COMCAST

2023 Summer Internship

# Page Personalization

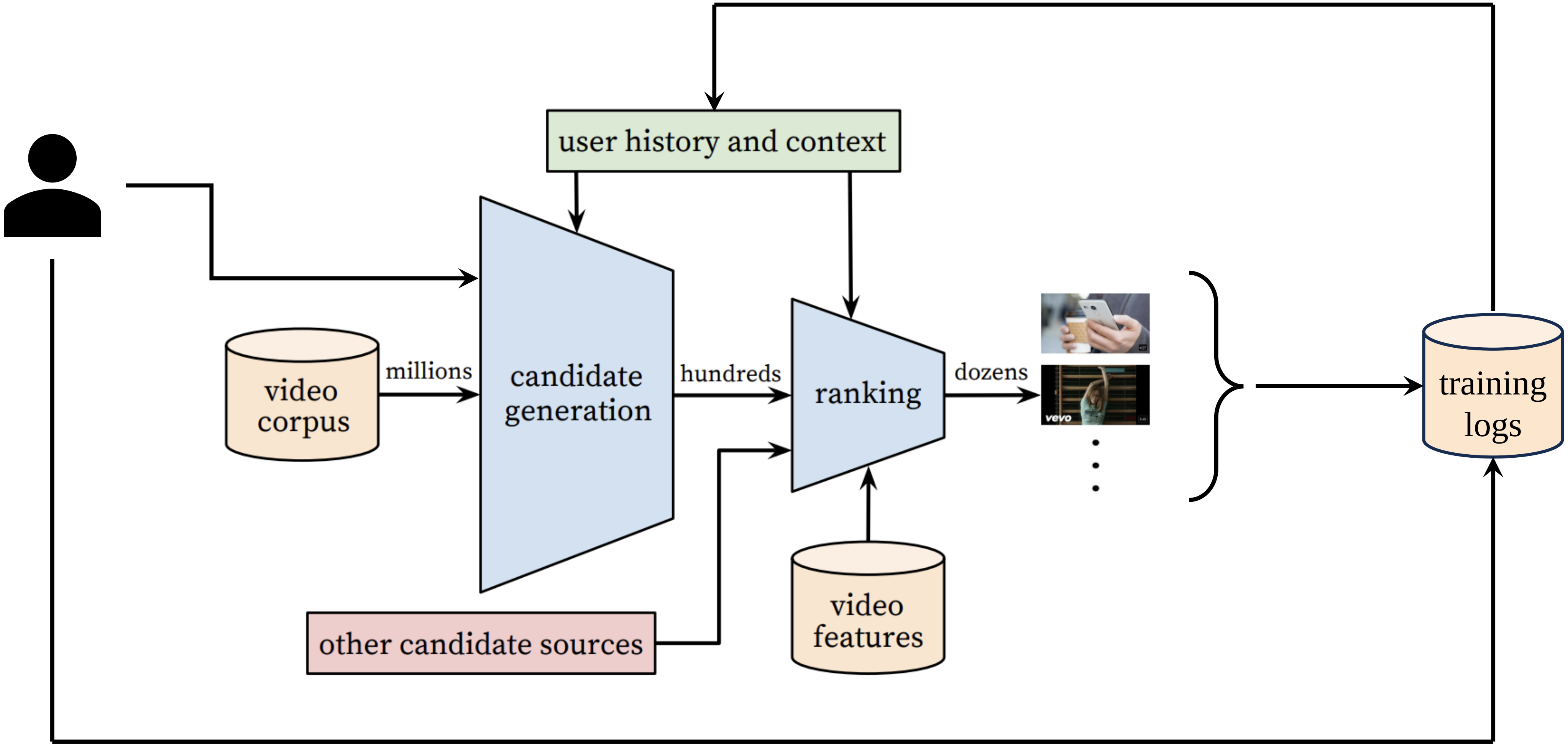
Meeshawn Marathe

MS in Artificial Intelligence

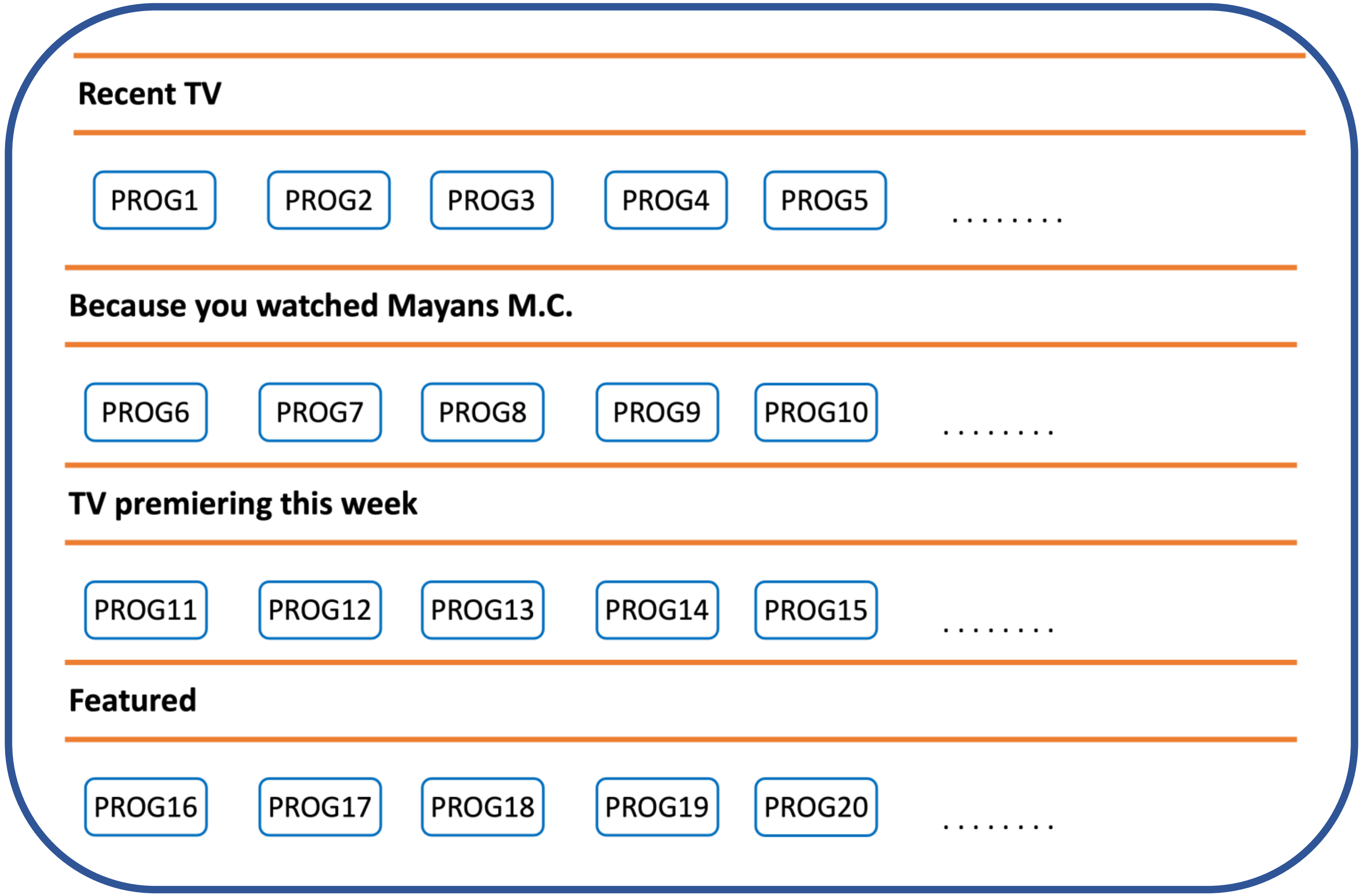
DEARBORN

COLLEGE OF ENGINEERING & COMPUTER SCIENCE
UNIVERSITY OF MICHIGAN-DEARBORN

# Recommendation Systems



# Problem Statement – **Page Personalization**



**Recent TV**

PROG1 PROG2 PROG3 PROG4 PROG5 ........

**Because you watched Mayans M.C.**

PROG6 PROG7 PROG8 PROG9 PROG10 ........

**TV premiering this week**

PROG11 PROG12 PROG13 PROG14 PROG15 ........

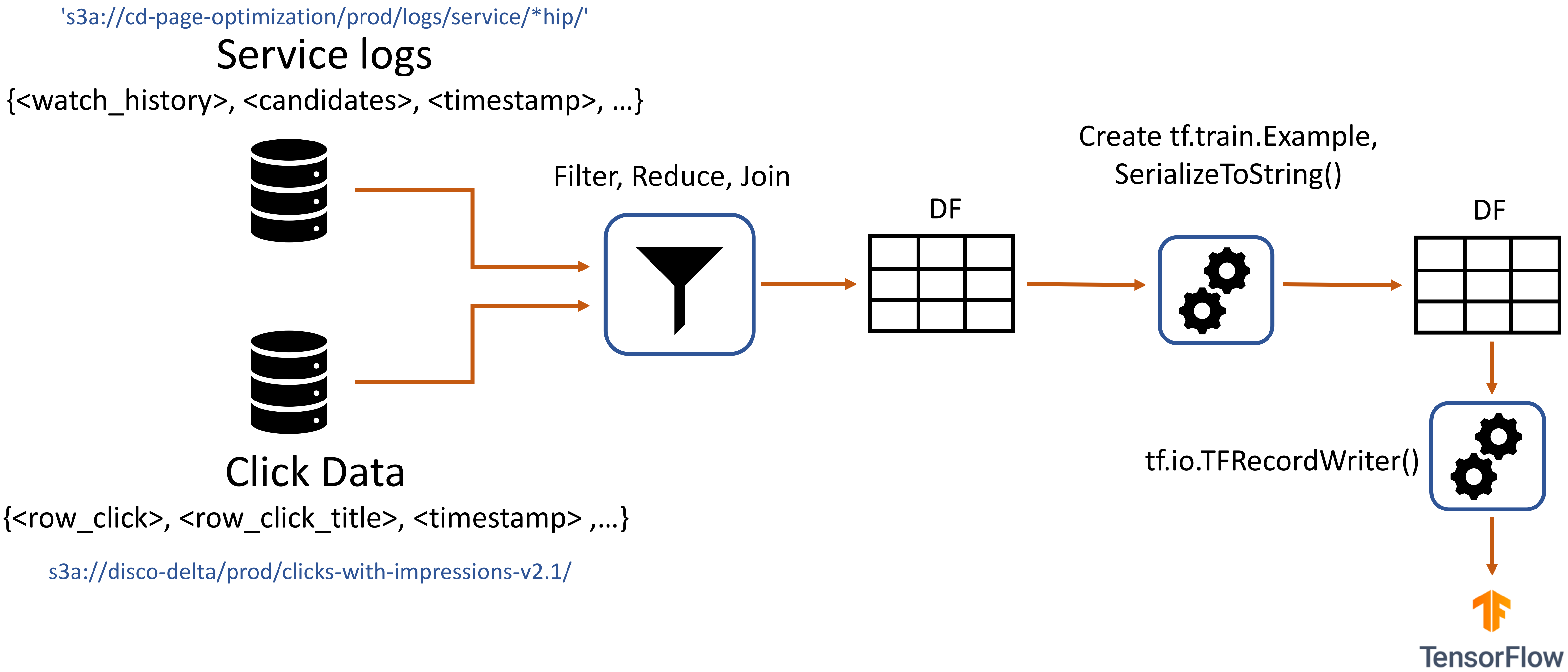
**Featured**

PROG16 PROG17 PROG18 PROG19 PROG20 ........

# Internship Goals

- Work with a new dataset
- Work with a new model:
  - Single Tower Model:
    - Full Softmax
    - Sampled Softmax
  - Two Tower Model:
    - Full Softmax
    - Sampled Softmax
- Work with a new library “*Tensorflow Recommenders*”
- Performance Comparison with existing model

# Methodology – Data Generation Pipeline



's3a://cd-page-optimization/prod/logs/service/*hip/'

Service logs

{<watch_history>, <candidates>, <timestamp>, …}

Filter, Reduce, Join

DF

Create tf.train.Example,
SerializeToString()

DF

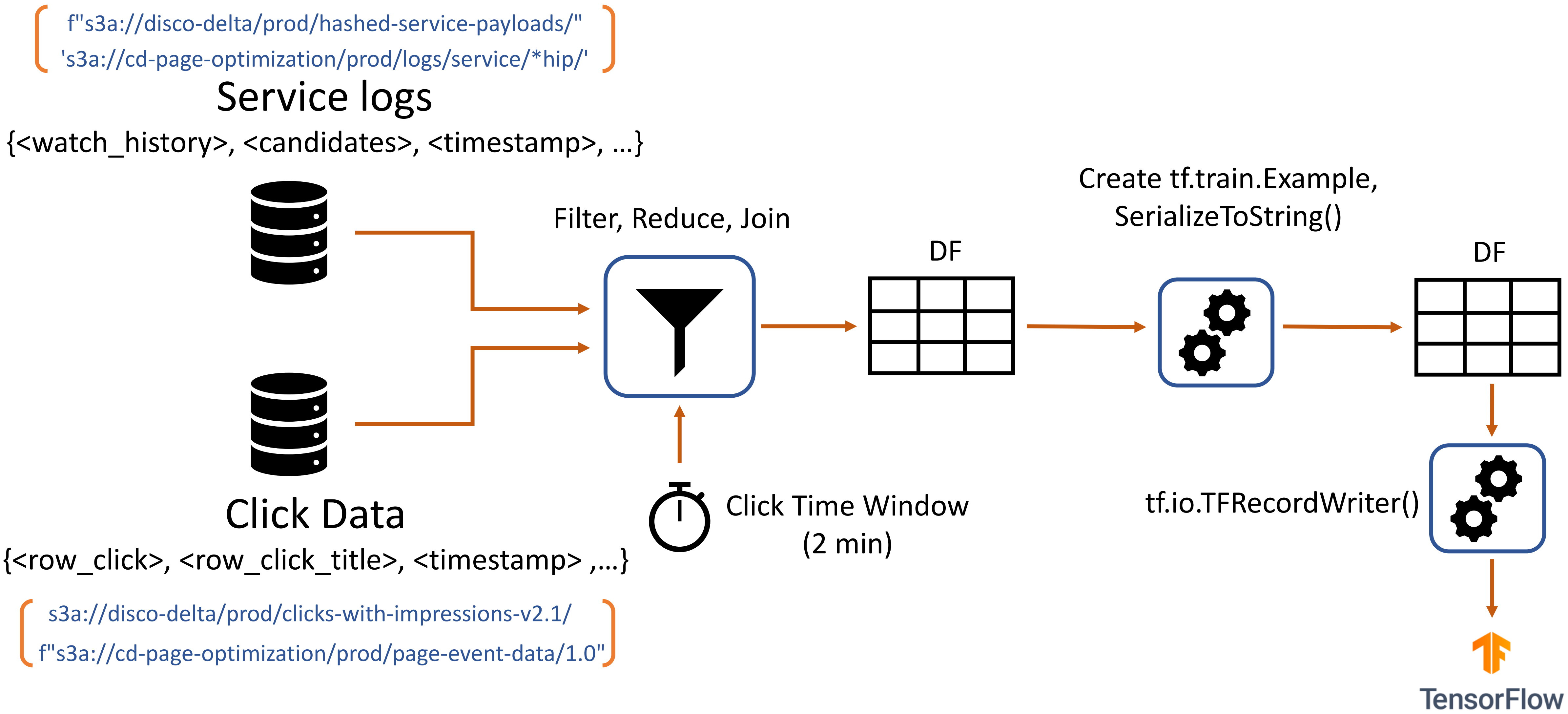
tf.io.TFRecordWriter()

Click Data

{<row_click>, <row_click_title>, <timestamp> ,…}

s3a://disco-delta/prod/clicks-with-impressions-v2.1/

# Methodology – Data Generation Pipeline



f"s3a://disco-delta/prod/hashed-service-payloads/"
's3a://cd-page-optimization/prod/logs/service/*hip/'

Service logs

{<watch_history>, <candidates>, <timestamp>, …}

Filter, Reduce, Join

DF

Create tf.train.Example,
SerializeToString()

DF

Click Time Window
(2 min)

tf.io.TFRecordWriter()

TensorFlow

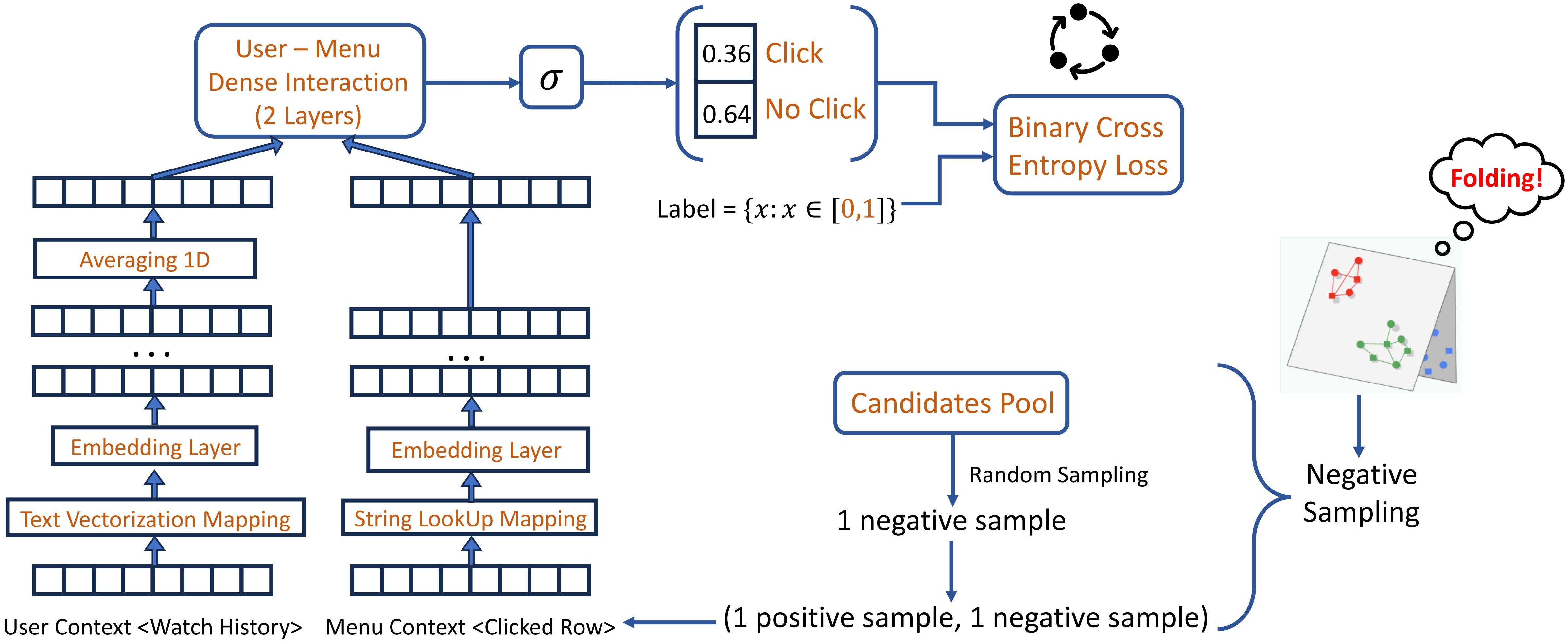
Click Data

{<row_click>, <row_click_title>, <timestamp> ,…}

s3a://disco-delta/prod/clicks-with-impressions-v2.1/
f"s3a://cd-page-optimization/prod/page-event-data/1.0"

# Methodology – Deep RecSys Training Pipeline

- Two-Tower Binary Classification Model (Click/No-Click)



User – Menu Dense Interaction (2 Layers)

$\sigma$

0.36 Click

0.64 No Click

Label = $\{x : x \in [0,1]\}$

Binary Cross Entropy Loss

Averaging 1D

Embedding Layer

Text Vectorization Mapping

Embedding Layer

String LookUp Mapping

User Context <Watch History>

Menu Context <Clicked Row>

Candidates Pool

Random Sampling

1 negative sample

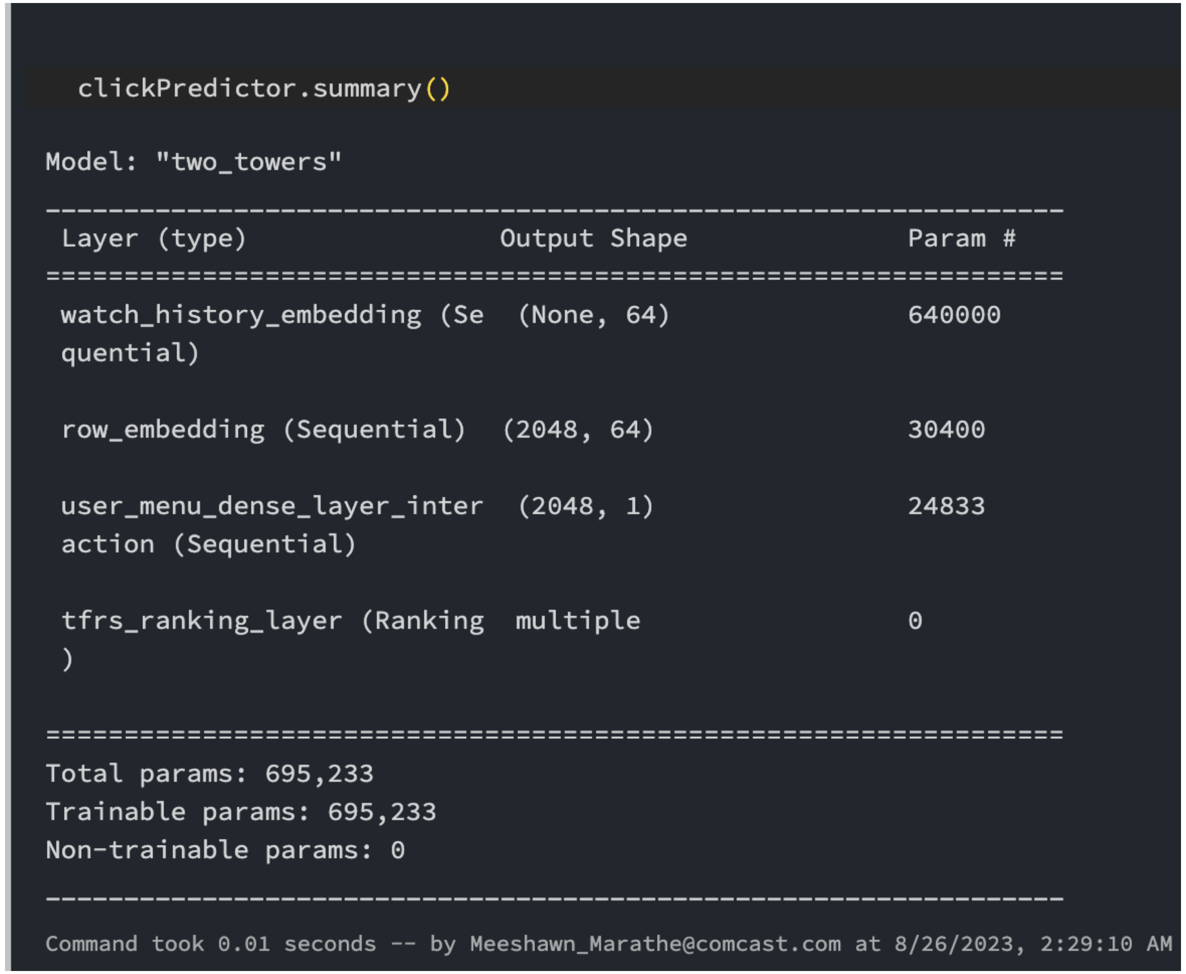
(1 positive sample, 1 negative sample)

Negative Sampling

Folding!

# Methodology – Deep RecSys Training Pipeline

- Two-Tower Binary Classification Model (Click/No-Click)



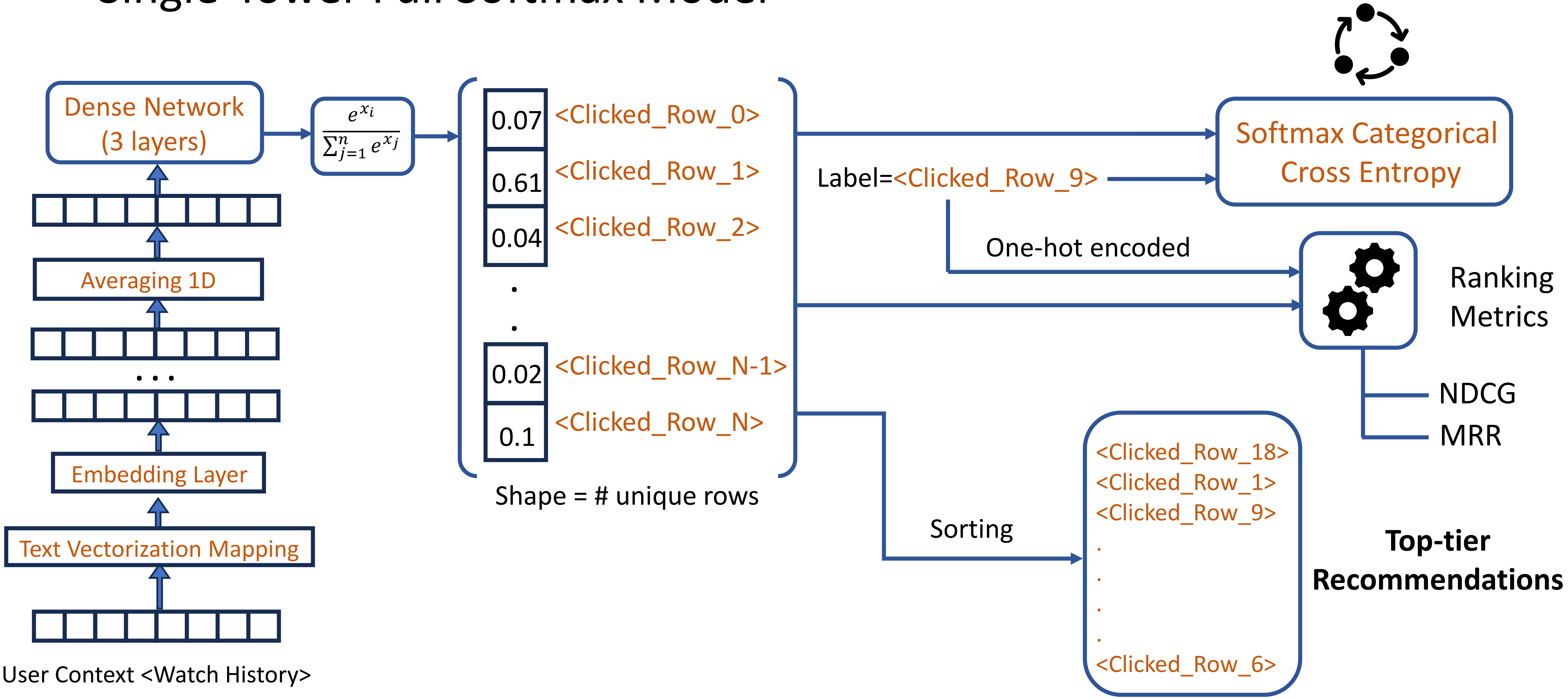
```
clickPredictor.summary()
```

```
Model: "two_towers"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 watch_history_embedding (Se  (None, 64)               640000    
 quential)                                                       
                                                                 
 row_embedding (Sequential)  (2048, 64)                30400     
                                                                 
 user_menu_dense_layer_inter  (2048, 1)                24833     
 action (Sequential)                                             
                                                                 
 tfrs_ranking_layer (Ranking  multiple                 0         
 )                                                               
                                                                 
=================================================================
Total params: 695,233
Trainable params: 695,233
Non-trainable params: 0
_________________________________________________________________
```

Command took 0.01 seconds -- by Meeshawn_Marathe@comcast.com at 8/26/2023, 2:29:10 AM

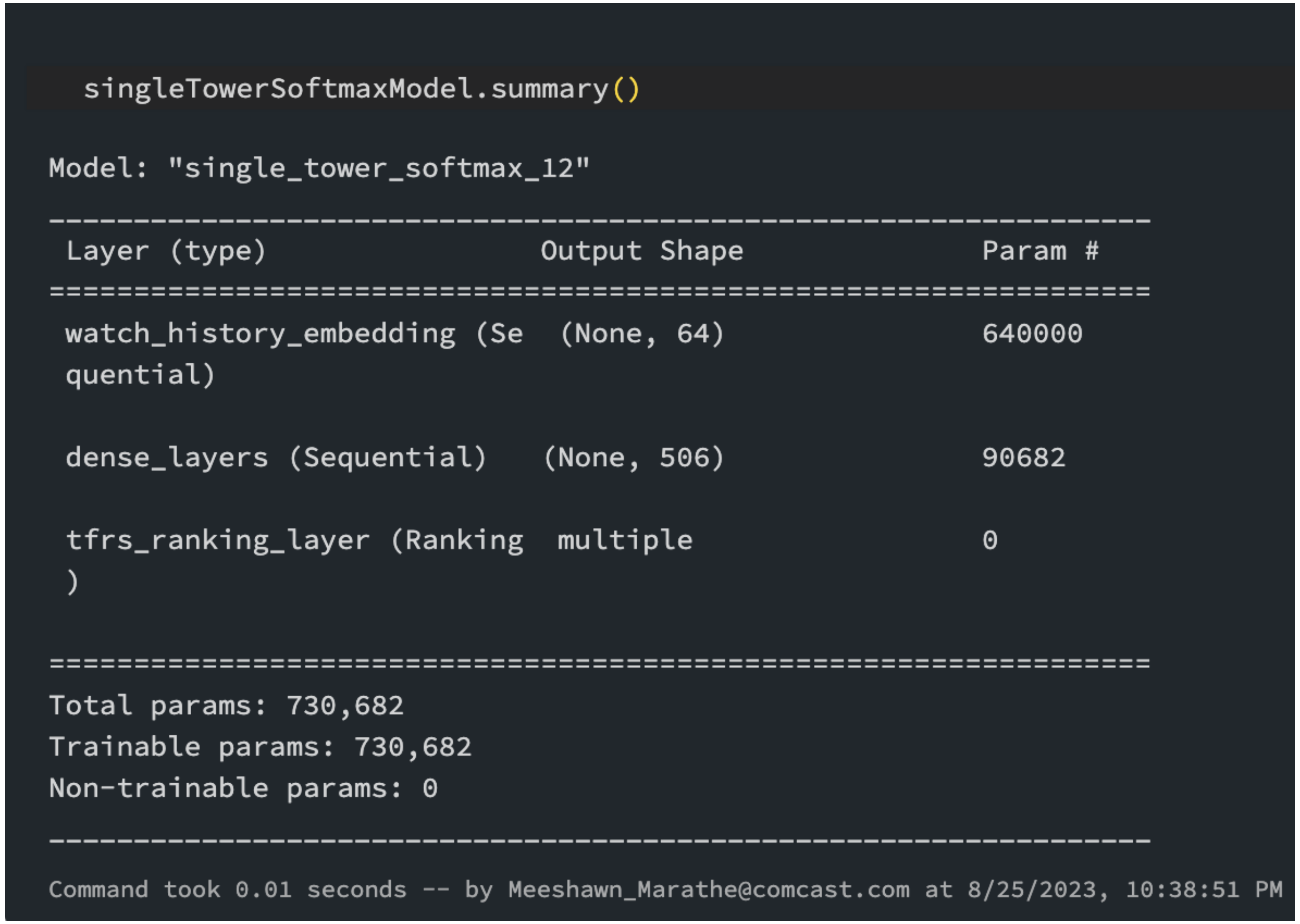
# Methodology – Deep RecSys Training Pipeline

- Single-Tower Full Softmax Model



# Methodology – Deep RecSys Training Pipeline

- Single-Tower Full Softmax Model

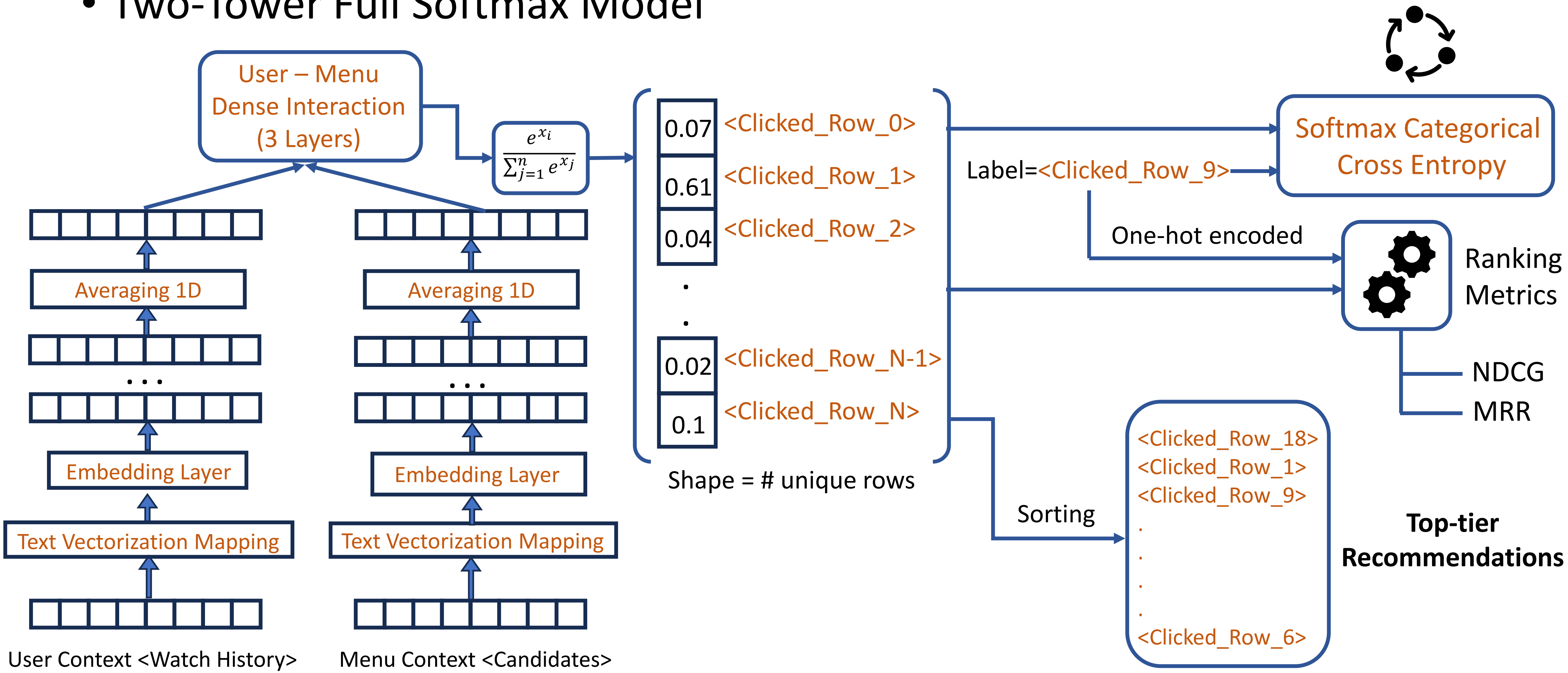


```
singleTowerSoftmaxModel.summary()
```

```
Model: "single_tower_softmax_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 watch_history_embedding (Se  (None, 64)               640000
 quential)

 dense_layers (Sequential)   (None, 506)               90682

 tfrs_ranking_layer (Ranking  multiple                 0
 )

=================================================================
Total params: 730,682
Trainable params: 730,682
Non-trainable params: 0
_________________________________________________________________
Command took 0.01 seconds -- by Meeshawn_Marathe@comcast.com at 8/25/2023, 10:38:51 PM
```

# Methodology – Deep RecSys Training Pipeline

- Two-Tower Full Softmax Model



User – Menu Dense Interaction (3 Layers)

$$\frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Averaging 1D

Embedding Layer

Text Vectorization Mapping

User Context <Watch History>

Averaging 1D

Embedding Layer

Text Vectorization Mapping

Menu Context <Candidates>

0.07 <Clicked_Row_0>

0.61 <Clicked_Row_1>

0.04 <Clicked_Row_2>

0.02 <Clicked_Row_N-1>

0.1 <Clicked_Row_N>

Shape = # unique rows

Softmax Categorical Cross Entropy

Label=<Clicked_Row_9>

One-hot encoded

Ranking Metrics

NDCG

MRR

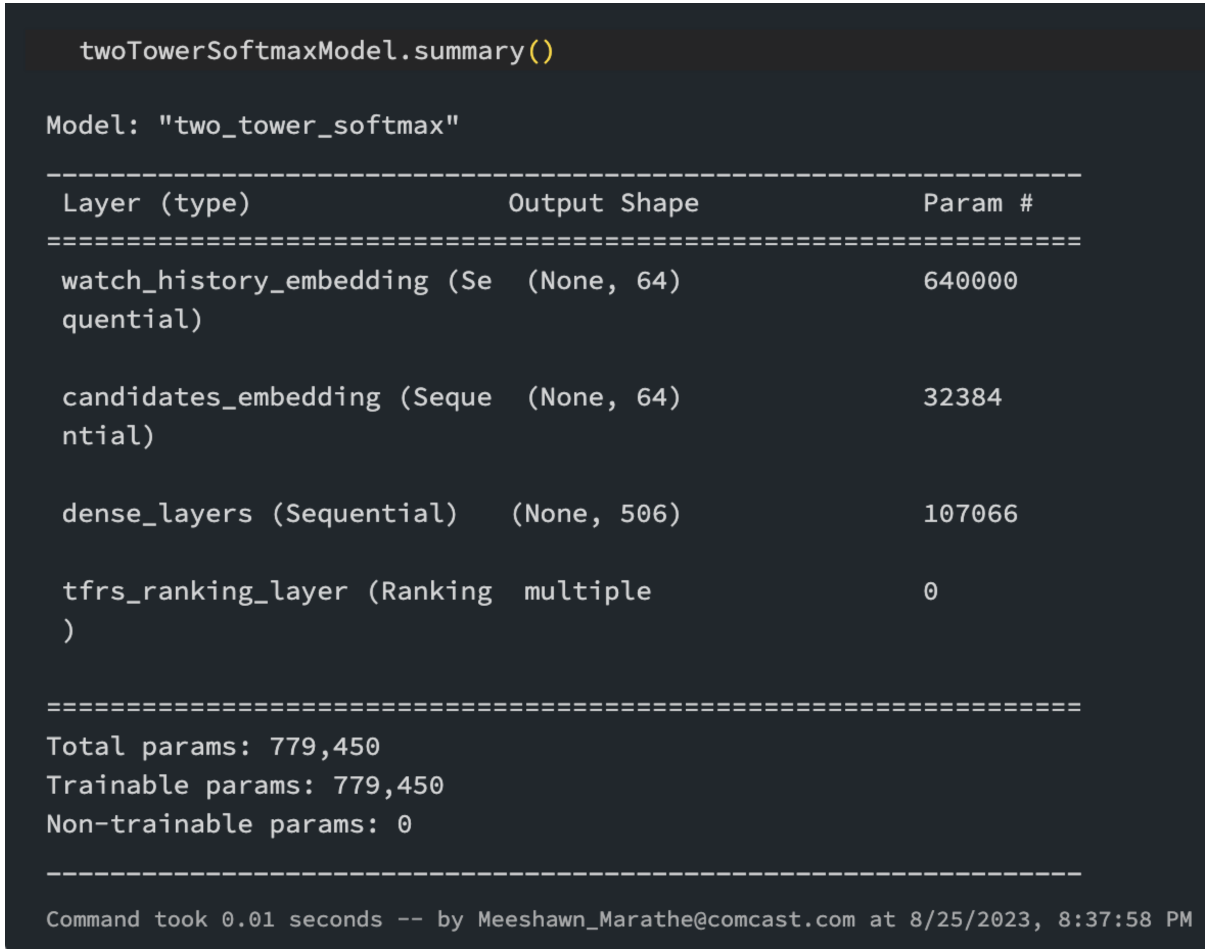
Sorting

<Clicked_Row_18>
<Clicked_Row_1>
<Clicked_Row_9>
.
.
.
.
<Clicked_Row_6>

**Top-tier Recommendations**

# Methodology – Deep RecSys Training Pipeline

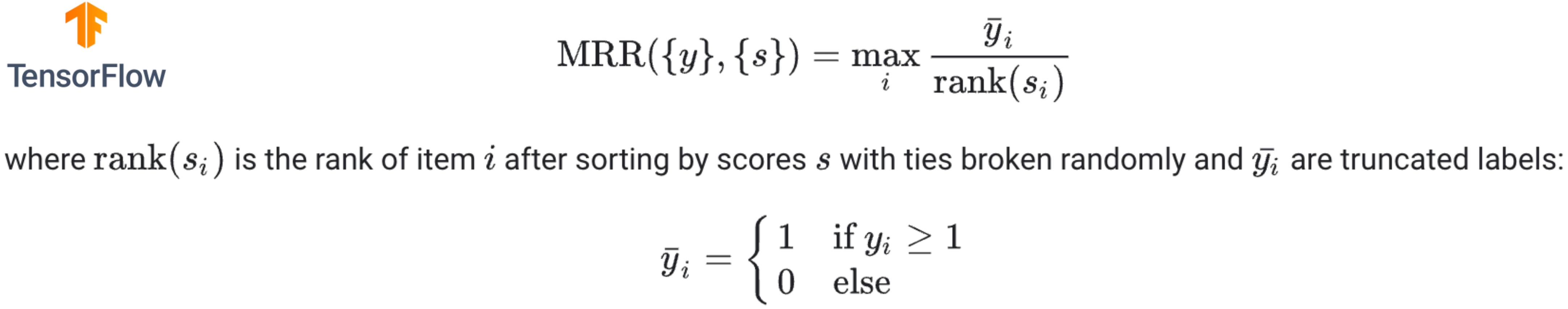
- Two-Tower Full Softmax Model



```
  twoTowerSoftmaxModel.summary()

Model: "two_tower_softmax"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 watch_history_embedding (Se  (None, 64)               640000
 quential)

 candidates_embedding (Seque  (None, 64)               32384
 ntial)

 dense_layers (Sequential)   (None, 506)               107066

 tfrs_ranking_layer (Ranking  multiple                 0
 )

=================================================================
Total params: 779,450
Trainable params: 779,450
Non-trainable params: 0
_________________________________________________________________
Command took 0.01 seconds -- by Meeshawn_Marathe@comcast.com at 8/25/2023, 8:37:58 PM
```

# Methodology - Metrics

TensorFlow

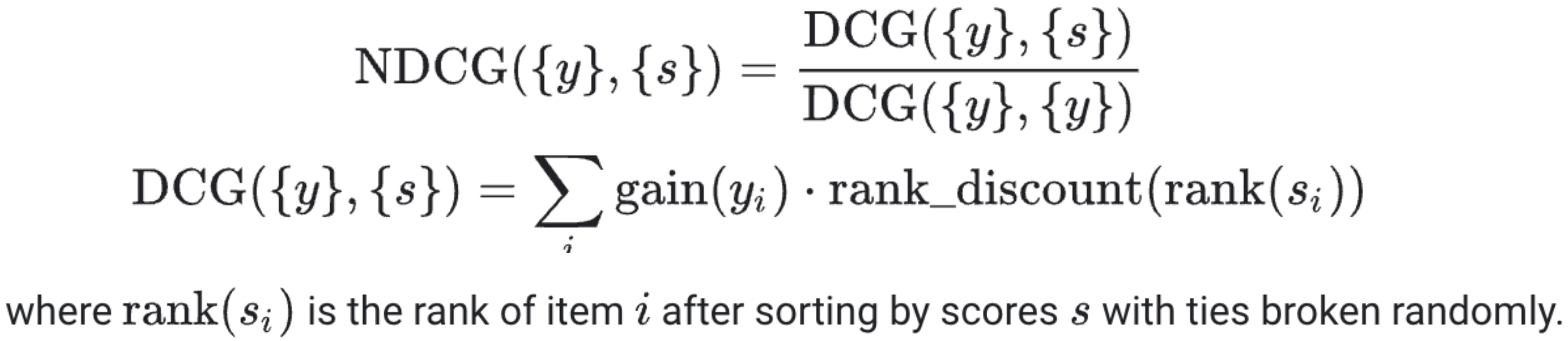


$$\mathrm{MRR}(\{y\}, \{s\}) = \max_i \frac{\bar{y}_i}{\mathrm{rank}(s_i)}$$

where $\mathrm{rank}(s_i)$ is the rank of item $i$ after sorting by scores $s$ with ties broken randomly and $\bar{y_i}$ are truncated labels:

$$\bar{y}_i = \begin{cases} 1 & \text{if } y_i \geq 1 \\ 0 & \text{else} \end{cases}$$

TensorFlow

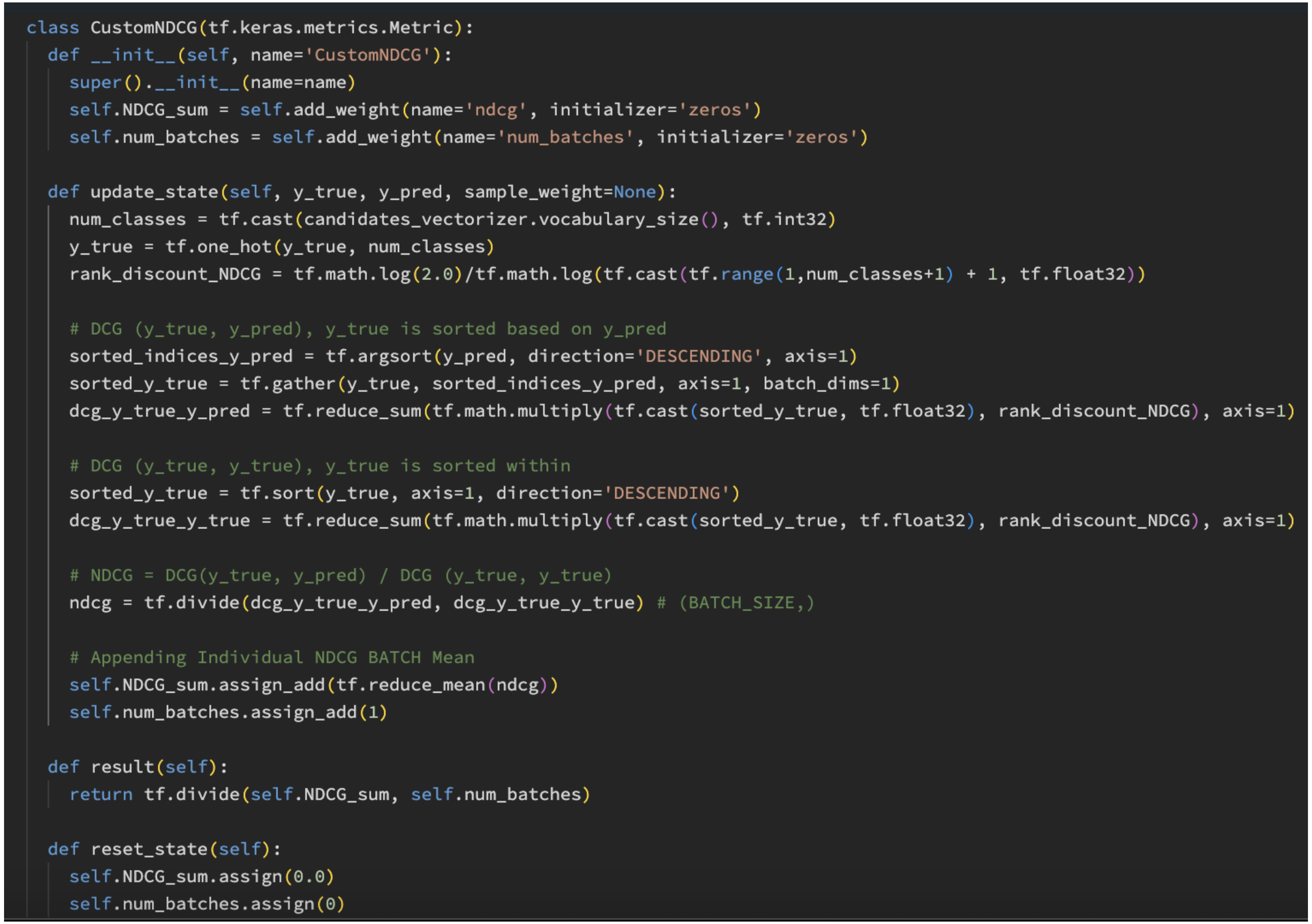


$$\mathrm{NDCG}(\{y\}, \{s\}) = \frac{\mathrm{DCG}(\{y\}, \{s\})}{\mathrm{DCG}(\{y\}, \{y\})}$$

$$\mathrm{DCG}(\{y\}, \{s\}) = \sum_i \mathrm{gain}(y_i) \cdot \mathrm{rank\_discount}(\mathrm{rank}(s_i))$$

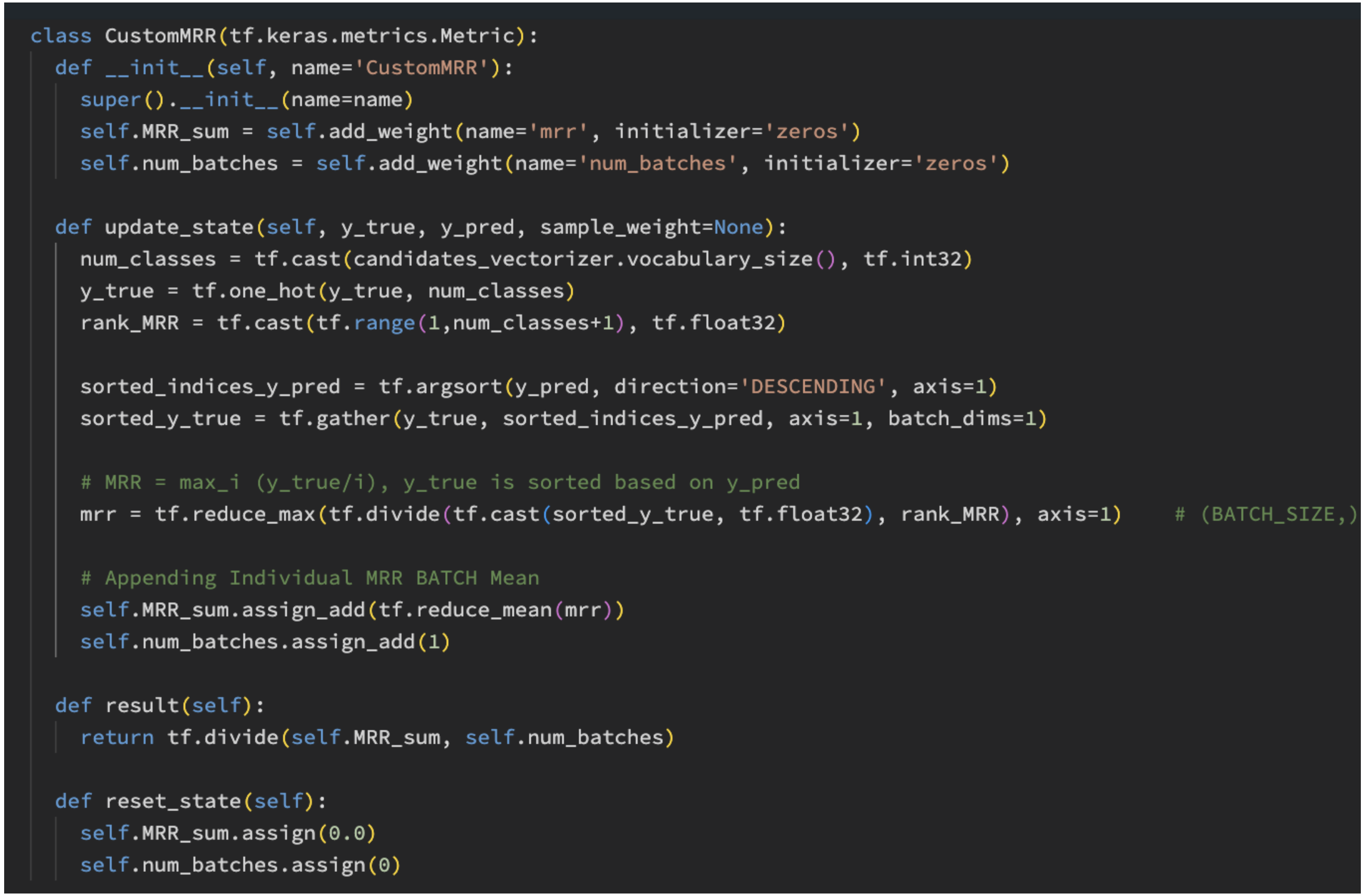
where $\mathrm{rank}(s_i)$ is the rank of item $i$ after sorting by scores $s$ with ties broken randomly.

# Methodology – Metrics: NDCG



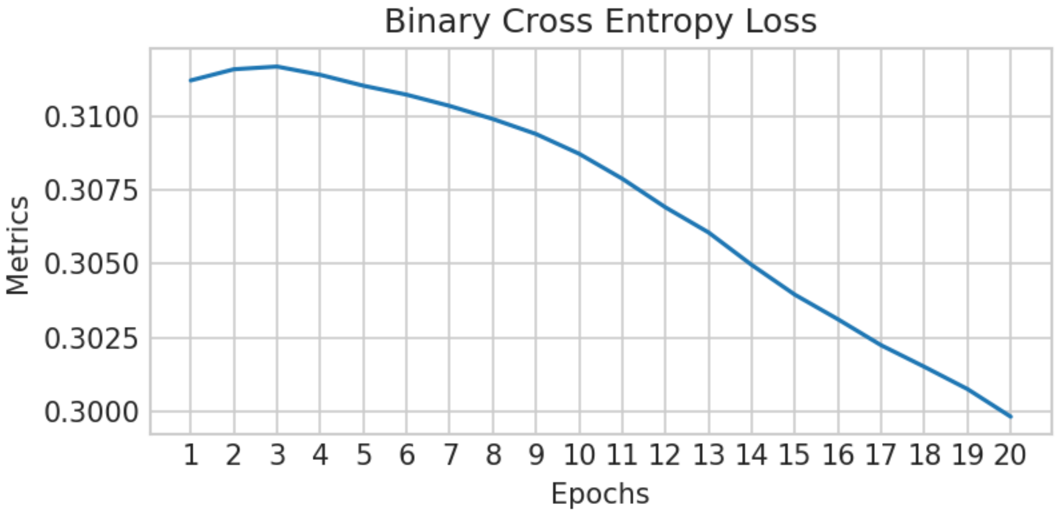
```
class CustomNDCG(tf.keras.metrics.Metric):
  def __init__(self, name='CustomNDCG'):
    super().__init__(name=name)
    self.NDCG_sum = self.add_weight(name='ndcg', initializer='zeros')
    self.num_batches = self.add_weight(name='num_batches', initializer='zeros')

  def update_state(self, y_true, y_pred, sample_weight=None):
    num_classes = tf.cast(candidates_vectorizer.vocabulary_size(), tf.int32)
    y_true = tf.one_hot(y_true, num_classes)
    rank_discount_NDCG = tf.math.log(2.0)/tf.math.log(tf.cast(tf.range(1,num_classes+1) + 1, tf.float32))

    # DCG (y_true, y_pred), y_true is sorted based on y_pred
    sorted_indices_y_pred = tf.argsort(y_pred, direction='DESCENDING', axis=1)
    sorted_y_true = tf.gather(y_true, sorted_indices_y_pred, axis=1, batch_dims=1)
    dcg_y_true_y_pred = tf.reduce_sum(tf.math.multiply(tf.cast(sorted_y_true, tf.float32), rank_discount_NDCG), axis=1)

    # DCG (y_true, y_true), y_true is sorted within
    sorted_y_true = tf.sort(y_true, axis=1, direction='DESCENDING')
    dcg_y_true_y_true = tf.reduce_sum(tf.math.multiply(tf.cast(sorted_y_true, tf.float32), rank_discount_NDCG), axis=1)

    # NDCG = DCG(y_true, y_pred) / DCG (y_true, y_true)
    ndcg = tf.divide(dcg_y_true_y_pred, dcg_y_true_y_true) # (BATCH_SIZE,)

    # Appending Individual NDCG BATCH Mean
    self.NDCG_sum.assign_add(tf.reduce_mean(ndcg))
    self.num_batches.assign_add(1)

  def result(self):
    return tf.divide(self.NDCG_sum, self.num_batches)

  def reset_state(self):
    self.NDCG_sum.assign(0.0)
    self.num_batches.assign(0)
```

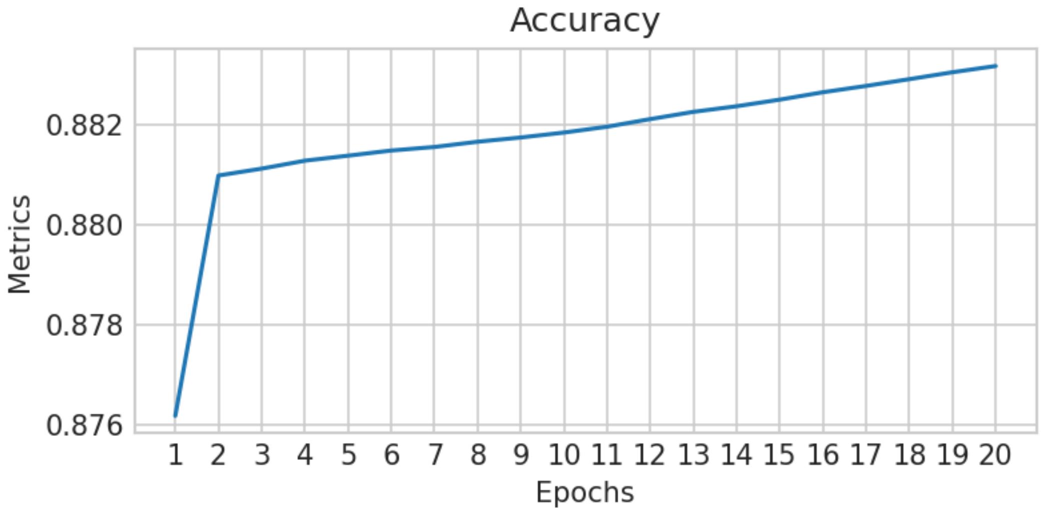
# Methodology – Metrics: MRR



```
class CustomMRR(tf.keras.metrics.Metric):
  def __init__(self, name='CustomMRR'):
    super().__init__(name=name)
    self.MRR_sum = self.add_weight(name='mrr', initializer='zeros')
    self.num_batches = self.add_weight(name='num_batches', initializer='zeros')

  def update_state(self, y_true, y_pred, sample_weight=None):
    num_classes = tf.cast(candidates_vectorizer.vocabulary_size(), tf.int32)
    y_true = tf.one_hot(y_true, num_classes)
    rank_MRR = tf.cast(tf.range(1,num_classes+1), tf.float32)

    sorted_indices_y_pred = tf.argsort(y_pred, direction='DESCENDING', axis=1)
    sorted_y_true = tf.gather(y_true, sorted_indices_y_pred, axis=1, batch_dims=1)

    # MRR = max_i (y_true/i), y_true is sorted based on y_pred
    mrr = tf.reduce_max(tf.divide(tf.cast(sorted_y_true, tf.float32), rank_MRR), axis=1)    # (BATCH_SIZE,)

    # Appending Individual MRR BATCH Mean
    self.MRR_sum.assign_add(tf.reduce_mean(mrr))
    self.num_batches.assign_add(1)

  def result(self):
    return tf.divide(self.MRR_sum, self.num_batches)

  def reset_state(self):
    self.MRR_sum.assign(0.0)
    self.num_batches.assign(0)
```

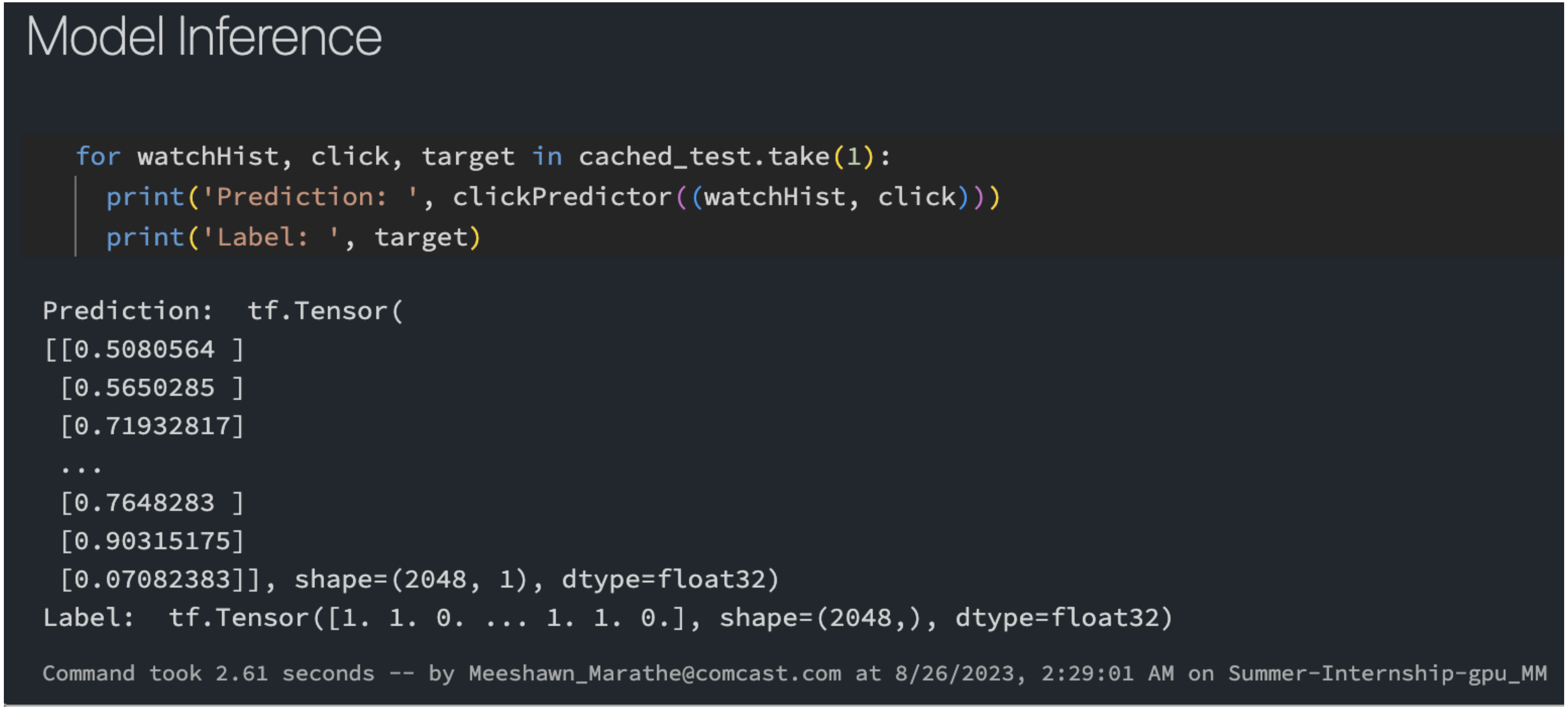
# Results & Discussion – Two-Tower Binary Classification





- **Batch Size:** 2048
- **Train Dataset:**
  - Period: 3rd - 4th July 2023
  - Size: ≈ 2 million x 2 (Negative sampling)
  - Training Time ≈ 1 hour
  - Accuracy = 0.8832
- **Test Dataset:**
  - Period: 5th July 2023
  - Size: ≈ 0.851 million
  - Accuracy = 0.8847

# Results & Discussion – Two-Tower Binary Classification



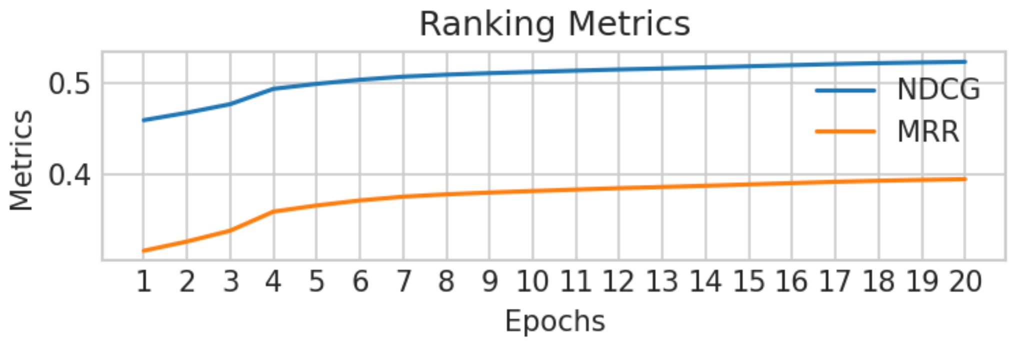
## Model Inference

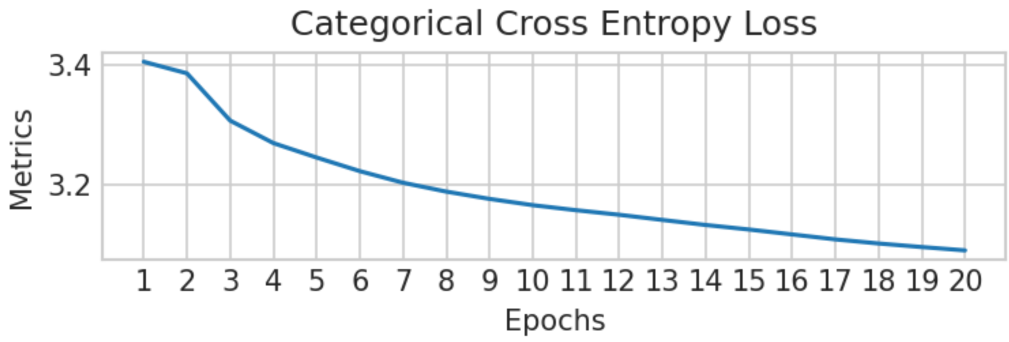
```
for watchHist, click, target in cached_test.take(1):
  print('Prediction: ', clickPredictor((watchHist, click)))
  print('Label: ', target)
```

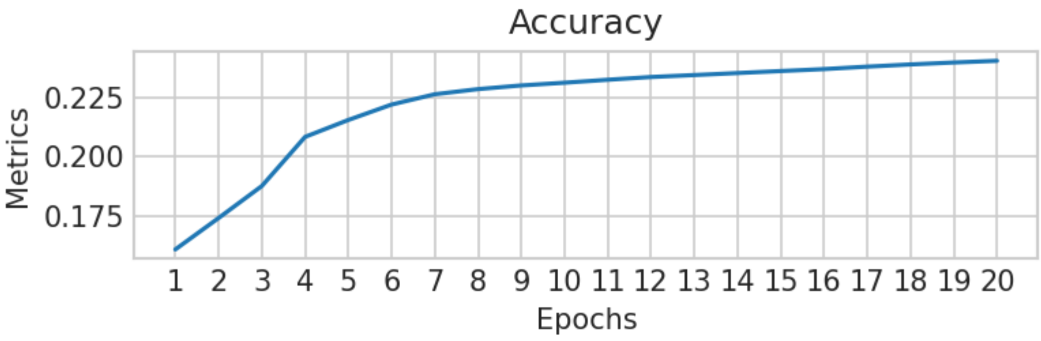
```
Prediction:  tf.Tensor(
[[0.5080564 ]
 [0.5650285 ]
 [0.71932817]
 ...
 [0.7648283 ]
 [0.90315175]
 [0.07082383]], shape=(2048, 1), dtype=float32)
Label:  tf.Tensor([1. 1. 0. ... 1. 1. 0.], shape=(2048,), dtype=float32)
```

Command took 2.61 seconds -- by Meeshawn_Marathe@comcast.com at 8/26/2023, 2:29:01 AM on Summer-Internship-gpu_MM

# Results & Discussion – Single-Tower Softmax





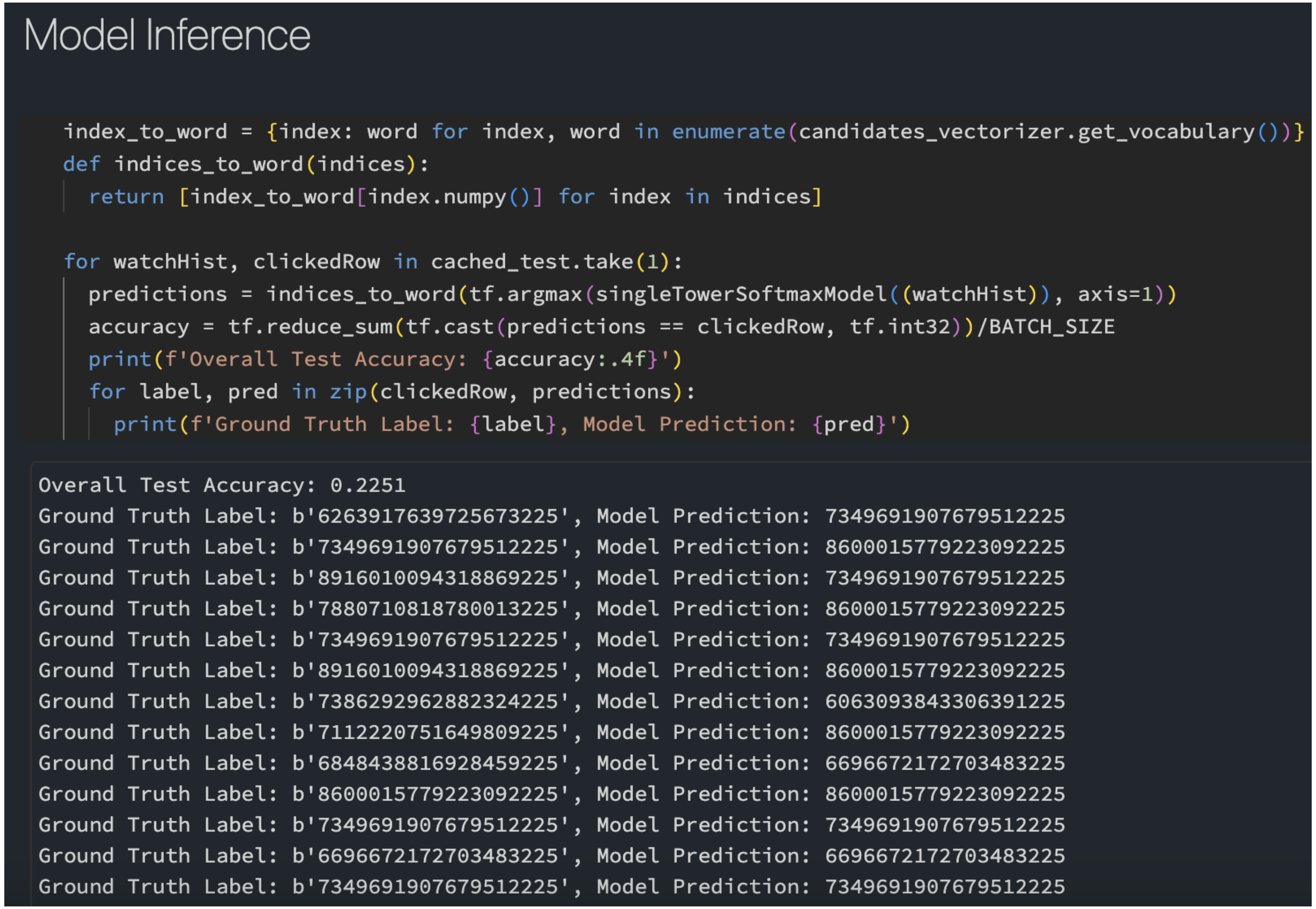


- **Batch Size:** 2048

- **Train Dataset:**
  - Period: 3rd - 4th July 2023
  - Size: ≈ 2 million
  - Training time: 35 min
  - Mean NDCG: 0.5229
  - Mean MRR: 0.3944
  - Accuracy: 0.2409

- **Test Dataset:**
  - Period: 5th July 2023
  - Size: ≈ 0.851 million
  - Mean NDCG: 0.5251
  - Mean MRR: 0.3970
  - Accuracy: 0.2428

# Results & Discussion – Single-Tower Softmax

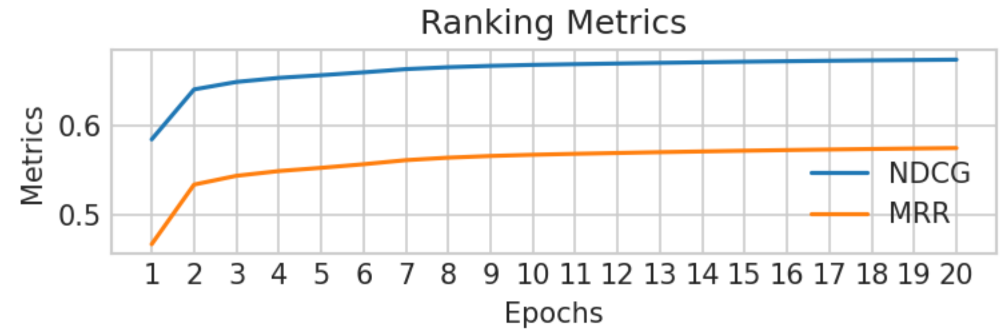


## Model Inference

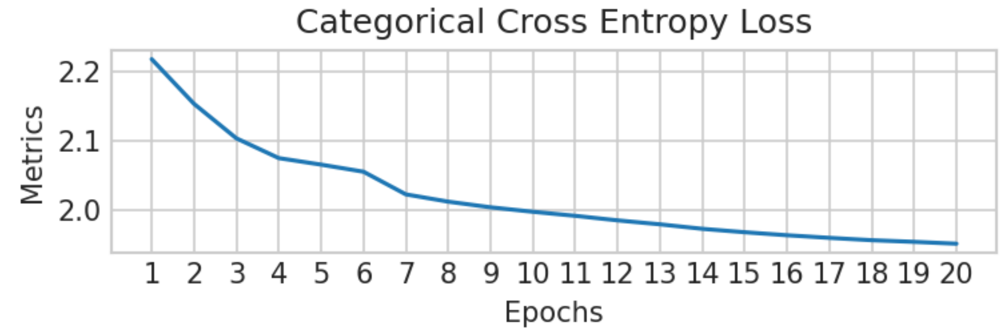
```python
index_to_word = {index: word for index, word in enumerate(candidates_vectorizer.get_vocabulary())}
def indices_to_word(indices):
  return [index_to_word[index.numpy()] for index in indices]

for watchHist, clickedRow in cached_test.take(1):
  predictions = indices_to_word(tf.argmax(singleTowerSoftmaxModel((watchHist)), axis=1))
  accuracy = tf.reduce_sum(tf.cast(predictions == clickedRow, tf.int32))/BATCH_SIZE
  print(f'Overall Test Accuracy: {accuracy:.4f}')
  for label, pred in zip(clickedRow, predictions):
    print(f'Ground Truth Label: {label}, Model Prediction: {pred}')
```

```
Overall Test Accuracy: 0.2251
Ground Truth Label: b'6263917639725673225', Model Prediction: 7349691907679512225
Ground Truth Label: b'7349691907679512225', Model Prediction: 8600015779223092225
Ground Truth Label: b'8916010094318869225', Model Prediction: 7349691907679512225
Ground Truth Label: b'7880710818780013225', Model Prediction: 8600015779223092225
Ground Truth Label: b'7349691907679512225', Model Prediction: 7349691907679512225
Ground Truth Label: b'8916010094318869225', Model Prediction: 8600015779223092225
Ground Truth Label: b'7386292962882324225', Model Prediction: 6063093843306391225
Ground Truth Label: b'7112220751649809225', Model Prediction: 8600015779223092225
Ground Truth Label: b'6848438816928459225', Model Prediction: 6696672172703483225
Ground Truth Label: b'8600015779223092225', Model Prediction: 8600015779223092225
Ground Truth Label: b'7349691907679512225', Model Prediction: 7349691907679512225
Ground Truth Label: b'6696672172703483225', Model Prediction: 6696672172703483225
Ground Truth Label: b'7349691907679512225', Model Prediction: 7349691907679512225
```

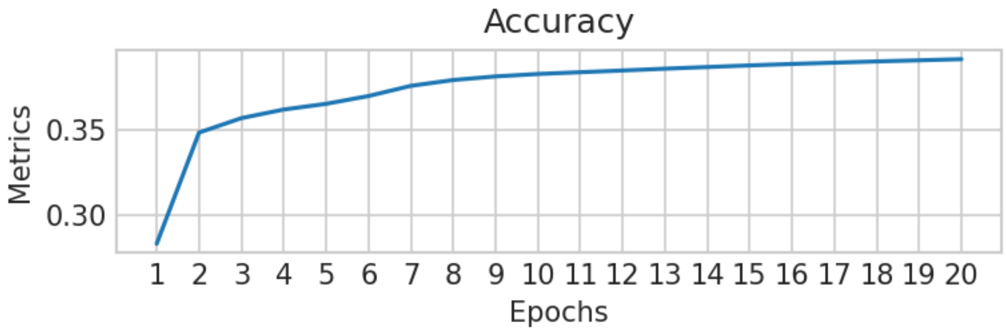
# Results & Discussion – Two-Tower Softmax



Ranking Metrics



Categorical Cross Entropy Loss



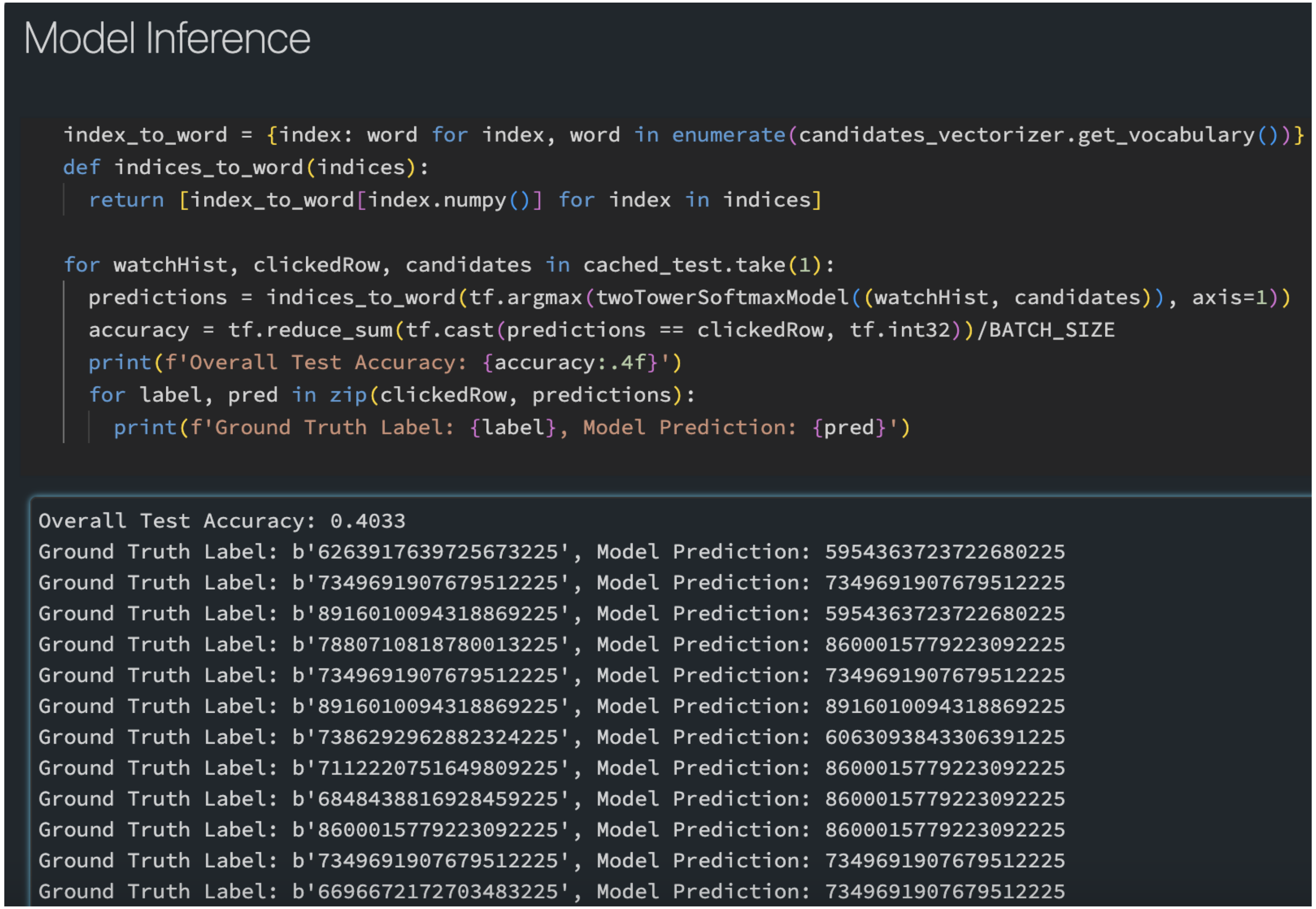
Accuracy

- **Batch Size:** 2048

- **Train Dataset:**
  - Period: 3rd - 4th July 2023
  - Size: ≈ 2 million
  - Training time: ≈ 35 min
  - Mean NDCG: 0.6751
  - Mean MRR: 0.5755
  - Accuracy: 0.3919

- **Test Dataset:**
  - Period: 5th July 2023
  - Size: ≈ 0.851 million
  - Mean NDCG: 0.6812
  - Mean MRR: 0.5833
  - Accuracy: 0.4020

# Results & Discussion – Two-Tower Softmax



## Model Inference

```python
index_to_word = {index: word for index, word in enumerate(candidates_vectorizer.get_vocabulary())}
def indices_to_word(indices):
  return [index_to_word[index.numpy()] for index in indices]

for watchHist, clickedRow, candidates in cached_test.take(1):
  predictions = indices_to_word(tf.argmax(twoTowerSoftmaxModel((watchHist, candidates)), axis=1))
  accuracy = tf.reduce_sum(tf.cast(predictions == clickedRow, tf.int32))/BATCH_SIZE
  print(f'Overall Test Accuracy: {accuracy:.4f}')
  for label, pred in zip(clickedRow, predictions):
    print(f'Ground Truth Label: {label}, Model Prediction: {pred}')
```

```
Overall Test Accuracy: 0.4033
Ground Truth Label: b'6263917639725673225', Model Prediction: 5954363723722680225
Ground Truth Label: b'7349691907679512225', Model Prediction: 7349691907679512225
Ground Truth Label: b'8916010094318869225', Model Prediction: 5954363723722680225
Ground Truth Label: b'7880710818780013225', Model Prediction: 8600015779223092225
Ground Truth Label: b'7349691907679512225', Model Prediction: 7349691907679512225
Ground Truth Label: b'8916010094318869225', Model Prediction: 8916010094318869225
Ground Truth Label: b'7386292962882324225', Model Prediction: 6063093843306391225
Ground Truth Label: b'7112220751649809225', Model Prediction: 8600015779223092225
Ground Truth Label: b'6848438816928459225', Model Prediction: 8600015779223092225
Ground Truth Label: b'8600015779223092225', Model Prediction: 8600015779223092225
Ground Truth Label: b'7349691907679512225', Model Prediction: 7349691907679512225
Ground Truth Label: b'6696672172703483225', Model Prediction: 7349691907679512225
```

# Performance Comparison with Existing Model

## Existing Model

Epochs: 40
Embedding Size: 100
Dropout: 0.2

```
Notebook exited: {
 "training_metrics": {
 "loss": 0.21750453114509583,
 "binary_accuracy": 0.8191505670547485,
 "val_loss": 0.2898108959197998,
 "val_binary_accuracy": 0.7796213626861572,
 "pop_mrr": 0.6567701101303101,
 "val_mrr": 0.6322186589241028,
 "pop_ndcg": 0.7404816746711731,
 "val_ndcg": 0.7120130062103271,
 "mrr_diff": -0.024551451206207275,
 "mrr_rel_diff": -0.03738212585449219,
 "ndcg_diff": -0.028468668460845947,
 "ndcg_rel_diff": -0.038446128368377686
 }
```

## Current Models

Epochs: 20
Embedding Size: 64
Dropout: NIL

**Two-Tower Full Softmax Model**

- **Train Dataset:**
  - Mean NDCG: 0.6751
  - Mean MRR: 0.5755
  - Accuracy: 0.3919
- **Test Dataset:**
  - Mean NDCG: 0.6812
  - Mean MRR: 0.5833
  - Accuracy: 0.4020

**Two-Tower Binary Classification Model**

- **Train Dataset:** Accuracy = 0.8832
- **Test Dataset:** Accuracy = 0.8847

# Summary

**Tasks:**

- Implemented 2 data generation pipelines
- Implemented 3 different Deep NN Ranking architectures:
  - Two-Tower Binary Classification (Click/No Click)
  - Single-Tower Full Softmax
  - Two-Tower Full Softmax
- Utilized Tensorflow Recommenders library for modeling & training
- Implemented Ranking metrics from scratch
  - NDCG
  - MRR
- Literature Survey (Wide & Deep Learning for RecSys, MMOE, Neural CF …)

# Summary

**Model Results:**

- Adding a menu tower to the softmax model **improved the ranking metrics** for the **same complexity** of the dense layers and with **similar training time.**
- The trained Two-Tower Full Softmax model produced **comparable NDCG & MRR** metrics when compared to the existing model with **lower embedding size (64 < 100)** and **no dropout.**
- The trained model provides an **alternate benchmark to compare results** with, which is a pivotal objective of the project.
- NDCG & MRR are a better indication of the ranking task than the overall classification accuracy.
- Hyperparameter tuning (layer size, # layers, embedding size…) and inclusion of additional attributes (timestamp, pinned, transaction, location) was not explored and might yield better results.

# Project Challenges & Learnings

- **Dataset:**
  - Finalizing the dataset to match service logs with clicks. Joining the two datasets.
- **Text Preprocessing:**
  - Creation of dictionaries (adapting) for Text Vectorization/String LookUp is time consuming ($\approx$ 1 hour/dictionary) [Create once, save and load it every time]
- **Tensorflow:**
  - Tensorflow – CUDA compatibility issue [Force install v2.10.0]
  - Tensorflow Recommenders – Tensorflow Ranking compatibility:
    - Issues using tfr NDCG/MRR metrics [Implemented NDCG & MRR from scratch]
  - Tensorflow Recommenders/Ranking Library Installation modifies existing TF installation [Install Tensorflow at the end, after all other libraries are installed]

# Project Challenges & Learnings

- **Tensorflow (continued . . .):**
  - Reading and writing tensorflow records files [tf.train.Example, SerializeToString(), tf.io.TFRecordWriter()]
  - Creation of tensorflow dataset from tfrecords
  - Sampled Softmax:
    - Keras tensor object – Tensorflow tensor object compatibility issue
- **Databricks:**
  - Writing Tensorflow datasets, pickle/paraquet files to S3 locations [Write locally, then copy to S3]
  - Reading and writing data and model files to dbfs and temp locations [Create temp/local folders exclusively and then write to them]

# Future Directions

- **Data Generation:**
  - Longer training period data
  - Cross check with click time window matched data
- **Existing Model Performance:**
  - Sampled-Softmax for reducing training time
  - Hyperparameter Tuning for optimal model parameters
  - Dropout & Regularization
- **Better Models:**
  - Cross-layer interaction for modeling complex interactions
  - Adding more context (timestamp, pinned, transaction,…)
  - Position Bias modeling and removal
  - Sequential models based on time-sequence of watch history

# Q/A

meeshawn@umich.edu

www.linkedin.com/in/meeshawnmarathe