NVIDIA

Partner Business Manager, Southern Europe

RECOMMENDER SYSTEMS DEMYSTIFIED

Matthieu Gasse-Hellio, Partner Business Manager at NVIDIA

Miguel Martínez, Sr. Data Scientist at NVIDIA

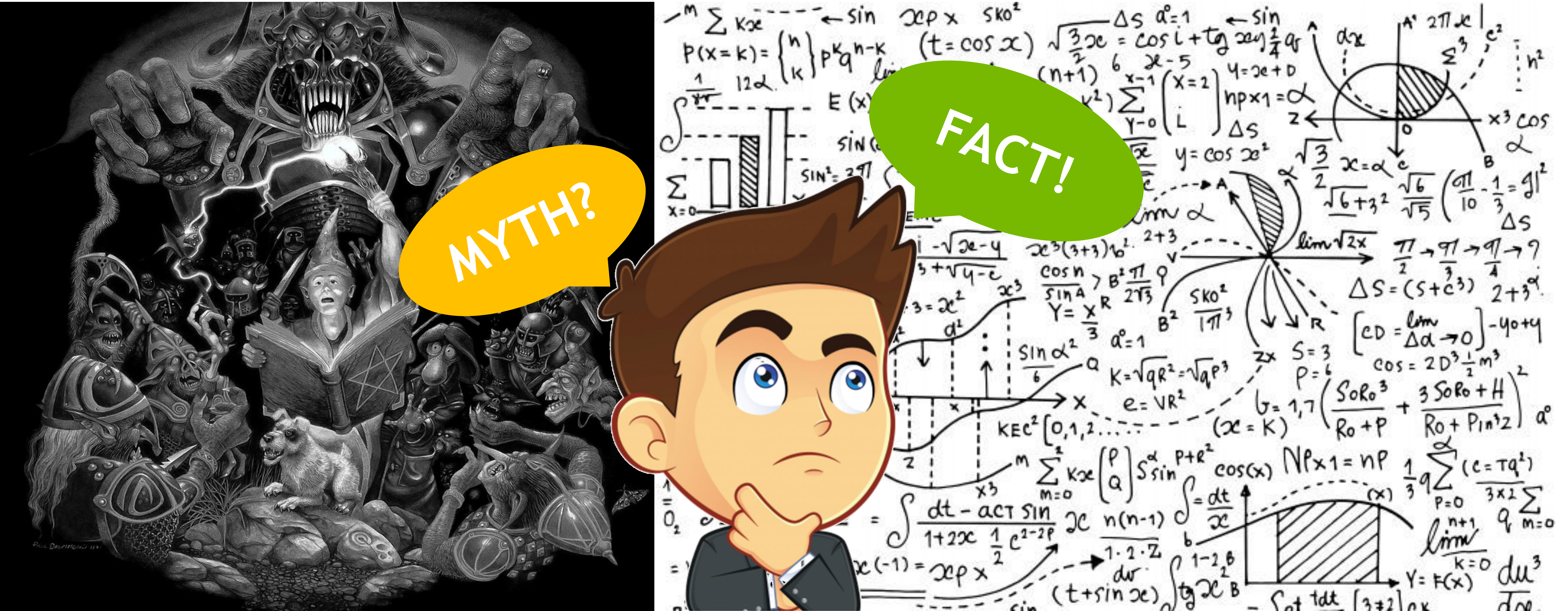
Today's Agenda

- A gentle introduction to RecSys
  - RecSys techniques
    - Collaborative filtering
    - Content-based filtering
  - Deep & Wide architecture
- NVIDIA Merlin framework
  - NVTabular
  - HugeCTR
  - Reference implementations
  - Triton Inference Server

Recommender Systems (RecSys)

Today’s talk is about

Everything you wanted to know about RecSys but were afraid to ask



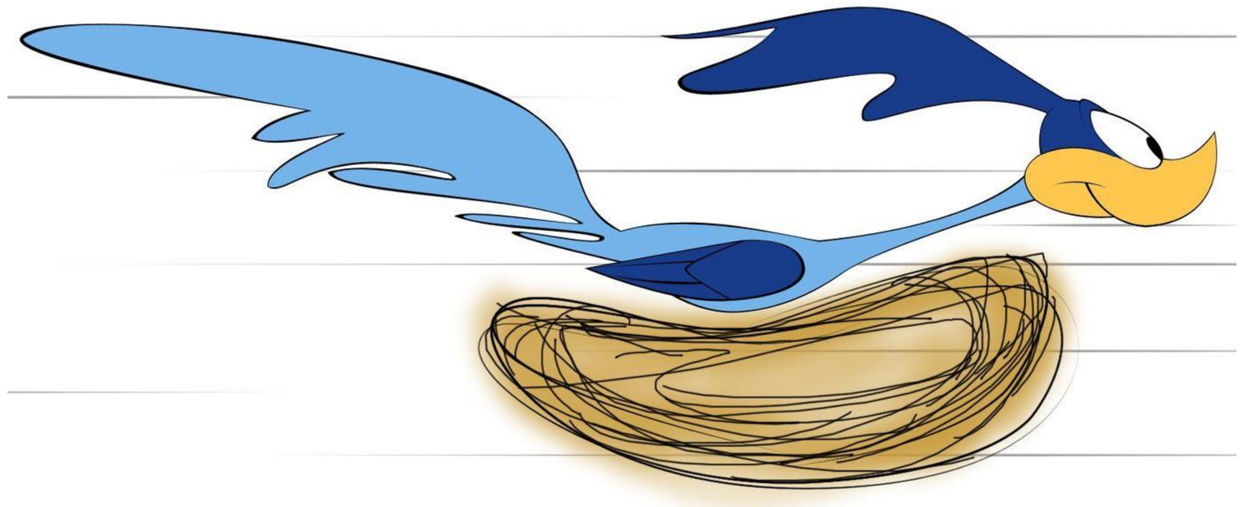
We will also learn

How to do this …

https://www.rinapiccolo.com/

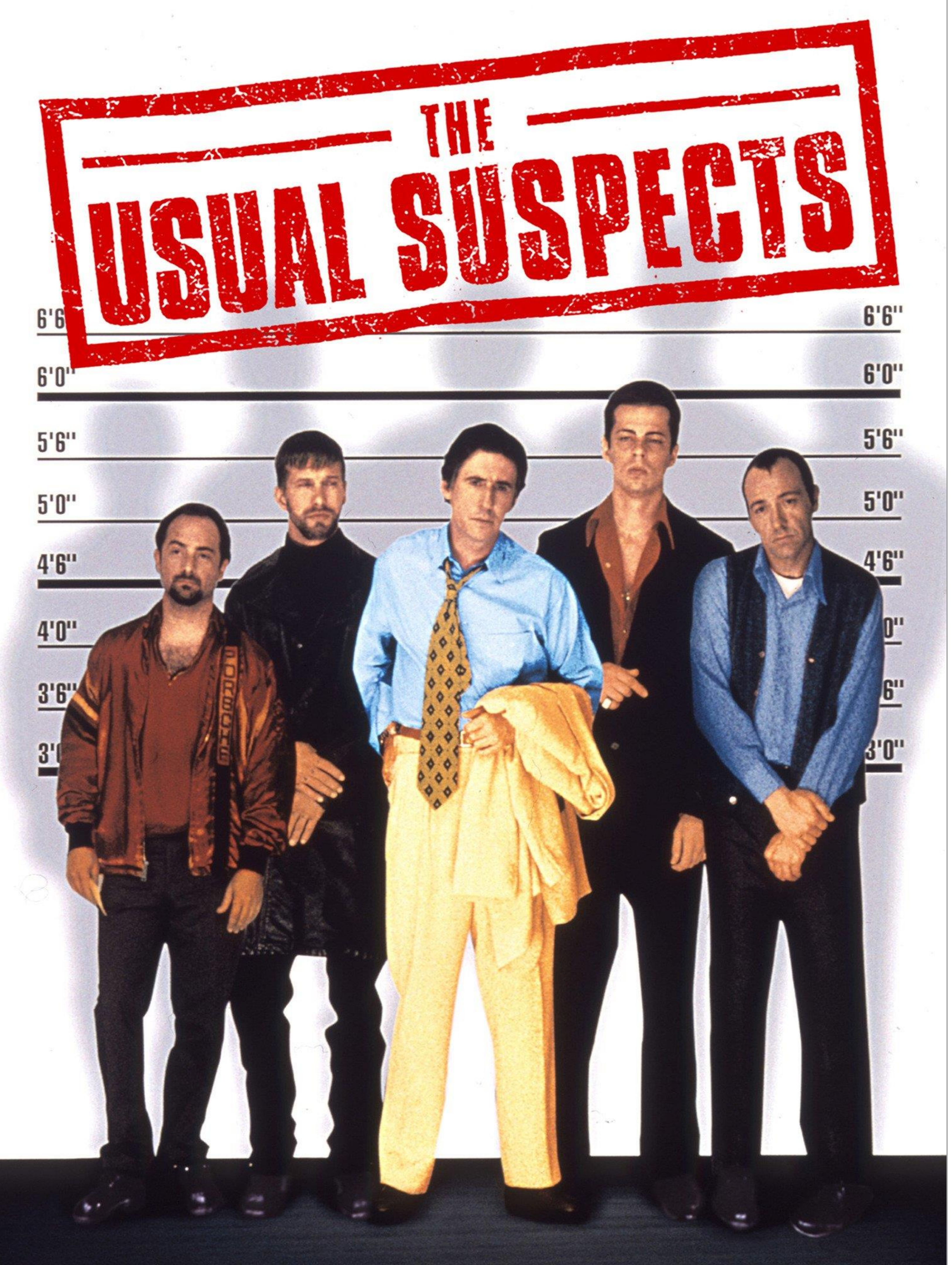
Cartoon by Rina Piccolo

… as fast as possible …



… with the help of …

NVIDIA
MERLIN



RecSys are Everywhere

The usual suspects

NETFLIX
Other movies you may enjoy

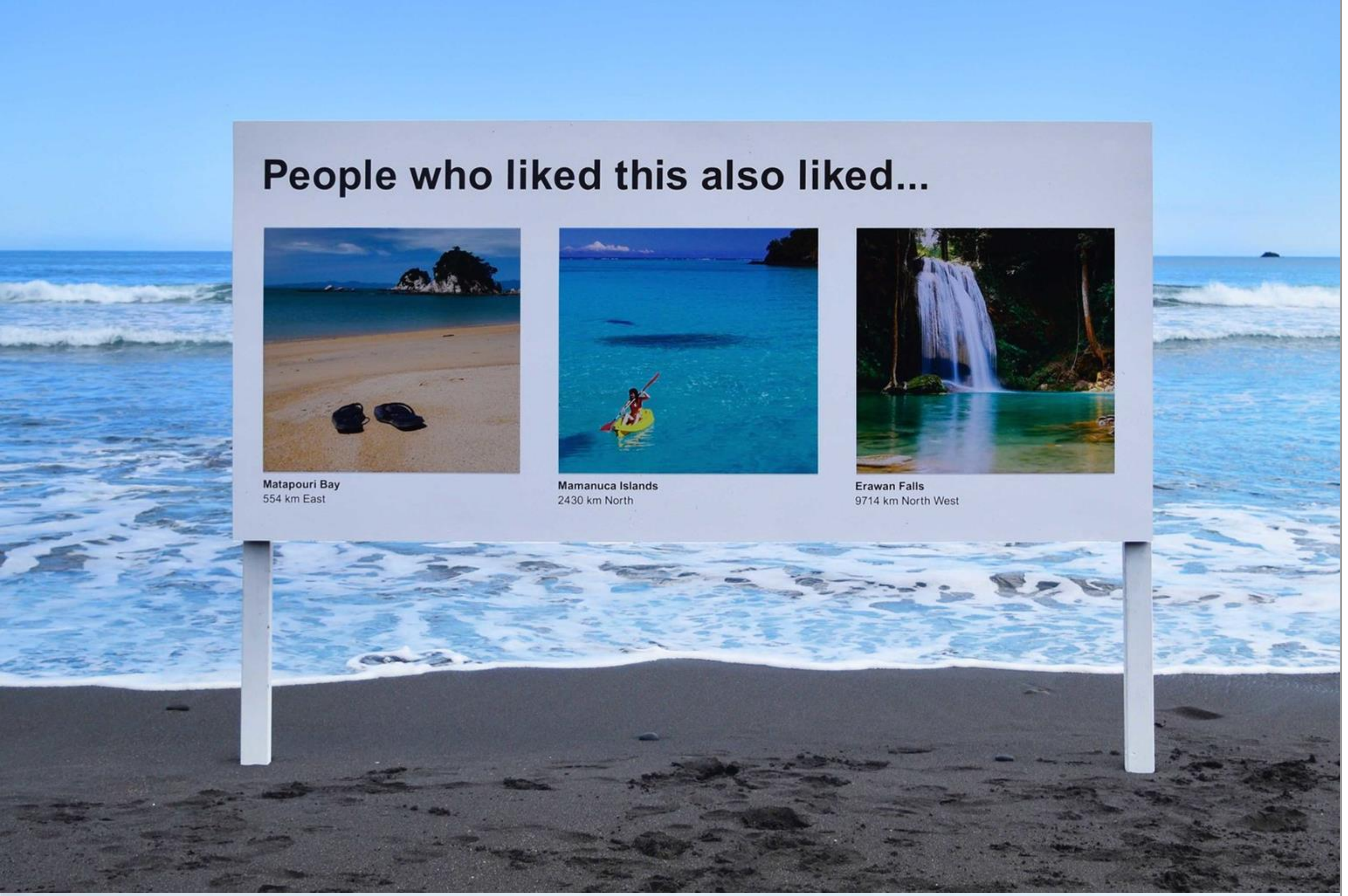
amazon
Customer who bought this item also bought

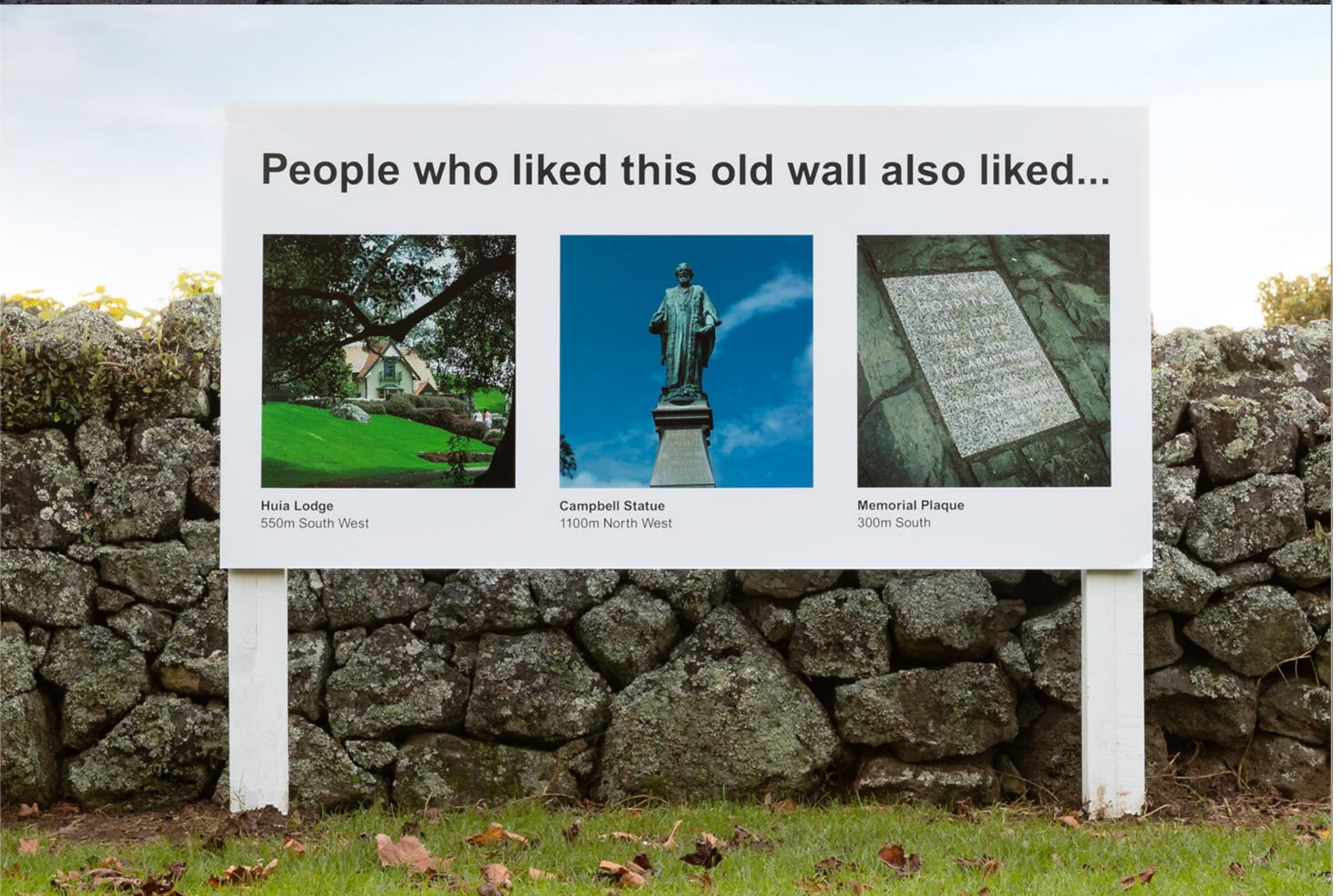
Spotify
New releases for you

facebook
People you may know

Google Play
Recommended for you

LinkedIn
Jobs you may be interested in





RecSys are Everywhere

Also in your non-digital life!

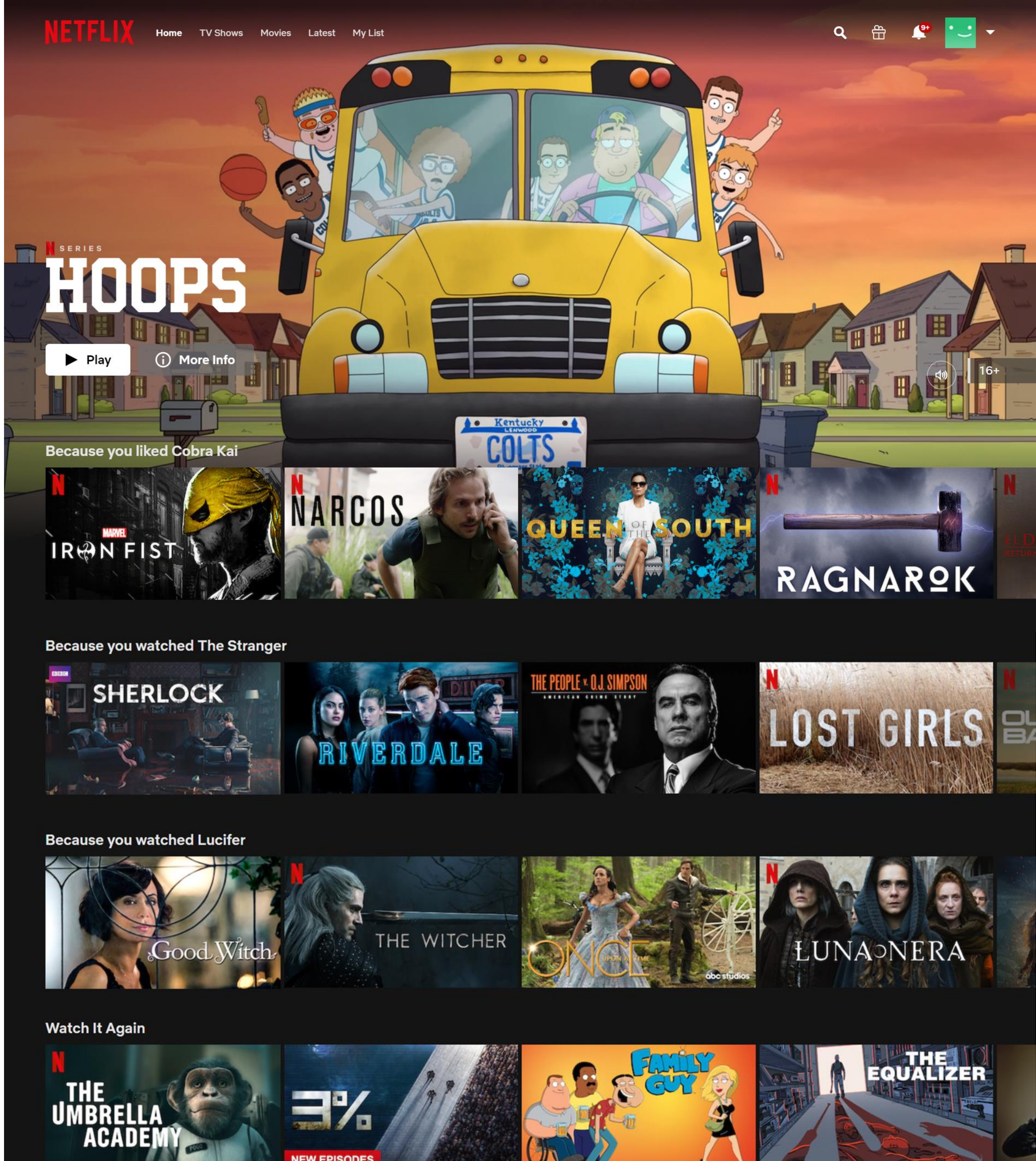
http://scottandbenorbenandscott.com/#/signs-of-the-times

Recommender Systems

My user experience

Recommendations based on TV shows:

- I have **liked** before.
- I have **watched** before.



Recommender Systems

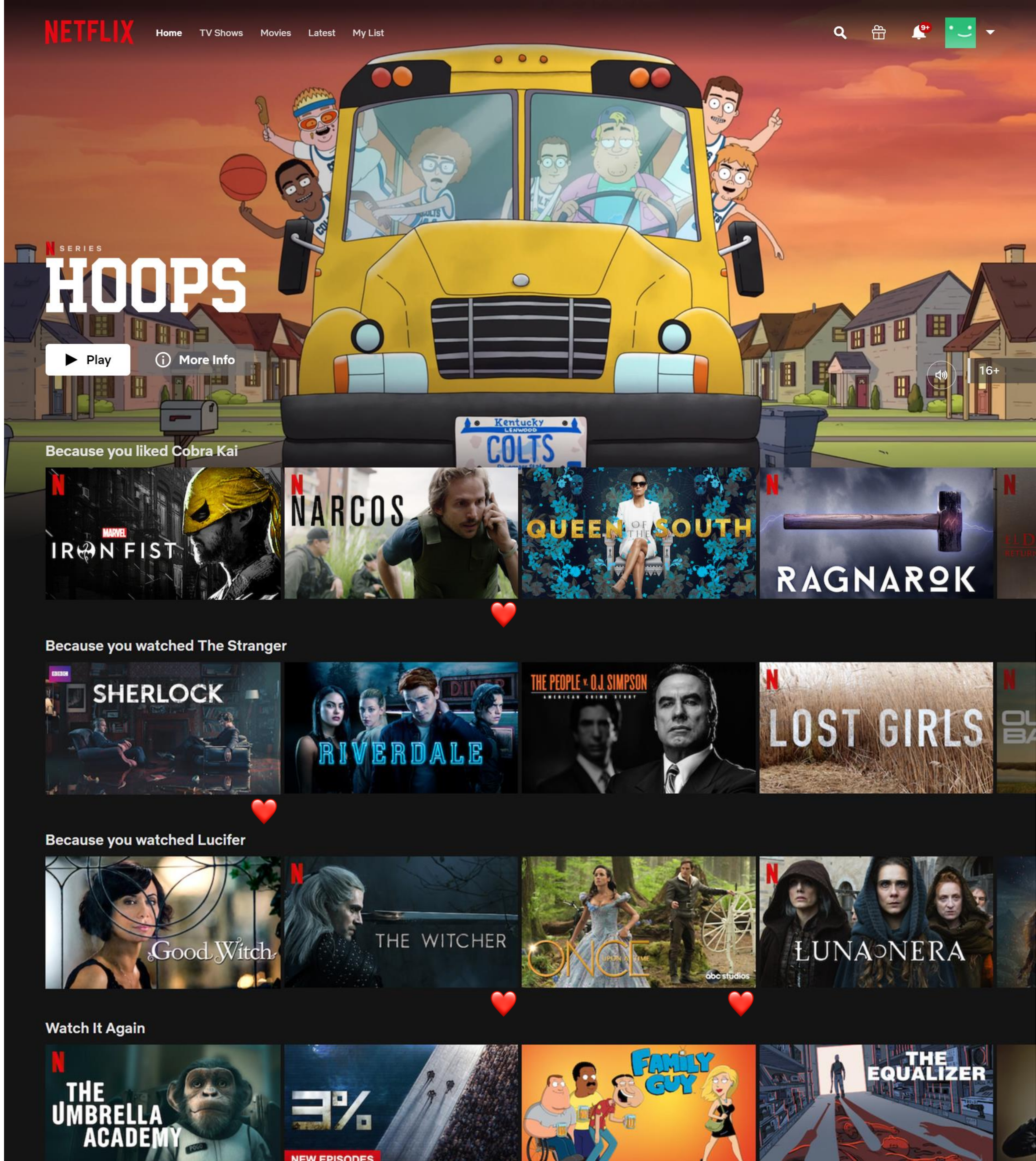
My user experience

Recommendations based on TV shows:

- I have **liked** before.
- I have **watched** before.

… and it works quite well:

- TV shows I have already watched and liked. ❤️: 33%



Recommender Systems

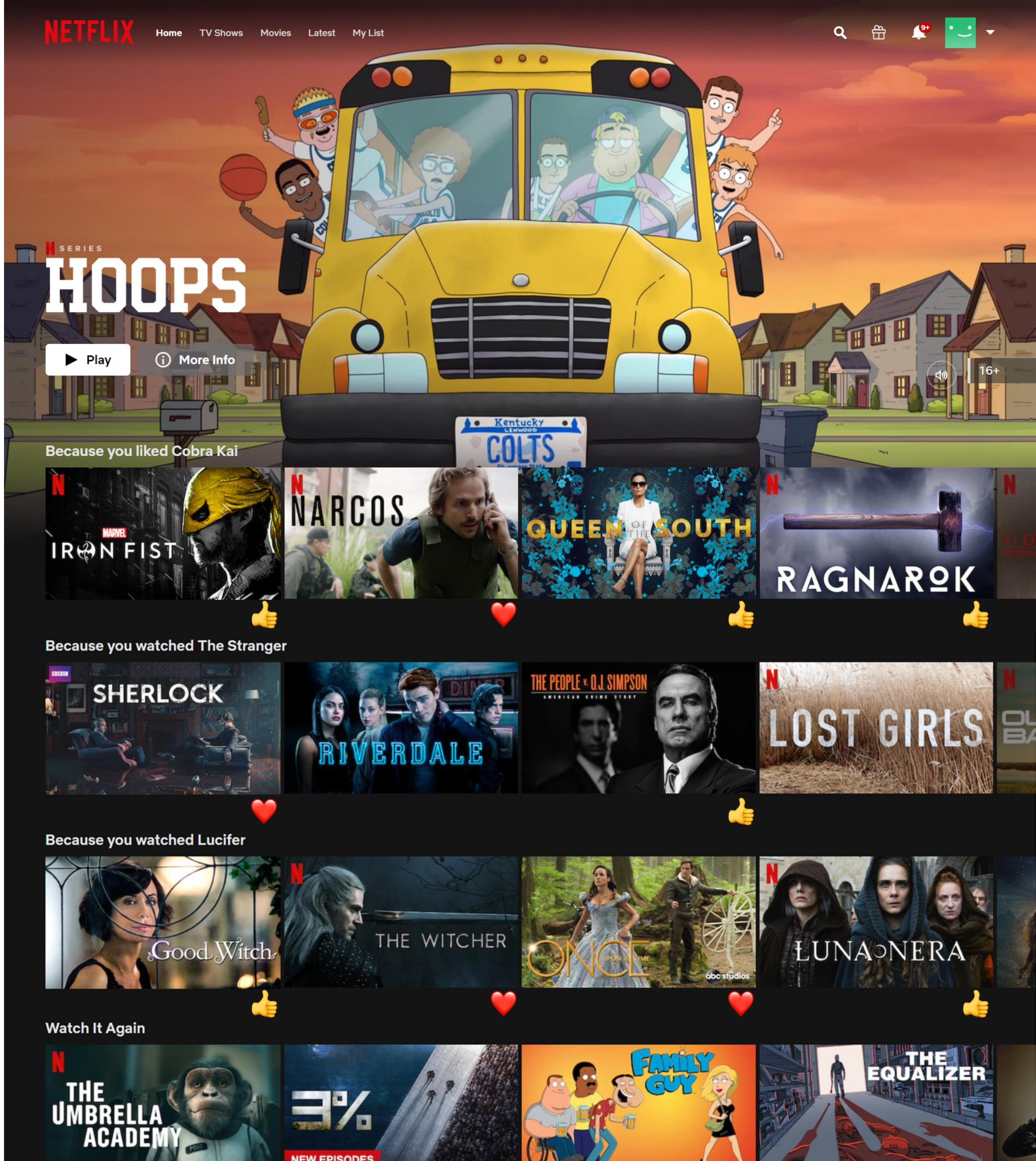
My user experience

Recommendations based on TV shows:

- I have **liked** before.
- I have **watched** before.

… and it works quite well:

- TV shows I have already watched and liked. ❤️: 33%
- TV shows I haven't watched yet, but I'd like to watch. 👍: 50%



Recommender Systems

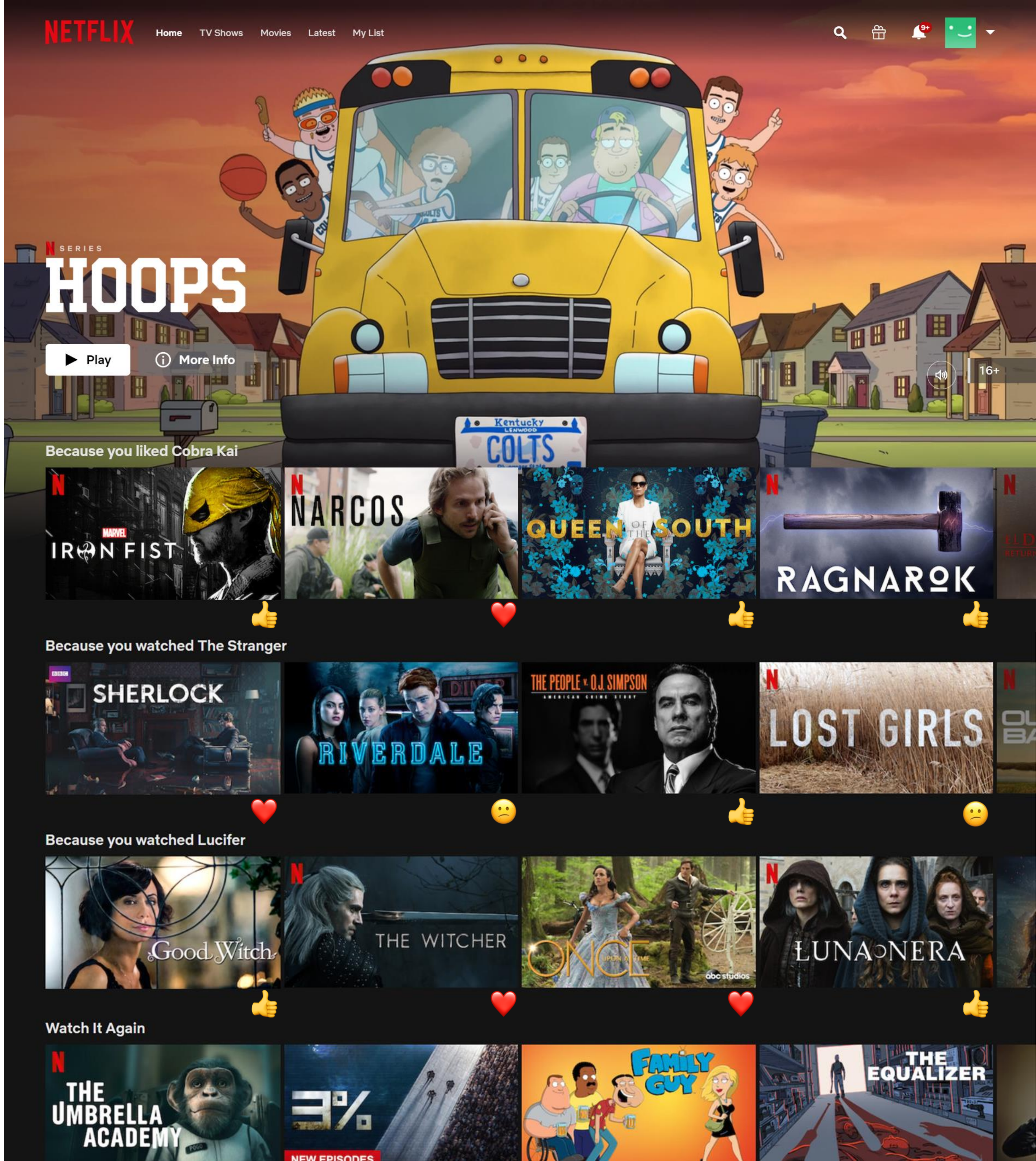
My user experience

Recommendations based on TV shows:

- I have **liked** before.
- I have **watched** before.

… and it works quite well:

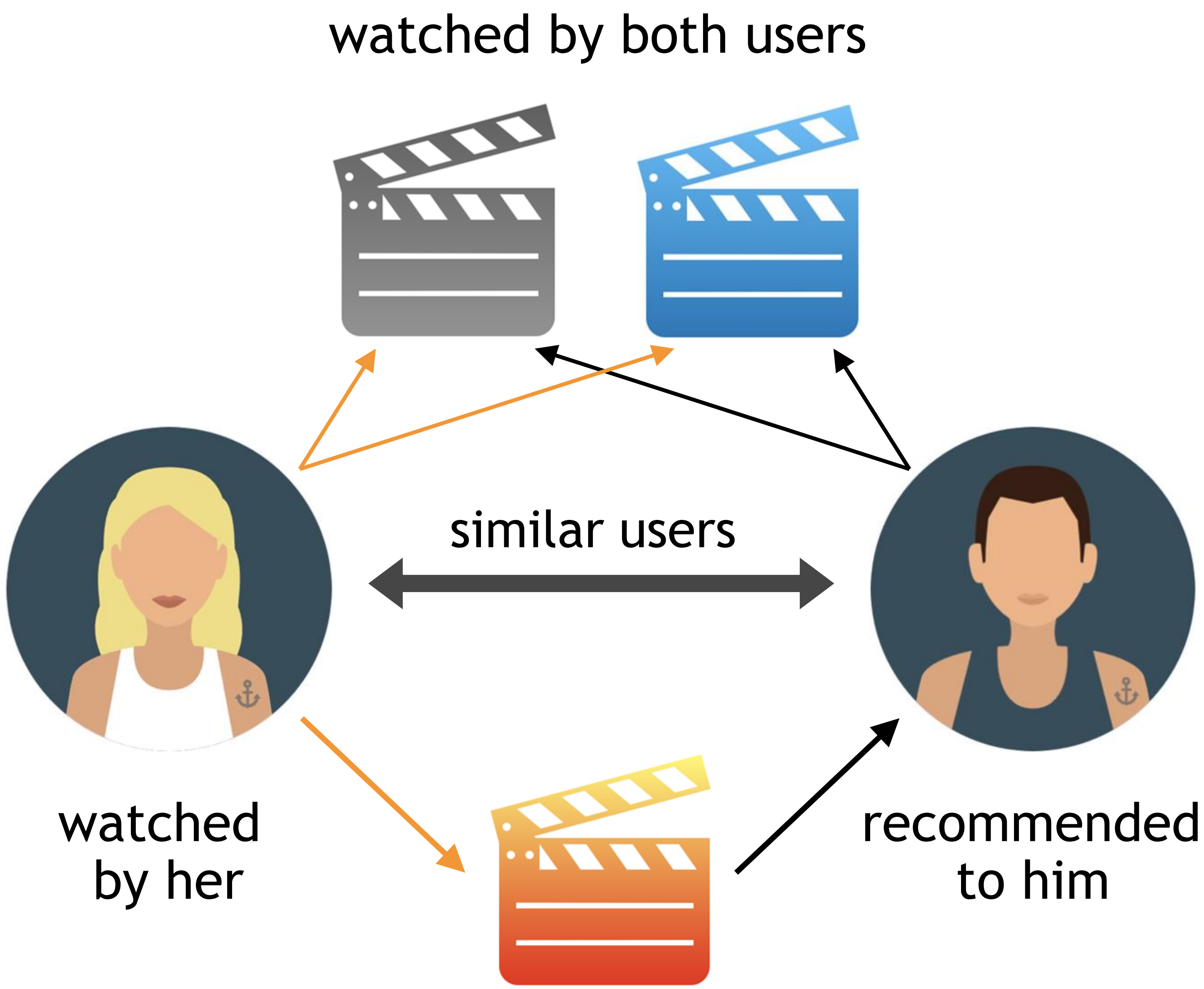
- TV shows I have already watched and liked. ❤️: 33%
- TV shows I haven’t watched yet, but I’d like to watch. 👍: 50%
- TV shows I am not sure I’d like to watch. 😕: 17%



RecSys Techniques

Multiple techniques mostly grouped in two categories

Collaborative Filtering

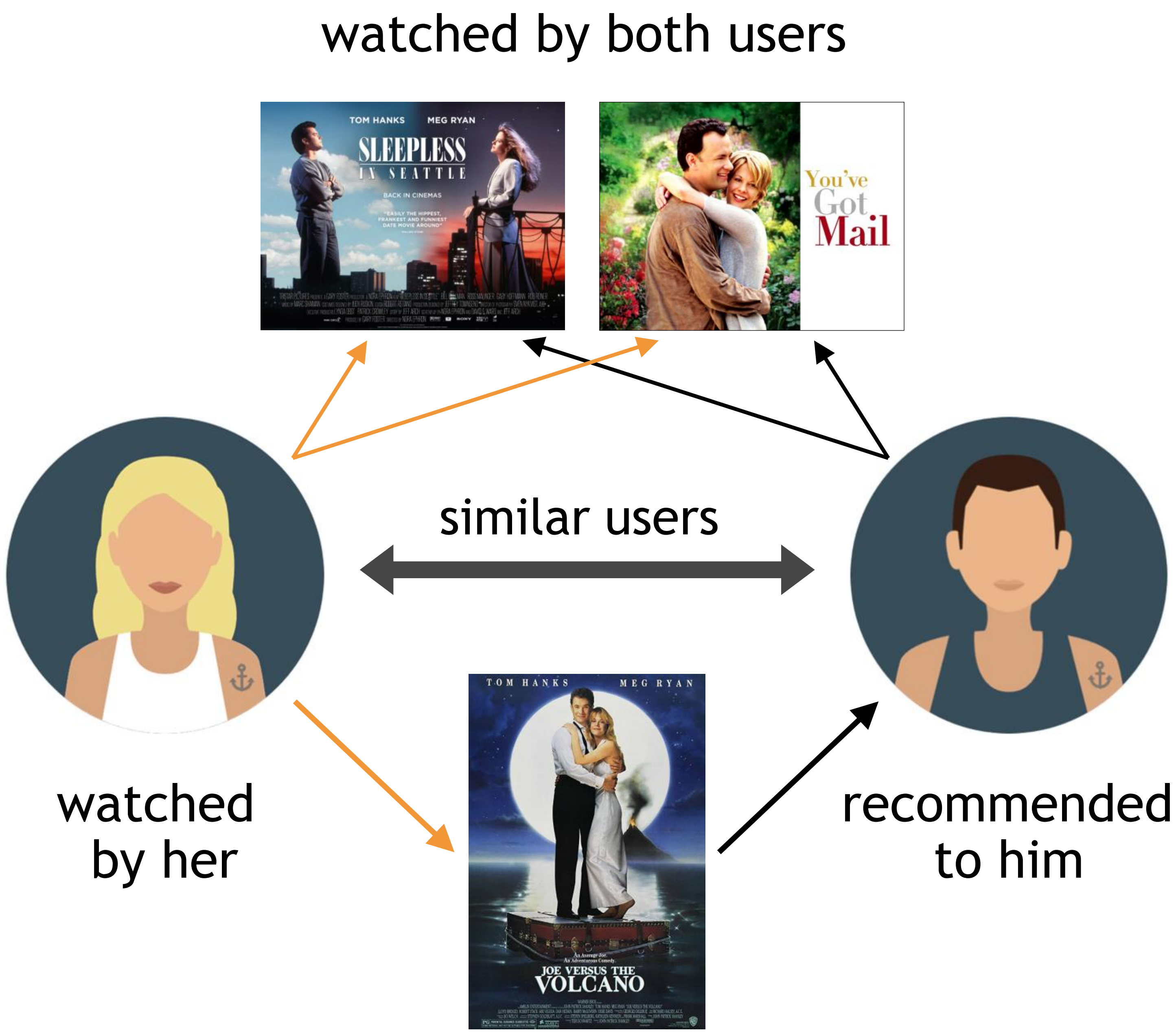


Content-based Filtering

Diagrams by Emma Grimaldi.

Multiple techniques mostly grouped in two categories

Collaborative Filtering

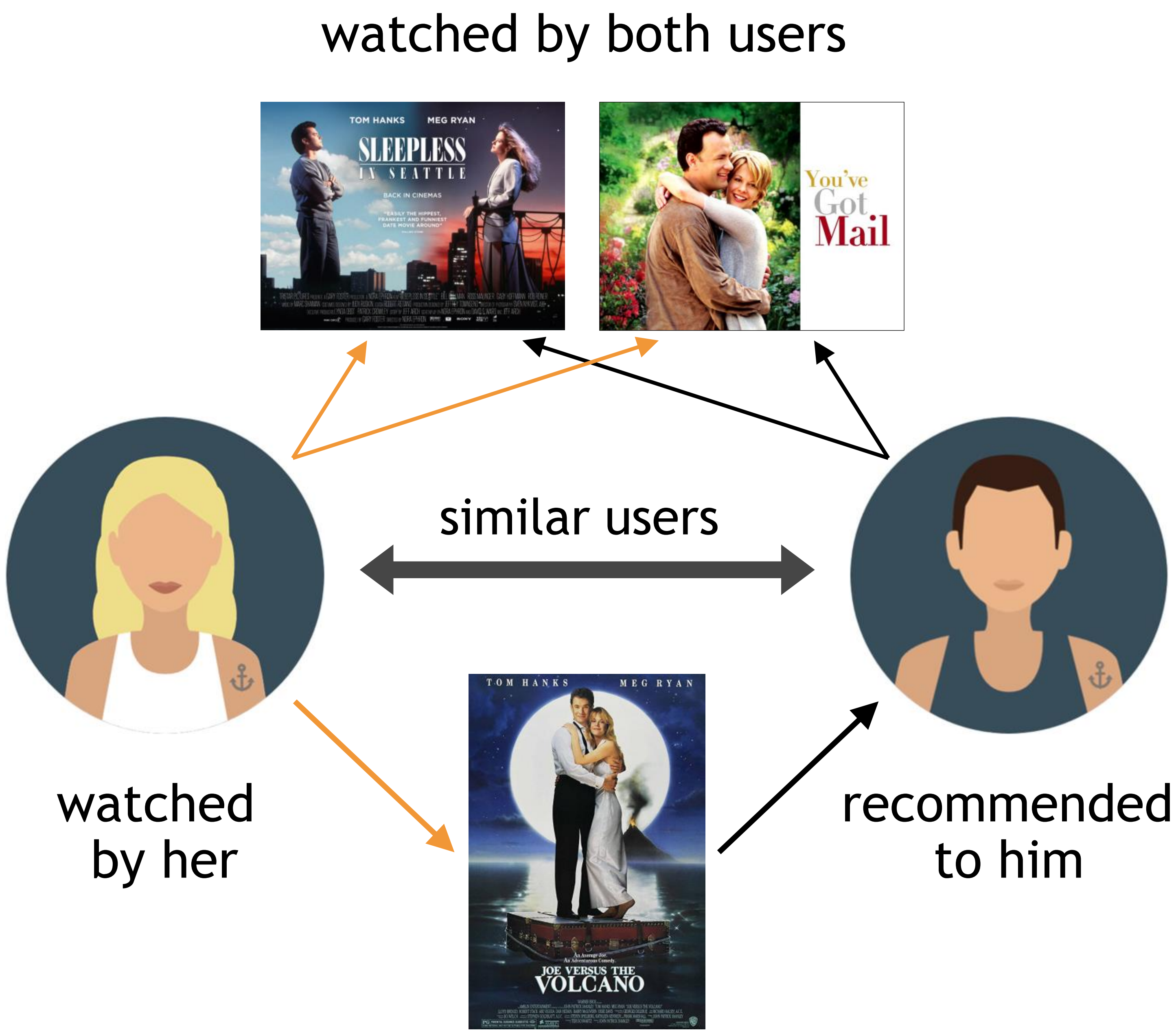


Content-based Filtering

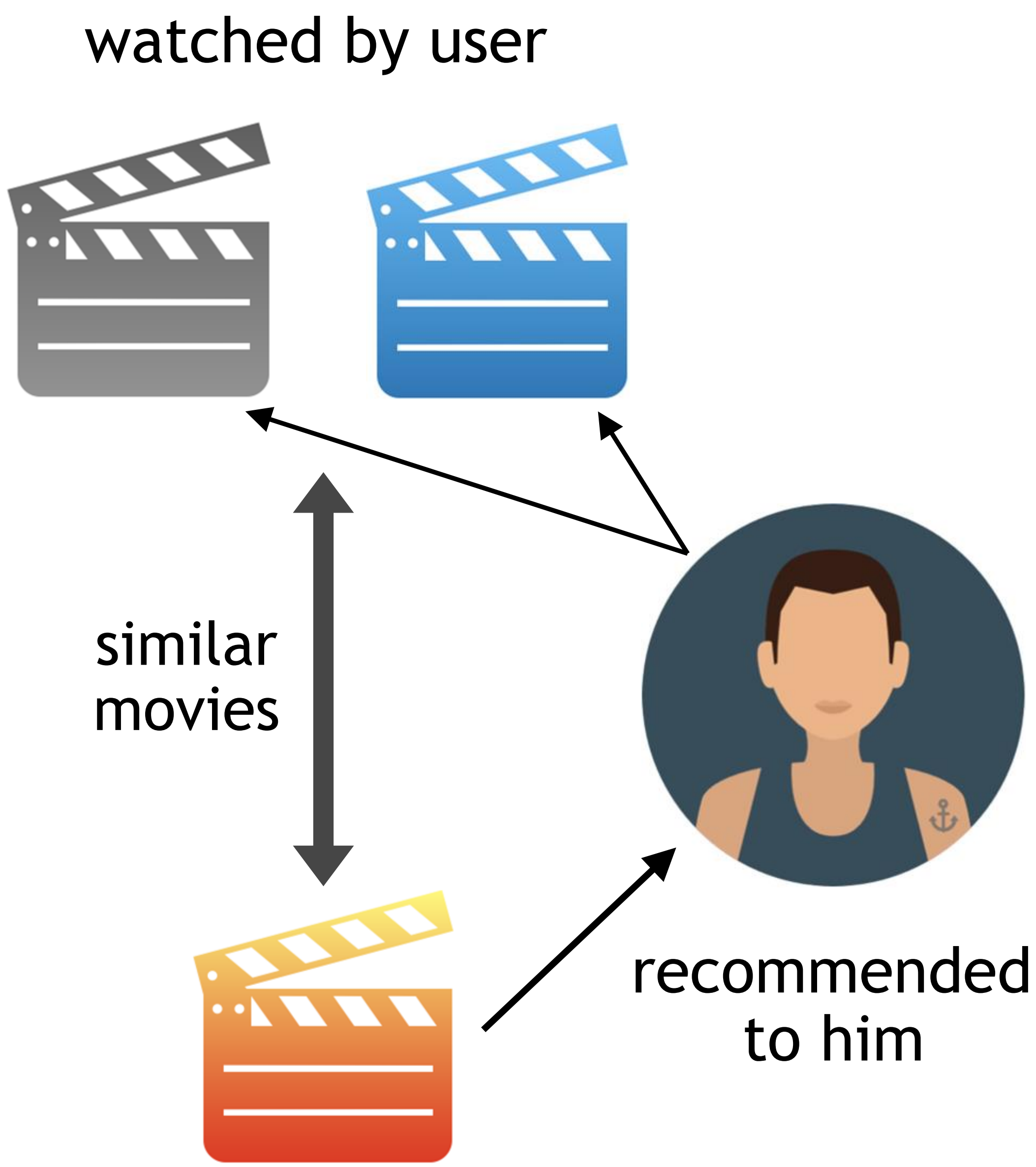
Diagrams by Emma Grimaldi.

Multiple techniques mostly grouped in two categories

Collaborative Filtering



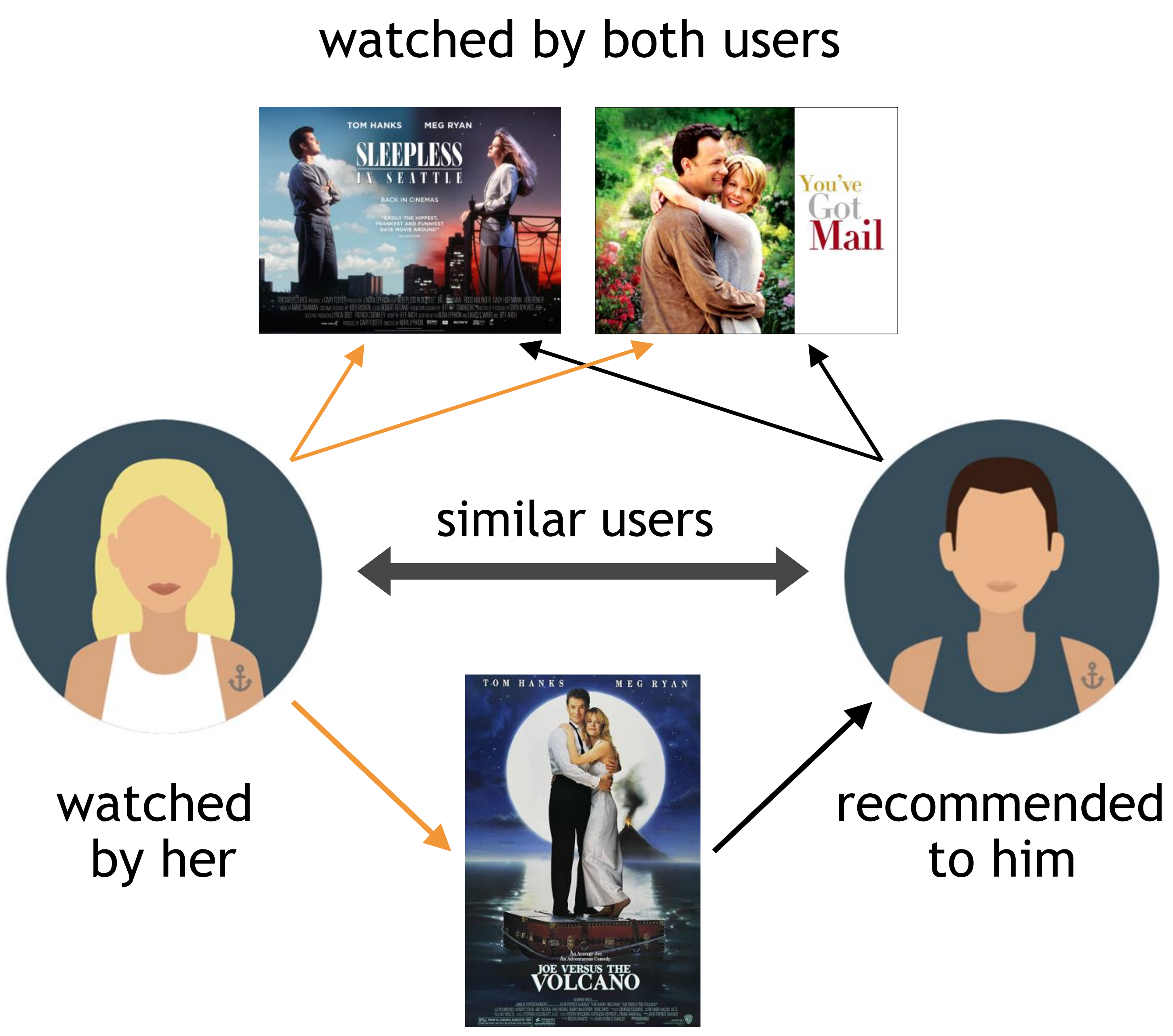
Content-based Filtering



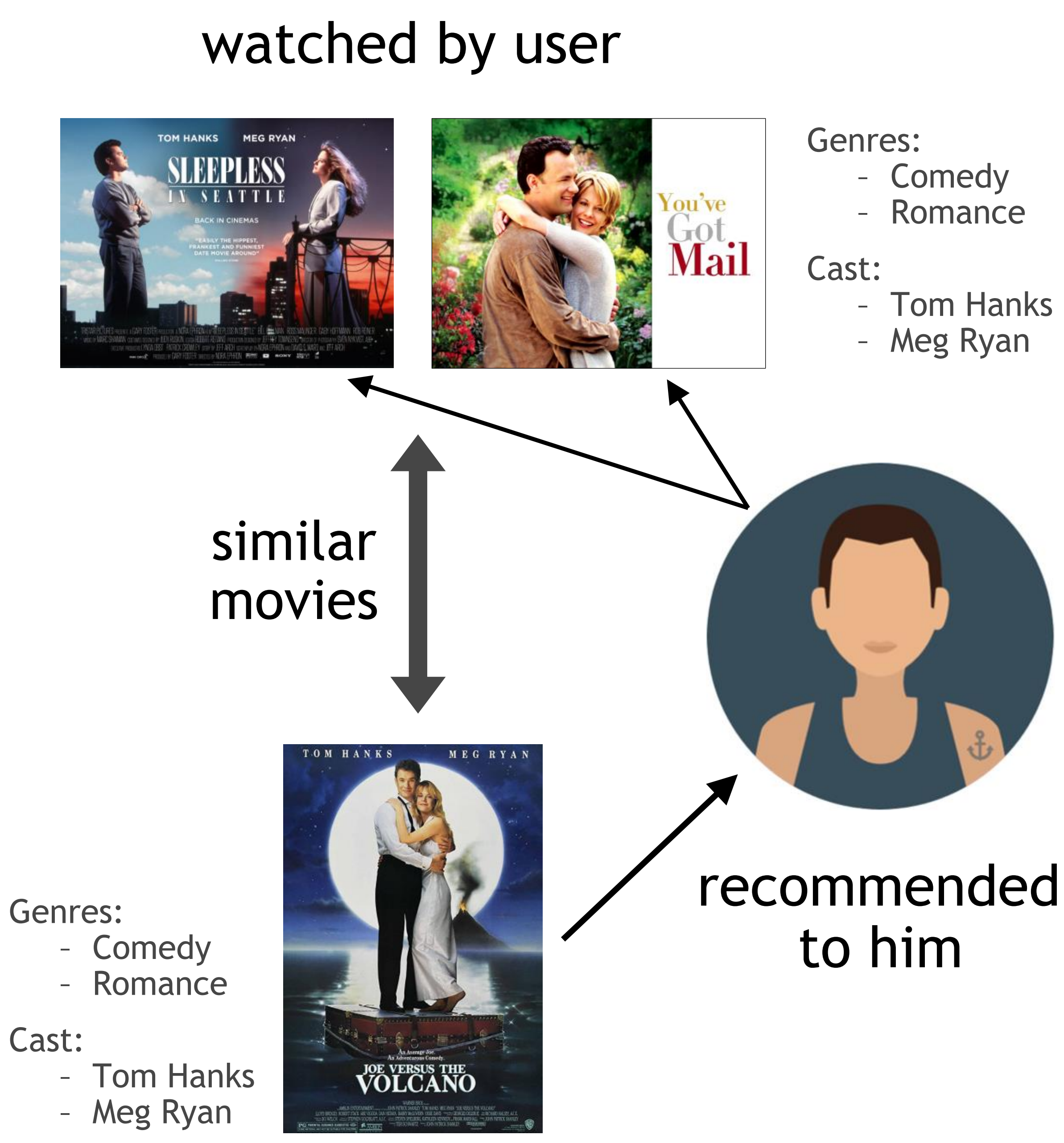
Diagrams by Emma Grimaldi.

Multiple techniques mostly grouped in two categories

Collaborative Filtering



Content-based Filtering

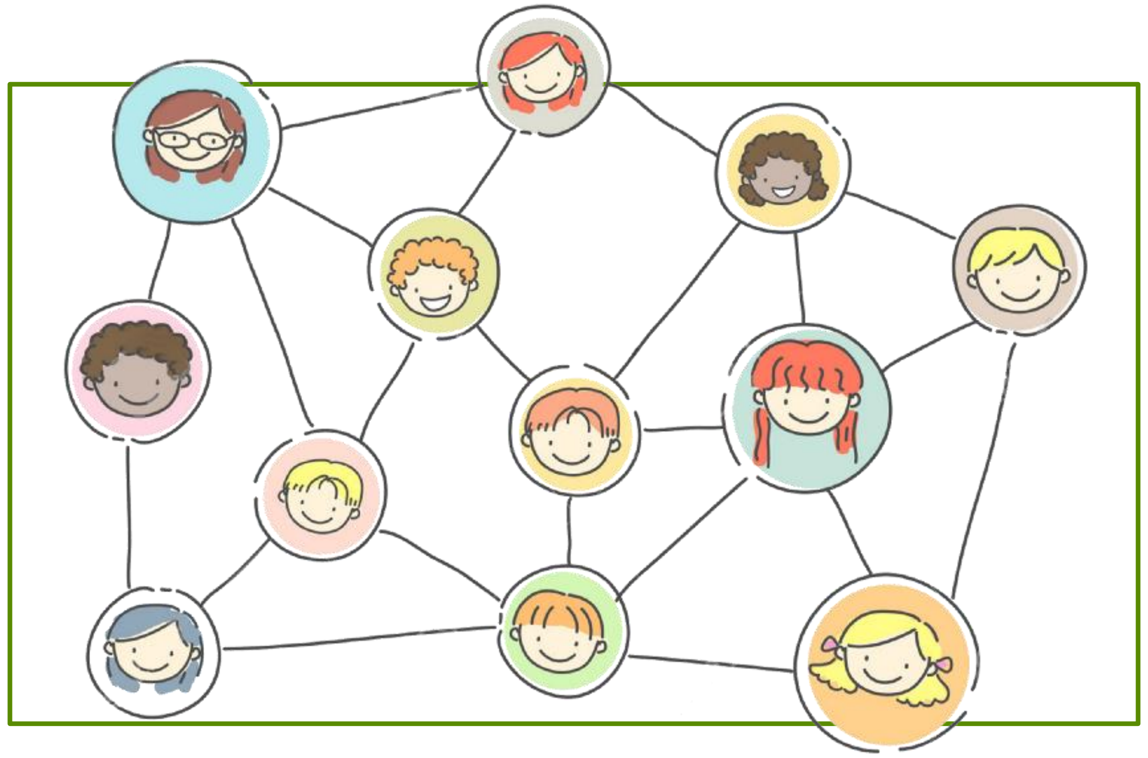


Diagrams by Emma Grimaldi.

Collaborative Filtering

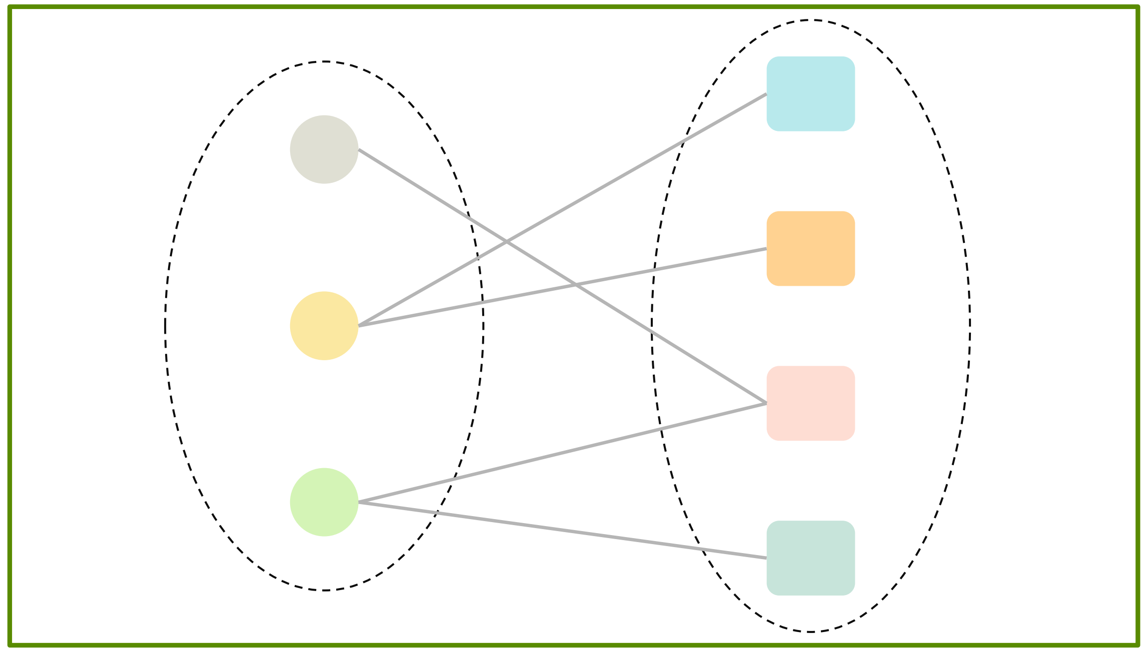
Collaborative Filtering

A graph-based approach
Jaccard Similarity vs Overlap Coefficient



A **graph** is a set of objects called *nodes* or *vertices* that are connected together.

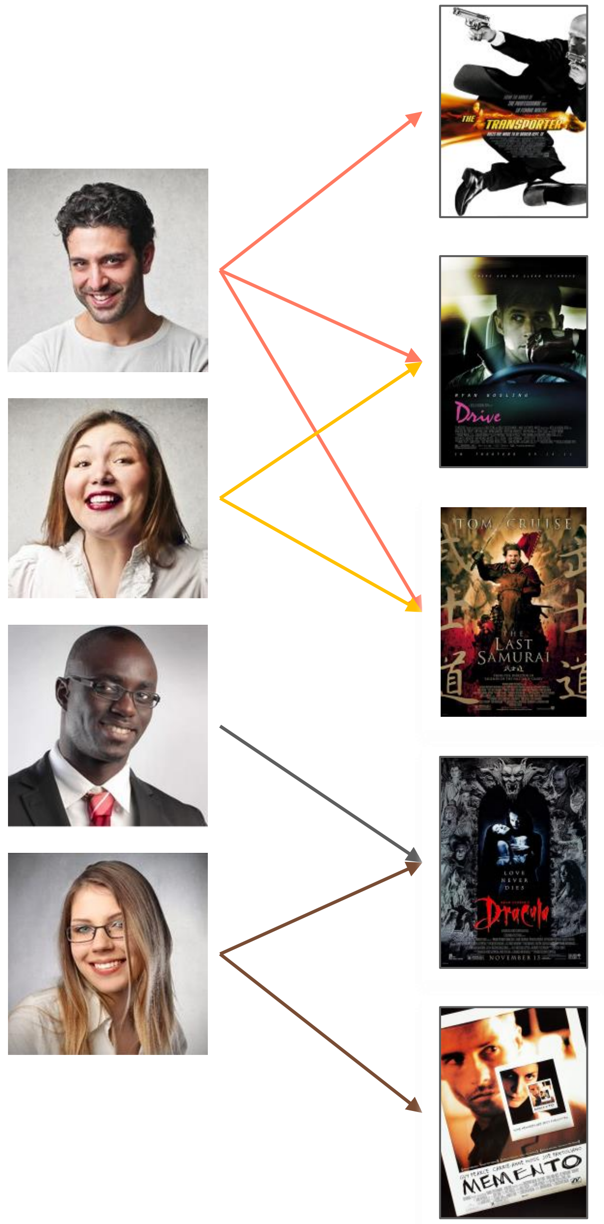
The connections between the nodes are known as *edges* or *links*.



A **bipartite graph**, or *bigraph*, is a graph whose nodes can be divided into two disjoint and independent sets, and such that every edge connects a node in one set to one node in the other set.

Collaborative Filtering

A graph-based approach
Jaccard Similarity vs Overlap Coefficient

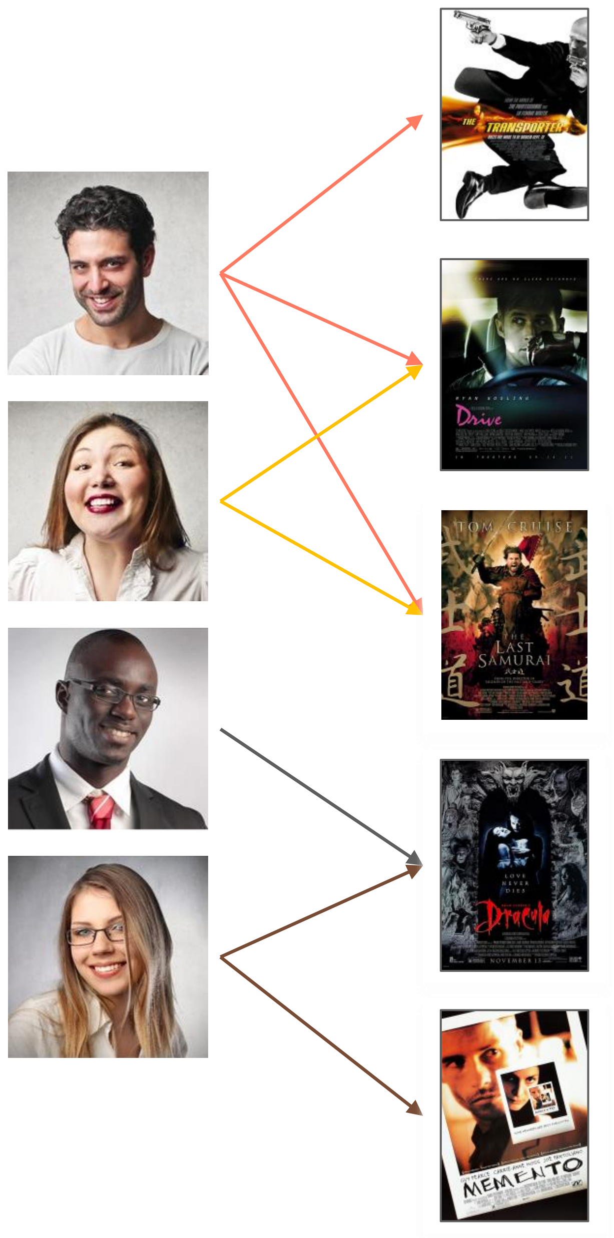


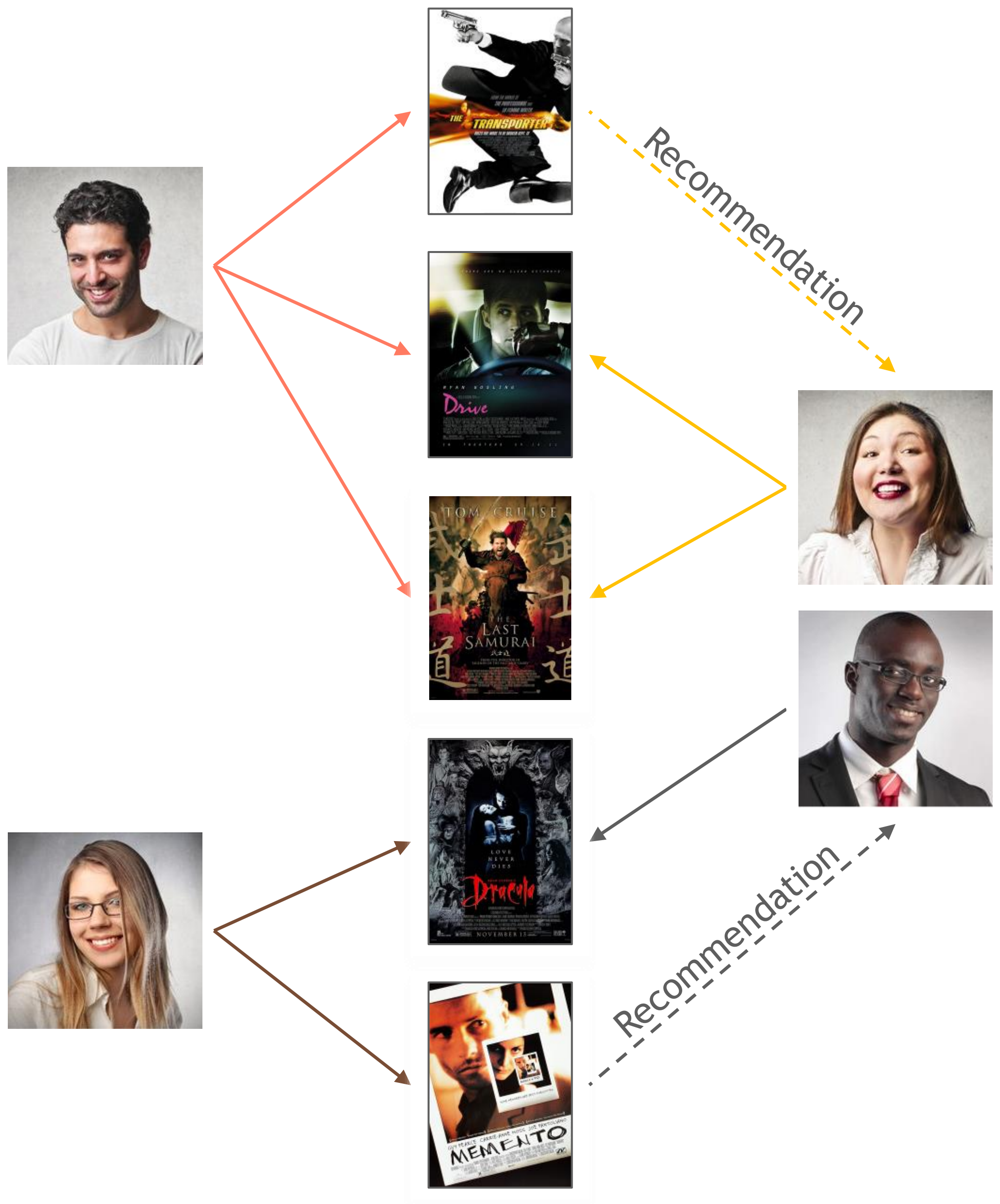
Collaborative Filtering

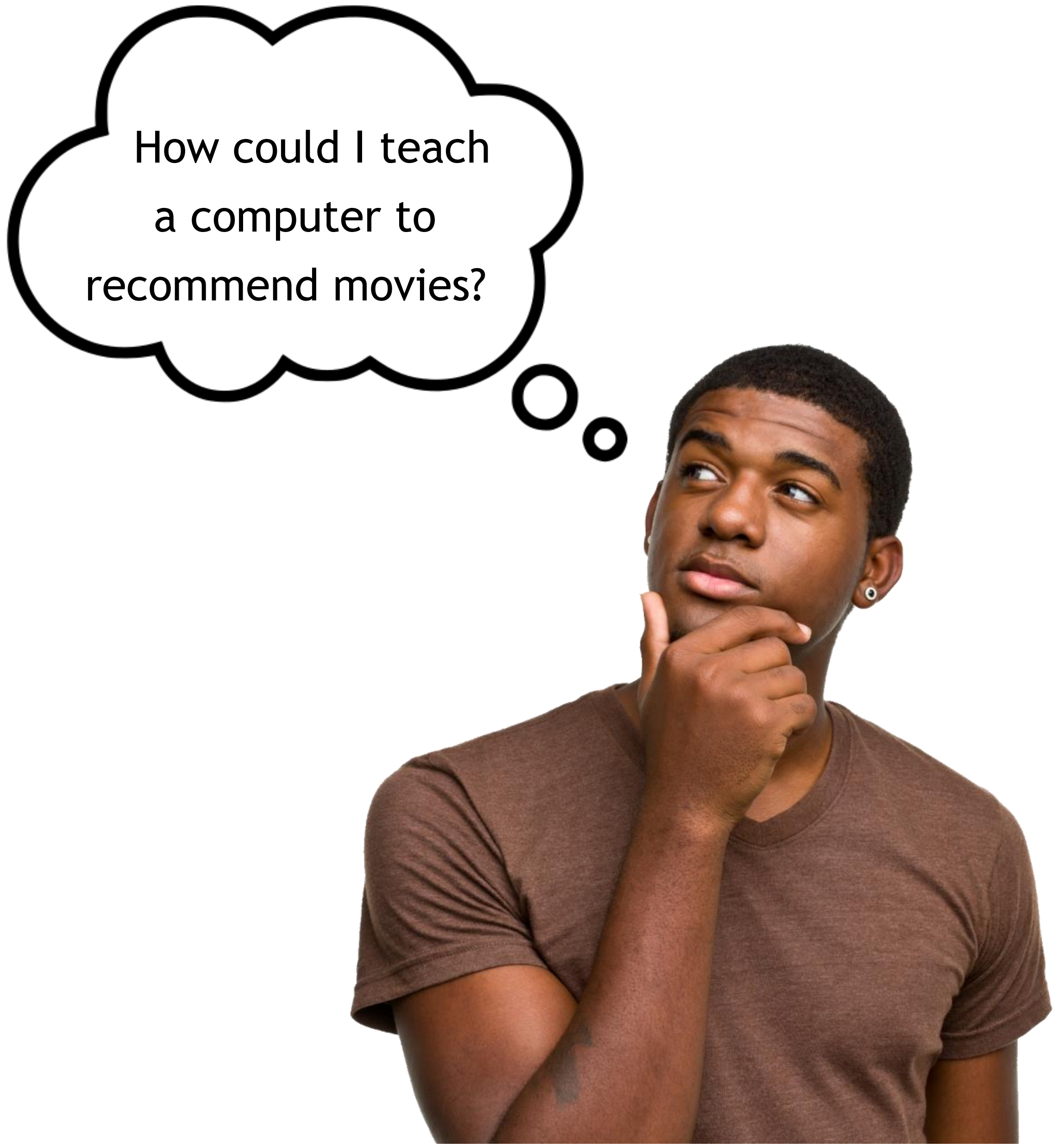
A graph-based approach
Jaccard Similarity vs Overlap Coefficient

Collaborative Filtering

A graph-based approach
Jaccard Similarity vs Overlap Coefficient





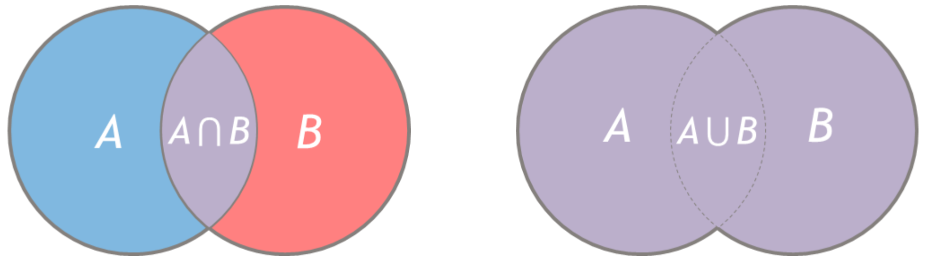


Collaborative Filtering

A graph-based approach
Jaccard Similarity vs Overlap Coefficient

Jaccard Similarity

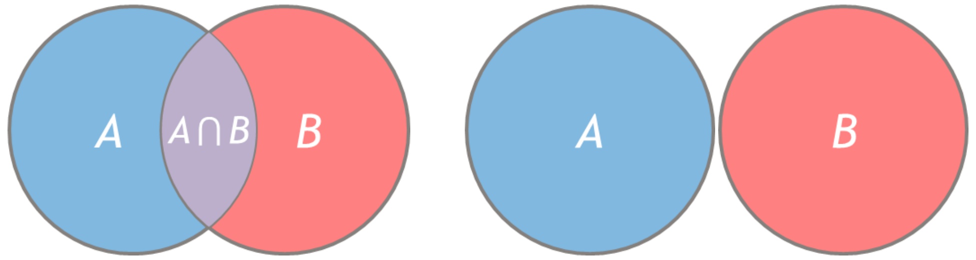
$$js(A,B) = \frac{|A \cap B|}{|A \cup B|}$$



Intersection *Union*

Overlap Coefficient

$$oc(A,B) = \frac{|A \cap B|}{\min(|A|, |B|)}$$



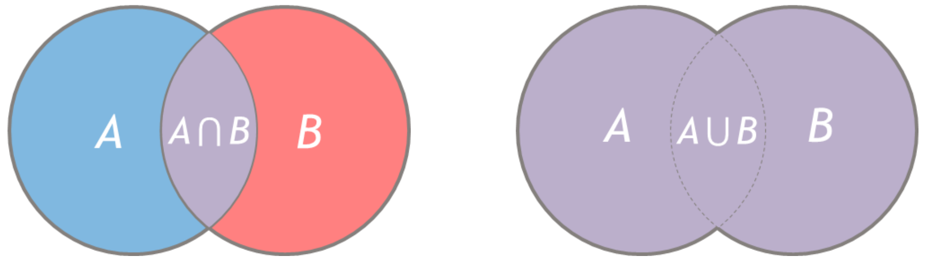
Intersection

Collaborative Filtering

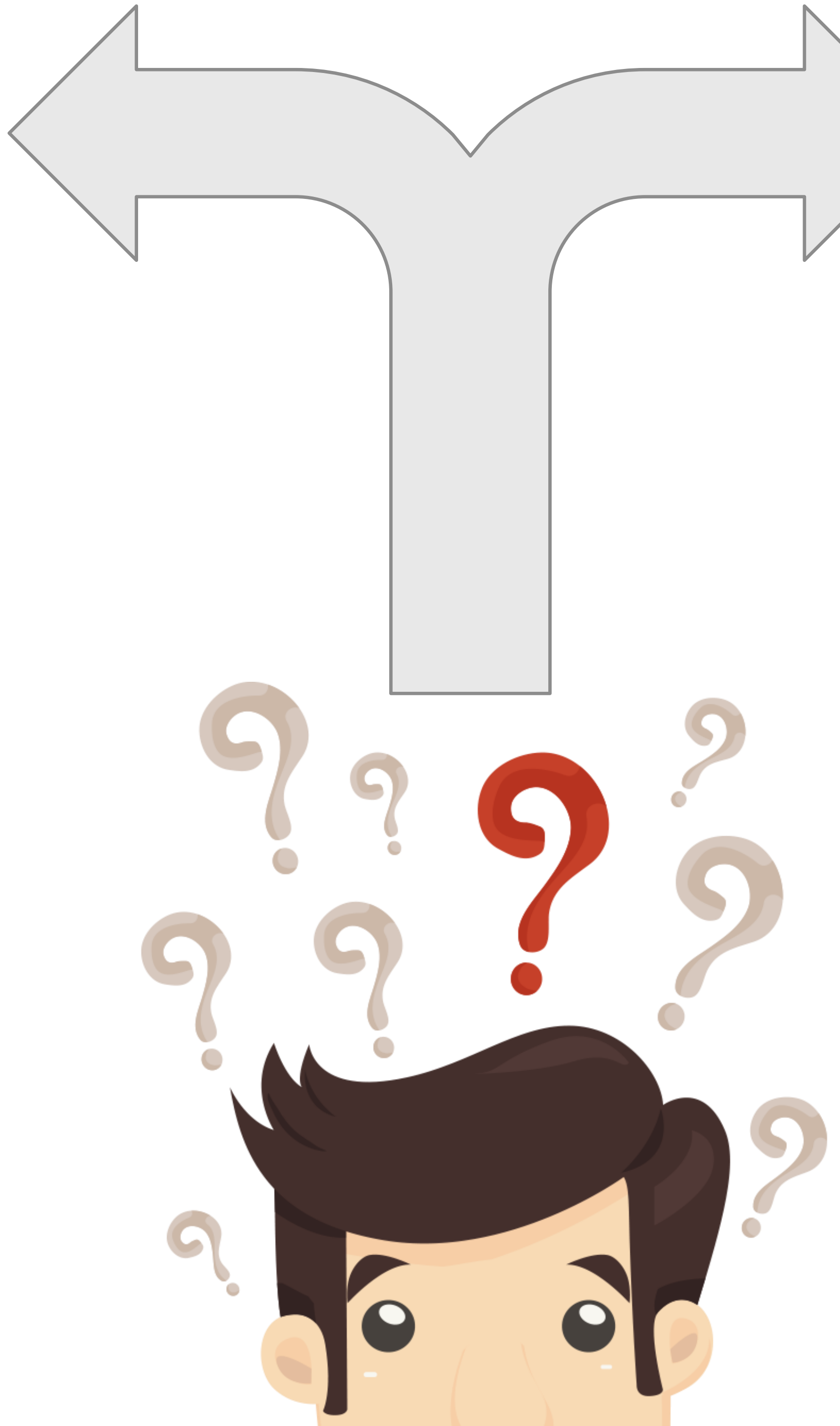
A graph-based approach
Jaccard Similarity vs Overlap Coefficient

Jaccard Similarity

$$js(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

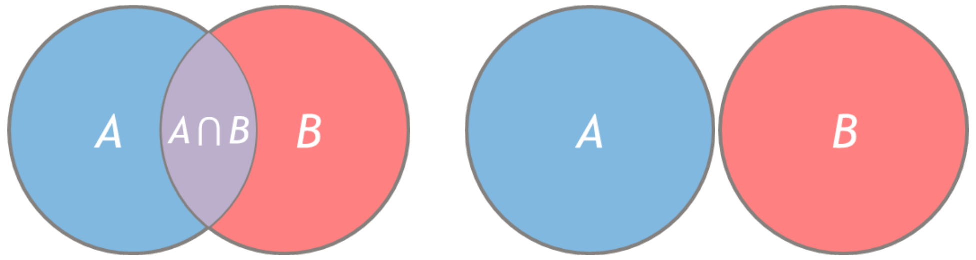


Intersection *Union*

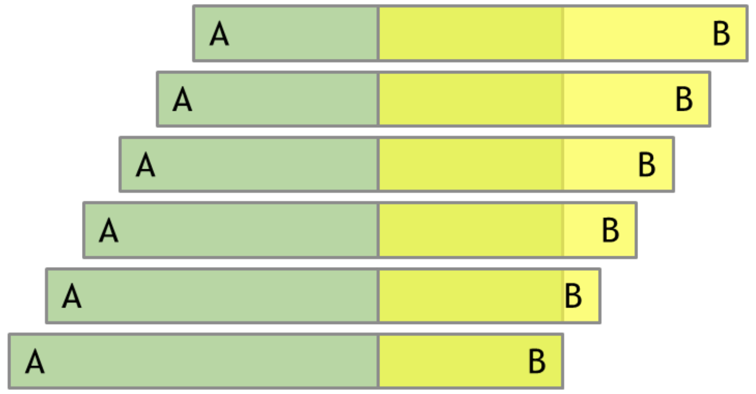


Overlap Coefficient

$$oc(A,B) = \frac{|A \cap B|}{\min(|A|,|B|)}$$



Intersection



A	B	Jaccard	Overlap
100	100	0.333	0.500
110	90	0.333	0.556
120	80	0.333	0.625
130	70	0.333	0.714
140	60	0.333	0.833
150	50	0.333	1.000

Collaborative Filtering

A clustering approach
k-Means + KNN

We will group the items we would like to recommend in different clusters.

Unsupervised machine learning task:

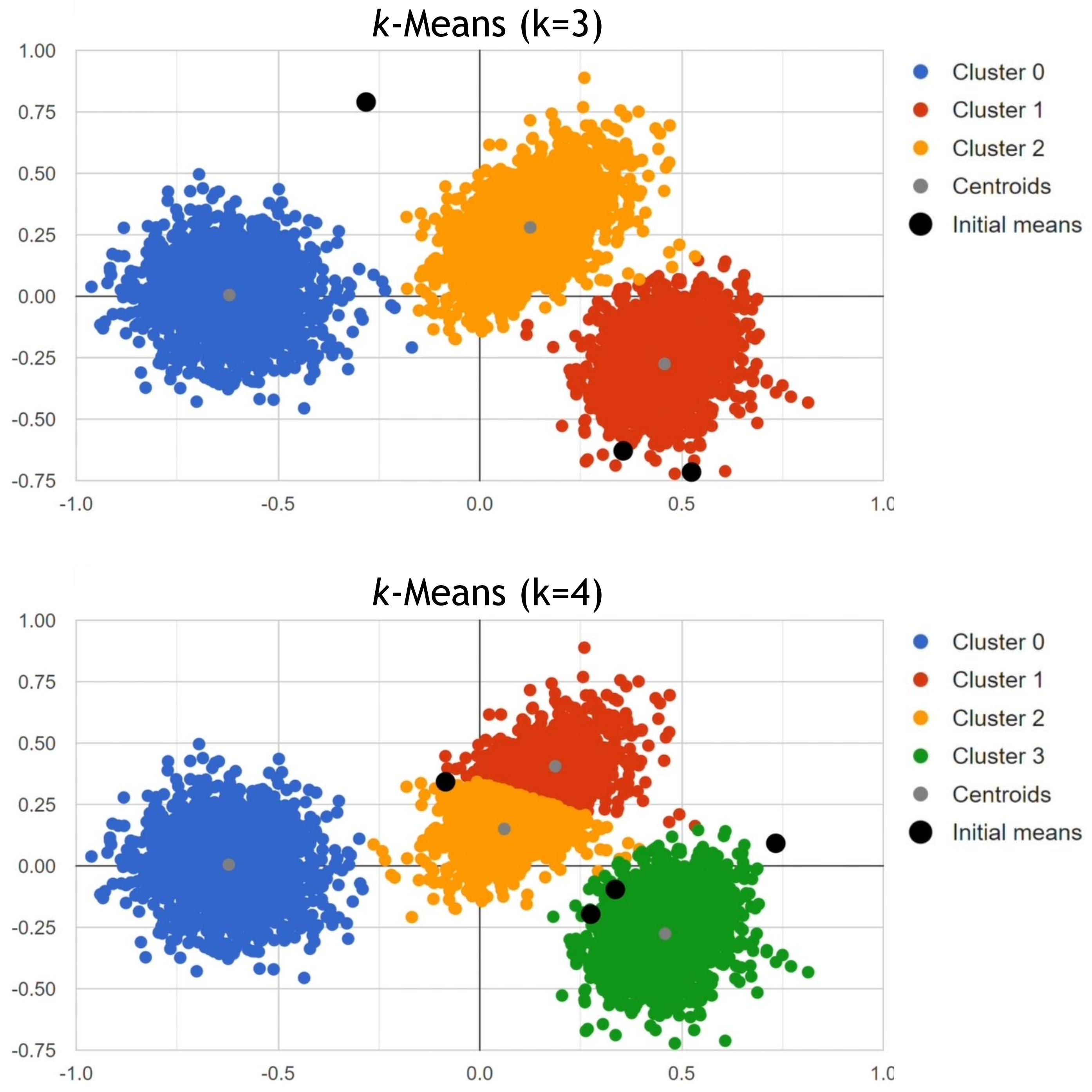
- Finding groups in a features space.

There are multiple clustering algorithms:

- *k*-Means, DBSCAN, Gaussian Mixture Model…

***k*-Means clustering**:

- The number of clusters is a hyperparameter.
- It is a non-deterministic algorithm:
  - Initial means are randomly generated.



http://syskall.com/kmeans.js

Collaborative Filtering

A clustering approach
k-Means + KNN

movie_id	title	year	genre	budget	country
0000	The Lion King	2019	Adventure	$260,000,000	USA
0001	Toy Story	1995	Comedy	$30,000,000	USA
…	…	…	…	…	…
3098	Seven Samuray	1954	Action	$1,129,178	Japan
3099	Good Will Hunting	1997	Drama	$10,000,000	USA

```python
import cudf
import cuml

df = cudf.read_csv('movies.csv')

kmeans = cuml.KMeans(n_clusters=10, max_iter=300, init='random')
kmeans.fit(df)

df['cluster'] = kmeans.labels_
```

k-Means (k=10)

Collaborative Filtering

A clustering approach
k-Means + KNN

user_id	user_name
00000	Camilla Manning
00001	Eloise Banks
…	…
21347	Nadia Reynolds
21348	Stephen Daniel

movie_id	title	…	cluster
0000	The Lion King	…	3
0001	Toy Story	…	3
…	…	…	…
3098	Seven Samuray	…	1
3099	Good Will Hunting	…	7

user_id	movie_id	rating
00000	2137	8
00000	1624	3
…	…	…
21348	3097	9
21348	1943	6

		user_id								
		00000	00001	00002	00003	…	21345	21346	21347	21348
movie_id	0000	0	0	0	0	…	0	9	0	0
	0001	0	0	0	4	…	0	0	0	0
	0002	0	0	0	0	…	7	0	0	0
	…	…	…	…	…	…	…	…	…	…
	3096	0	0	0	0	…	0	0	0	0
	3097	0	0	0	0	…	0	8	0	4
	3098	6	0	0	0	…	0	0	0	0
	3099	0	0	0	0	…	0	0	0	0

user ratings

Collaborative Filtering

A clustering approach
k-Means + KNN

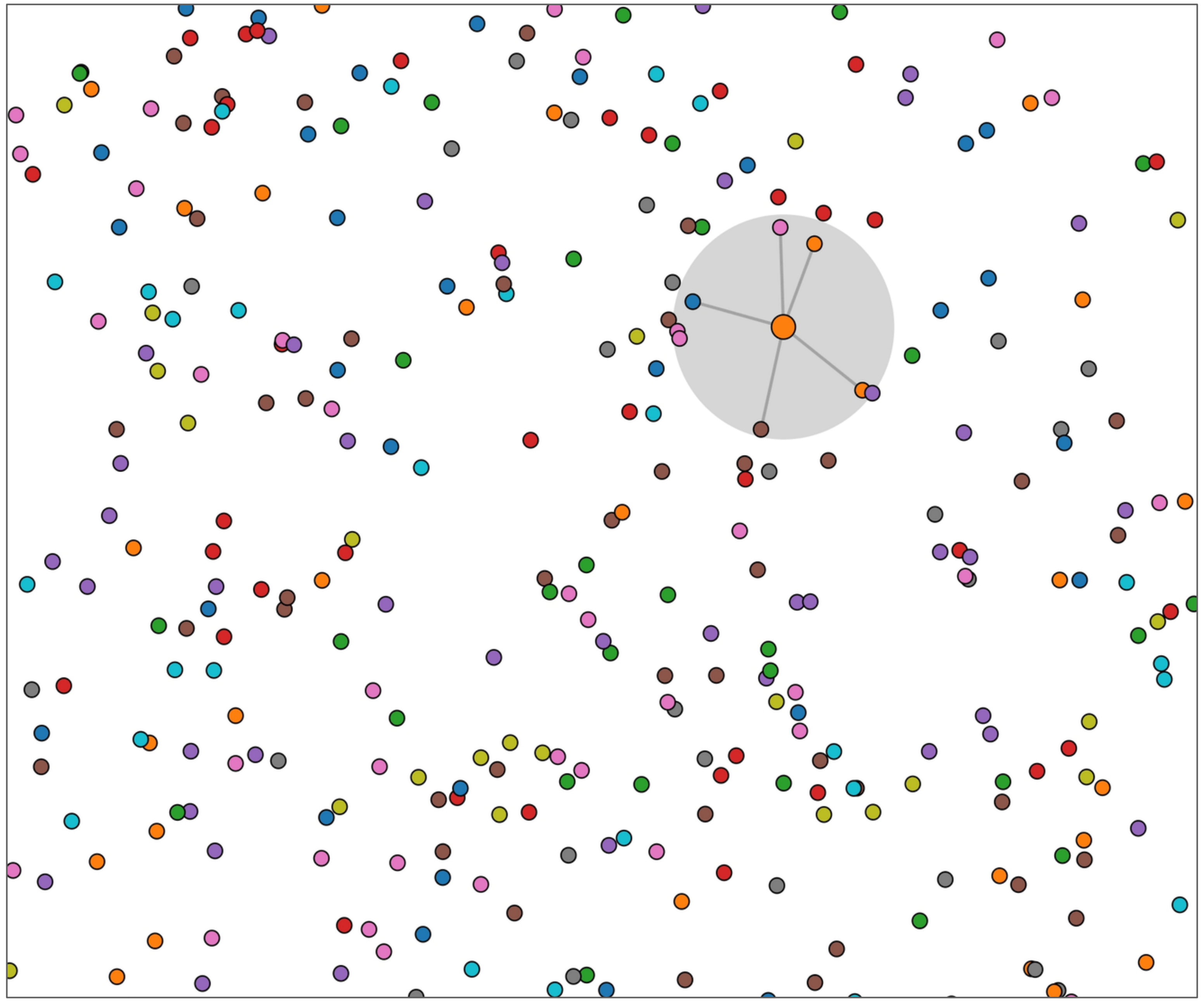
***k*-Nearest Neighbours (*k*-NN)** finds a predefined number of training samples closest in distance to a point, and predicts its label from these.

- The 'euclidean' metric is unhelpful in high dimensions.
- The user ratings matrix is sparse → use 'cosine' metric.

```
import cudf
import cuml

df = cudf.read_csv('movie_ratings.csv')

knn = cuml.NearestNeighbors(n_neighbors=5, metric='cosine')
knn.fit(df)
# ...
distances, movies = knn.kneighbors(user_fav_movies_df)
```



New movie classification

Content-based Filtering

Content-based Filtering

A sweet example

Candy	Chocolate	Round	Colorful	Fruity	Caramel	Chewy
M&Ms	✓	✓	✓	✗	✗	✗
Skittles	✗	✓	✓	✓	✗	✗
Snickers	✓	✗	✗	✗	✓	✗
Laffy Taffy	✗	✗	✓	✓	✗	✓
Caramel Chew	✗	✗	✗	✗	✓	✓

Content-based Filtering

A sweet example

Candy	Chocolate	Round	Colorful	Fruity	Caramel	Chewy
M&Ms	✓	✓	✓	✗	✗	✗
Skittles	✗	✓	✓	✓	✗	✗
Snickers	✓	✗	✗	✗	✓	✗
Laffy Taffy	✗	✗	✓	✓	✗	✓
Caramel Chew	✗	✗	✗	✗	✓	✓

Lara's Rating	Candy	Chocolate	Round	Colorful	Fruity	Caramel	Chewy
3	M&Ms	3	3	3	0	0	0
-	Skittles	-	-	-	-	-	-
5	Snickers	5	0	0	0	5	0
-	Laffy Taffy	-	-	-	-	-	-
-	Caramel Chew	-	-	-	-	-	-
	Total:	**8**	**3**	**3**	**0**	**5**	**0**

$\Sigma = 19$

Content-based Filtering

A sweet example

Candy	Chocolate	Round	Colorful	Fruity	Caramel	Chewy
M&Ms	✓	✓	✓	✗	✗	✗
Skittles	✗	✓	✓	✓	✗	✗
Snickers	✓	✗	✗	✗	✓	✗
Laffy Taffy	✗	✗	✓	✓	✗	✓
Caramel Chew	✗	✗	✗	✗	✓	✓

Lara's Rating	Candy	Chocolate	Round	Colorful	Fruity	Caramel	Chewy
3	M&Ms	3	3	3	0	0	0
-	Skittles	-	-	-	-	-	-
5	Snickers	5	0	0	0	5	0
-	Laffy Taffy	-	-	-	-	-	-
-	Caramel Chew	-	-	-	-	-	-
	Total:	8	3	3	0	5	0

$\Sigma = 19$

Divide by Total Sum

$\div 19$

Candy	Chocolate	Round	Colorful	Fruity	Caramel	Chewy
Total:	0.42	0.16	0.16	0	0.26	0

Content-based Filtering

A sweet example

Candy	Chocolate	Round	Colorful	Fruity	Caramel	Chewy
M&Ms	✓	✓	✓	✗	✗	✗
Skittles	✗	✓	✓	✓	✗	✗
Snickers	✓	✗	✗	✗	✓	✗
Laffy Taffy	✗	✗	✓	✓	✗	✓
Caramel Chew	✗	✗	✗	✗	✓	✓

Lara's Rating
3
-
5
-
-

Candy	Chocolate	Round	Colorful	Fruity	Caramel	Chewy
M&Ms	3	3	3	0	0	0
Skittles	-	-	-	-	-	-
Snickers	5	0	0	0	5	0
Laffy Taffy	-	-	-	-	-	-
Caramel Chew	-	-	-	-	-	-
Total:	8	3	3	0	5	0

$\Sigma = 19$

Divide by Total Sum

$\div 19$

Candy	Chocolate	Round	Colorful	Fruity	Caramel	Chewy
Total:	0.42	0.16	0.16	0	0.26	0

Lara's Prediction
-
0.32
-
0.16
0.26

Candy	Chocolate	Round	Colorful	Fruity	Caramel	Chewy
M&Ms	-	-	-	-	-	-
Skittles	0	0.16	0.16	0	0	0
Snickers	-	-	-	-	-	-
Laffy Taffy	0	0	0.16	0	0	0
Caramel Chew	0	0	0	0	0.26	0

Collaborative Filtering
vs
Content-base Filtering

Collaborative Filtering

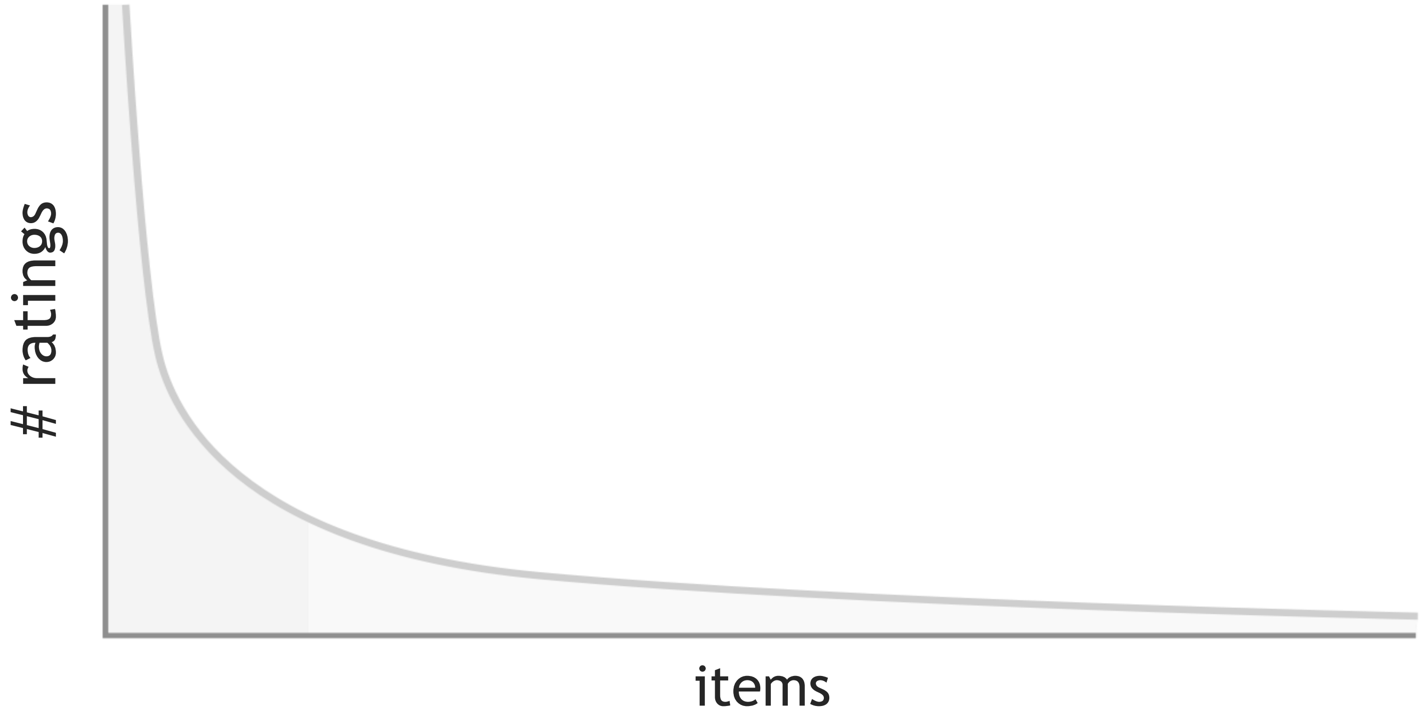
A few things to consider

Popularity bias:

- Popular items are more likely to be recommended.

Cold start issues:

- New community:
  - Refers to the system startup.
  - No data available the recommender can rely on.
- New user:
  - The system cannot rely on the user's past interactions to provide any recommendation.
- New item:
  - New items added to the catalogue have either none or very little interactions.



Long tail ratings distribution

Content-base Filtering

A few things to consider

~~Popularity bias:~~

- ~~Popular items are more likely to be recommended.~~

Cold start issues:

- New community:
  - Refers to the system startup.
  - No data available the recommender can rely on.
- New user:
  - The system cannot rely on the user’s past interactions to provide any recommendation.
- ~~New item:~~
  - ~~New items added to the catalogue have either none or very little interactions.~~

New community issue

Collaborative Filtering vs Content-based Filtering

No need to choose

Feature	Collaborative Filtering	Content-based Filtering
No Human Feature Engineering	✓	✗
Good at Expanding User's Interest	✓	✗
Can Recommend Highly Specific Items	✗	✓
Can Recommend New Items	✗	✓

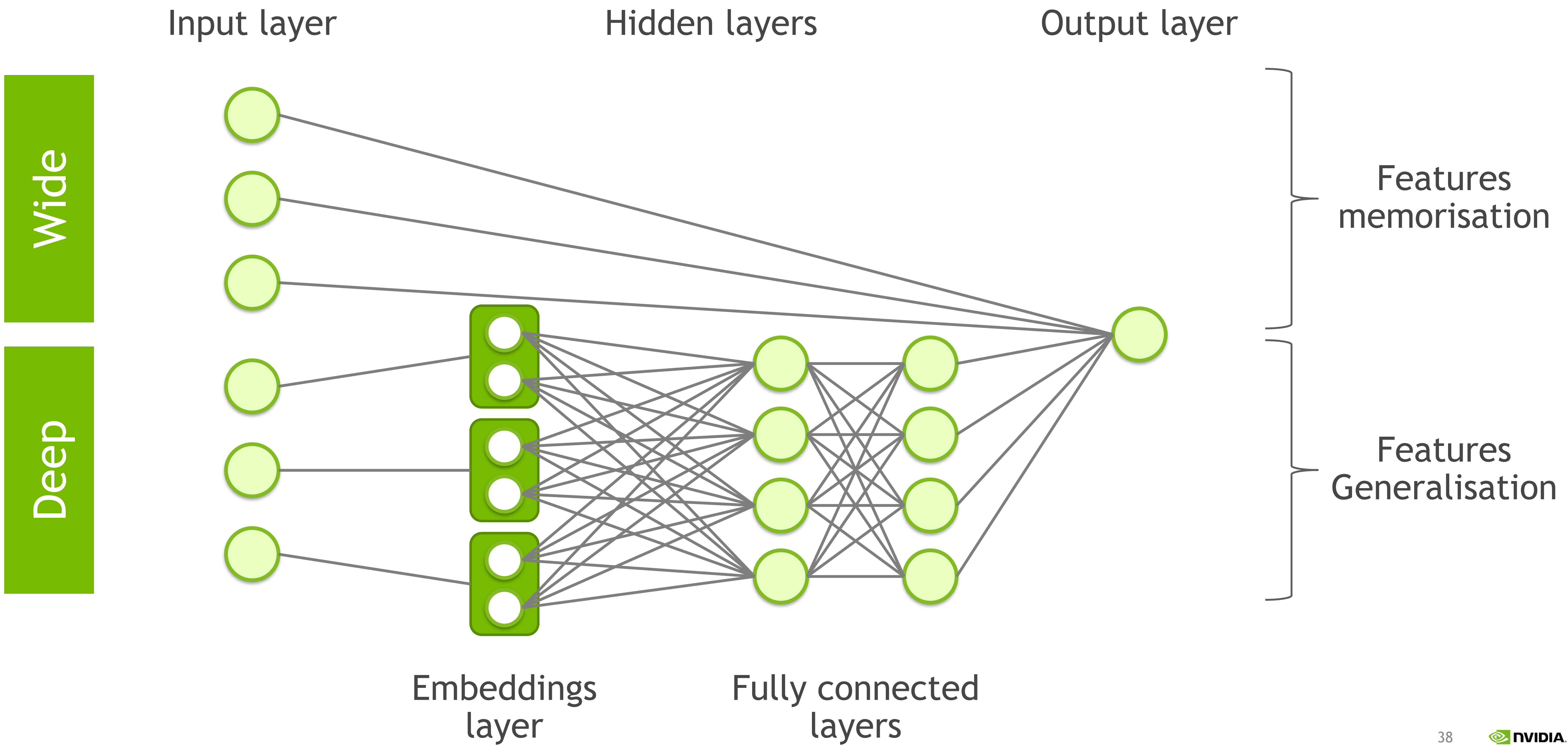
Solution: **hybrid approach** combining both techniques.

Diagrams by Emma Grimaldi.

Deep Learning Based Recommender Systems

Wide & Deep Neural Architecture

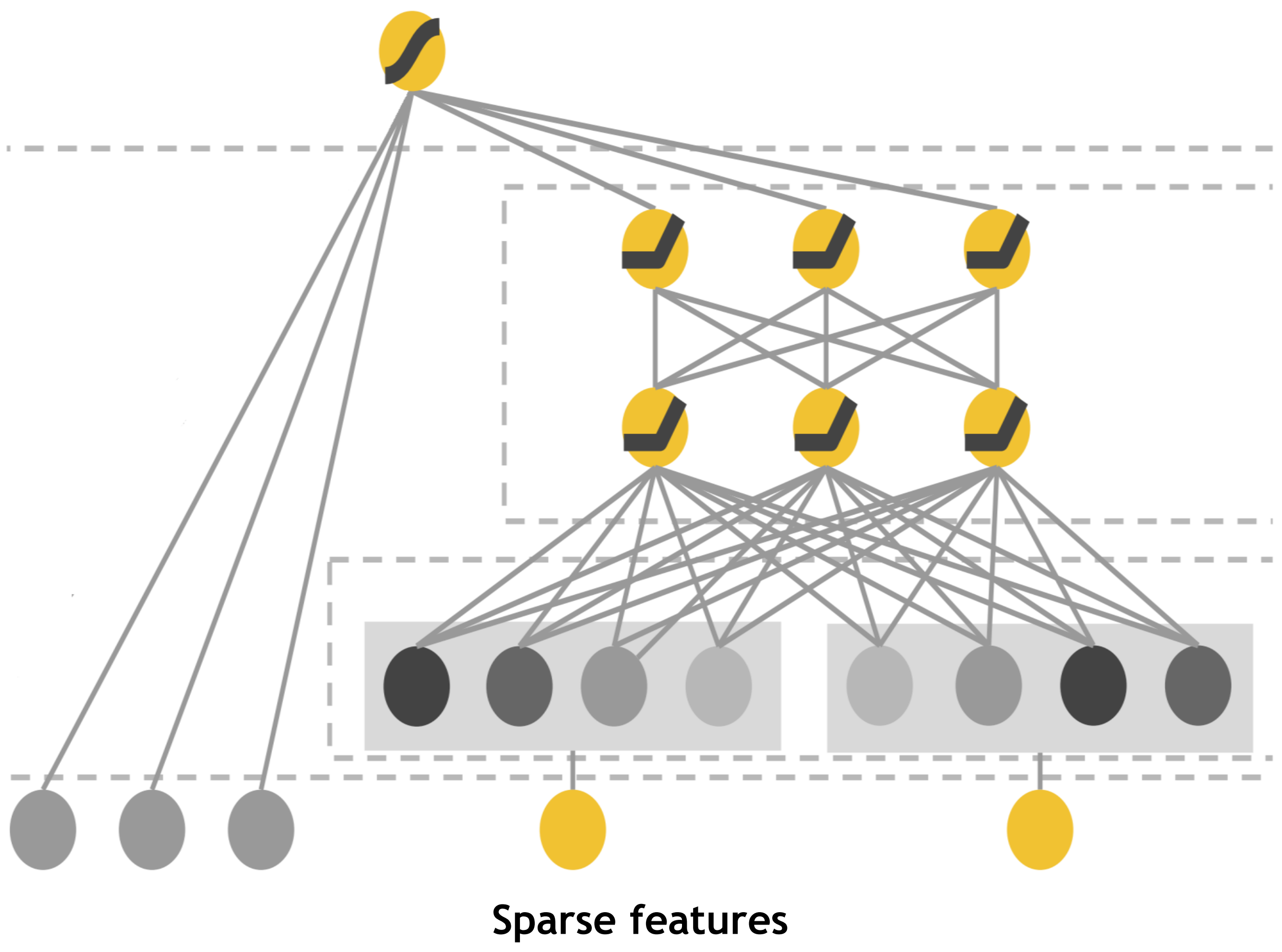
Learning how not to forget



Wide & Deep Neural Usage Examples

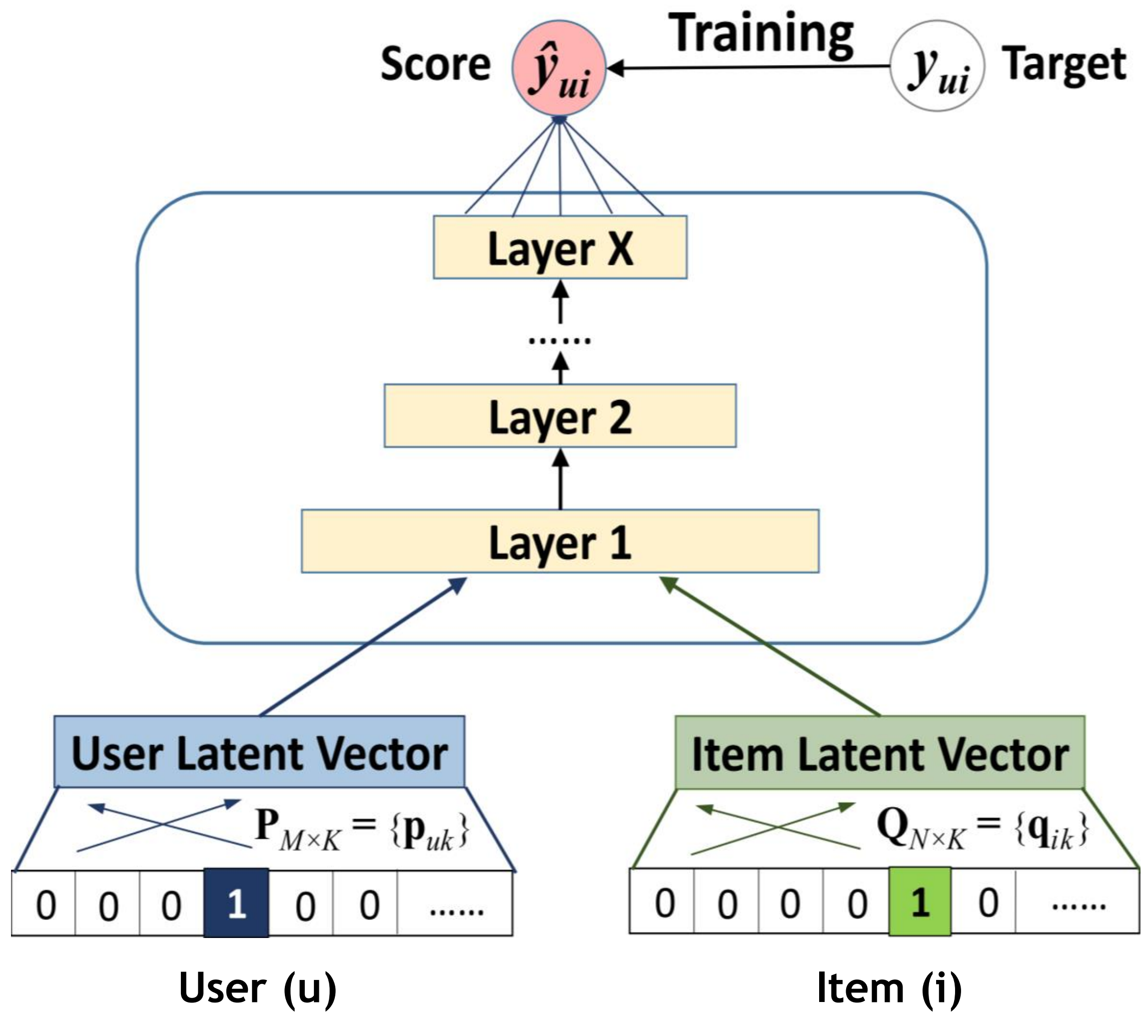
A *widely* and *deeply* used architecture

Google Wide & Deep



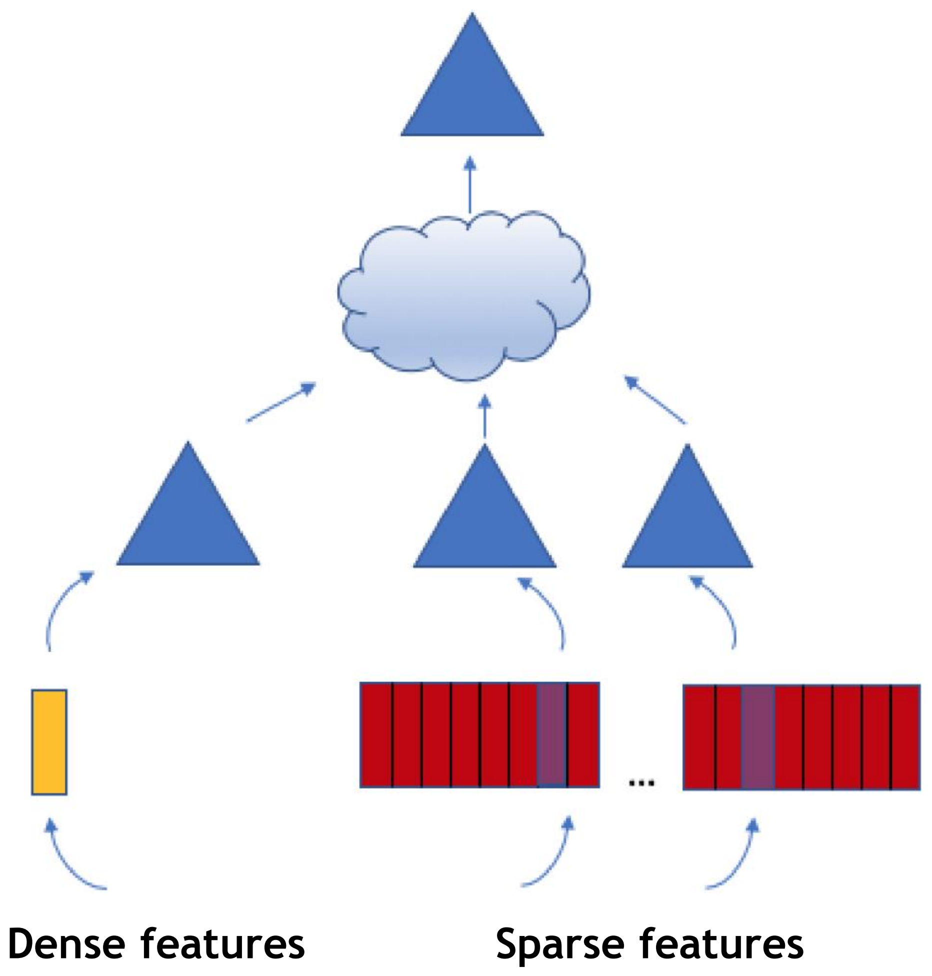
https://arxiv.org/abs/1606.07792

Neural Collaborative Filtering



https://arxiv.org/abs/1708.05031

Facebook DLRM



https://arxiv.org/pdf/1906.00091.pdf

Embeddings, Embeddings, Embeddings.

Huge Number of Customers and Products

Someone said embeddings?

10^9 users

user_id
200650946
447945298
473849537
…
329521575

→ → →

One-hot encoding ➔ large sparse matrix

user_id_1	user_id_2	user_id_3	…	user_id_10^9
1	0	0	…	0
0	1	0	…	0
0	0	1	…	0
…	…	…	…	…
0	0	0	…	1

→ → →

→ → →

Smaller dense embedding matrix

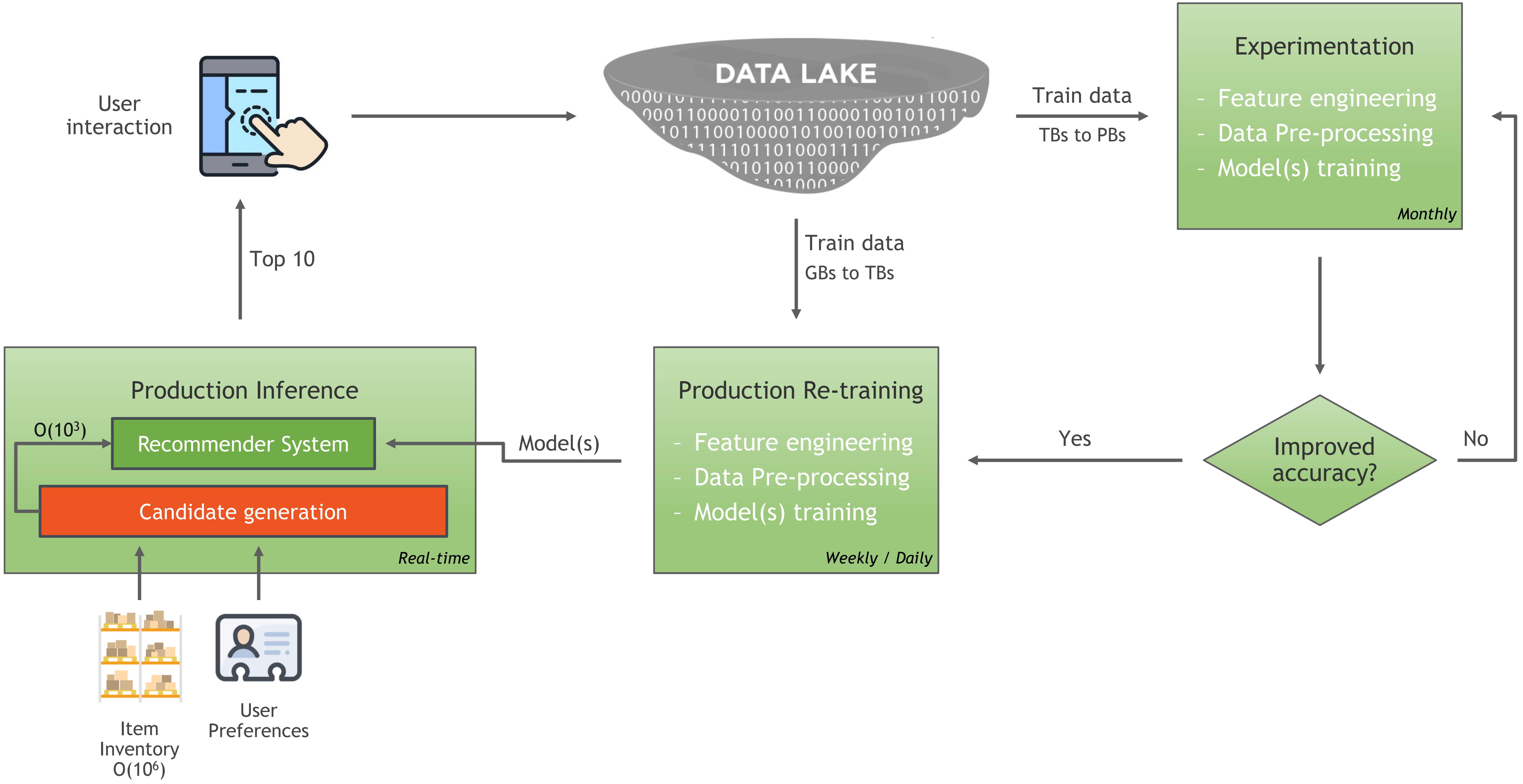
feat_1	feat_2	feat_3	…	feat_10^3
0.03244	0.20043	0.15545	…	0.97712
0.61397	0.65557	0.15097	…	0.74054
0.84247	0.35971	0.68802	…	0.97123
…	…	…	…	…
0.00279	0.57935	0.15785	…	0.89202

$10^9 >>> 10^3$

NVIDIA Merlin

Recommendation Pipelines

Example



Recommendation Pipelines

Challenges

Data (ETL)

Feature exploration

- Huge data sets: TBs, PBs or more.
- Complex data preprocessing and feature engineering pipelines.
- Many iterations required.

Data loading

- Data loading can be > 50% of total training time.
- Tabular data loading scales poorly with an item-by-item approach.

Recommendation Pipelines

Challenges

Data (ETL)

Feature exploration

- Huge data sets: TBs, PBs or more.
- Complex data preprocessing and feature engineering pipelines.
- Many iterations required.

Data loading

- Data loading can be > 50% of total training time.
- Tabular data loading scales poorly with an item-by-item approach.

Training

Huge embedding tables

- Large embedding tables exceed single GPU memory.
- Sub-optimal lookups ops implementation.

Performance & Accuracy

- Hard to achieve high scaling efficiency with both model and data parallelism.
- Longer iteration cycles reduce the ability to reach higher accuracies quickly.

Recommendation Pipelines

Challenges

Data (ETL)

Feature exploration

- Huge data sets: TBs, PBs or more.
- Complex data preprocessing and feature engineering pipelines.
- Many iterations required.

Data loading

- Data loading can be > 50% of total training time.
- Tabular data loading scales poorly with an item-by-item approach.

Training

Huge embedding tables

- Large embedding tables exceed single GPU memory.
- Sub-optimal lookups ops implementation.

Performance & Accuracy

- Hard to achieve high scaling efficiency with both model and data parallelism.
- Longer iteration cycles reduce the ability to reach higher accuracies quickly.

Inference

Throughput & Latency

- Difficult to have high throughput and low latency when ranking a huge number of items.

NVIDIA Merlin

DATA (ETL)

NVTabular

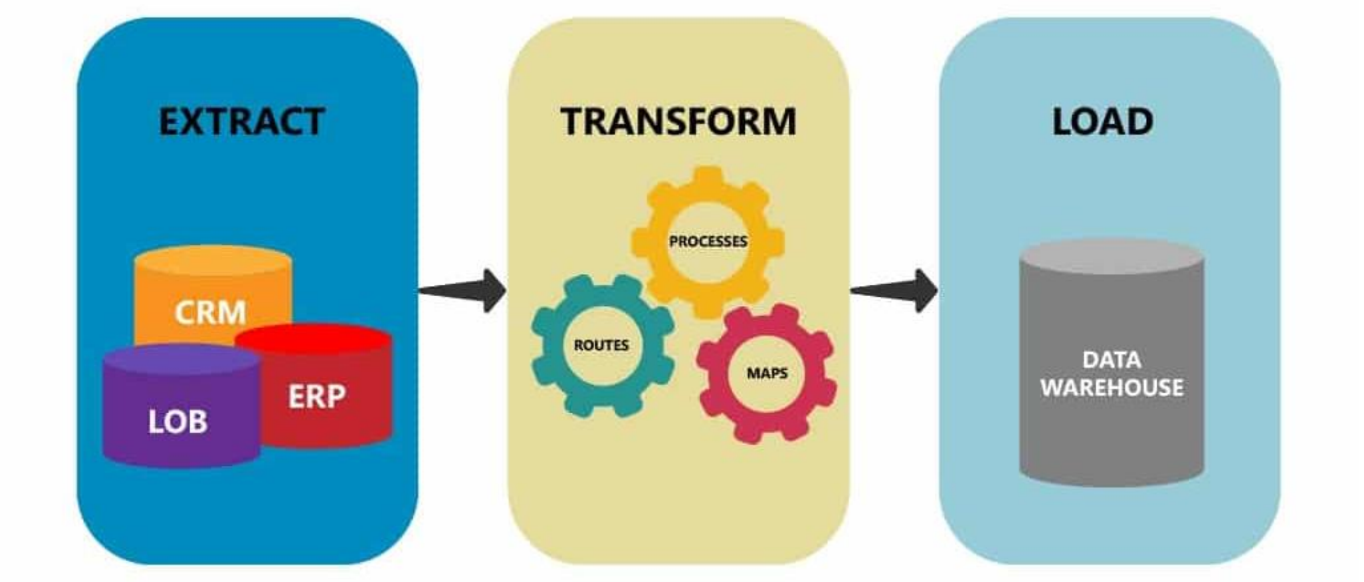
What it is:

Feature engineering and preprocessing library designed to quickly and easily manipulate terabytes of tabular data.

What it's capable of:

- **Scale** – No limit on dataset size (not bound by GPU or CPU memory).
- **Speed -** GPU acceleration, 10x speedup compared to CPU, eliminate input bottleneck.
- **Usability** - Higher level abstraction, recommender systems oriented, fewer API calls are required to accomplish the same processing pipeline.
- **Interoperability** with PyTorch, TensorFlow, and HugeCTR.

NVTabular



ETL – Extract, Transform, Load

NVIDIA Merlin

Training (1 of 2)

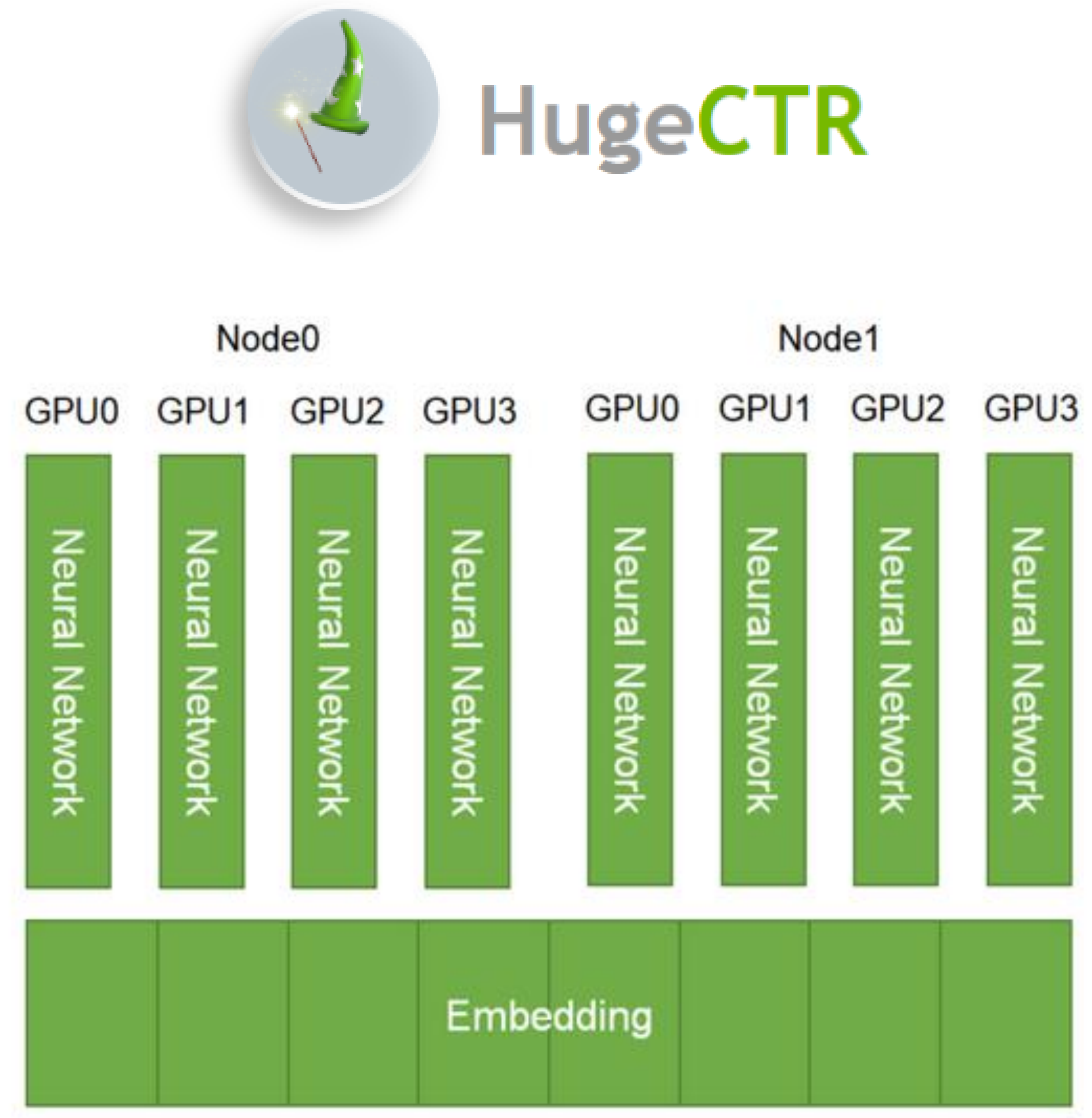
HugeCTR

What it is:

Highly efficient C++ GPU framework and reference design dedicated for recommendation workload training.

What it's capable of:

- **Model and Data parallelism.**
- **Scale embedding** across multiple GPUs and multiple nodes.
- **Designed for distributed training** with model-parallel embedding tables and data-parallel neural networks.
- Supports **a range of model architectures:**
    - DCN,
    - DeepFM,
    - DLRM,
    - W&D.



NVIDIA Merlin

Training (2 of 2)

Reference Implementations

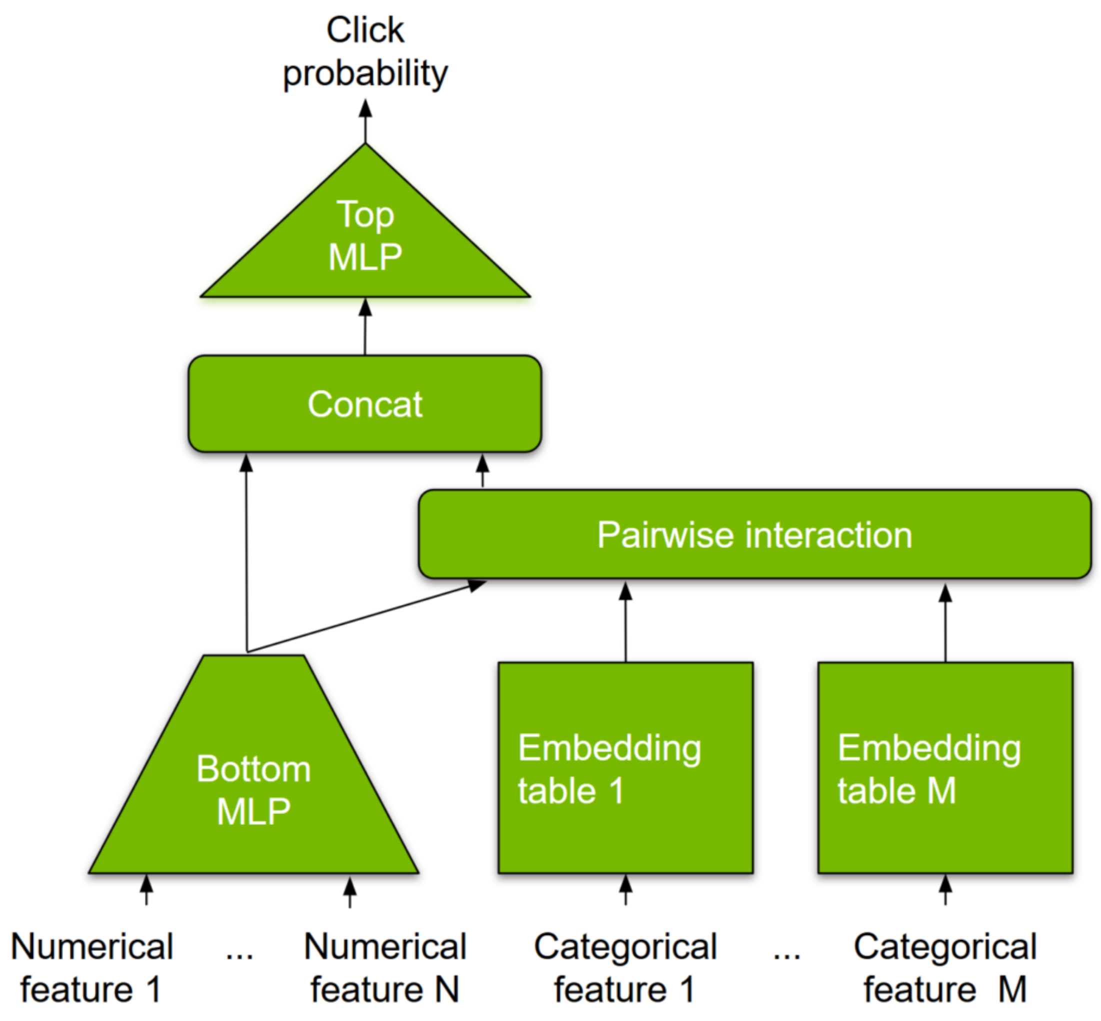
What it is:

Open source reference implementations for popular DL recommender models in TensorFlow and PyTorch.

What it's capable of:

- **State-of-the-art accuracy** on public datasets.
- Up to **67x acceleration** compared to CPU implementation.
- Supports **a range of model architectures:**
  - DLRM (PyTorch),
  - NCF (TensorFlow, PyTorch),
  - VAE-CF (TensorFlow),
  - W&D (TensorFlow).

Example: DLRM architecture



NVIDIA Merlin

Inference

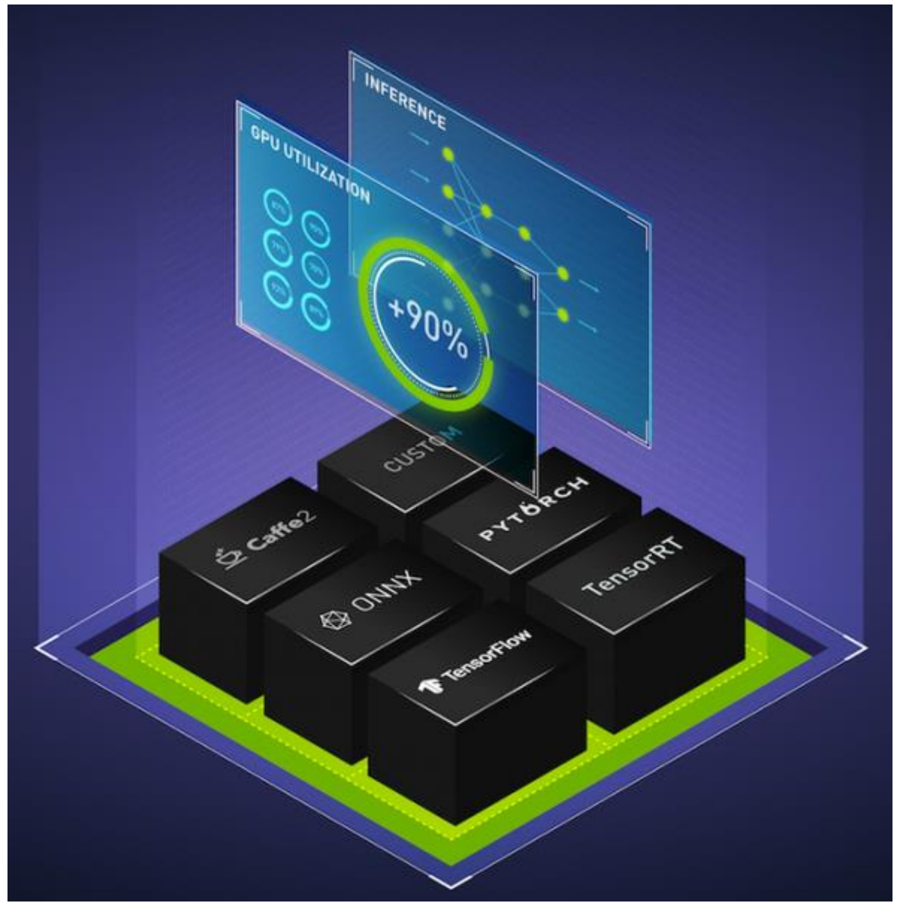
TensorRT and Triton

What it is:

TensorRT is an SDK for high performance DL inference.
Triton Server is a GPU-optimized inferencing solution.

What it's capable of:

- Maximizes GPU utilization.
- Maximizes throughput at the desired latency.
- For instance, W&D TensorRT Inference pipeline, compared to an equivalent CPU solution, provides:
    - Up to 18x reduction in latency,
    - Up to 17.6x improvement in throughput.



NVIDIA Merlin

Components Summary

Data (ETL)

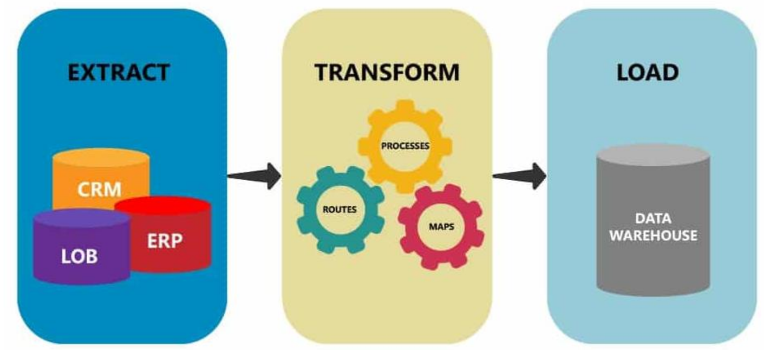
NVTabular

A feature engineering and preprocessing library designed to quickly and easily manipulate petabytes of tabular data.

Scales Easily: No limit on dataset size (not bound by CPU or GPU memory).

GPU-accelerated, eliminating CPU bottlenecks.

Dataloader acceleration and interoperable with PyTorch, Tensorflow, and HugeCTR.



ETL - Extract, Transform, Load

Training

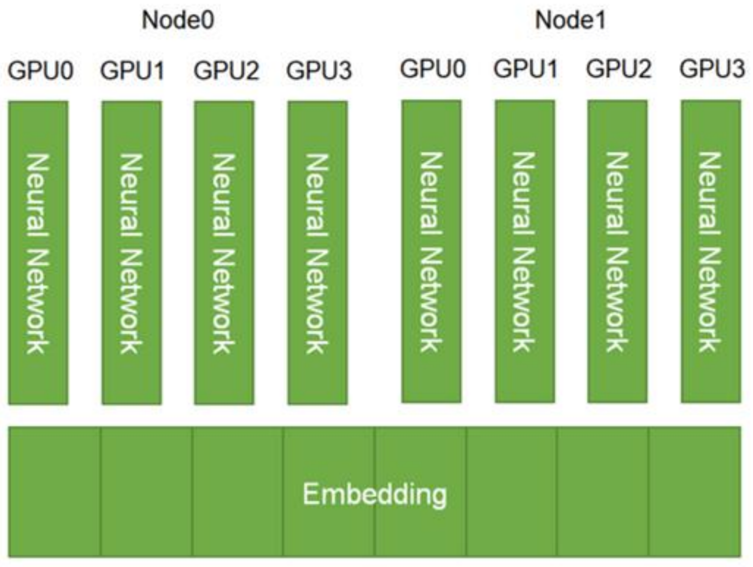
HugeCTR

A highly efficient C++ GPU framework and reference design dedicated for recommendation workload training.

Supports multiple model architectures:

- DCN
- DeepFM
- DLRM
- W&D

Designed for distributed training.



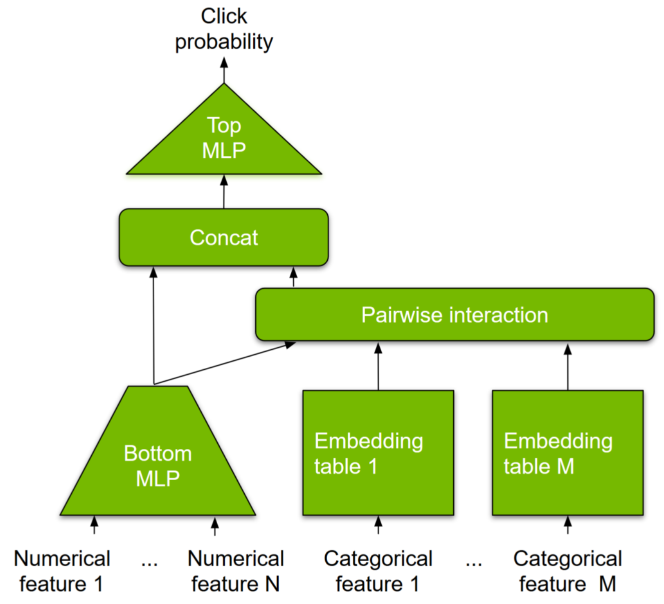
Reference Implementations

Get started with open source reference implementations and achieve state-of-the-art accuracy on public datasets.

Supports multiple model architectures:

- DLRM (PyTorch)
- NCF (TensorFlow, PyTorch)
- VAE-CF (TensorFlow)
- W&D (TensorFlow)

Up to 67x acceleration, compared to CPU-based implementations.



Inference

TensorRT and Triton

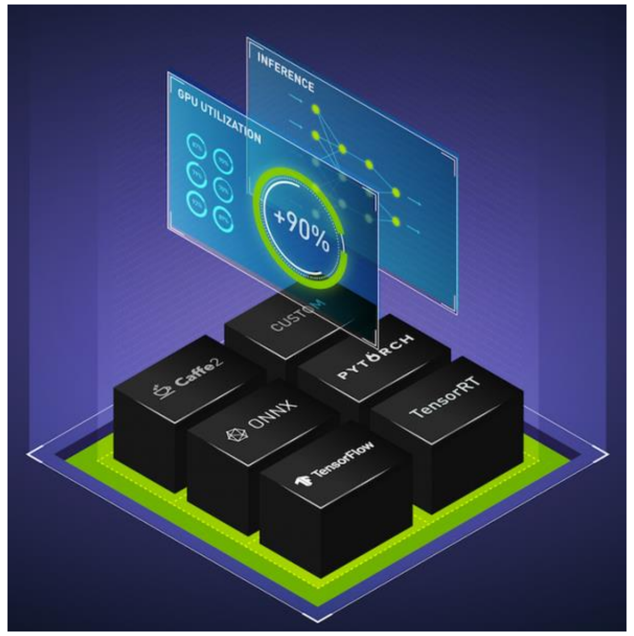
Take advantage of TensorRT and Triton to run inference efficiently on GPUs by maximizing throughput with the right combination of latency and GPU utilization.

Compare 10x the number of candidates at the same SLA cost for less.

Acceleration example for Wide & Deep:

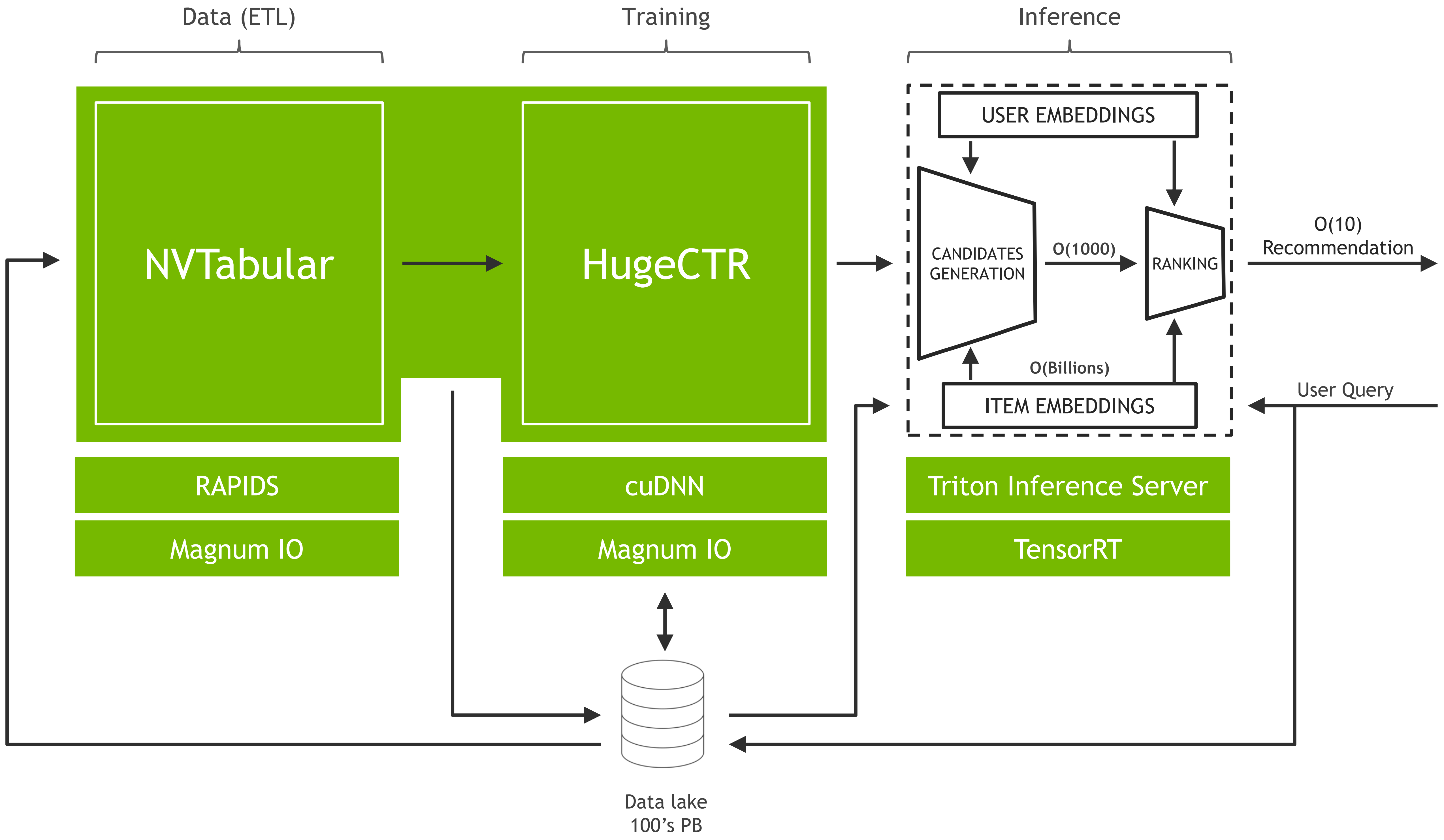
- 18x reduction in latency,
- 17x improvement in throughput,

compared to an equivalent CPU solution.



NVIDIA Merlin

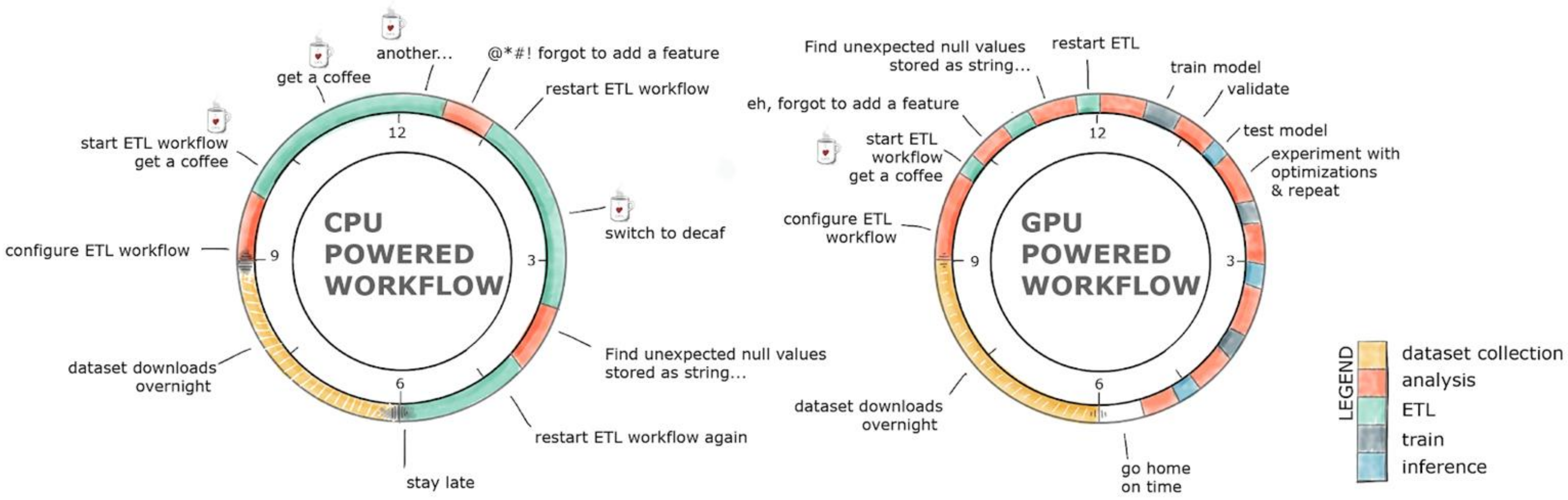
Deep Recommender Application Framework



NVTabular

GPU-Accelerated ETL

The average data scientist spends up to 80% of their time in ETL, as opposed to training models



Built on top of RAPIDS

CPU version

ETL

Model Training

Viz

Dask

Pandas
Analytics

Scikit-Learn
Machine Learning

NetworkX
Graph Analytics

Pytorch, MxNext, TensorFlow
Deep Learning

Matplotlib, Plotly
Visualization

CPU Memory

GPU-Accelerated

ETL

Model Training

Viz

Dask

cuDF
Analytics

cuML
Machine Learning

cuGraph
Graph Analytics

Pytorch, MxNext, TensorFlow
Deep Learning

pyViz, Plotly, cuXfilter
Visualization

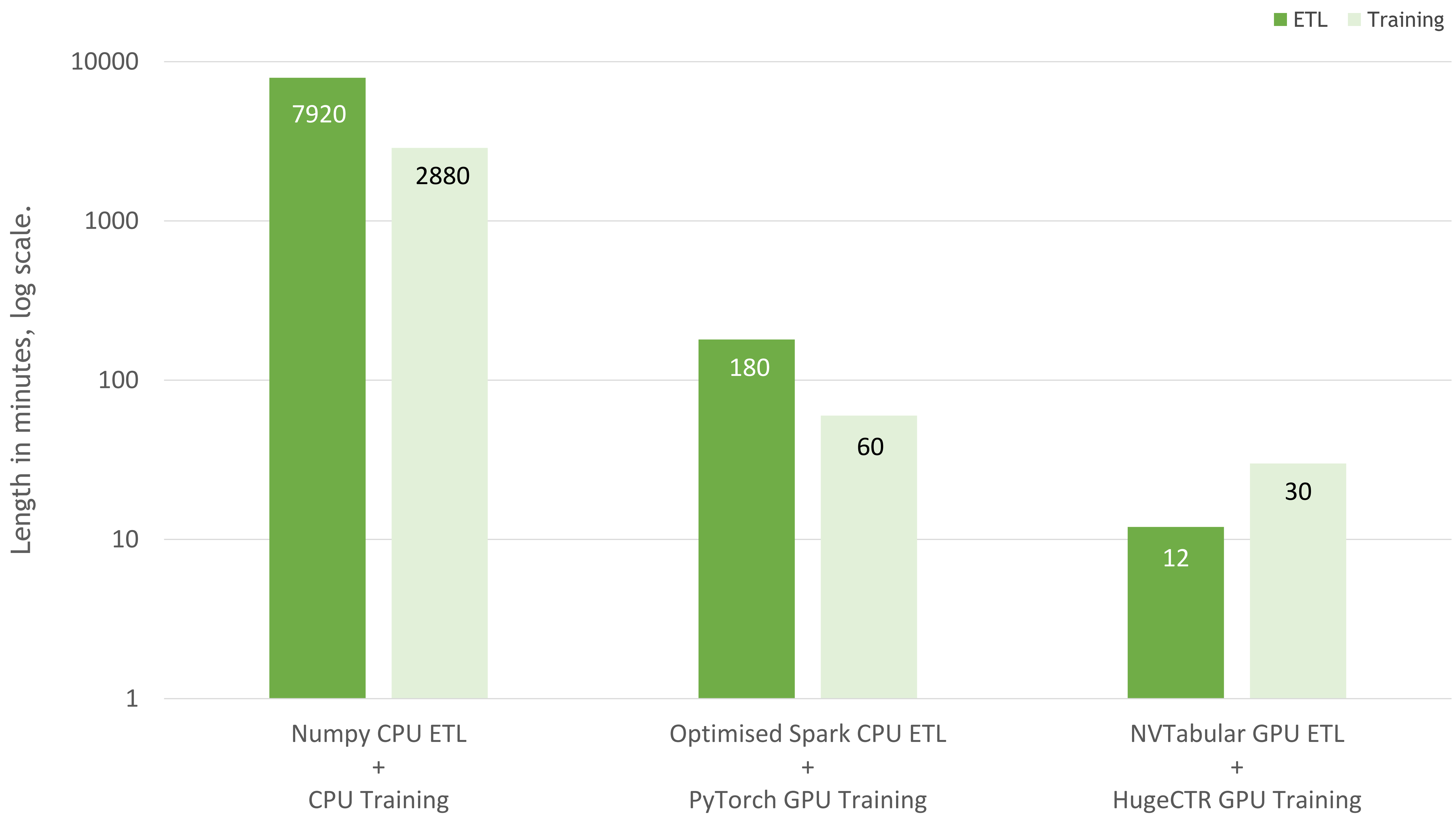
Apache Arrow

GPU Memory

www.rapids.ai

Case Study: 1TB Ads Dataset

ETL 660x faster. Training 96x faster.



https://github.com/NVIDIA/NVTabular/blob/master/examples/criteo-example.ipynb

CPU: AWS r5d.24xl, 96 cores, 768 GB RAM
GPU: 1 x NVIDIA V100 32GB

NVTabular Key Features

Faster and Easier GPU-based ETL

- Focused on recommendation use cases. It requires fewer API calls to accomplish the same tasks.
- GPU-accelerated, eliminating CPU bottlenecks.
- Out-of-core execution. No GPU memory limits and reduced I/O through lazy execution.
- Pytorch, TensorFlow and HugeCTR compatible.
- Triton Inference Server support.

	NVTabular	pandas
Dataset size limitation	Unlimited	CPU Memory
Code complexity	Simple	Moderate
Lines of code	10 - 20	100 - 1000
Flexibility	Domain specific	General
Data loading Transforms	Yes	No
Inference Transforms	Yes	No

NVTabular vs Pandas code

100x fewer lines of code required

```
import glob
import nvtabular as nvt

# Create datasets from input files
train_files = glob.glob("./dataset/train/*.parquet")
valid_files = glob.glob("./dataset/valid/*.parquet")

train_ds = nvt.Dataset(train_files, gpu_memory_frac=0.1)
valid_ds = nvt.Dataset(valid_files, gpu_memory_frac=0.1)
```

Import libraries.

Create training and validation datasets.

```
# Initialise workflow
cat_names = ["C" + str(x) for x in range(1, 27)]   # Specify categorical feature names
cont_names = ["I" + str(x) for x in range(1, 14)]  # Specify continuous feature names
label_name = ["label"] # Specify target feature

proc = nvt.Workflow(cat_names=cat_names, cont_names=cont_names, label_name=label_name)
```

Initialise workflow specifying categorical, and continuous data.

```
# Add feature engineering and pre-processing ops to workflow
proc.add_cont_feature([nvt.ops.ZeroFill(), nvt.ops.LogOp()])
proc.add_cont_preprocess(nvt.ops.Normalize())
proc.add_cat_preprocess(nvt.ops.Categorify(use_frequency=True, freq_threshold=15))
```

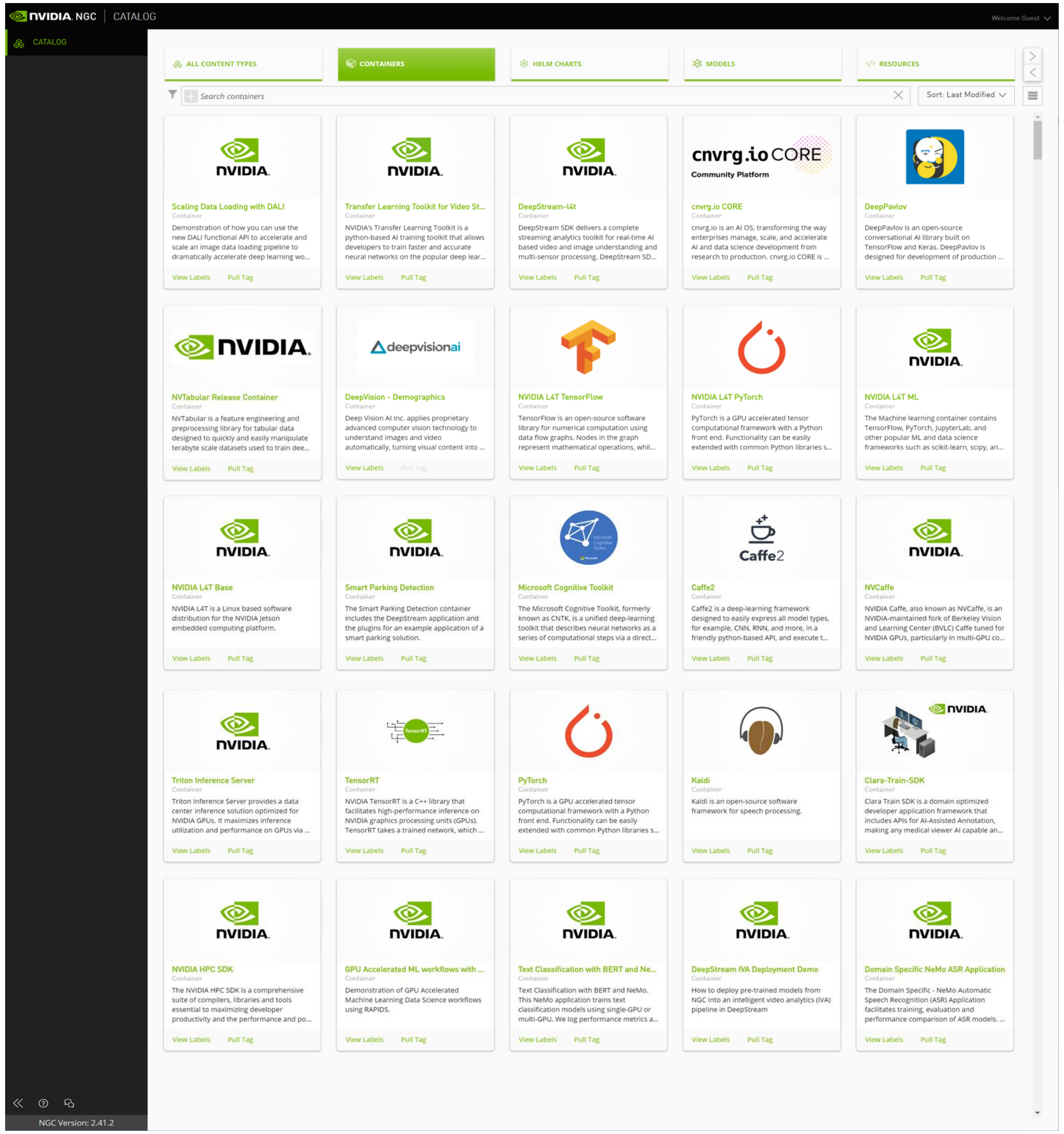
Zero fill any nulls, log transform and normalise continuous variables. Encode categorical data.

```
# Compute statistics, transform data, and export to disk
proc.apply(train_dataset, shuffle=True,  output_path="./processed_data/train", num_out_files=len(train_files))
proc.apply(valid_dataset, shuffle=False, output_path="./processed_data/valid", num_out_files=len(valid_files))
```

Apply the operations, creating new shuffled training and validation datasets.

Equivalent Pandas/Numpy code: https://github.com/facebookresearch/dlrm/blob/master/data_utils.py

Getting Started



https://ngc.nvidia.com

NVIDIA NGC + GitHub

- Pull containers from NVIDIA NGC:
  https://ngc.nvidia.com/catalog/containers/nvidia:merlin:merlin-training
  https://ngc.nvidia.com/catalog/containers/nvidia:merlin:merlin-inference
- Run examples / Jupyter notebooks:
  https://github.com/NVIDIA/NVTabular/tree/master/examples
- Getting started documentation:
  https://github.com/NVIDIA/NVTabular#installation

Or alternatively

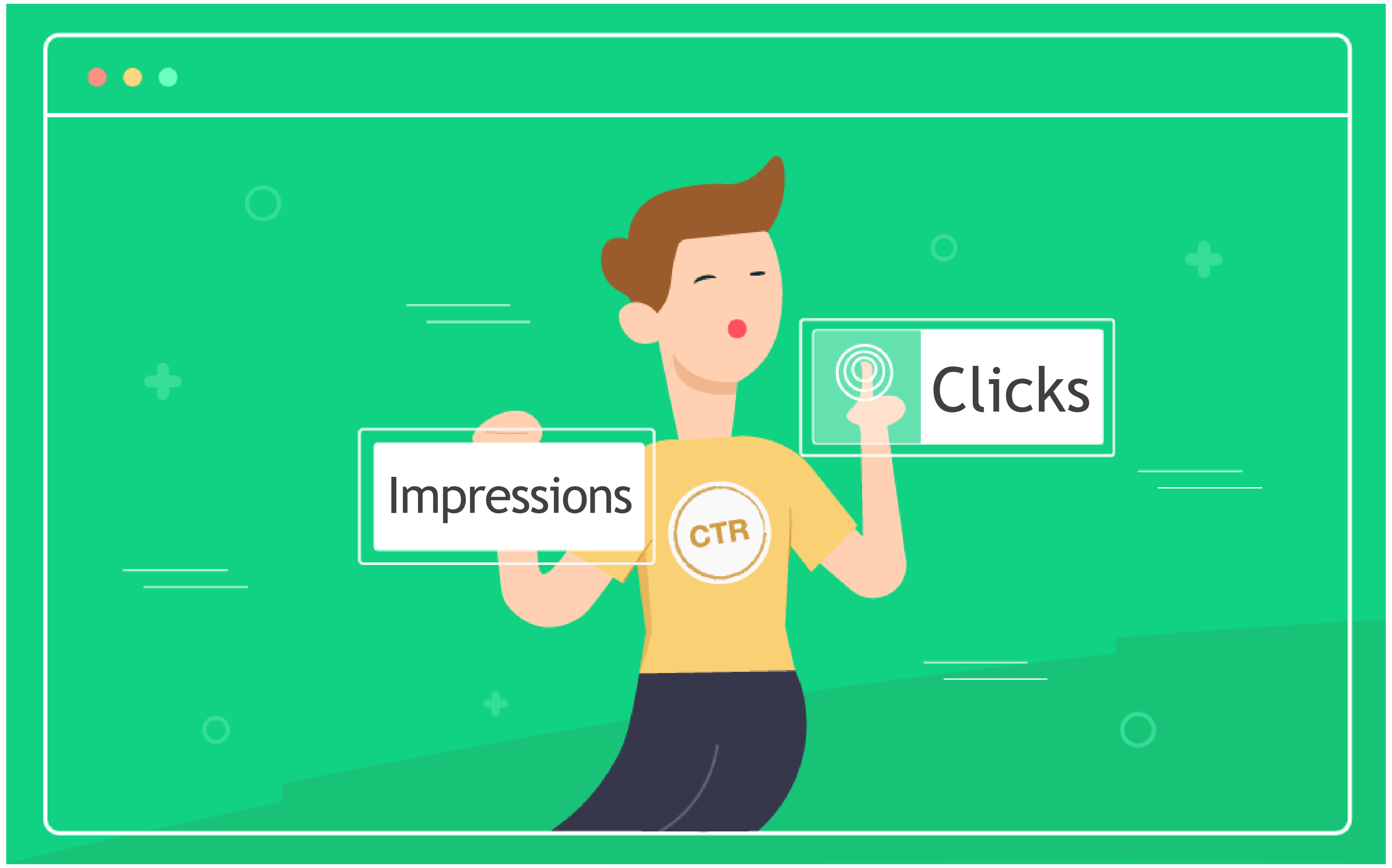
https://anaconda.org/nvidia/nvtabular

https://pypi.org/project/nvtabular

HugeCTR

HugeCTR

A Deep Recommender Training Framework



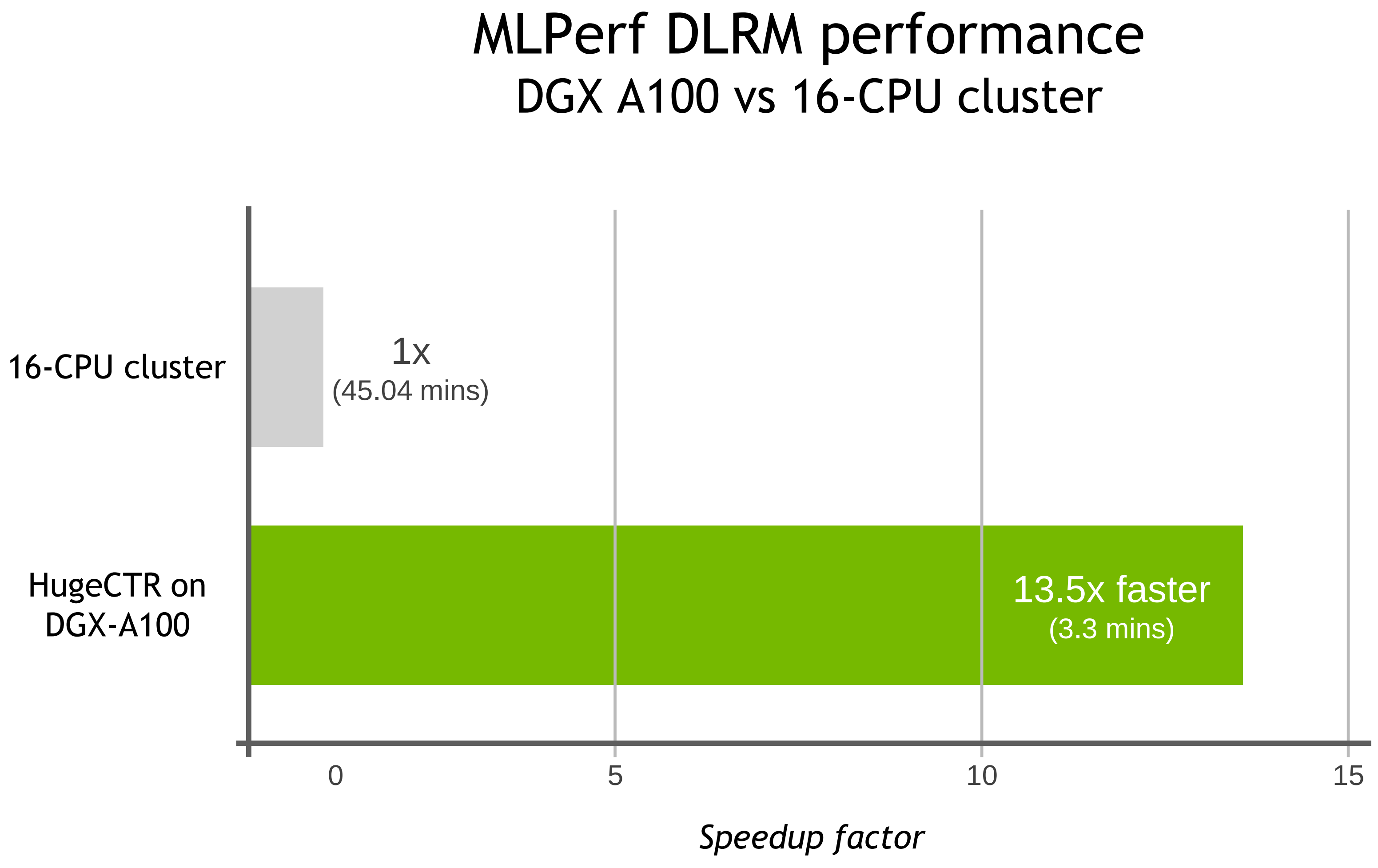
$$CTR = \frac{\text{\# of clicks}}{\text{\# of impressions}}$$

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

- It is fast, very fast:
  - Speedup of up to 114x over TensorFlow on a 40-core CPU node.
  - Speedup of up to 8.3x over TensorFlow on a single NVIDIA V100 GPU.
- Supports GPU accelerated recommender specific operations, i.e. GPU hash table, fused layers, etc.
- Optimized multi-node functionality, syncing GPUs via UCX P2P.
- Easy to use.

Case Study: MLPerf 0.7 Win

The Fastest on MLPerf RecSys Benchmark



https://mlperf.org/training-results-0-7

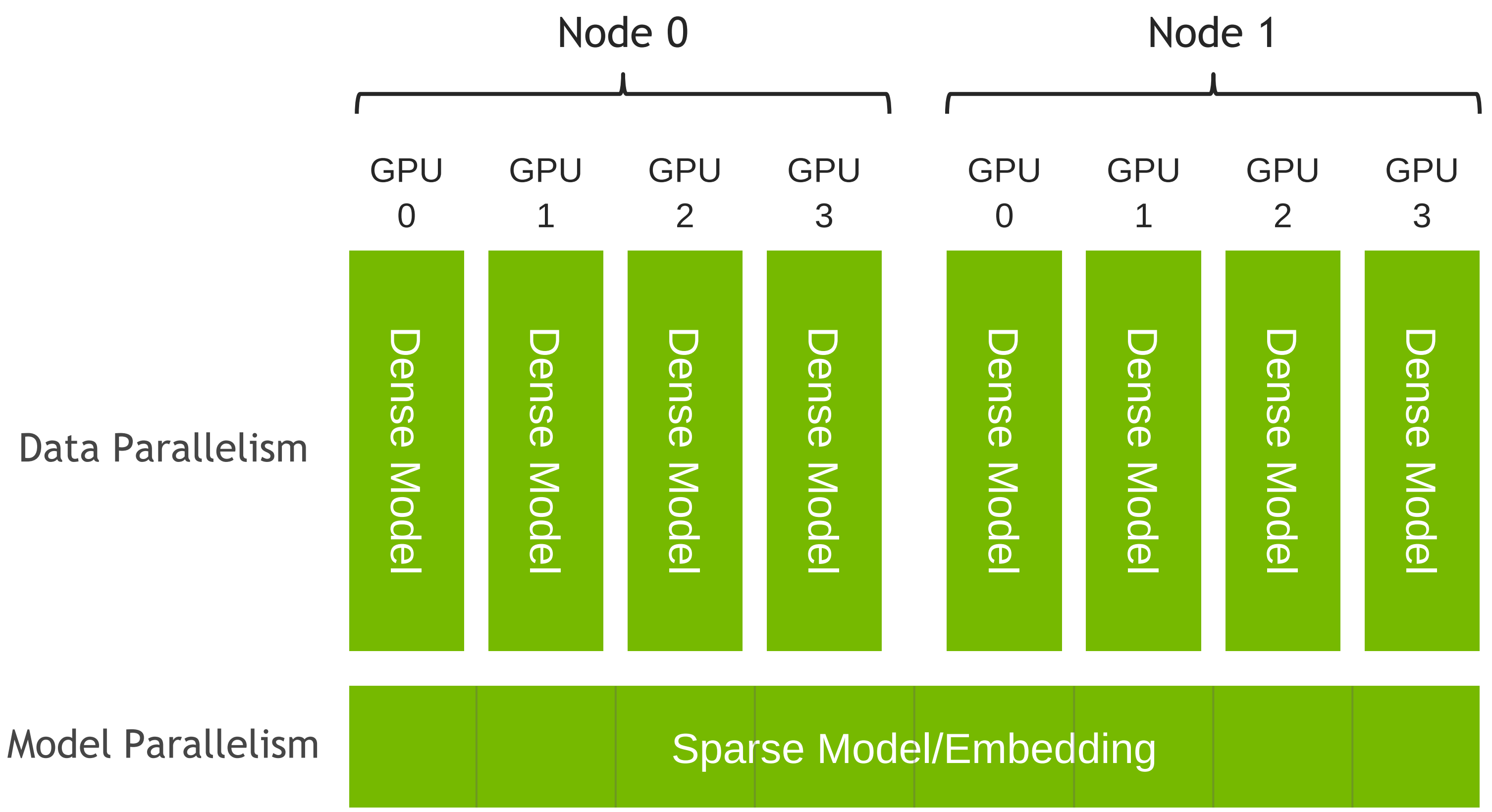
HugeCTR is the key driver behind the recent NVIDIA MLPerf recommender records.

- HugeCTR v2.2 running on DGX-A100 is 13.5x faster than Intel's 16-CPU cluster submission.
- At 3.33 minutes, HugeCTR on DGX-A100 is the fastest commercially available system on the MLPerf recommender benchmark.

Recommender task (training DLRM on the Criteo 1TB dataset). Bars represent speedup factor compared to a 4 CPU-node cluster. The higher the better. HugeCTR v2.2 running on DGX-A100 with 8x A100 40GB GPU. Intel's CPU submission based on 4 nodes, each with 4X 3rd Gen Intel® Xeon® Platinum processor (28core, 2.70GHz, pre-production) with 6 UPI for a total of 16 CPUs.

Highlighted Features

Model Parallel Embedding + Data Parallel Neural Network

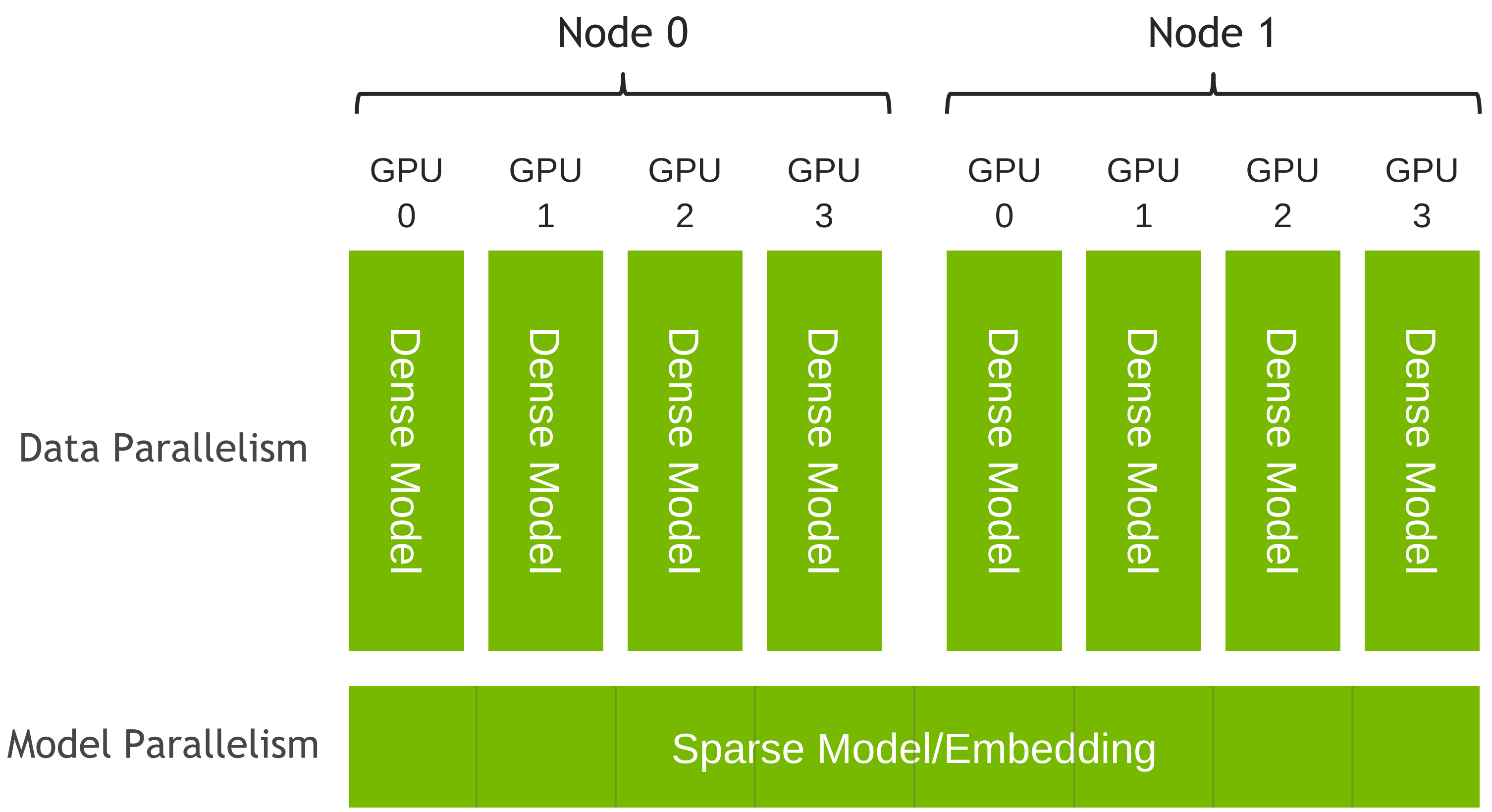


The full network is divided into dense and sparse models.

The large embedding will be shared between multiple GPUs and multiple nodes, so any model with any size can be trained with enough GPUs.

Highlighted Features

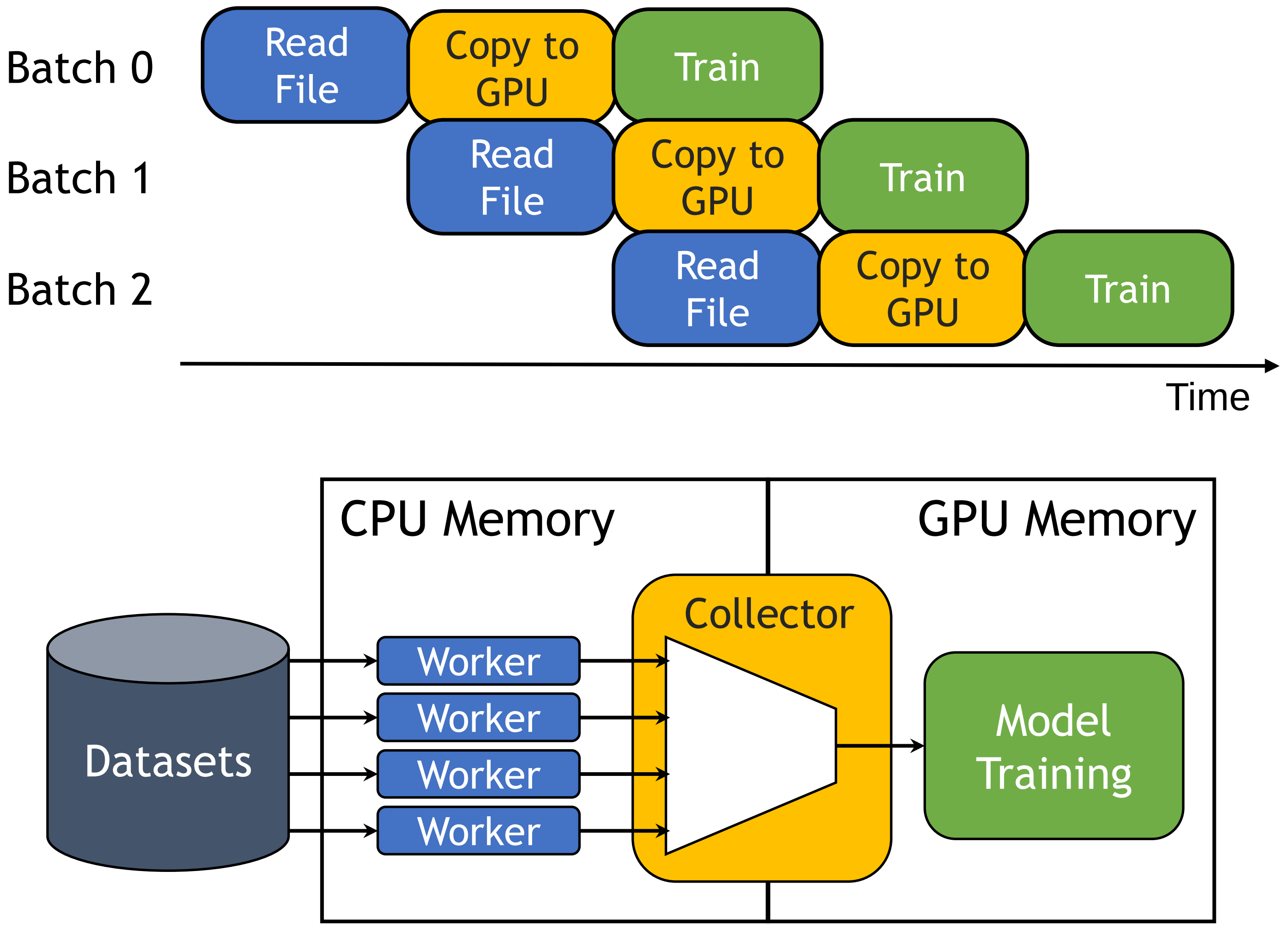
Model Parallel Embedding + Data Parallel Neural Network



The full network is divided into dense and sparse models.

The large embedding will be shared between multiple GPUs and multiple nodes, so any model with any size can be trained with enough GPUs.

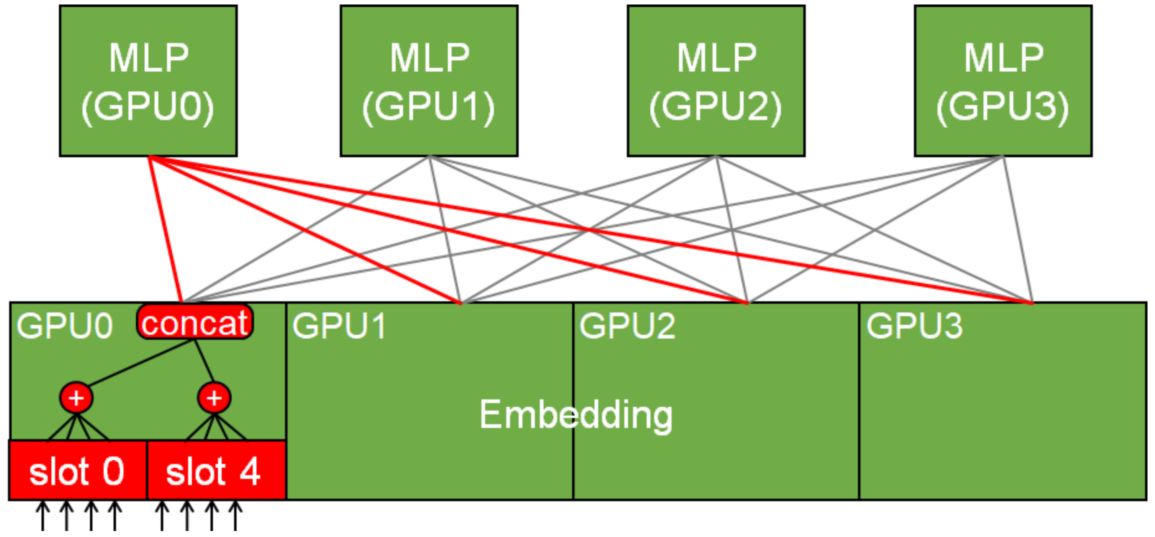
Asynchronous, Multi-threaded Data Pipeline



HugeCTR's asynchronous and multi-threaded file reader reduces data loading bottlenecks, by overlapping disk to CPU memory operations, data transfers from CPU to GPU, and model training.

Highlighted Features

Multi-Slot Embedding Support

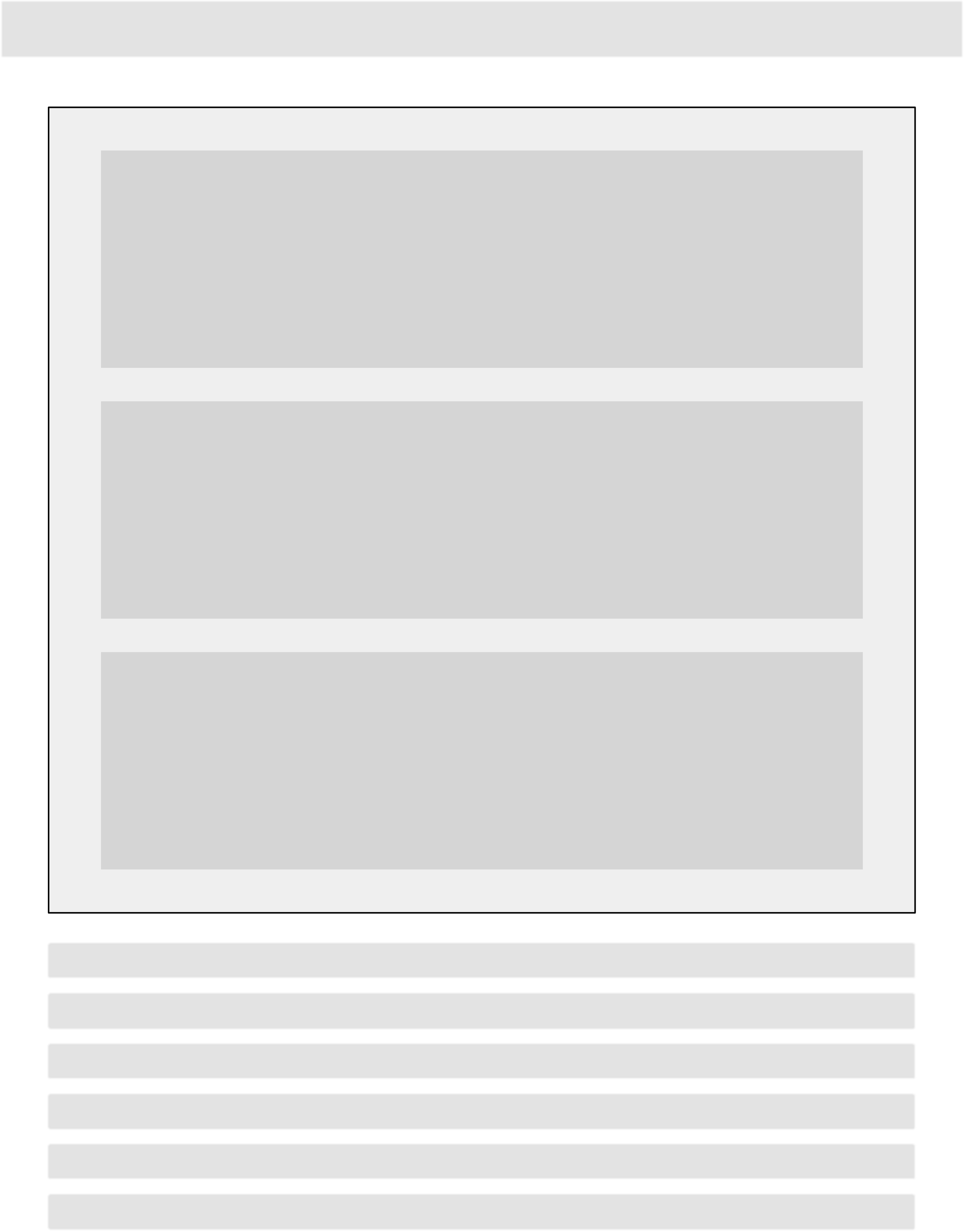


The embedding table can be segmented into multiple slots.

The multi-slot embedding improves the inter-GPU bandwidth utilization in the two ways:

- Helps reduce the number of effective features within each slot to a manageable degree when there exist extremely many features in the dataset.
- The number of transactions across GPUs is reduced, which facilitates more efficient communication.

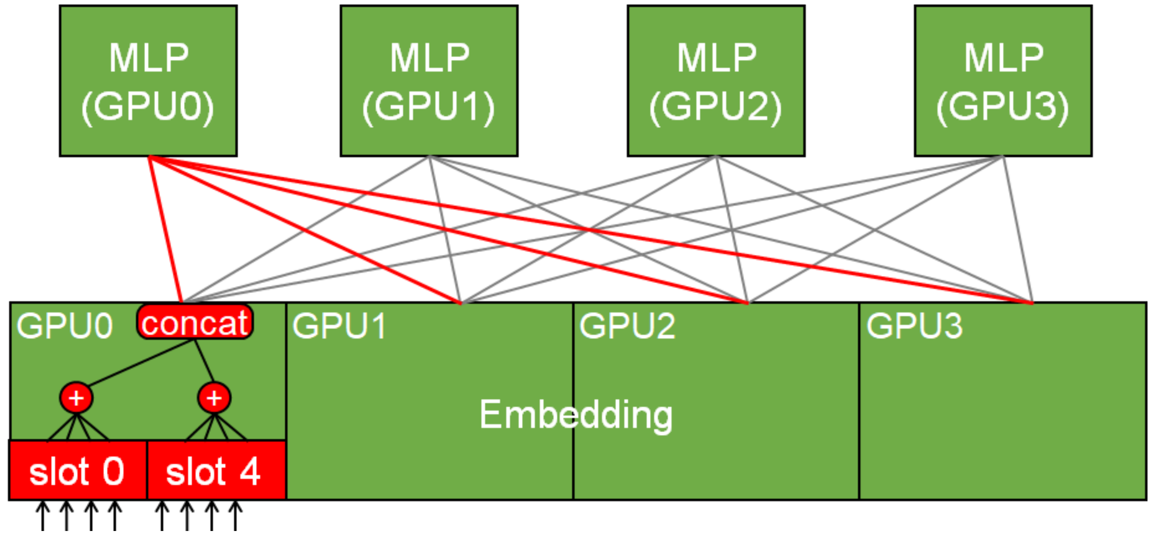
The multi-slot embedding is also useful in expressing a linear model by just setting both the number of slots and the embedding dimension to 1.



Check out Wide & Deep example for the detailed information.

Highlighted Features

Multi-Slot Embedding Support



The embedding table can be segmented into multiple slots.

The multi-slot embedding improves the inter-GPU bandwidth utilization in the two ways:

- Helps reduce the number of effective features within each slot to a manageable degree when there exist extremely many features in the dataset.
- The number of transactions across GPUs is reduced, which facilitates more efficient communication.

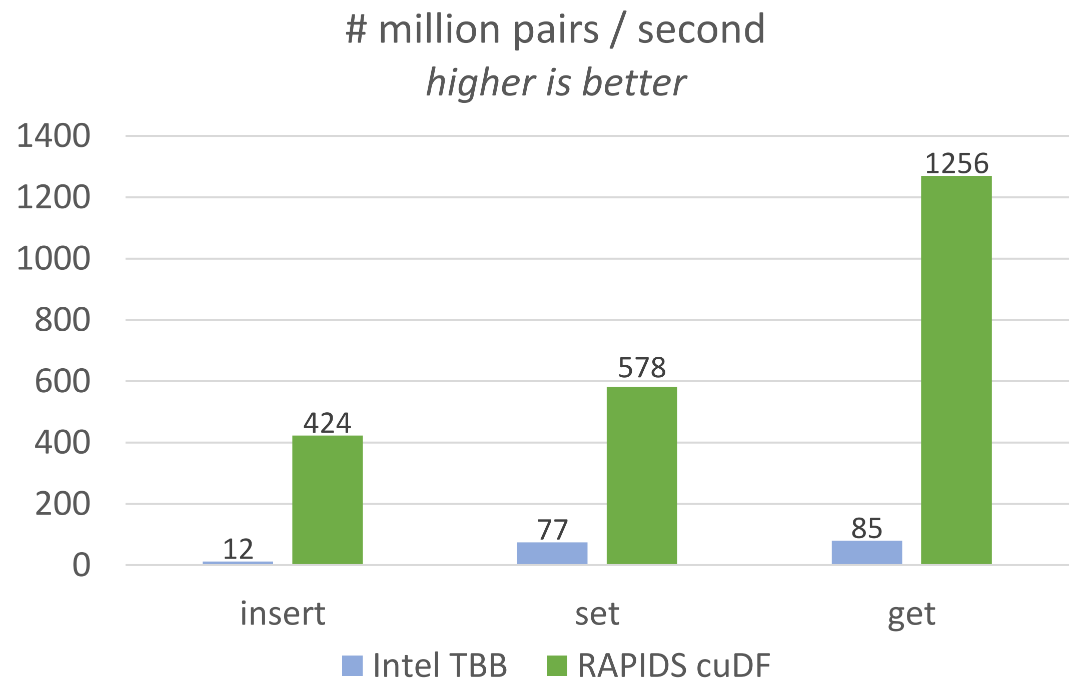
The multi-slot embedding is also useful in expressing a linear model by just setting both the number of slots and the embedding dimension to 1.

Check out Wide & Deep example for the detailed information.

Hash table Support for Embeddings

Custom embedding layer which includes a high performant GPU hash table based on RAPIDS cuDF.

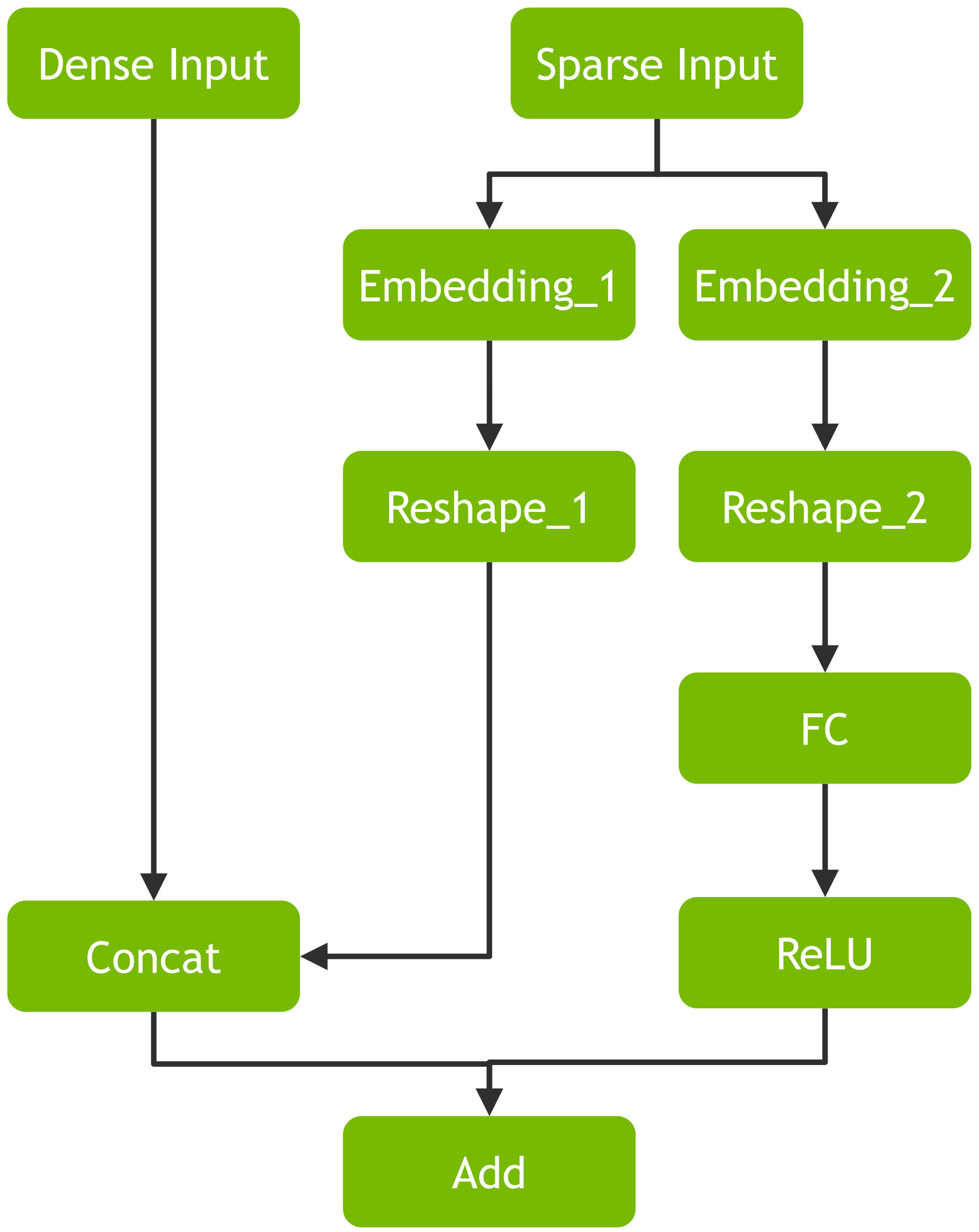
- Supports dynamic insertion.
- Sorted based parameter update to reduce memory footprint.
- Fused reduction for multiple feature fields (slots).
- Up to 35x speedup over *concurrent_hash_map* from Intel's Threading Building Blocks (TBB).



1M key-value pairs; load factor: 0.8
CPU: tbb::concurrent_hash_map on Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
GPU: cudf on V100

Highlighted Features

Model definition in Keras-like API or JSON



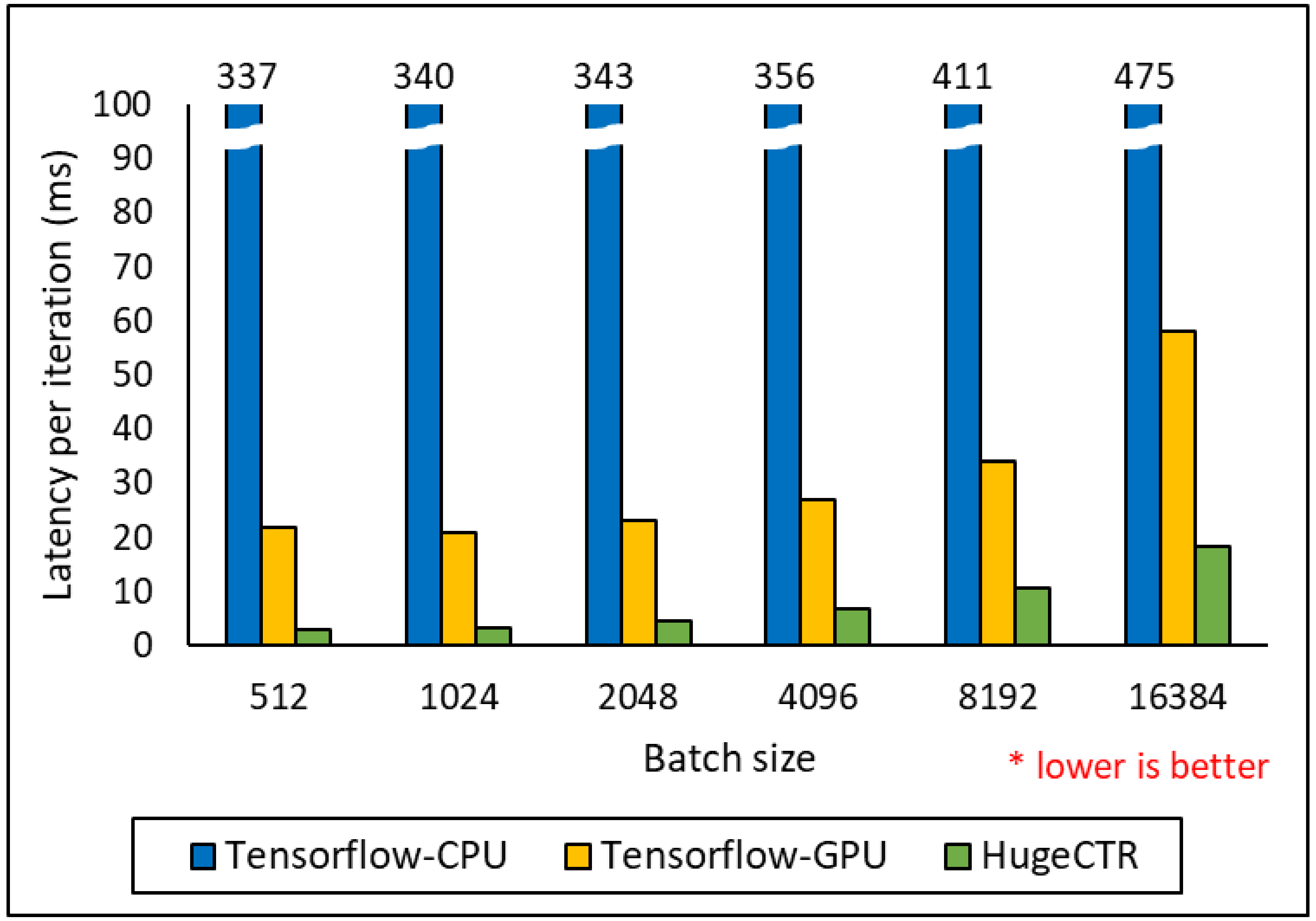

```
"layers": [
    ...
    {
        "name": "embedding_1",
        "type": "DistributedSlotSparseEmbeddingHash",
        "bottom": "wide_sparse_input",
        ...
    },
    {
        "name": "embedding_2",
        "type": "DistributedSlotSparseEmbeddingHash",
        "bottom": "deep_sparse_input",
        ...
    },
    {
        "name": "concat",
        "type": "Concat",
        "bottom": ["dense_input", "reshape_1"],
        ...
    },
    ...
]
```

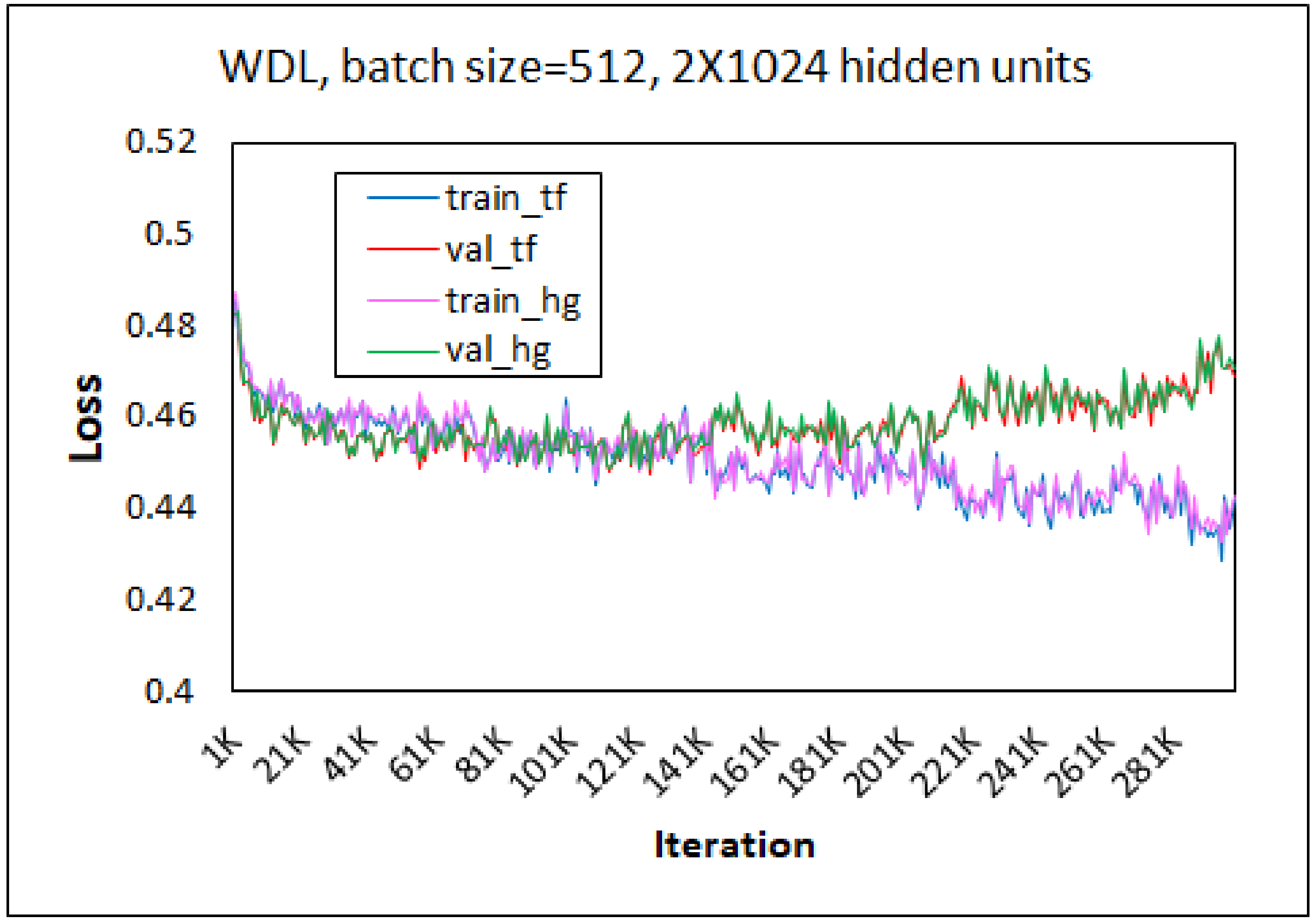
Network configuration documentation

https://github.com/NVIDIA/HugeCTR/tree/master/samples

HugeCTR vs TensorFlow

Wide & Deep Network (WDL)



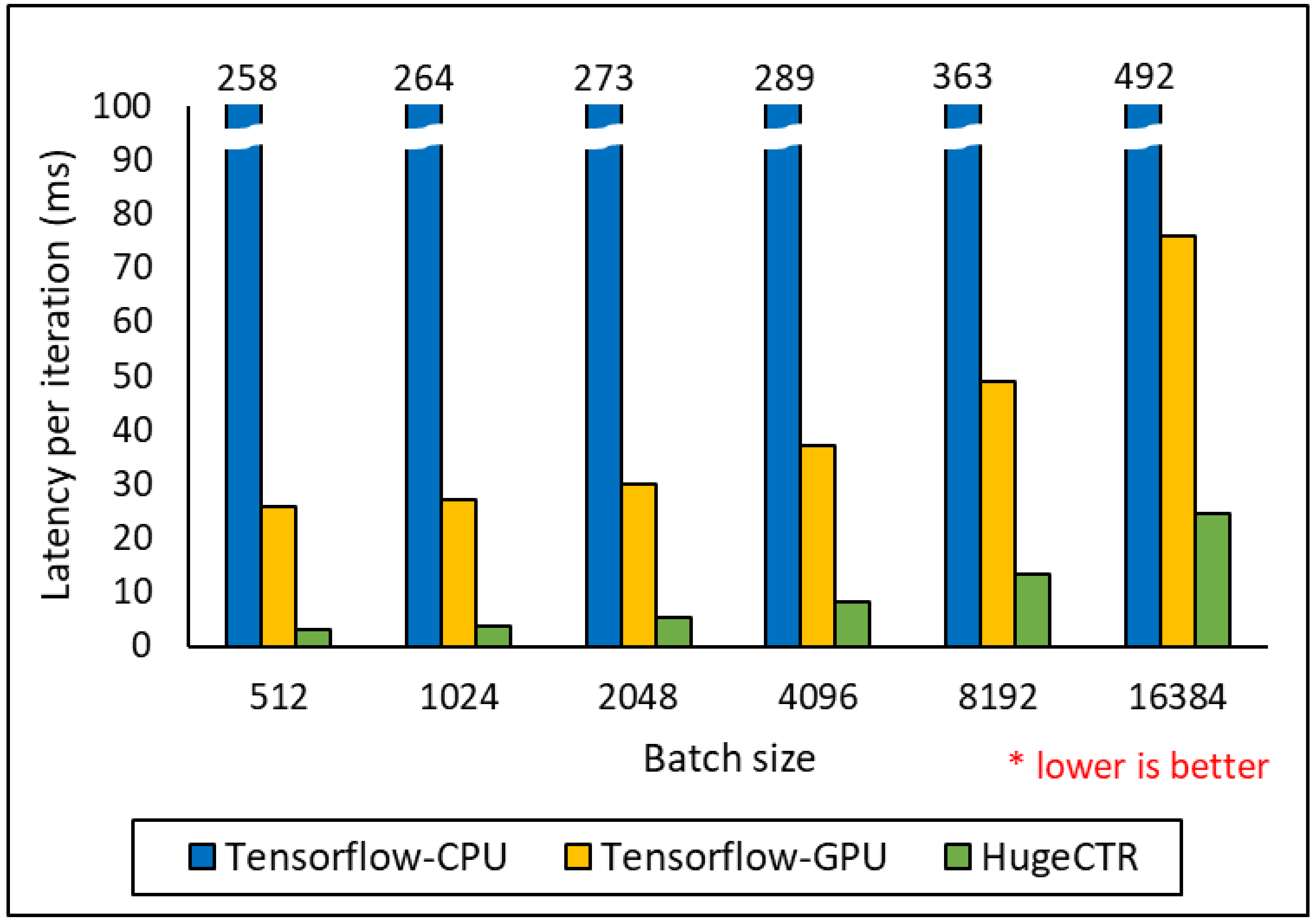


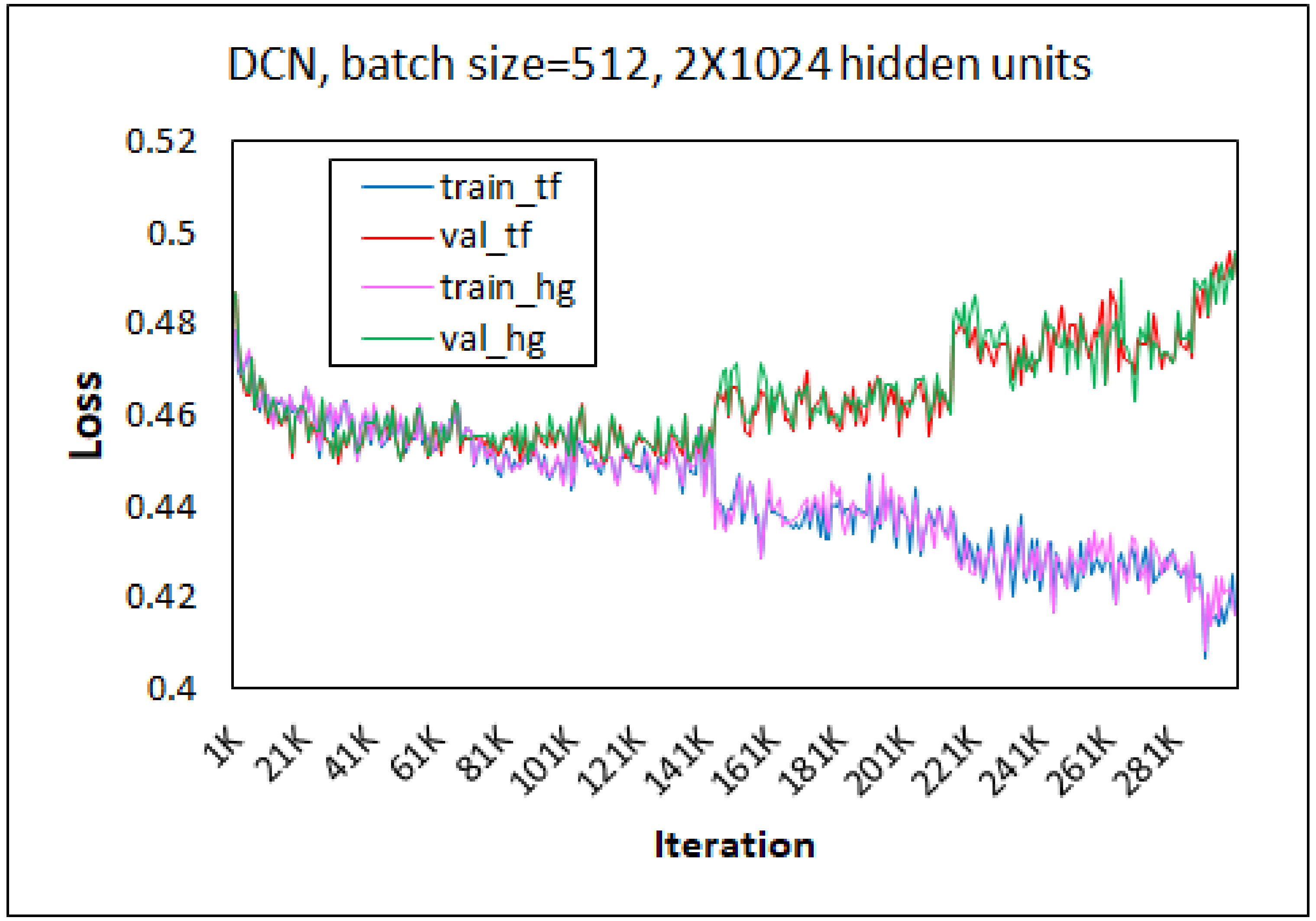
https://github.com/NVIDIA/HugeCTR/tree/master/samples/wdl

GPU: NVIDA Tesla V100 16GB
CPU: Dual 20-core Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
Dataset: Criteo (26 categorical features and 13 integer features)
Model: 2x 1024-unit FC layers with ReLU and dropout, emb_dim: 16
Optimizer: Adam for both Linear and DNN models

HugeCTR vs TensorFlow

Deep Cross Network (DCN)

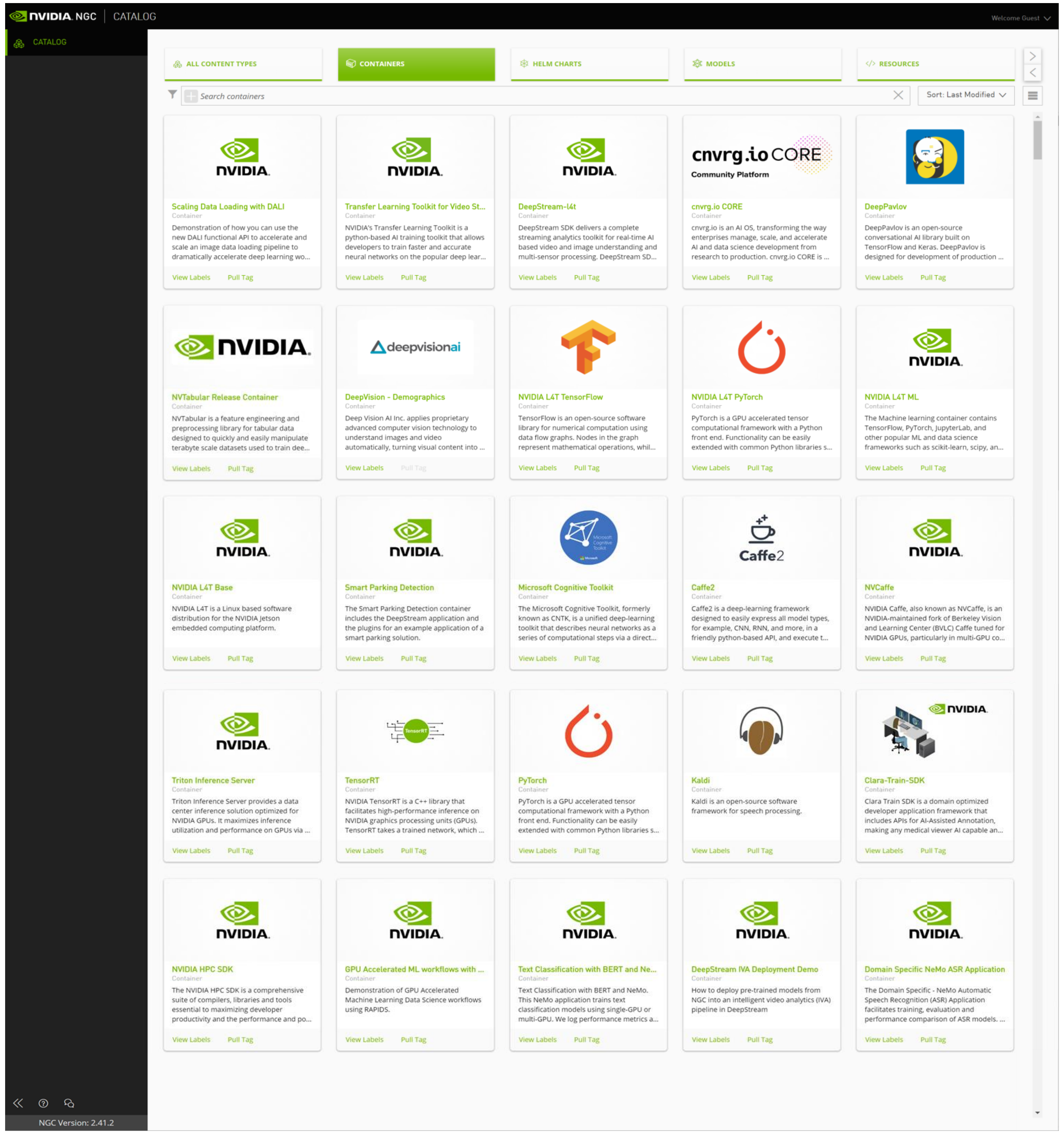




https://github.com/NVIDIA/HugeCTR/tree/master/samples/dcn

GPU: NVIDA Tesla V100 16GB
CPU: Dual 20-core Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
Dataset: Criteo (26 categorical features and 13 integer features)
Model: 2x 1024-unit FC layers with ReLU and dropout, emb_dim: 16
Optimizer: Adam for both Linear and DNN models

Getting Started



https://ngc.nvidia.com

NVIDIA NGC + GitHub

- Pull containers from NVIDIA NGC:

  https://ngc.nvidia.com/catalog/containers/nvidia:merlin:merlin-training

  https://ngc.nvidia.com/catalog/containers/nvidia:merlin:merlin-inference

- Run examples / Jupyter notebooks:

  https://github.com/NVIDIA/HugeCTR/tree/master/notebooks

  https://github.com/NVIDIA/HugeCTR/tree/master/samples

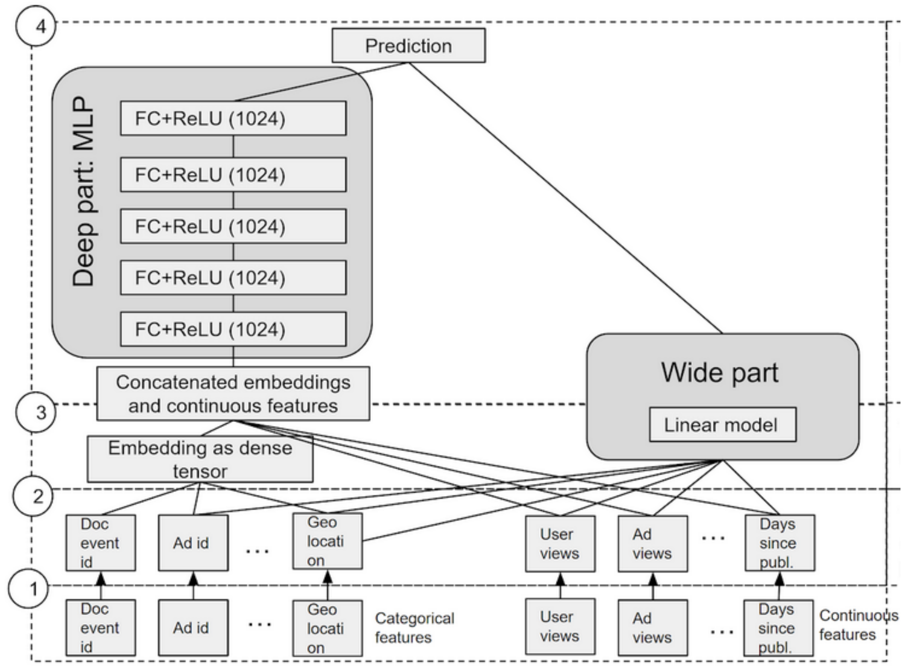
- Getting started documentation:

  https://github.com/NVIDIA/HugeCTR#getting-started

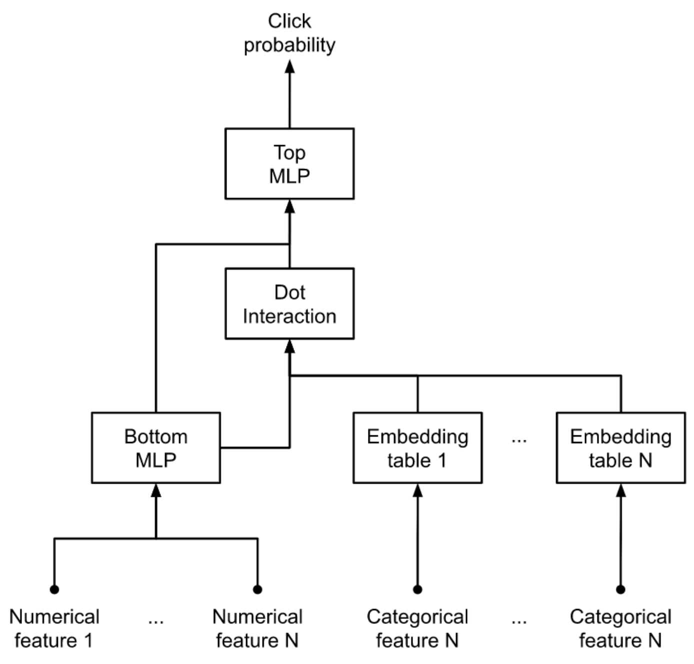
Reference Implementations

Deep Learning based Recommender Examples

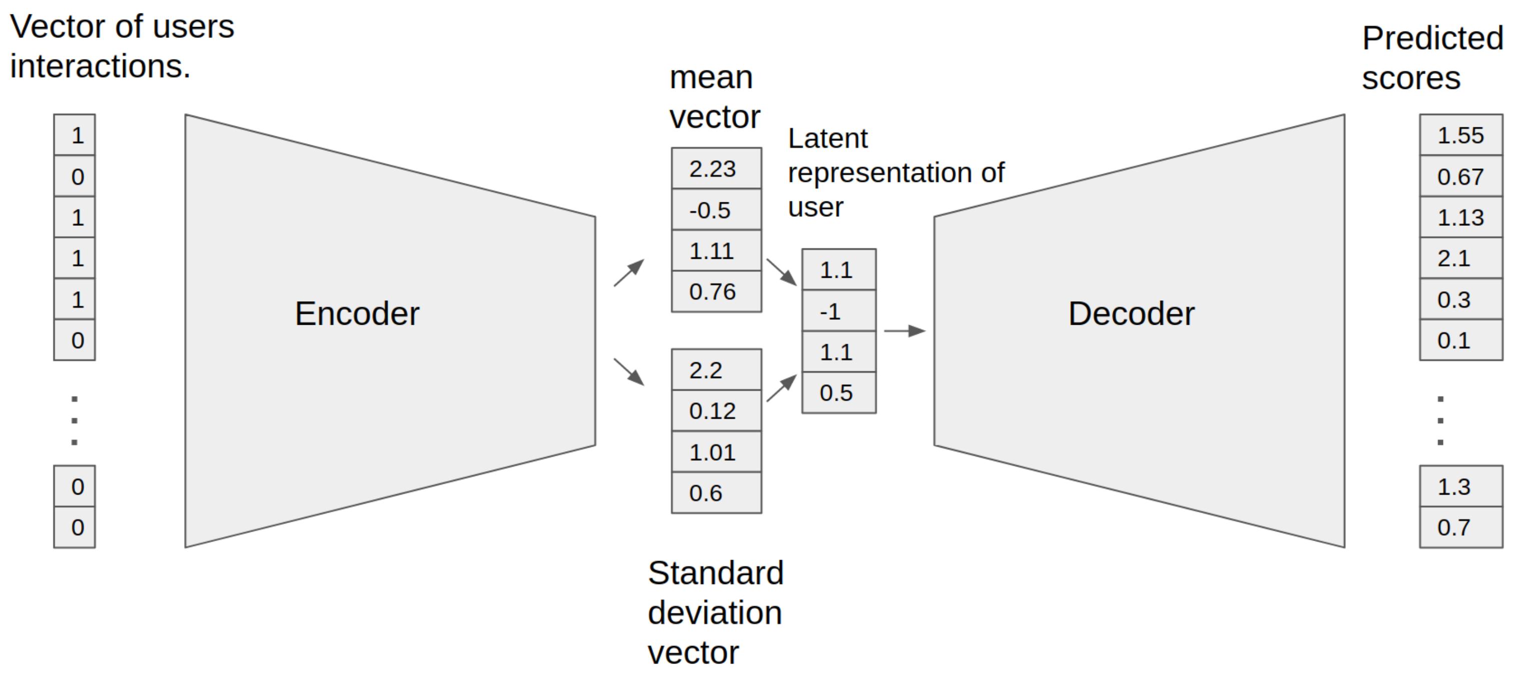
Improving state-of-the-art models



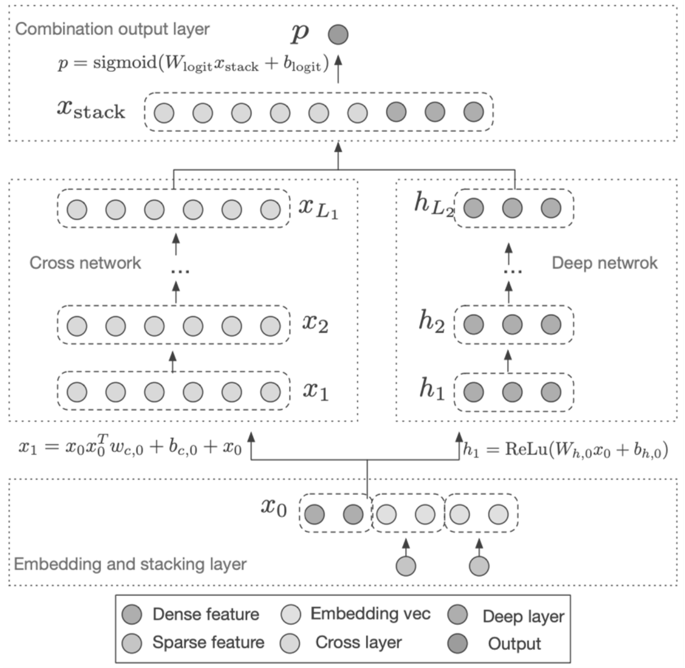
W&D



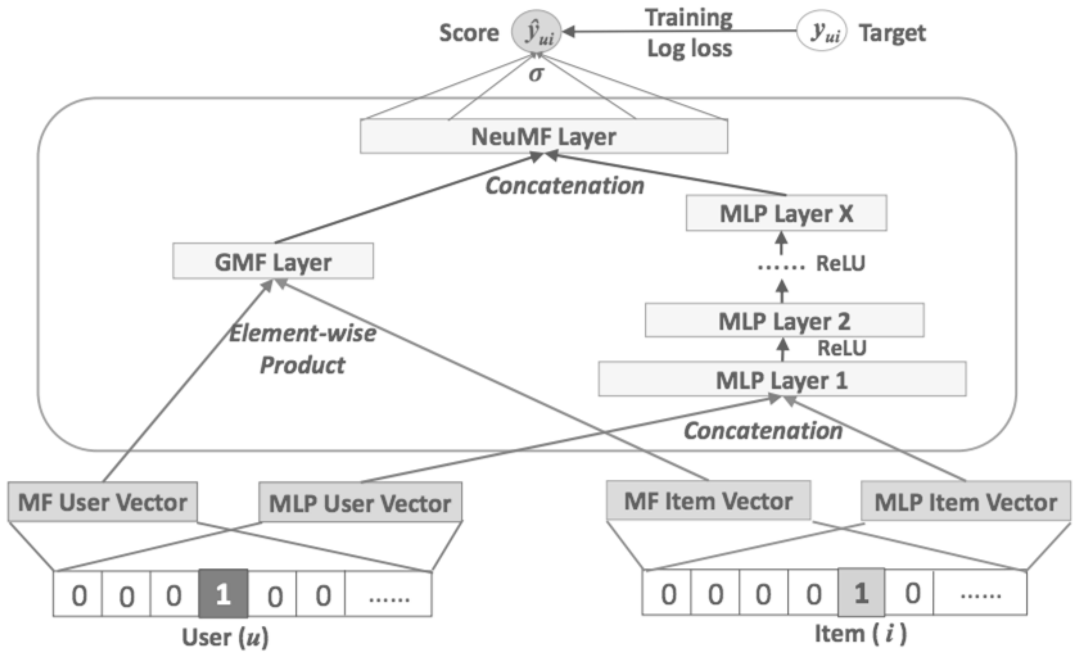
DLRM



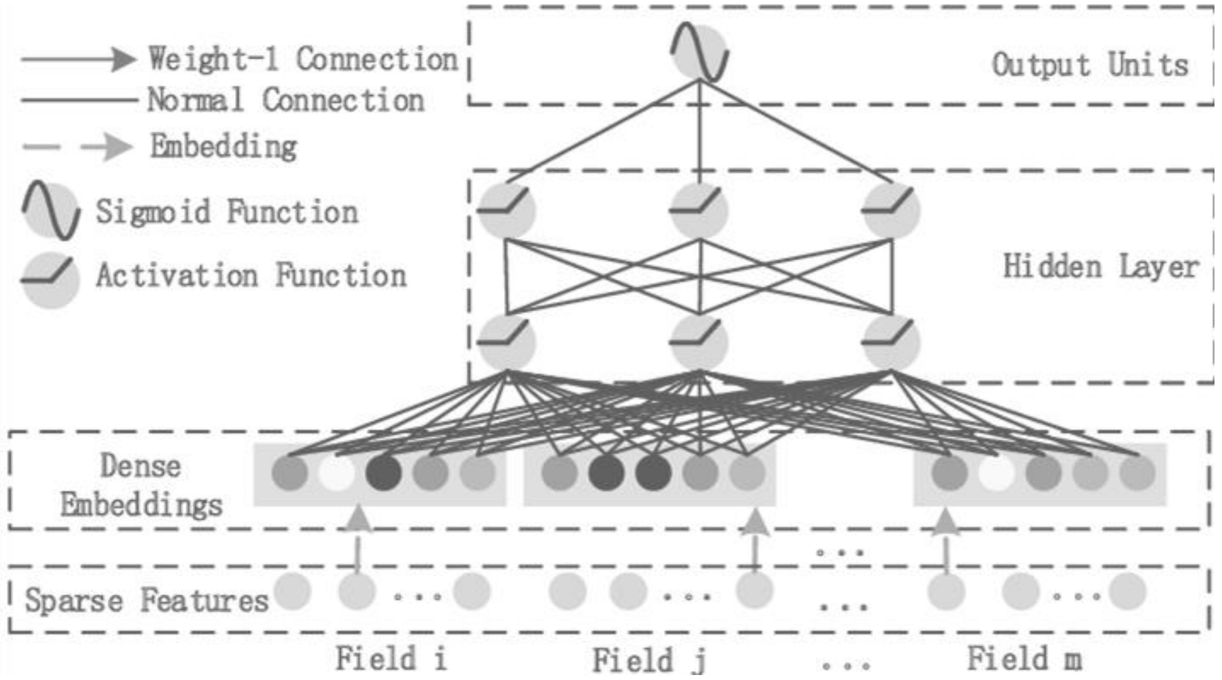
VAE-CF



DCN



NCF



DeepFM

Example: Wide & Deep

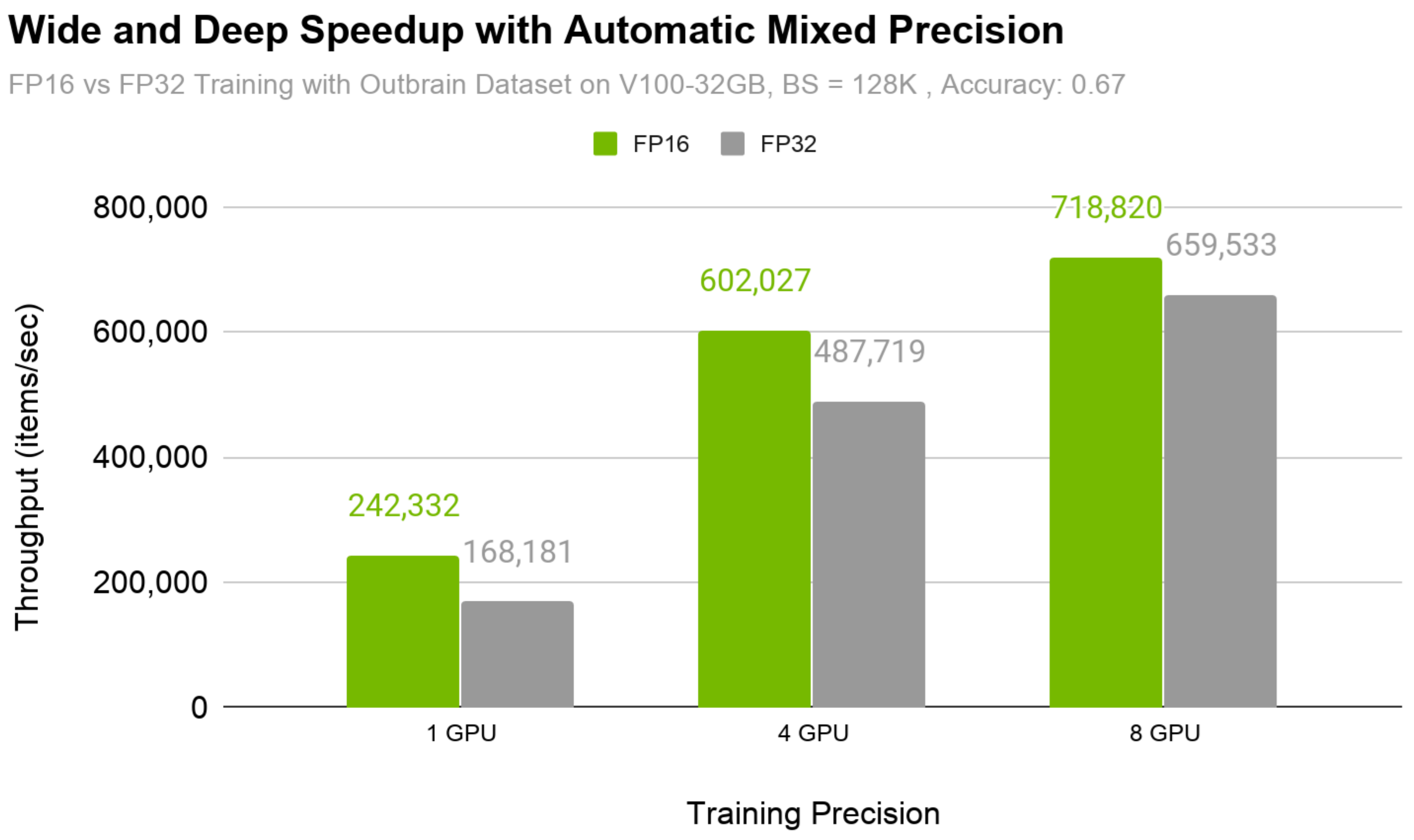
Training with TensorFlow

Model Description:

- Combines the memorization of the Wide part and generalization of the Deep part of the network.
- AMP and Horovod Multi-GPU
- Click Through Rate Prediction

What's New:

- The original model had 3 layers of 1024, 512, and 256 neurons.
- Our model consists of 5 layers each of 1024 neurons.



Example: DLRM

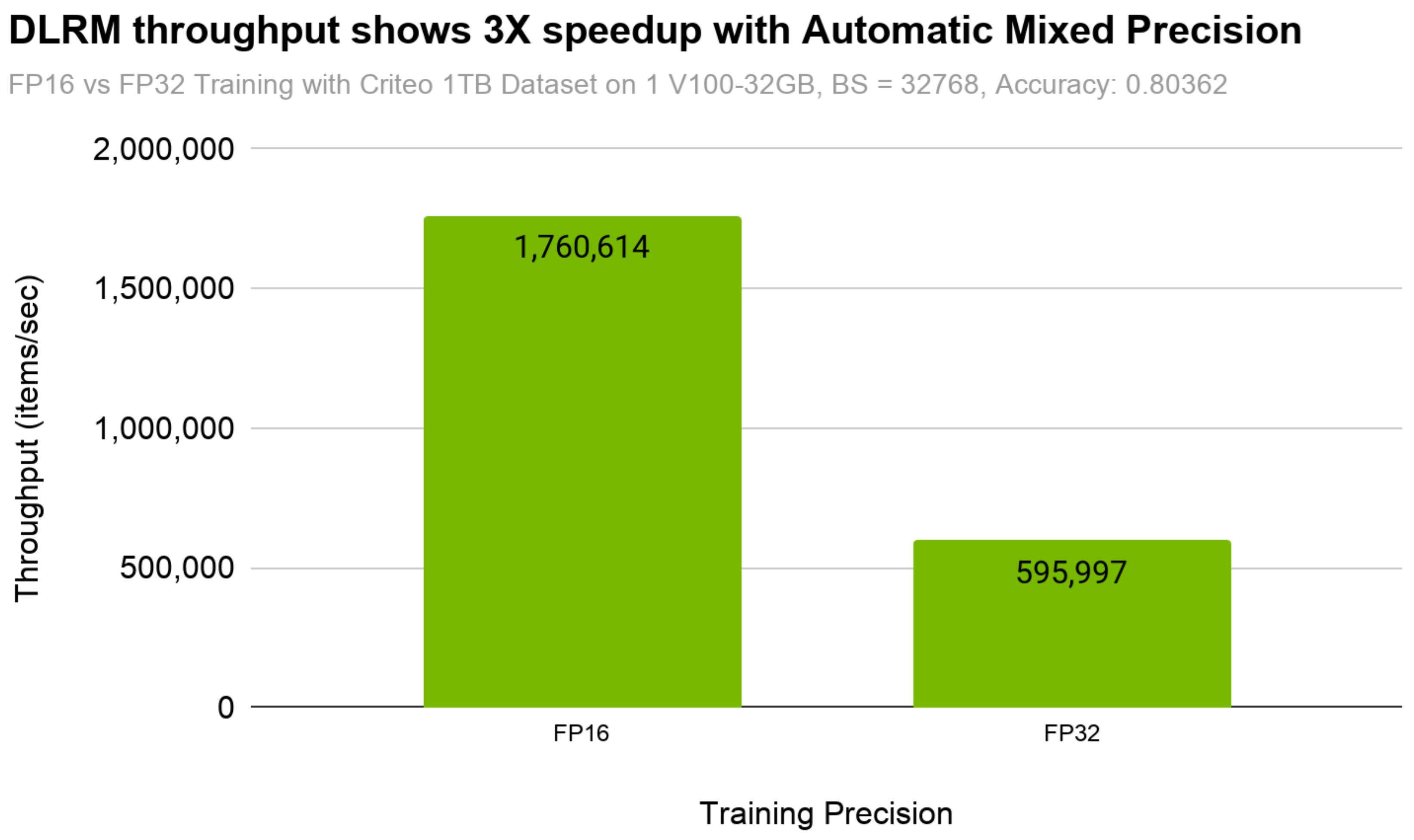
Training with Pytorch

Model Description:

- Provides state-of-art results while enables GPUs to work efficiently with production-scale data.
- Efficient GPU processing of categorical features using embeddings.
- Continuous features are efficiently processed with a bottom multilayer perceptron.

What's New:

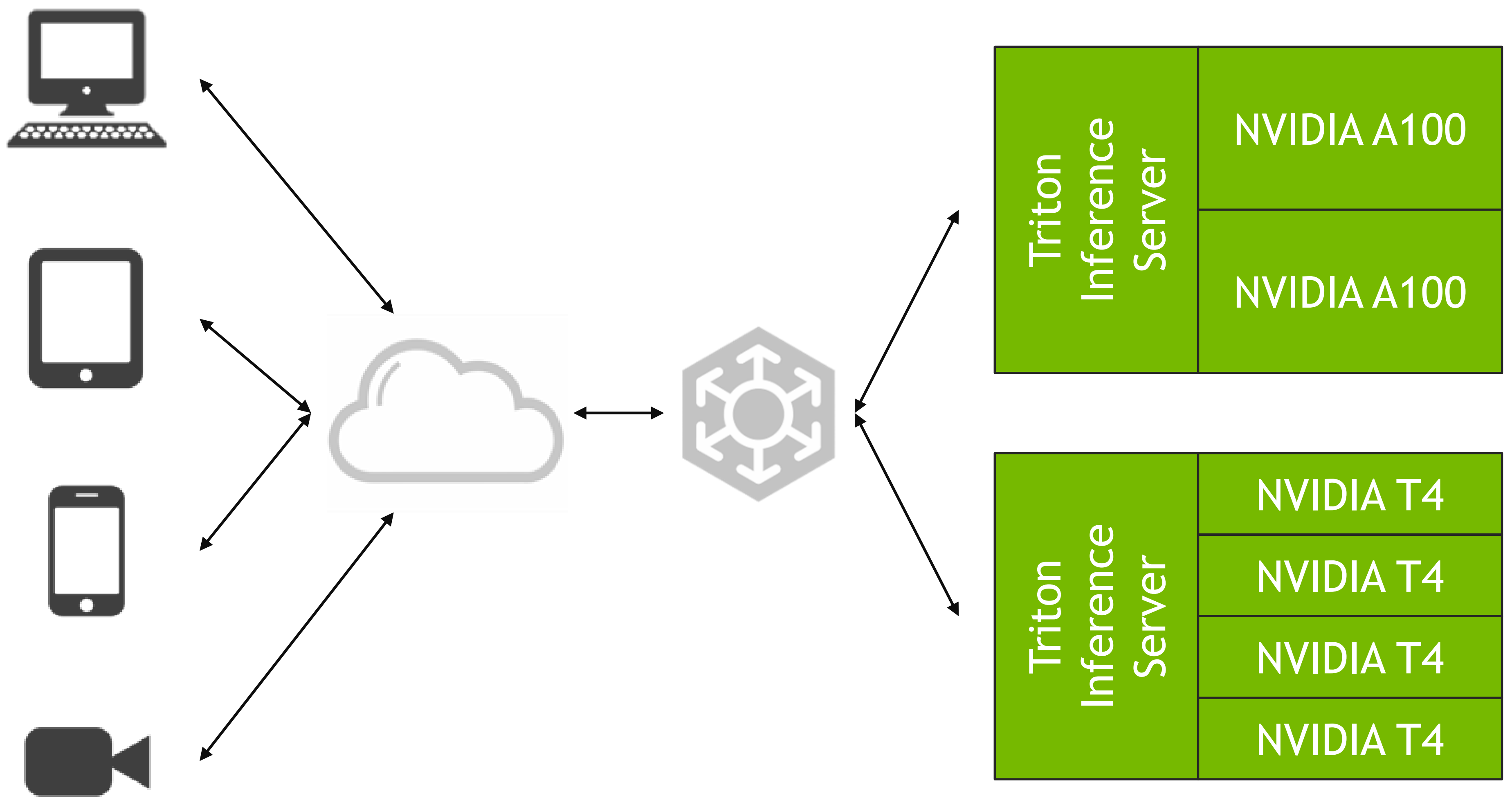
- 3x speedup with Automatic Mixed Precision.



Triton Inference Server

NVIDIA Triton

Production-Ready Inference Server



Maximize GPUs real-time inference performance.

Quickly deploy and manage multiple models per GPU per node.

Easily scale to heterogeneous GPUs and multi-GPU nodes.

Integrates with orchestration systems and auto-scalers via latency and health metrics.

Now open source for thorough customization and integration.

NVIDIA Triton

Key Features

Concurrent Model Execution

Multiple models (or multiple instances of same model) may execute on GPU simultaneously.

CPU Model Inference Execution

Framework native models can execute inference requests on the CPU.

Metrics

Utilization, count, memory, and latency.

Custom Backend

Custom backend allows the user more flexibility by providing their own implementation of an execution engine through the use of a shared library.

Model Ensemble

Pipeline of one or more models, connecting input and output tensors between those models.

Dynamic Batching

Inference requests can be batched up by the inference server to 1) the model-allowed maximum or 2) the user-defined latency SLA.

Multiple Model Format Support

- PyTorch JIT (.pt)
- TensorFlow GraphDef/SavedModel
- TensorFlow and TensorRT GraphDef
- ONNX graph (ONNX Runtime)
- TensorRT Plans
- Caffe2 NetDef (ONNX import path)

CMake build

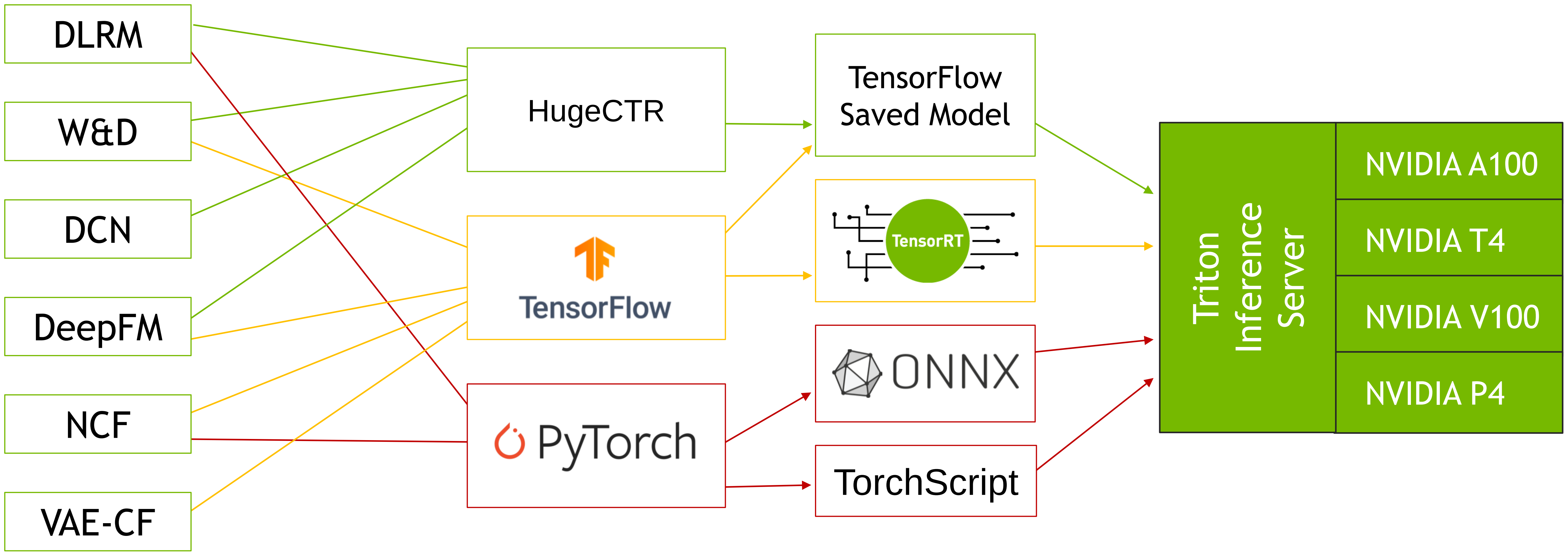
Build the inference server from source making it more portable to multiple OSes and removing the build dependency on Docker.

Streaming API

Built-in support for audio streaming input e.g. for speech recognition.

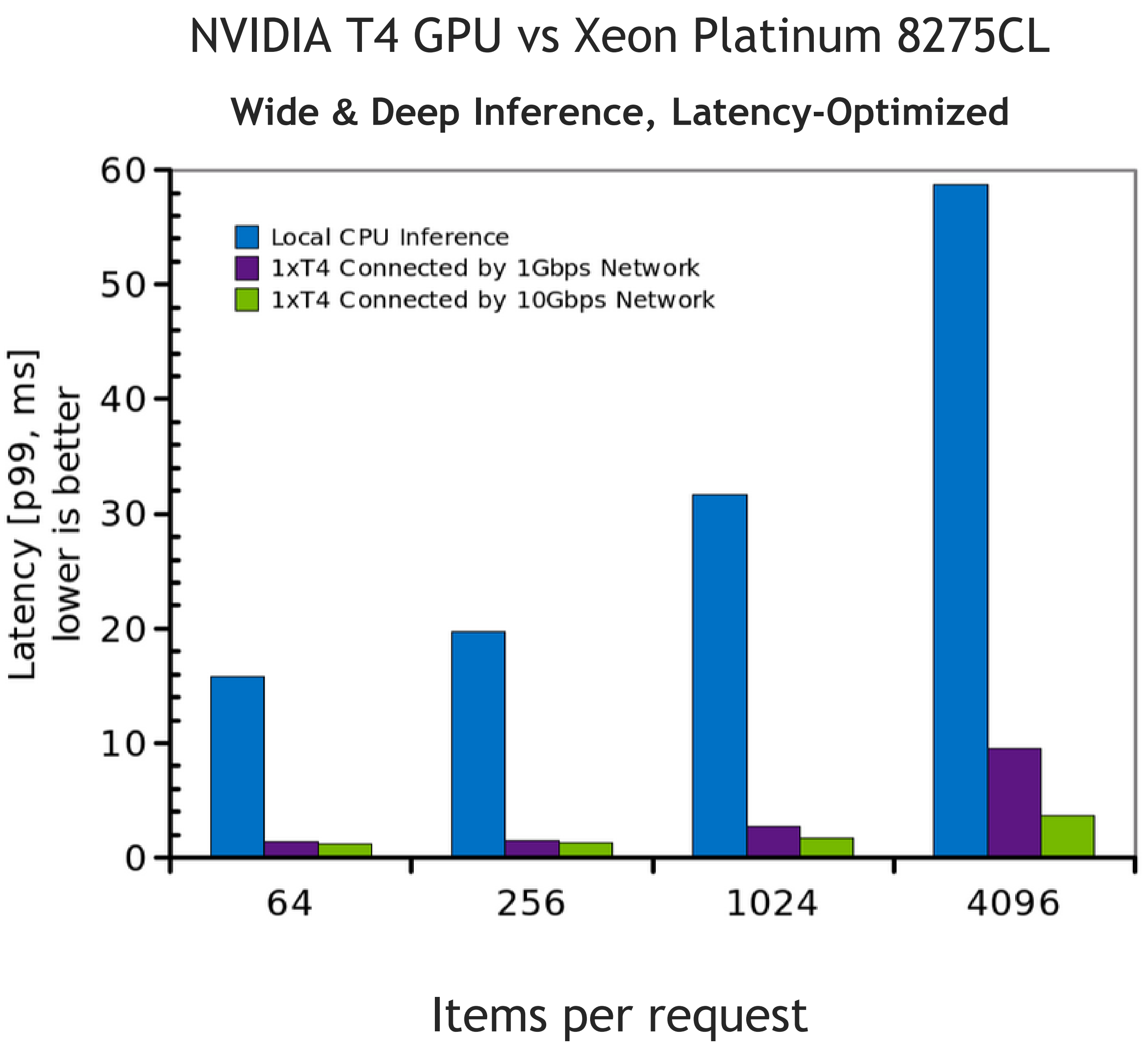
NVIDIA Triton

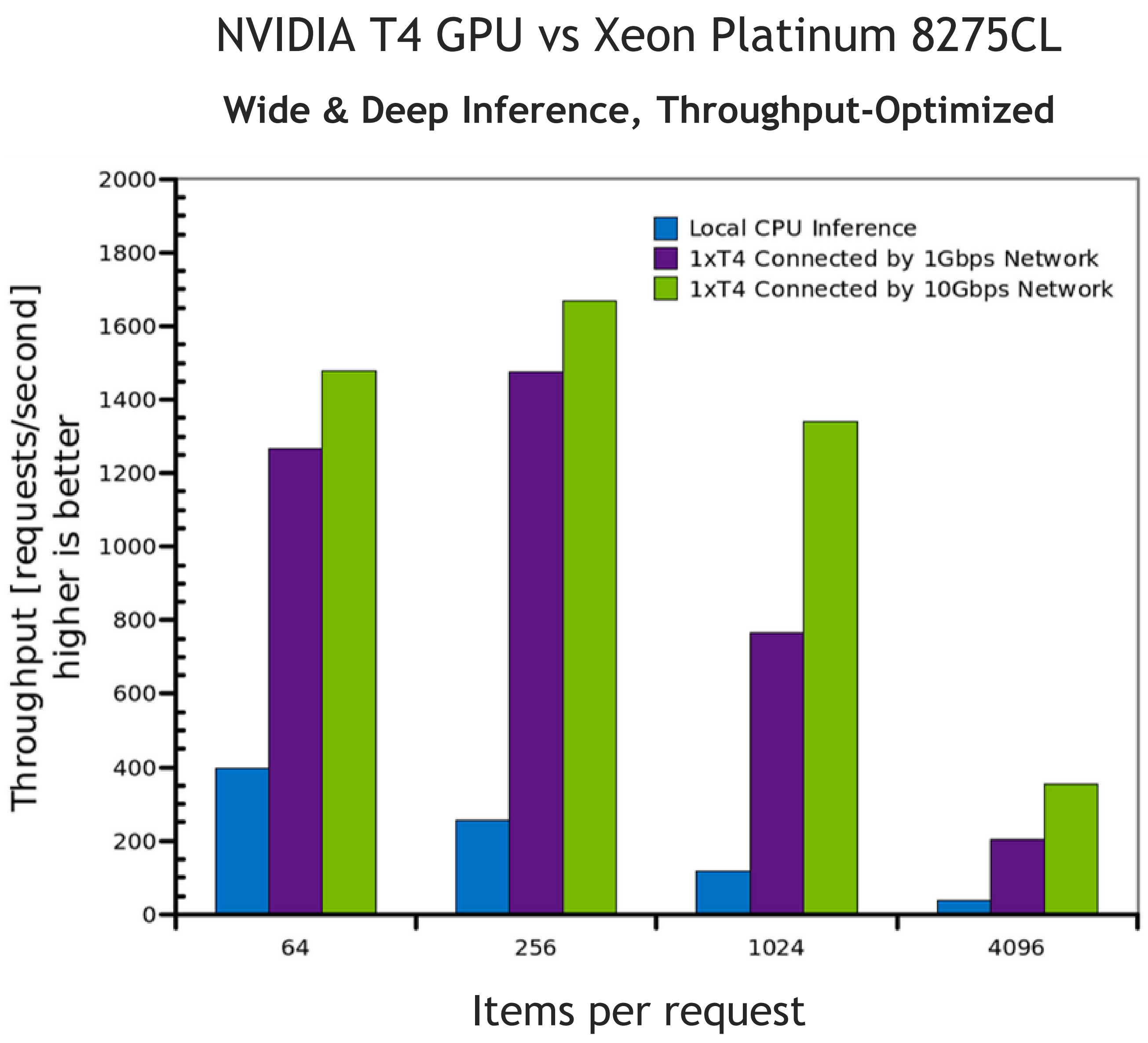
Path to Production



Example: Wide & Deep

Inference in Triton with TensorRT



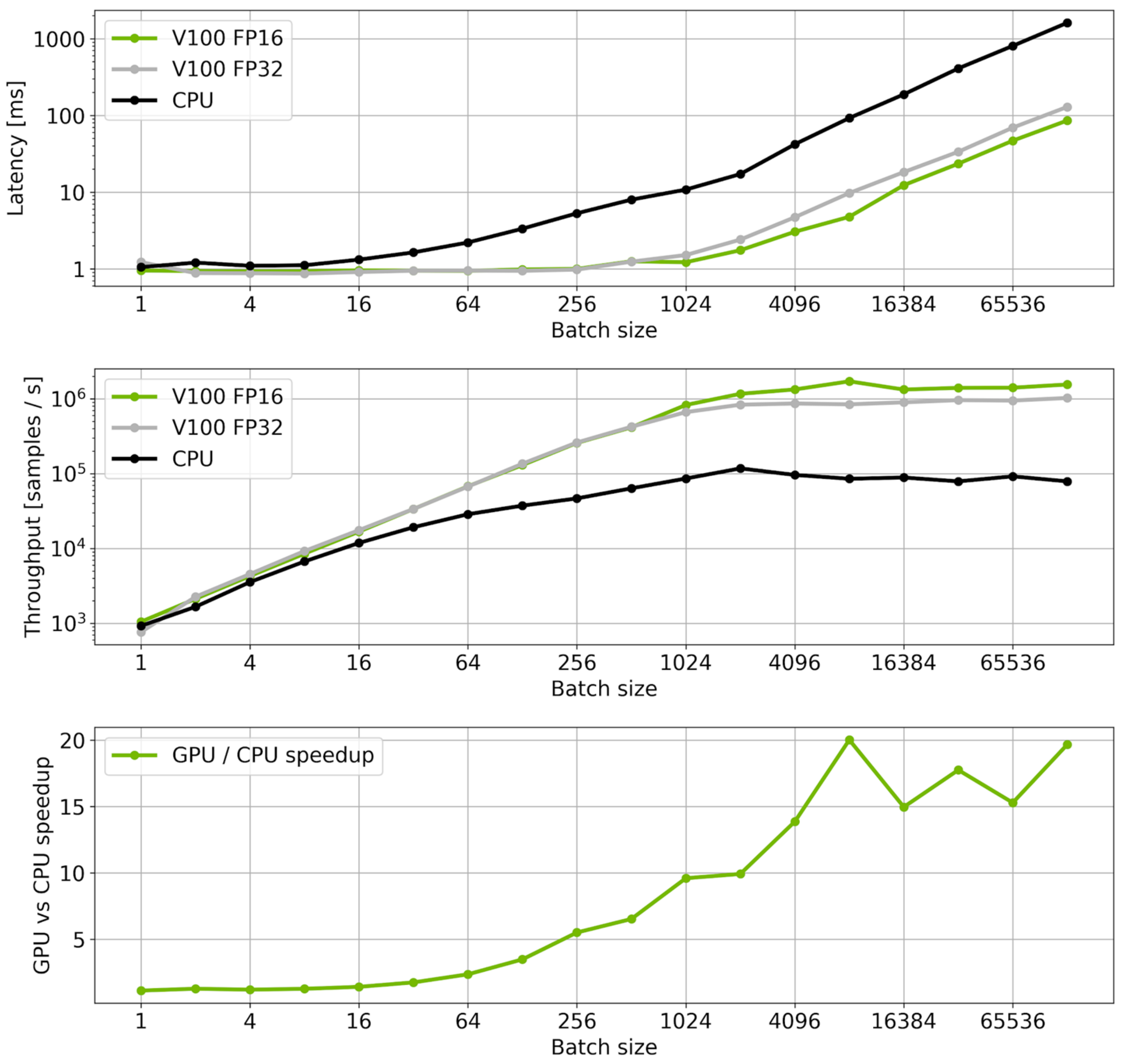


https://github.com/NVIDIA/HugeCTR/tree/masterhttps://devblogs.nvidia.com/accelerating-wide-deep-recommender-inference-on-gpus/samples/wdl

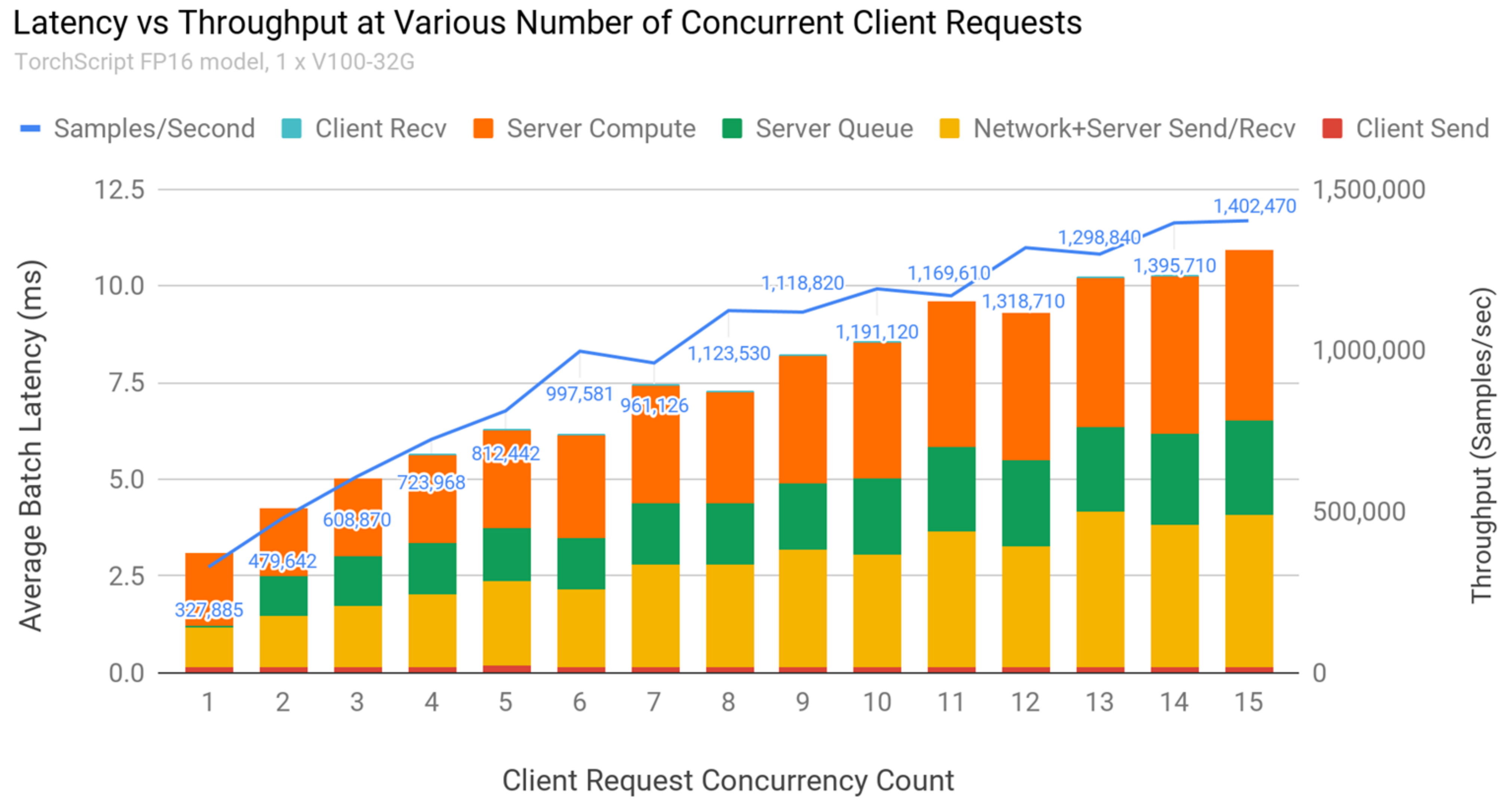
CPU: FP32; AWS c5d.24xlarge, single socket
GPU: NVIDA T4, mixed precision; AWS g4dn.16xlarge

Example: DLRM

Inference in Triton with TensorRT



Optimal batch size for running inference is 65536.



- Platform: 1x V100-32G.
- Release: 20.06-py3.
- Throughput is measured in recommendations/seconds.
- Latency is measured in milliseconds.

Summary

Recommender Systems Demystified

What we have seen today

RecSys are everywhere.

There are multiple techniques.

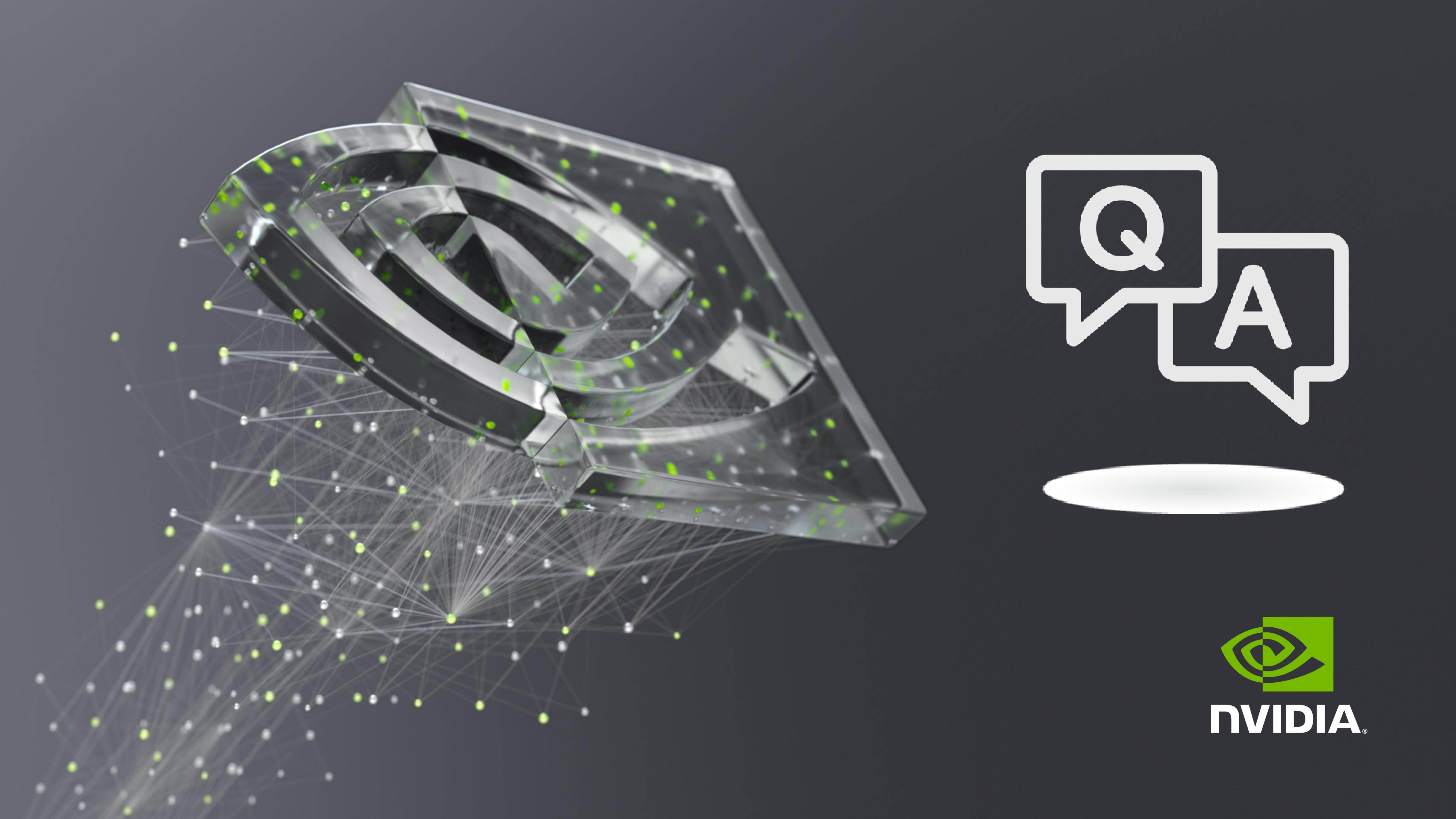
Collaborative Filtering + Content-Based Filtering.

Deep Learning Based Recommender Systems: Wide & Deep.

NVTabular + HugeCTR + Optimised Examples + Triton Inference Server.

Finally, the game cover in the third slide corresponds to Simon the Sorcerer.

That’s all folks!


Q
A
NVIDIA

Thank You!

NVIDIA