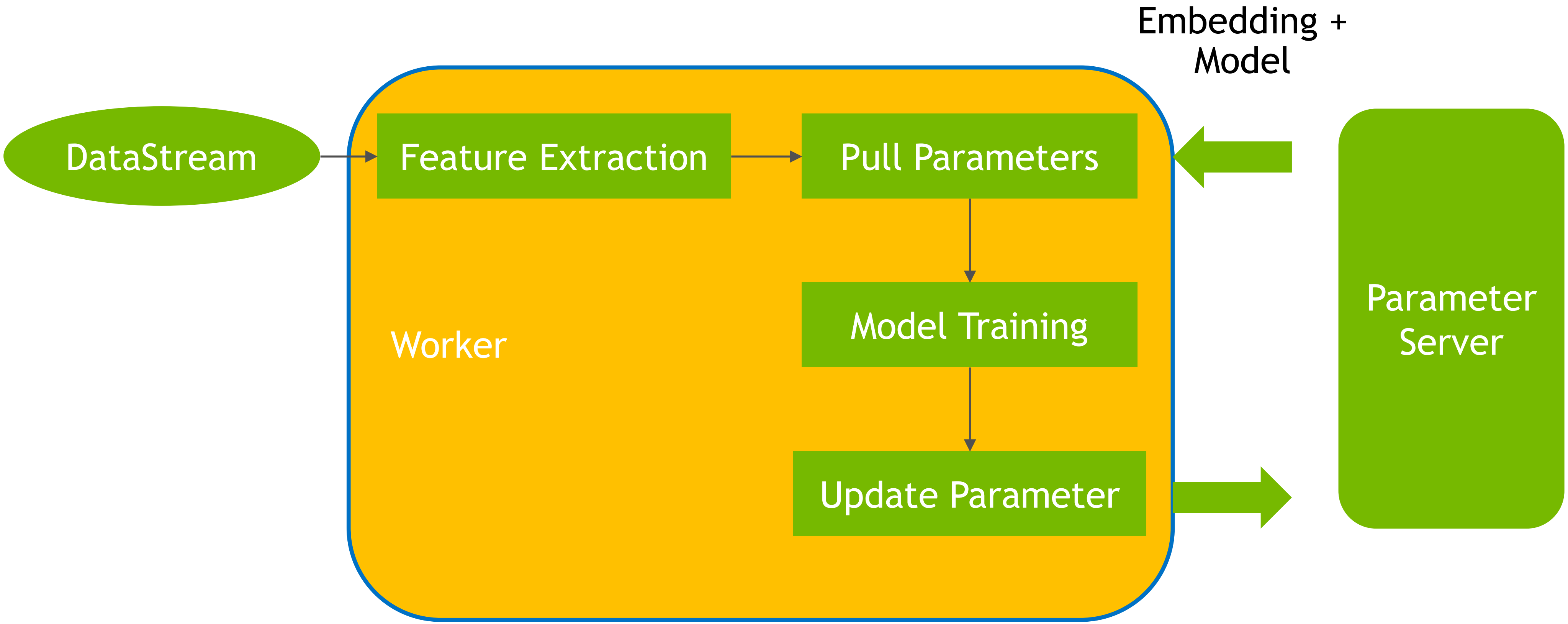
- CTR
- RDTM
- PCR

CTR INFERENCE WORKFLOW



CTR TRAINING WORKFLOW

Parameter Server Based



MODEL

Without DNN: Logistic Regression / Factor Machine

With DNN: Embedding+MLP / Wide Deep Learning / DeepFM / DCN / DIN / DIEN

CHALLENGES IN CTR TRAINING

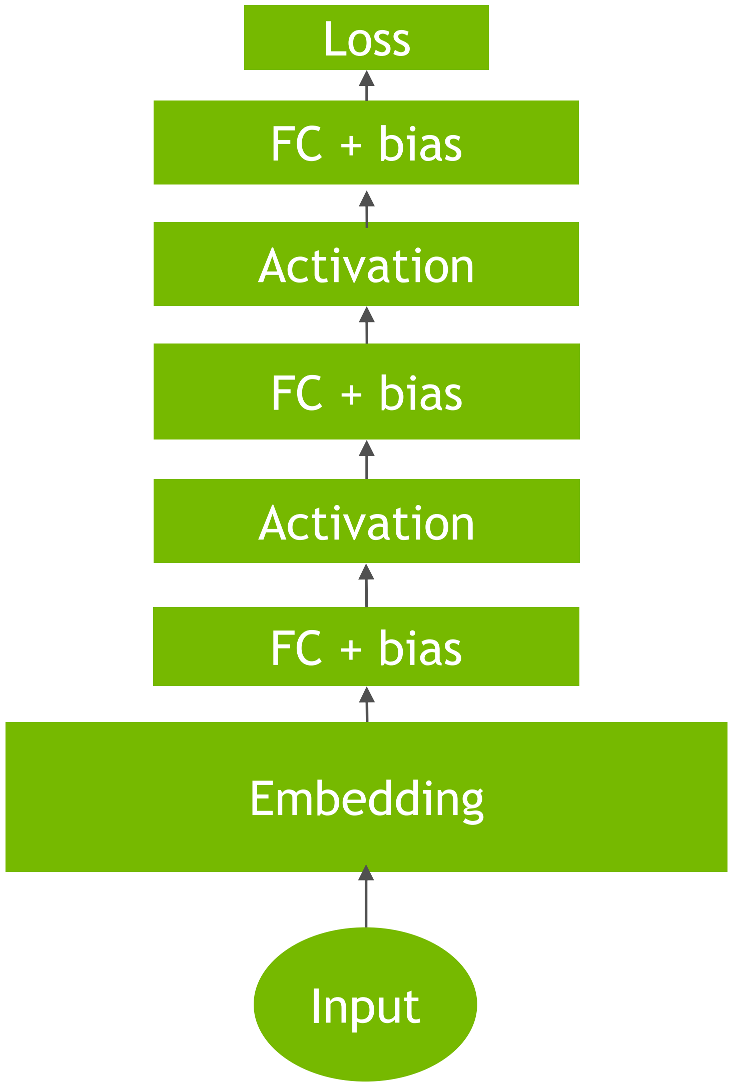
EMBEDDING + MLP

Standard Network

Large Embedding table: E_MEM = GBs to TBs

Small FC layers:

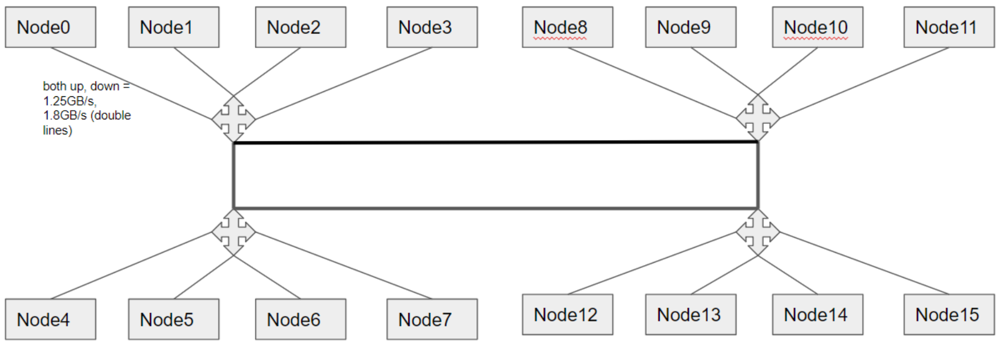
FC_MEM = #Layers * 100s * 100s
(Suppose 5*500*500*4B = 5MB



CTR SOLUTION

CPU

- 100 Nodes, connected with Ethernet (1.25-1.8GB/s)
- Each forward/backward exchange whole the dense model ~10MB per node: 5.6ms*
- Compute time = ~2ms (BS=2000)
- Overall time = compute + data exchange = 7.6ms

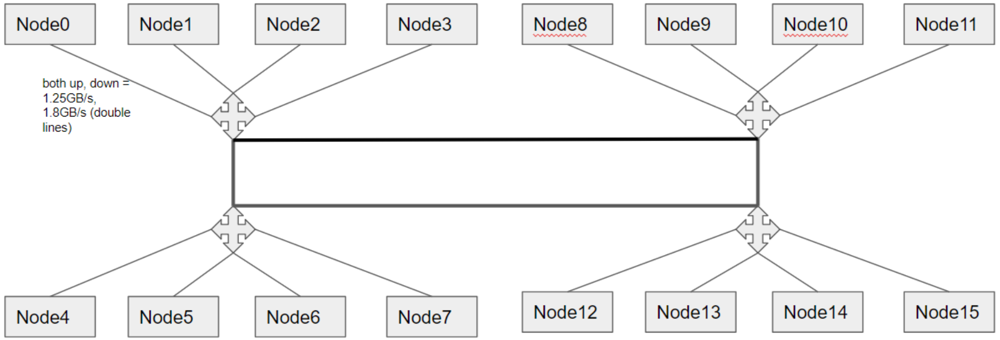


* Suppose 1.8GB/s Ethernet and CPU with 6TFlops per node

CTR SOLUTION

CPU

- 100 Nodes, connected with Ethernet (1.25-1.8GB/s)
- Each forward/backward exchange whole the dense model ~10MB per node: 5.6ms
- Compute time = ~2ms (BS=2000)
- Overall time = compute + data exchange = 7.6ms

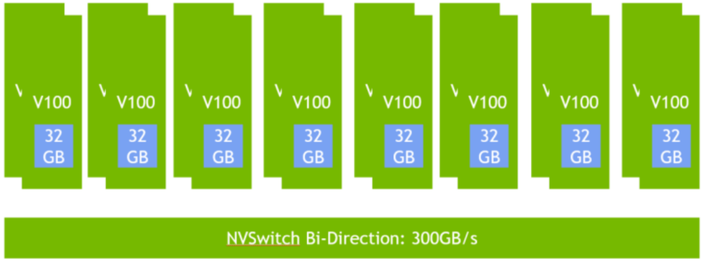


Bottle Neck is Network

CTR SOLUTION

Single GPU Node

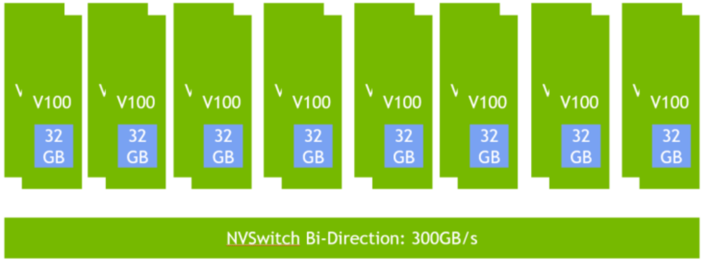
- Single Node
  - Within GPU server: model exchange is >83x faster (0.067ms)
  - Compute Time: 6ms (batchsize=2x10^5)
  - Total Time = 6ms (1.26x 100 CPU Nodes)



CTR SOLUTION

Single GPU Node

- Single Node
  - Within GPU server: model exchange is >83x faster (0.067ms)
  - Compute Time: 6ms (batchsize=2x10^5)
  - Total Time = 6ms (1.26x 100 CPU Nodes)

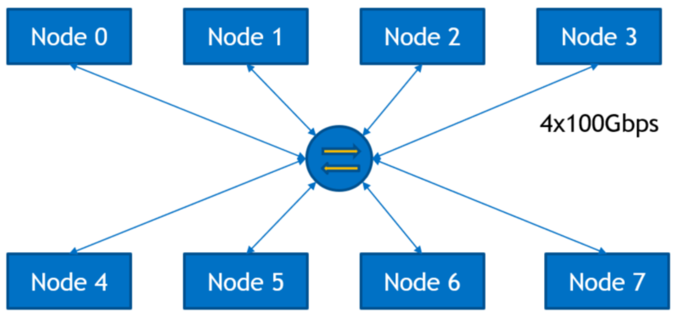


Bottle Neck is Compute

CTR SOLUTION

Multi GPU Nodes

- Multi Node
  - Within GPU server: model exchange is 27.8x faster than CPU
  - Compute Time: 6ms/#Node (batchsize=2x10^5/#Node)
  - Total Time = 6ms/#Node + 0.2ms (linear scale if Nodes < 10)



CHALLENGES FOR GPU SOLUTION

Streaming Training: Dynamic Hashtable Insertion

Very big hashtable (GBs~TBs)

Large data I/O for data reading

Very shallow networks (3~20 layers)

Not a typical DNN training can be handled by current frameworks like pytorch TensorFlow

CHALLENGES FOR GPU SOLUTION

Challenges:

- Streaming Training: Dynamic Hashtable Insertion
- Very big hashtable (GBs~TBs)
- Large data I/O for data reading
- Very shallow networks (3~20 layers)

HugeCTR:

- Flexible GPU Hashtable
- Multi-Node training
- Efficient Three Stage Pipeline

HUGECTR INTRODUCTION

WHAT IS HUGECTR

HugeCTR is a high efficiency GPU framework designed for Click-Through-Rate (CTR) estimating training.

Key Features in 2.0:

- GPU Hashtable and dynamic insertion
- Multi-node training and enabling very large embedding
- Mixed precision training

HOW HUGECTR HELP

1. Prototype: Showing performance and possibility on GPUs. (v1.0)
2. Reference Design: Developers and NV can work together to modify HugeCTR according to the specific requirements (v2.0 current stage)
3. Framework: Developers can train their model easily on HugeCTR (v3.0)

NETWORK SUPPORTED

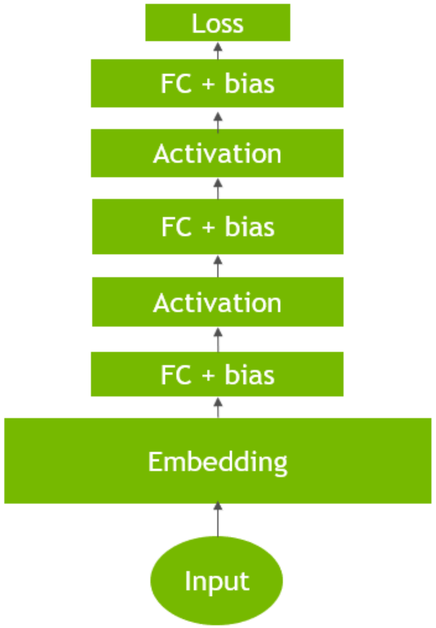
Embedding + MLP

Multi slot embedding: Sum / Mean

Layers: Concat / Fully Connected / Relu / BatchNorm / elu

Optimizer: Adam/ Momentum SGD/ Nesterov

Loss: CrossEngtropy/ BinaryCrossEntropy



* Supporting multiple labels and each label will have a unique weight

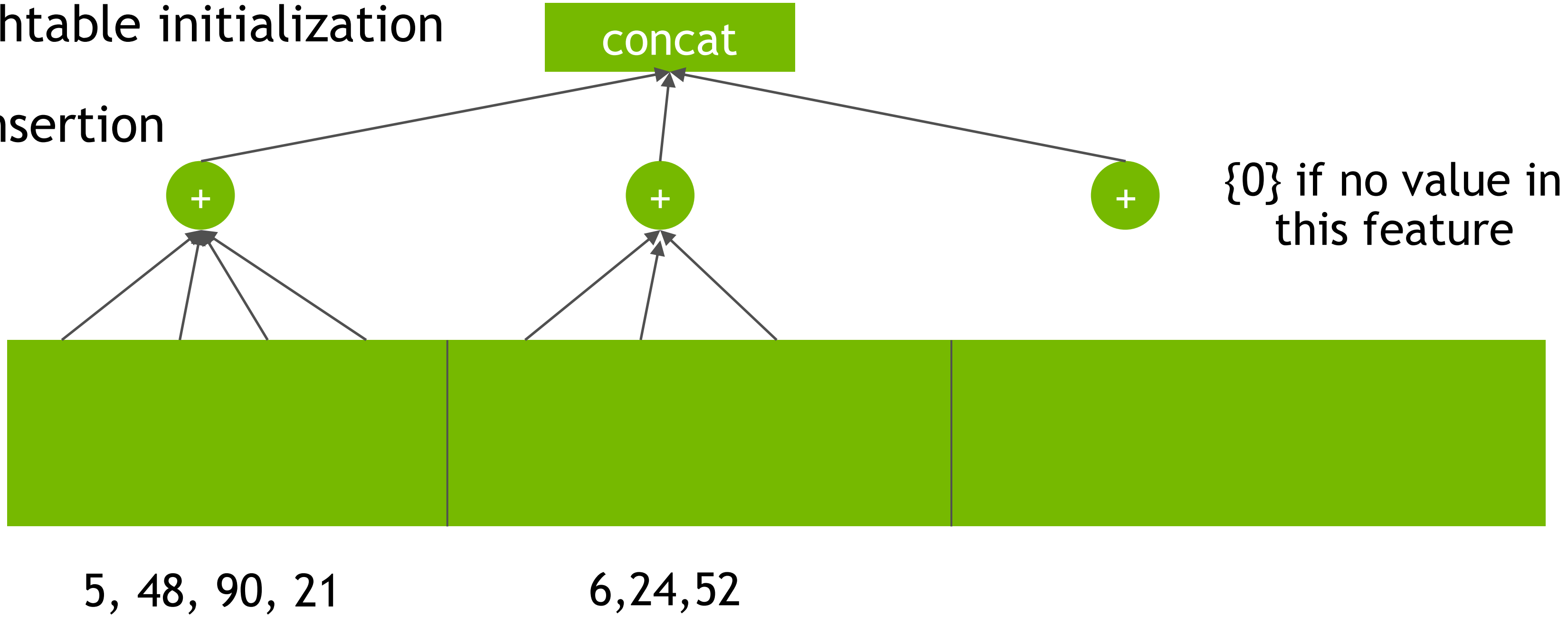
NETWORK SUPPORTED

Sparse Model

Supported reduce ‘+’: sum / mean

Empty Hashtable initialization

Dynamic insertion



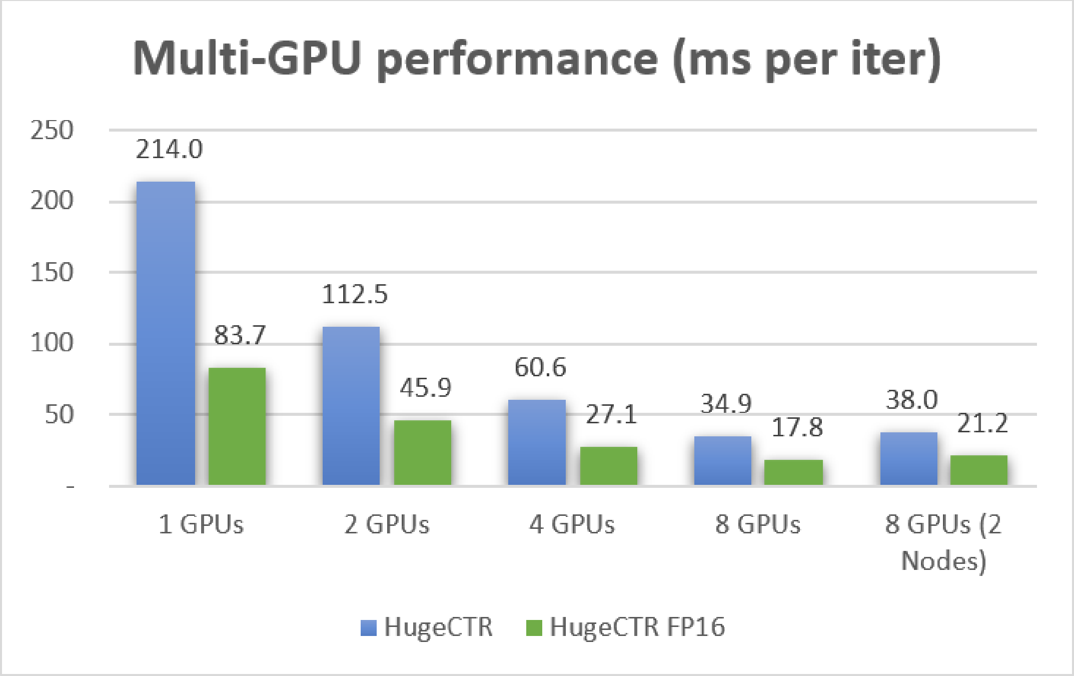
PERFORMANCE

Good Scalability

NCCL 2.0

Three stages pipeline:

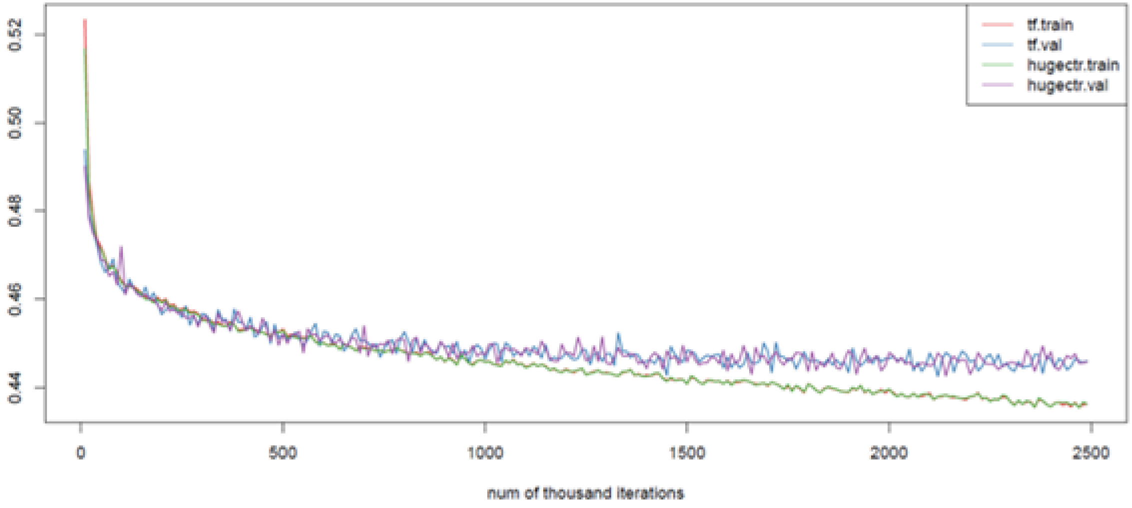
- reading from file
- host to device data transaction (inter / intra nodes)
- GPU training

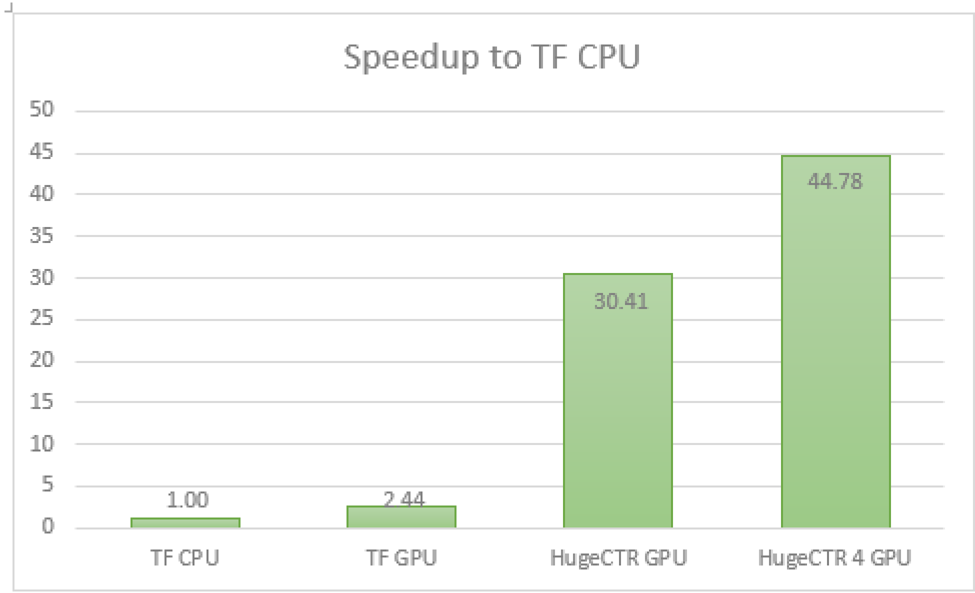


*MLP Layers: 12 / MLP Output: 1024 / Embedding Vector: 64 / Table Number: 1

PERFORMANCE
TensorFlow

44x Speedup to CPU TF and same loss curve





Embedding Vector: 64/ Layers: 4 / MLP Output: 200 / Table Number: 1

PERFORMANCE

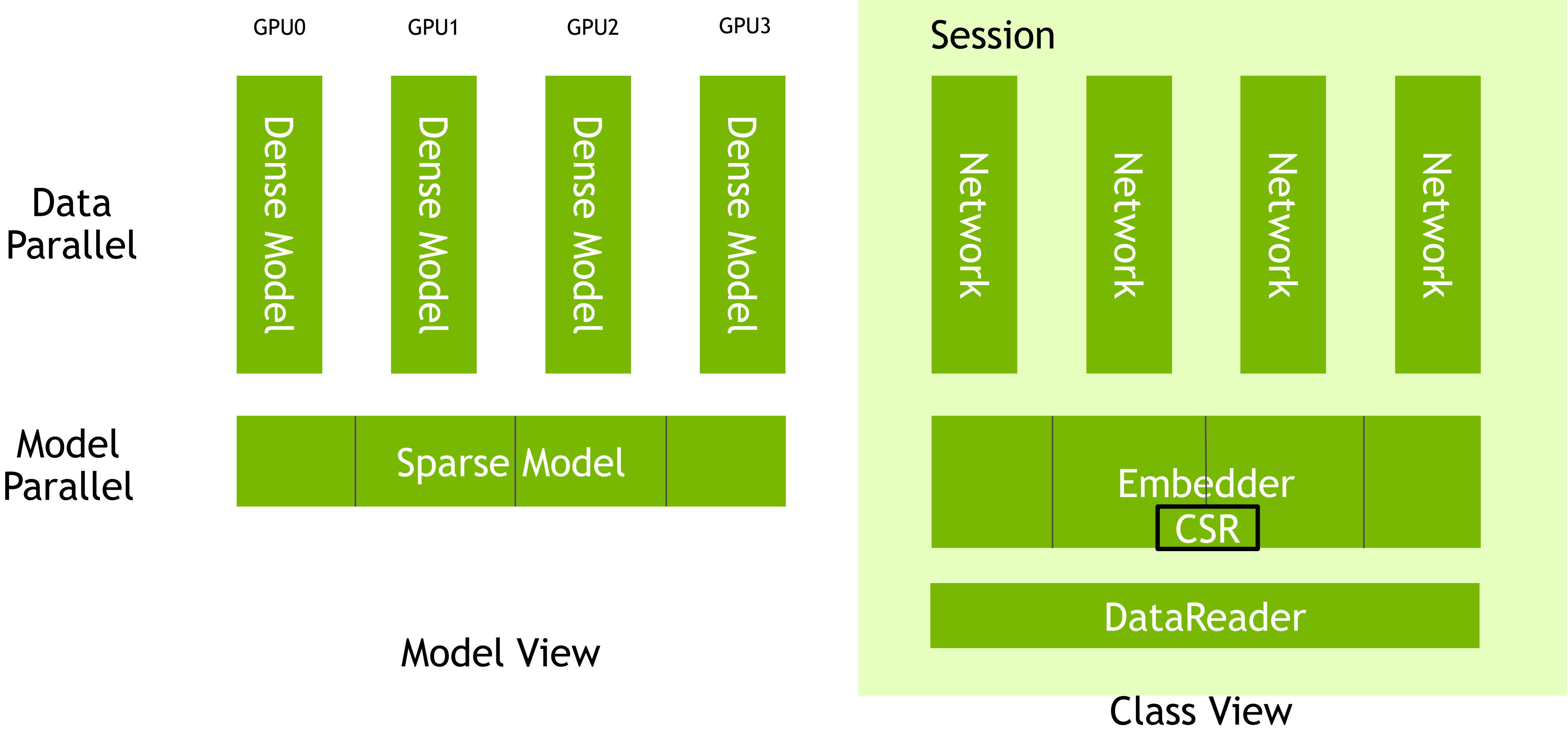
Pytorch DLRM

HugeCTR	slot_num	embedding_vec	num_layers	output of MLP
	64	64	4	512

GPUs	Batchsize	HugeCTR Time (s per 200iters)	DLRM (200iters)	Speedup
1	40960	13.5	17.7	131%
2	40960	10.3	19.4	188%
4	40960	6.3	17.3	275%
8	40960	4.3	33.8	786%
1	4096	1.6	4.5	281%
2	4096	1.34	6.5	485%
4	4096	0.9	8.4	933%
8	4096	0.75	13.7	1827%

Embedding Vector: 64 / Layers: 4 / MLP Output: 512 / Table number: 64

SYSTEM


GPU0
GPU1
GPU2
GPU3
Data Parallel
Dense Model
Dense Model
Dense Model
Dense Model
Model Parallel
Sparse Model
Model View
Session
Network
Network
Network
Network
Embedder
CSR
DataReader
Class View

HOW TO USE

A Simplified Framework For Ranking or Retrieval

Weight initialization: generate a file with initialized weight according to the name in config file

$ huge_ctr –-init config.json

Training:

$ huge_ctr –-train config.json

All the network, solver and dataset is configured under config.json

HOW TO USE

Config.json

Configuration file is in json format, and has four parts:

Solver

Optimizer

Data

Network

```
{
  "solver": {
    "lr_policy": "fixed",
    "display": 200,
    "max_iter": 50000,
    "gpu": [[0],[0]],
    "batchsize": 40960,
    "snapshot": 10000,
    "snapshot_prefix": "./",
    "eval_interval": 1000,
    "eval_batches": 100,
    "model_file": "./criteo.model"
  },
```

```
"optimizer": {
  "type": "Adam",
  "adam_hparam": {
    "alpha": 0.005,
    "beta1": 0.9,
    "beta2": 0.999,
    "epsilon": 0.000001
  }
},
```

```
 "data": {
   "source": "./file_list.txt",
   "eval_source": "./file_list_test.txt",
   "max_feature_num_per_sample": 100,
   "label_dim": 1,
   "slot_num": 1
 },
```

```
"layers": [
  {
    "name": "sparse_embedding1",
    "type": "SparseEmbeddingHash",
    "top": "sparse_embedding1",
    "sparse_embedding_hparam": {
      "vocabulary_size": 1603616,
      "load_factor": 0.75,
      "slot_num":10,
      "embedding_vec_size": 64,
      "combiner": 0
    }
  },

  {
    "name": "concat1",
    "type": "Concat",
    "bottom": "sparse_embedding1",
    "top": "concat1"
  },

  {
    "name": "fc1",
    "type": "InnerProduct",
    "bottom": "concat1",
    "top": "fc1",
     "fc_param": {
      "num_output": 200
    }
  },

  {
    "name": "relu1",
    "type": "ReLU",
    "bottom": "fc1",
    "top": "relu1"

  },
```

HOW TO USE

Dataset

Dataset contains two kinds of files:

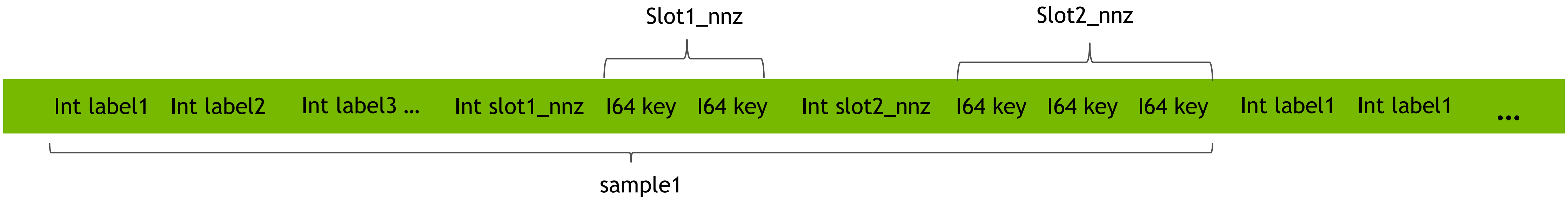
1. File list: includes the number of files and file name list with text format. A file name could be either of a relative address or absolute address. The file names are separated with ‘\n’

2. Data files: includes a bunch of files with binary format.

HOW TO USE

Data File

Training Set Format (RAW data with header):



Header:

```
typedef struct DataSetHeader_{
  long long number_of_records; //the number of samples in this data file
  long long label_dim; //dimension of label
  long long slot_num; //the number of slots in each sample
  long long reserved; //reserved for future use
} DataSetHeader;
```

ROADMAP

1.0

- Early 2019
- RAW buffer Embedder
- Embedder+MLP

2.0

- September 2019
- HashTable Embedding
- Multi-node
- Mixed Precision Training
- More Layers

3.0

- TBD
- Support TensorFlow
- optimized slot reduction
- Dense input
- Inference support
- Input data check

RESOURCES

源码：

https://github.com/NVIDIA/HugeCTR

公众号文章：

https://mp.weixin.qq.com/s/OieuhvtZvzFEfKklTHiuOg

KEY CONTRIBUTORS


Fan Yu
Hashtable


Yong Wang
Algorithm
Advisor


Ryan Jeng
Competitive
Study


Joey Wang
Project
Management


Xiaoying Jia
Mixed
Precision


Minseok
Lee
Multi-Node


David Wu
Embedding

NVIDIA GTC

2019 年 12 月 16 日 – 19 日
苏州，中国

沟通

与来自 NVIDIA 和其他业界领先组织的技术专家互动。

学习

通过百余场讲座、动手实验和研究海报获取宝贵见解和实践培训。

发现

了解 GPU 技术如何为深度学习等重要领域带来重大突破，描绘最新 AI 世界观。

创新

共同探索改变世界的颠覆性创新，定义未来。

立即注册，扫码立享 75 折邀请优惠购票
或使用我的优惠邀请码：NVZEHUANW
前往 www.nvidia.cn/gtc/ 完成报名

RYAN JENG
NVIDIA 高級工程師

扫码注册，经典课程+全新主题，
AI实践经验升级

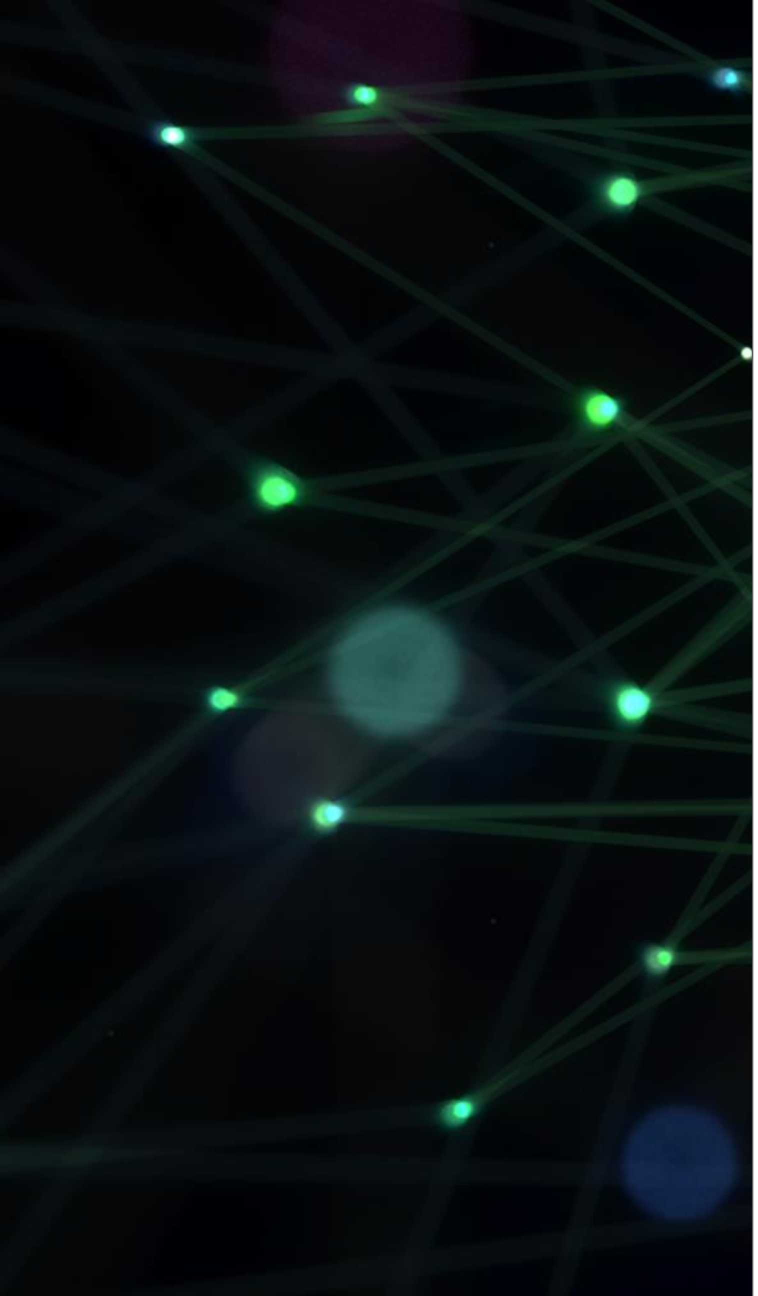
CUDA PYTHON

探讨如何使用 Numba（即时，专用类型的 Python 函数编译器）在 NVIDIA 大规模并行运算的 GPU 上加速 Python 应用程序。

您将学习如何：

- **使用 Numba 从NumPy ufuncs 编译 CUDA 内核**
- **使用 Numba 创建和启动自定义 CUDA 内核**
- **应用关键的GPU内存管理技术**

完成本课程后，您将能够使用Numba编译并启动 CUDA 内核，以加速 NVIDIA GPU上的 Python 应用程序。



zehuanw@nvidia.com

NVIDIA