# Deep Learning

## Deep Recommender System - Lab

**U Kang**
**Seoul National University**

# In This Lecture

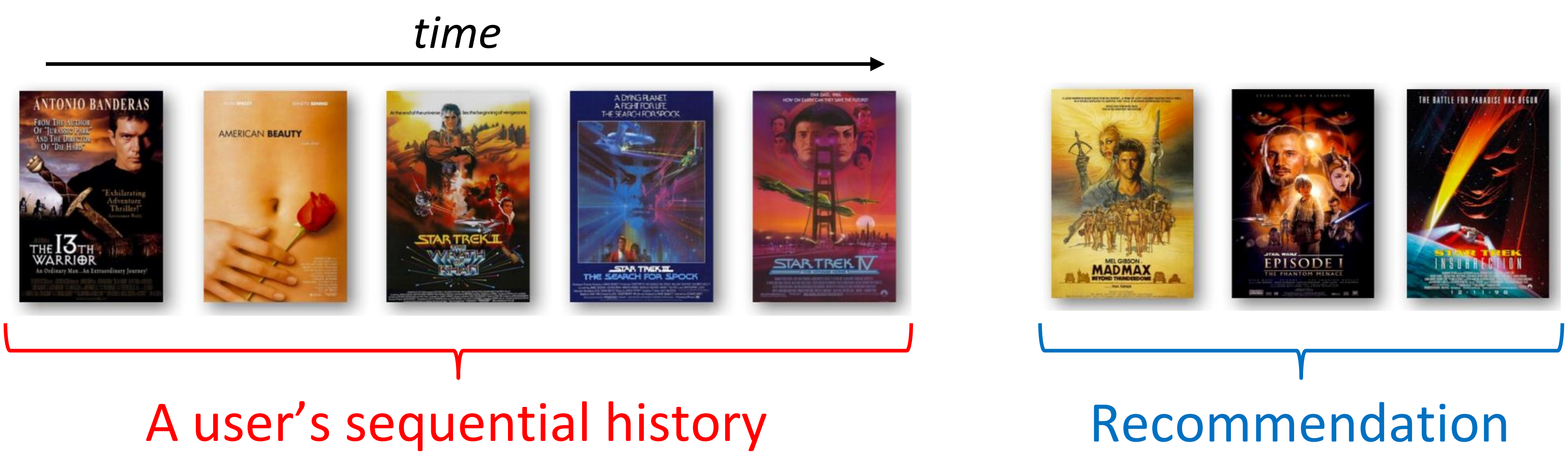
- RNN based Recommender System
  - Sequential recommendation
  - Data preparation
  - Model implementation
  - Model training / evaluation

# Outline

- ☐ **Sequential Recommendation**
- ☐ Data Preparation
- ☐ Model Implementation
- ☐ Model Training / Quantitative evaluation
- ☐ Qualitative Evaluation

# Sequential Recommendation (1)

- ***Given***
  - Users' sequential history (buy, watch, etc.)
- ***Goal***
  - Predict items that maximize her/his future needs

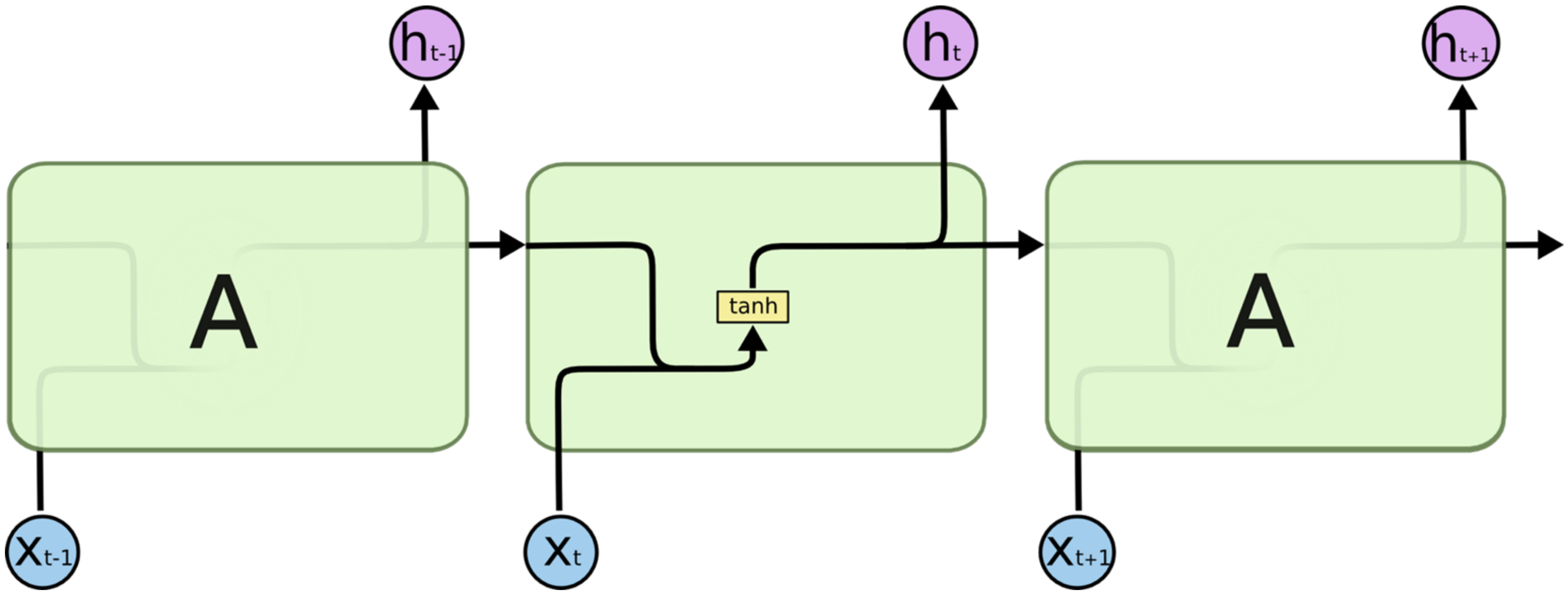


# Sequential Recommendation (2)

- A user's past interaction sequence is significant information
- Also have to consider personal preference

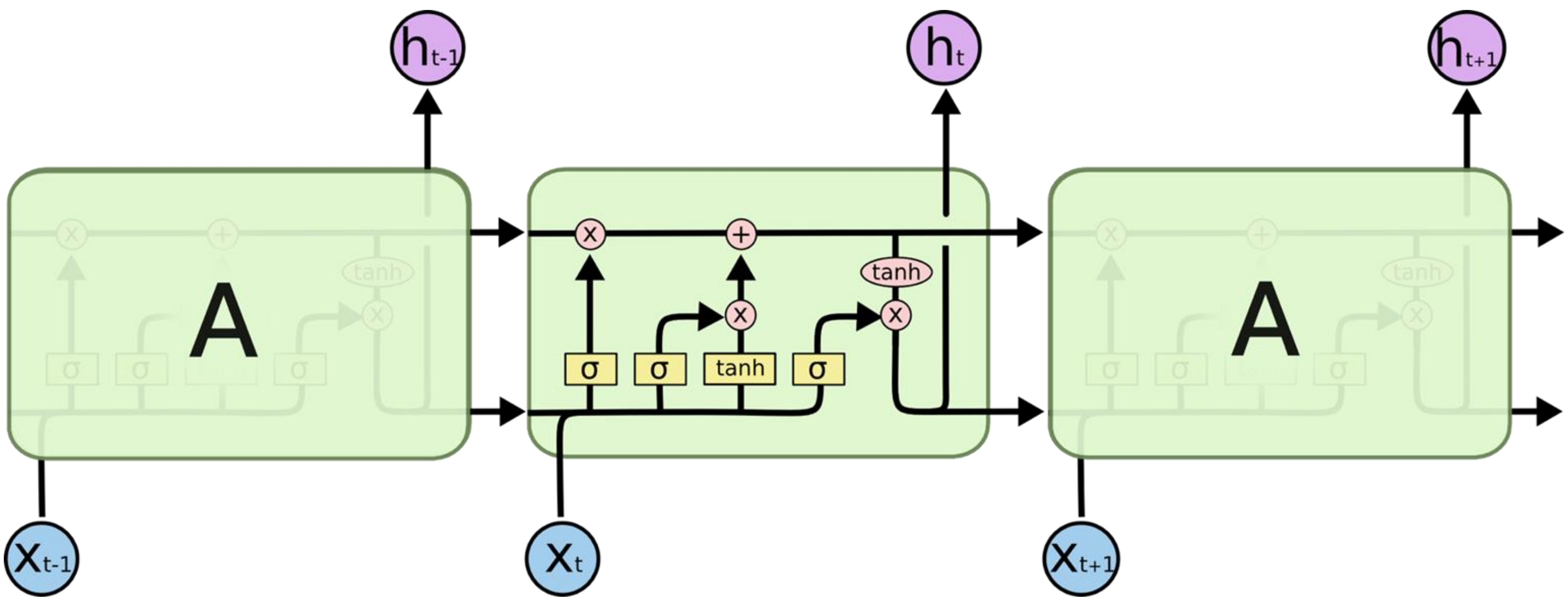
# Recurrent Neural Network (RNN)

- A deep learning structure for sequential data
- Contains a **cell**, which is a repeated structure
- Stores and passes **states** through a sequence

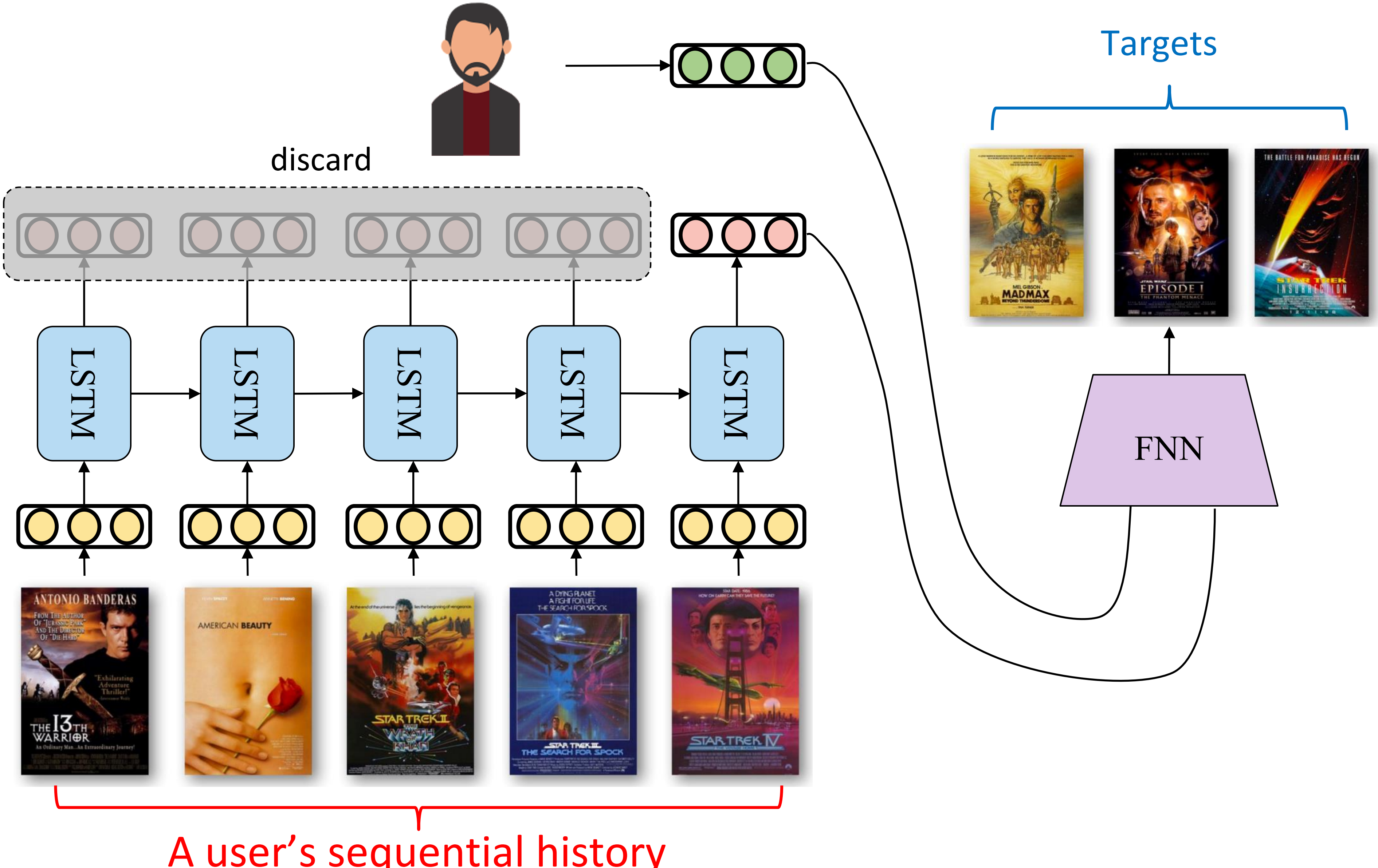


# Long Short-term Memory (LSTM)

- An advanced RNN structure
- It avoids the long-term dependency problem



# Architecture



# Outline

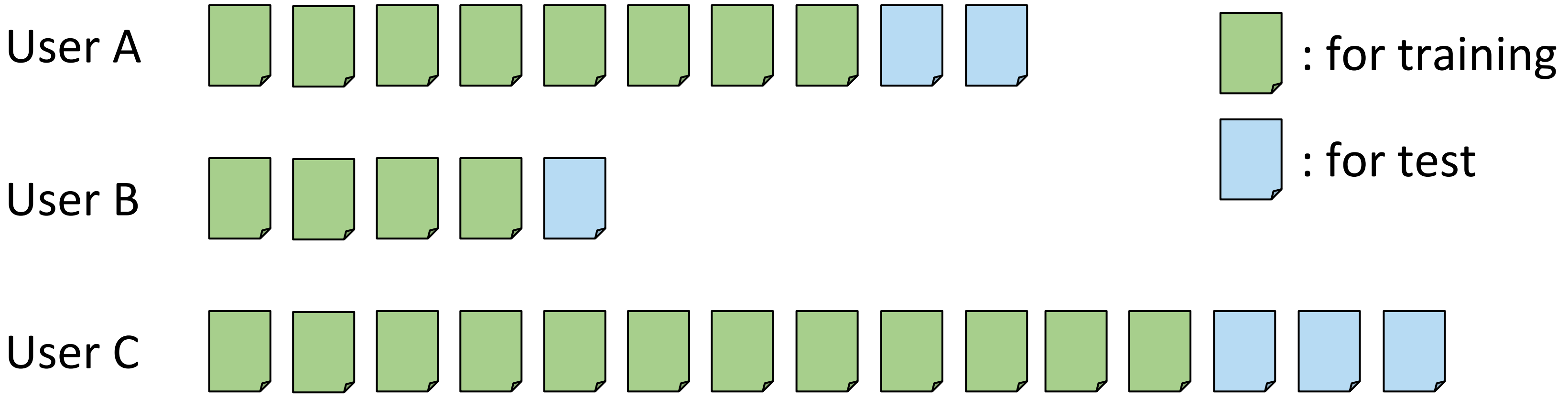
- ☑ Sequential Recommendation
- ☐ **Data Preparation**
- ☐ Model Implementation
- ☐ Model Training / Quantitative evaluation
- ☐ Qualitative Evaluation

# Dataset

- We use one of the most famous datasets in recommendation community: **MovieLens-1M**
  - Number of interactions: 1,000,209
  - Number of users: 6,040
  - Number of items: 3,952
  - Users gave ratings between 1 and 5 to items
  - Each user has at least 20 ratings
  - Logs between 1997/09/19 ~ 1998/04/22

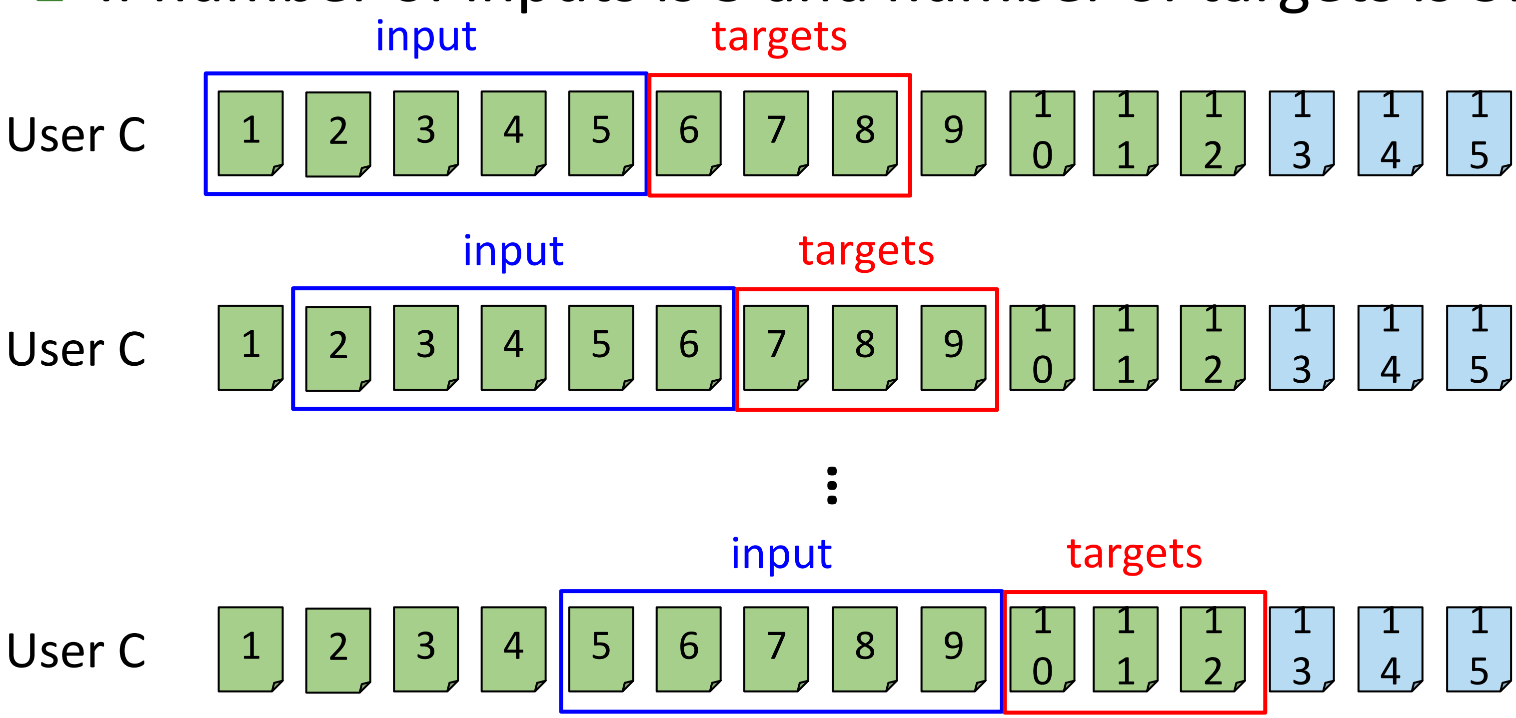
# Data Preparation (1)

- Let’s define training/test data
  - For each user, we use the first 80% of interactions as a training set
  - The remaining 20% of interactions are used as a test set



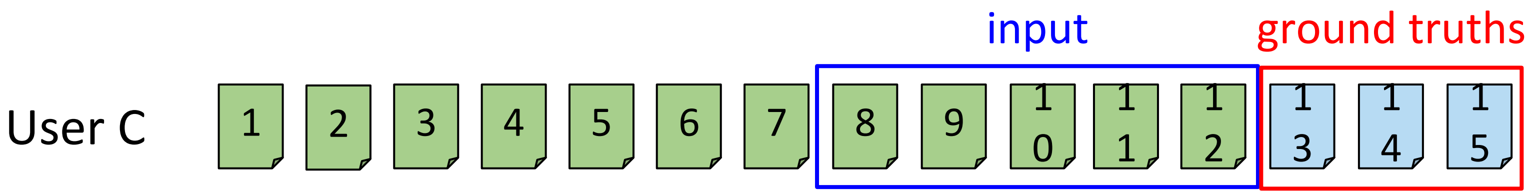
# Data Preparation (2)

- For each user, we get multiple training instances using a fixed size of window
  - If number of inputs is 5 and number of targets is 3:



# Data Preparation (3)

- A model is trained to predict the target items given input items
- When testing the model, we feed the last 5 items to the model and predict items that a user will interact with
- Then compare the predicted items with ground truths



# Reading Data File (1)

- Set the data path and split ratio
  - “ratings.dat” contains interaction logs

```
ratio = 80

in_path = './data/ml-1m-raw/'
ratings_file = in_path + 'ratings.dat'
```

# Reading Data File (2)

- Read data file
  - Format of “ratings.dat”
    - user_id::item_id::rating::time_stamp

```
# Load the input file in memory
raw = []
with open(ratings_file, 'r') as f_read:
    for line in f_read.readlines():
        line_list = line.split('::')
        raw.append(line_list)
```

# Data Analysis (1)

- Let's analyze the dataset
- **User** skewness
  - X-axis: number of interactions
  - Y-axis: number of **users**
- **Item** skewness
  - X-axis: number of interactions
  - Y-axis: number of **items**

# Data Analysis (2)

- Import numpy and pyplot

```
import numpy as np
import matplotlib.pyplot as plt
```

- Define the plot size

```
plt.rcParams["figure.figsize"] = (15,4)
```

# Data Analysis (3)

- Define user plot

```
raw = np.array(raw, dtype=int)
user_freq = np.bincount(raw[:, 0]) # [user1's freq, user2's freq, ..., usern's freq]
user_freq = [i for i in user_freq if i>0] # exclude dummy users
user_freq = np.bincount(user_freq)
user_x_axis = np.array(range(len(user_freq)))
print(f'users` max freq: {len(user_freq)-1}')
```

- Define item plot

```
item_freq = np.bincount(raw[:, 1]) #[item1's freq, item2's freq, ..., itemm's freq]
item_freq = [i for i in item_freq if i>0] # exclude dummy items
item_freq = np.bincount(item_freq)
item_x_axis = np.array(range(len(item_freq)))
print(f'items` max freq: {len(item_freq)-1}')
```

# Data Analysis (4)

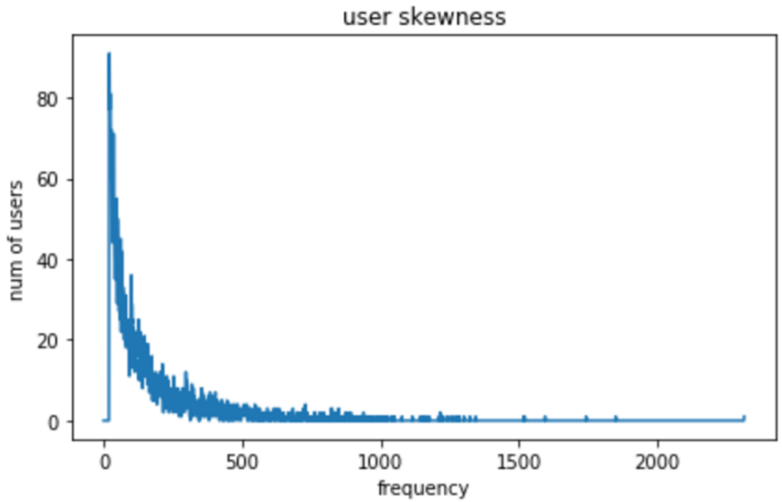
- Draw the plots

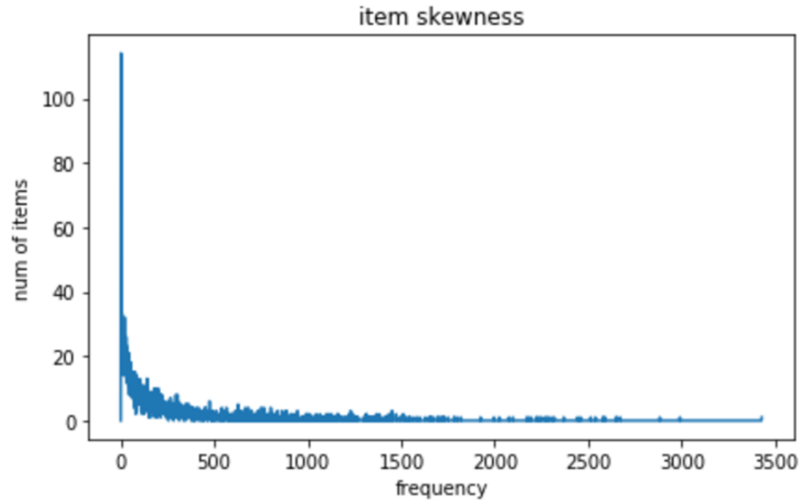
```
fig, axs = plt.subplots(1, 2)
axs[0].plot(user_x_axis, user_freq)
axs[0].set_title('user skewness')
axs[0].set_xlabel('frequency')
axs[0].set_ylabel('num of users')
axs[1].plot(item_x_axis, item_freq)
axs[1].set_title('item skewness')
axs[1].set_xlabel('frequency')
axs[1].set_ylabel('num of items')
plt.show()
```

# Data Analysis (5)

- The dataset is extremely skewed, which makes it difficult to make a personalized recommendation
  - A model will have a low loss even if it simply recommends the popular items to users

```
users` max freq: 2314
items` max freq: 3428
```



# Data Sorting

- Sort interactions by (user_id, timestamp) since we will split the data into training/test set based on each user's sequence

```
raw_sorted = np.array(sorted(raw, key=lambda x: (x[0], x[3])))
print(f'num of interactions: {len(raw_sorted)}')
```

```
num of interactions: 1000209
```

# Assign New IDs

- We need new ids that start from 0

```
user_ids = list()
item_ids = list()
user_map = dict() # raw -> new
item_map = dict() # raw -> new

user_ids = np.unique(raw[:, 0])
item_ids = np.unique(raw[:, 1])

user_map = {v: i for (i, v) in enumerate(user_ids)}
item_map = {v: i for (i, v) in enumerate(item_ids)}

new_sorted = [[user_map[u], item_map[i]] for (u, i)
              in zip(raw_sorted[:, 0], raw_sorted[:, 1])] # new array
```

# Split Dataset

- Split the dataset into training/test sets

```
trn_list = list()
test_list = list()

index = new_sorted[0][0]
buf = [] # Sequence for each user
for i in range(len(new_sorted)):
    if index == new_sorted[i][0]:
        buf.append(new_sorted[i])
        continue
    split_i = round(len(buf) * float(ratio/100))
    trn = buf[:split_i]
    test = buf[split_i:]
    for line in trn:
        trn_list.append([line[0], line[1]])
    for line in test:
        test_list.append([line[0], line[1]])
    index = new_sorted[i][0]
    buf = [new_sorted[i]]
split_i = int(len(buf) * float(ratio / 100))
trn = buf[:split_i]
test = buf[split_i:]
for line in trn:
    trn_list.append([line[0], line[1]])
for line in test:
    test_list.append([line[0], line[1]])
```

# Side Information

- Construct dictionary of items' side information
  - "movies.dat" contains title/genres of every item
  - Format: item_id::title::genres
- Note that we only use it for evaluating the model, not for training

```
movies_file = in_path + 'movies.dat'
meta_dict = dict()

with open(movies_file, 'r',  encoding='ISO-8859-1') as f_read:
    for line in f_read.readlines():
        line_list = line.split('::')
        raw_id = int(line_list[0].strip())
        try:
            new_id = item_map[raw_id]
        except KeyError:
            continue
        meta_dict[new_id] = [line_list[1].strip(), line_list[2].strip()]
```

# Data instances (1)

- Data instance parameters
- We also need negative samples to train a model

```
feed_len = 5
target_len = 3
neg_samples = 20
```

# Data instances (2)

- Define two functions to generate training instances

```
def generate_training_instances(user_ids, items, indices, max_len):
    for i in range(len(indices)):
        # set start_idx and stop_idx for each user
        start_idx = indices[i]
        if i >= len(indices) - 1:
            stop_idx =None
        else:
            stop_idx = indices[i + 1]

        for seq in sliding_window(items[start_idx:stop_idx], max_len):
            yield (user_ids[i], seq)

def sliding_window(tensor, window_size):
    for i in range(len(tensor)):
        if i + window_size > len(tensor):
            break
        else:
            yield tensor[i:i+window_size]
```

# Data instances (3)

- Prepare placeholders for training instances
- We also prepare “test input instances ”

```
trn_list = np.array(trn_list)
test_list = np.array(test_list)

# user_ids: unique user ids
# indices: first index of each user
# counts: number of each user's interactions
user_ids, indices, counts = np.unique(trn_list[:, 0],
                                      return_index=True, return_counts=True)
items = trn_list[:, 1]

max_len = feed_len + target_len
num_sequences = sum([c - max_len + 1 if c >= max_len else 1 for c in counts])
num_users = len(user_ids)

trn_feed_sequences = np.zeros(shape=(num_sequences, feed_len), dtype=np.int32)
trn_positive_targets = np.zeros(shape=(num_sequences, target_len), dtype=np.int32)
trn_users = np.empty(num_sequences, dtype=np.int32)
test_feed_sequences = np.zeros(shape=(num_users, feed_len), dtype=np.int32)
test_users = np.empty(num_users, dtype=np.int32)
```

# Data instances (4)

- Generate training instances
- We also generate “test input instances”

```
for i, (uid, item_seq) in enumerate(
    generate_training_instances(user_ids, items, indices, max_len)):
    trn_feed_sequences[i][:] = item_seq[:feed_len]
    trn_positive_targets[i][:] = item_seq[-target_len:]
    trn_users[i] = uid
    test_feed_sequences[uid][:] = item_seq[-feed_len:]
    test_users[uid] = uid
```

# Data instances (5)

- Define a function to generate test instances

```
def generate_test_instances(user_ids, items, indices):
        for i in range(len(indices)):
            # set start_idx and stop_idx for each user
            start_idx = indices[i]
            if i >= len(indices) - 1:
                stop_idx =None
            else:
                stop_idx = indices[i + 1]

            yield(user_ids[i], items[start_idx:stop_idx])
```

# Data instances (6)

- Generate test instances (only ground truths)
- Test input instances were generated while generating training instances

```
user_ids, indices, counts = np.unique(test_list[:, 0], return_index=True, return_counts=True)

items = test_list[:, 1]
test_targets = []

for uid, item_seq in generate_test_instances(user_ids, items, indices):
    test_targets.append(item_seq)
```

# Data instances (7)

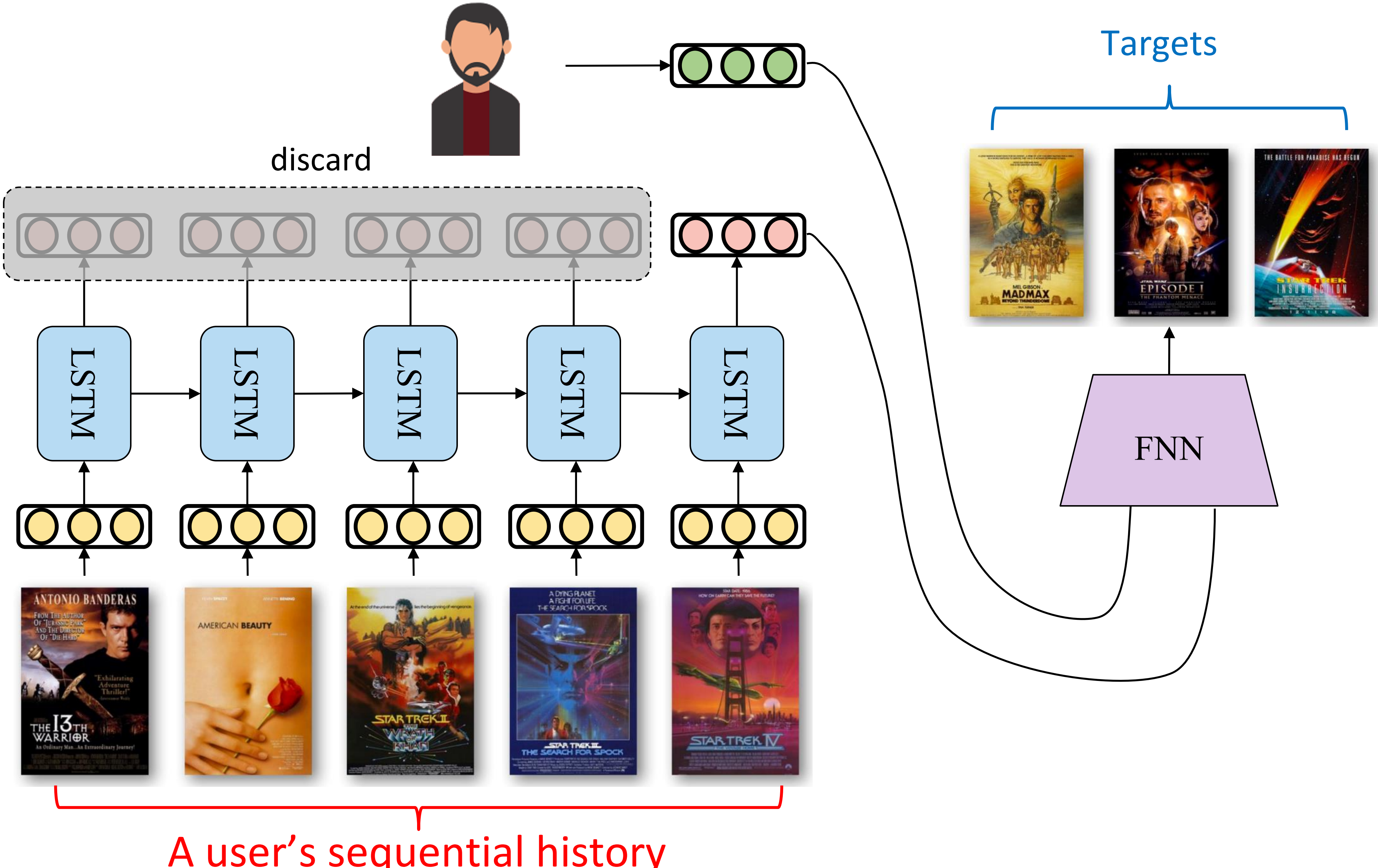
- We need negative samples to learn meaningful item embedding vectors.
- Otherwise, dissimilar items will have similar embedding vectors
- Randomly select items for negative samples

```
num_users = len(user_map)
num_items = len(item_map)
trn_negative_targets = np.random.randint(num_items,
                                         size=(num_sequences, target_len * neg_samples))
```

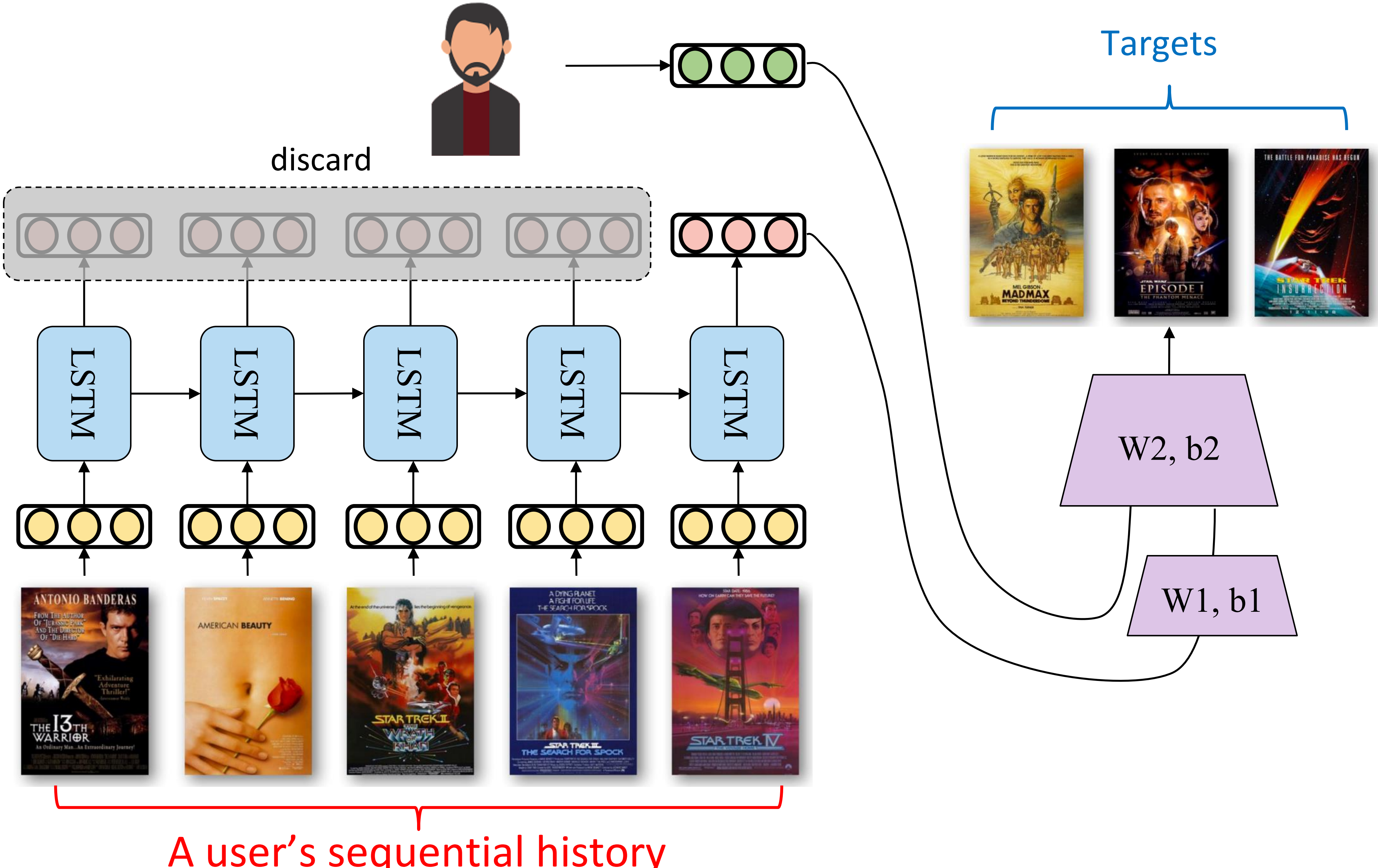
# Outline

- ☑ Sequential Recommendation
- ☑ Data Preparation
- ☐ **Model Implementation**
- ☐ Model Training / Quantitative evaluation
- ☐ Qualitative Evaluation

# Architecture (1)



# Architecture (2)



# What You Need to Know

- RNN based Recommender System
  - Sequential recommendation
  - Data preparation
  - Model implementation
  - Model training / Evaluation

# Questions?