Training and Deploying Multi-Stage Recommender Systems with Merlin

ACM RecSys 2022 Conference

Recommender Systems

Personalization Engine of Online Services

DIGITAL CONTENT

4.3B Watch Videos Online

E-COMMERCE

3.7B Shop Online

SOCIAL MEDIA

4.3B Active Users

DIGITAL ADVERTISING

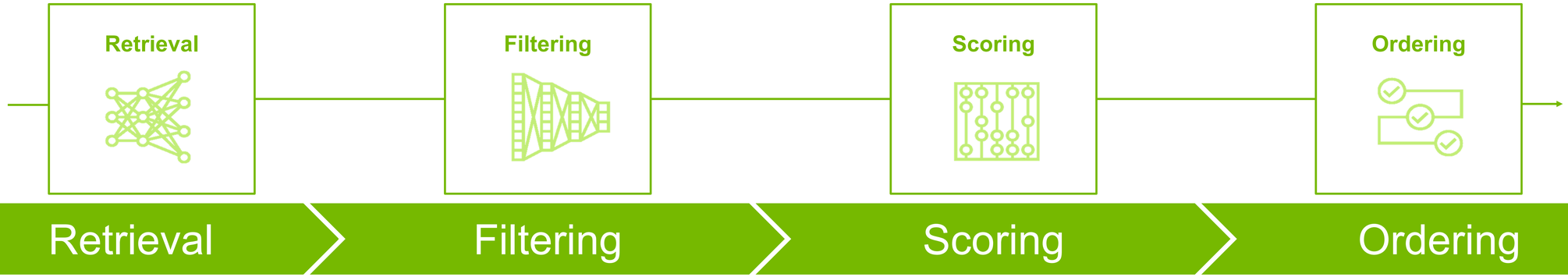
4.7B Internet Users

"Already, 35 percent of what consumers purchase on Amazon and 75 percent of what they watch on Netflix come from product recommendations based on such algorithms."

Source: McKinsey

Building Recommender Systems End-To-End is complex and requires 4 stages

NVIDIA MERLIN



Generating ~1000-10000 candidates used for recommendations from all available items.

Scoring all available items is computential too expensive, therefore only a subset should be scored

Removing invalid candidates

For example, products out of stock

More advanced models scores the candidates and determines the best recommendations

Reordering the recommendations by other business logic

Examples for the 4 stage recommender systems

	Retrieval	Filtering	Scoring	Ordering
Music Discovery	Find similar songs based on nearest neighbour search	Remove tracks users listened before	Predict likelihood that a user will listen to a song	Trade-Off between score, similarity, BPM, etc
Social Media	Find new posts in user's network	Remove posts from blocked and muted users	Predict likelihood that a user will interact with it	Change order that adjust posts from different authors
Online Store	Find items which are usually co-purchased	Remove items which are out of stock	Predict likelihood that a user will purchase an item	Reorder items based on price points
Streaming Service	Find items based on different rows/shelves/topics	Remove items which are not available for user's country	Predict user's stream time per item	Organize recommendations to fit genre distributions

Merlin easily accelerate and scale recommender systems pipelines end-to-end

Merlin

End-To-End

Merlin supports all steps of recommender systems from ETL to deployment.

Easy-To-Use

Merlin's high-level API and end-to-end design enables complex workflows with only few lines of code.

Scale

Merlin can process >100TB datasets.

Speed

Merlin runs on single GPU, multi-GPU and multi-node GPU systems.

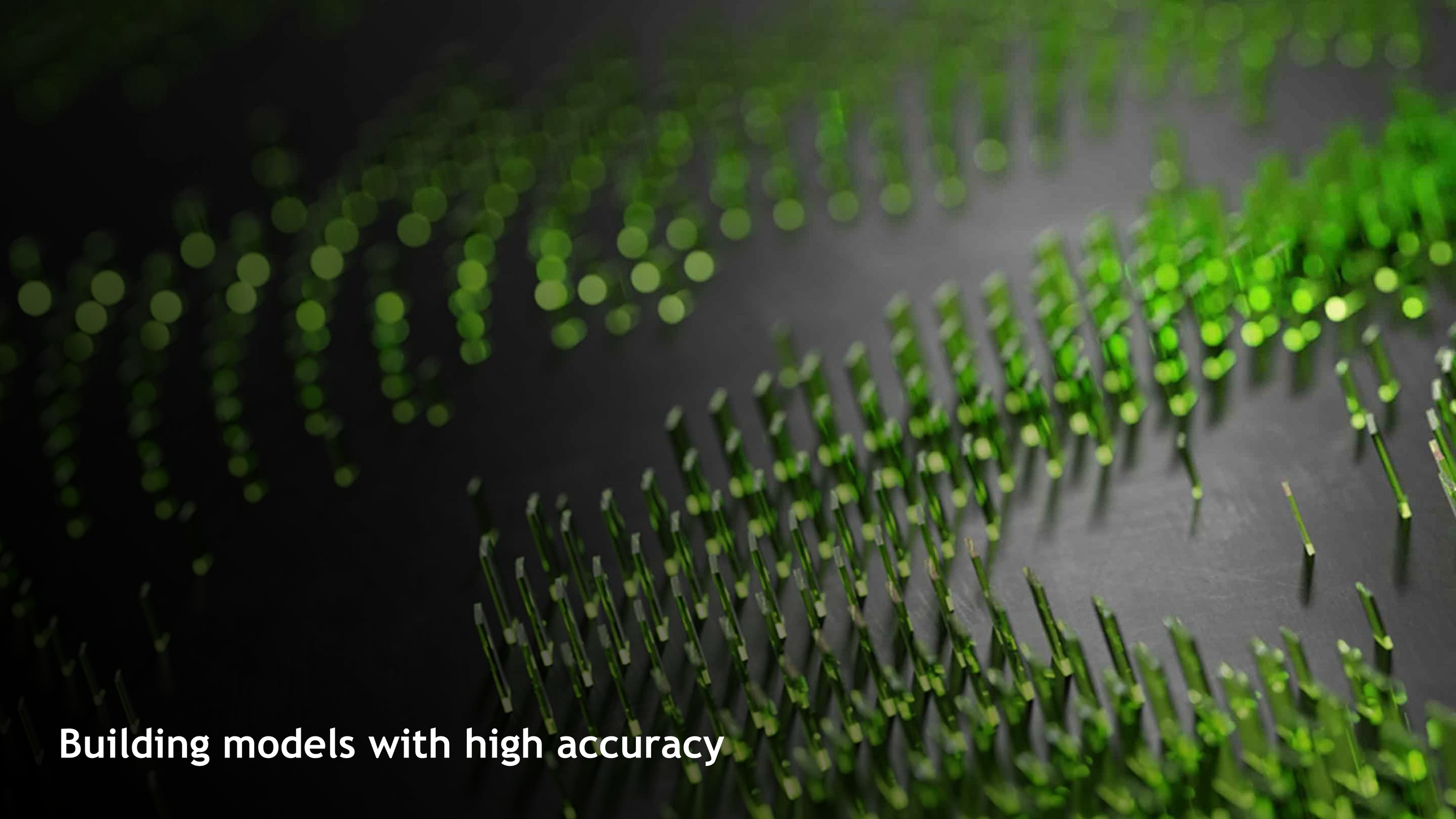
Merlin’s benefits at different steps of Recommender Systems

	Model Development & Evaluation	Deployment	Production (1)
Goal	Building models with high accuracy	Deploying new models into production systems	Supporting many recommender systems
Merlin Features	▪ Accelerating pipelines for fast experimentation cycle ▪ Close integration of ETL and model training & evaluation ▪ Implementations of common architectures, loss functions, negative sampling strategies, etc.	▪ Simple API to push to production systems ▪ Deployment of ETL and model as ensemble ▪ Many available components such as retrieval or filtering ▪ Scalable and accelerated components	▪ Standardize production workflow for all use cases ▪ Integration to other components for logging, feature storage, etc.

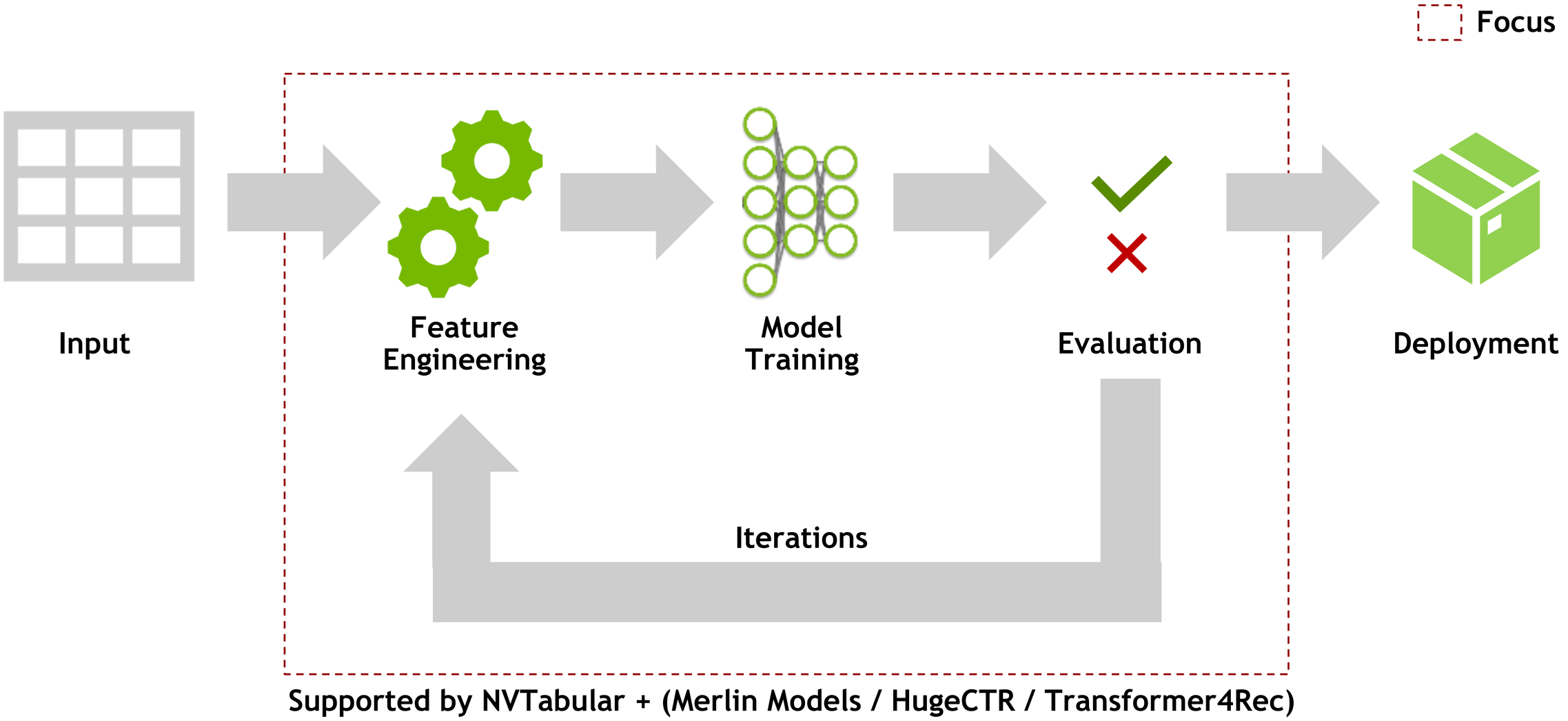
Hands-On Lab 1+2

Hands-On Lab 3+4

(1) Coming soon in Merlin v2.0


Building models with high accuracy

Merlin enables quick experimentation cycles to find high accuracy models



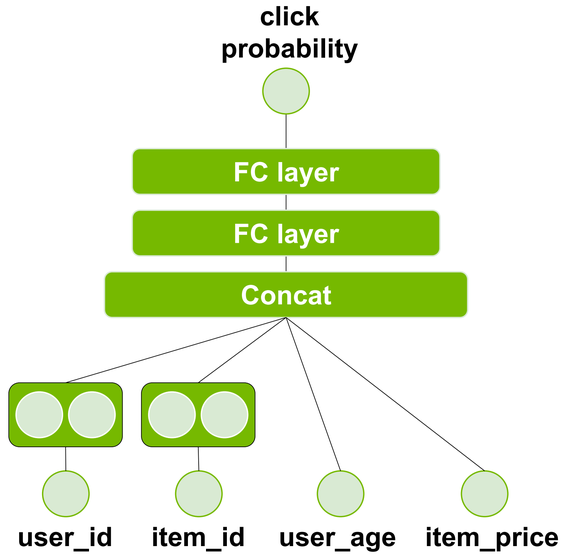
Review: Recommender System with Deep Learning

Example Dataset

	user_id	user_age	item_id	item_price	click
0	0	22	100	22	0
1	0	22	105	25	1
2	1	25	101	100	0
3	1	25	105	25	1
4	1	25	100	22	0
5	2	23	103	50	0
6	2	23	100	22	0
7	2	23	107	10	1
8	3	30	100	22	1
9	3	30	102	25	1
10	3	30	101	100	0
11	3	30	105	25	1

Predicting if a user will click a item

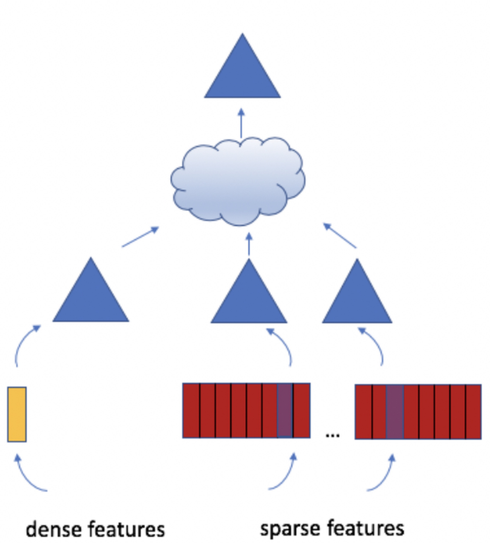
Example Architecture



- Categorical features (user_id, item_id) are fed through embedding layer
- Numerical features are normalized
- Embedding output and numerical features are concatenated and fed through fully-connected layers

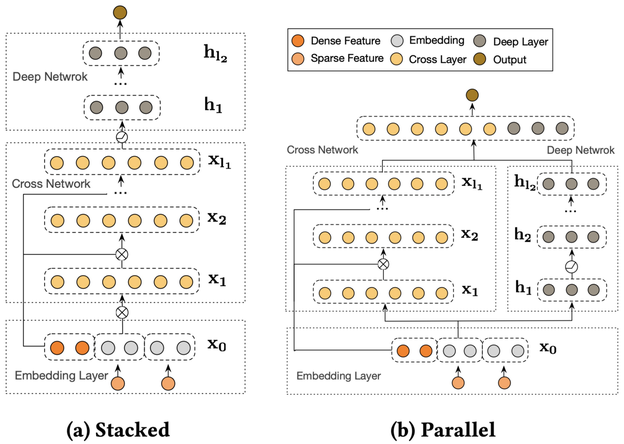
Review: Recommender System with Deep Learning

Facebook DLRM(1)



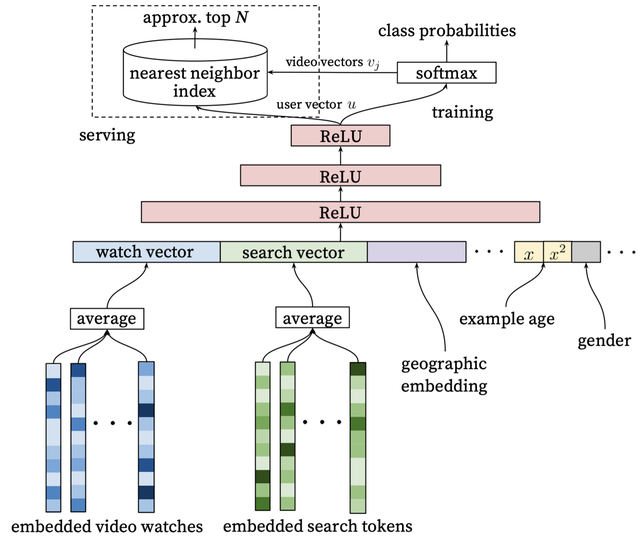
- Model inputs are both dense and sparse features
- Feature Interaction layer: dot product between all combinations of embedding vectors and processed dense features
- Parallel training setup - model and data parallelism

Google DCN(2)



- DCN v2 is an architecture for learning-to-rank: improves the original DCN model
- Leverages explicit & implicit feature interactions of the inputs via cross layers and DNNs
- Achives an healthier trade-off between model performance and latency via mixture of low-rank DCN

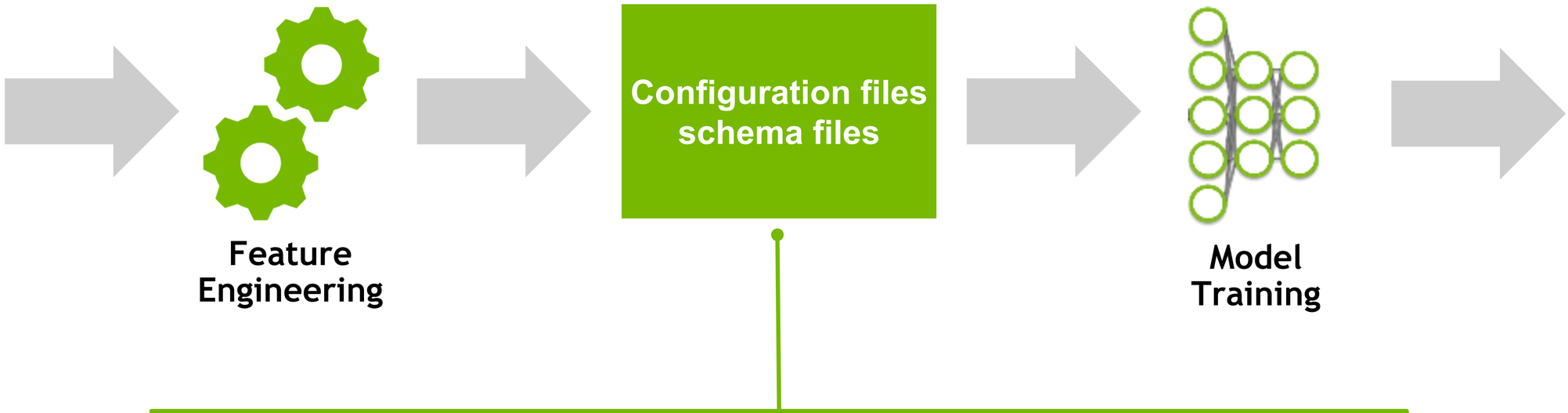
YouTube DNN(3)



- Two-stage Recommender Systems: candidate generation and ranking
- Candidate sampling strategy was used
- Original paper predicts `watch time` as target by modifying classic logistic regression

1) DLRM: Deep Learning Recommendation Model for Personalization and Recommendation Systems
2) DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems
3) YouTube DNN: Deep Neural Networks for YouTube Recommendations

Close integration between feature engineering and training simplifies workflow



- Model training requires information about the dataset, for example which columns are categorical and numerical, cardinality of categorical features, etc.
- Feature engineering step can collect the information and provide them in a defined schema file
- Model can be defined based on the provided information from the schema file
- Updating the feature engineering (for example adding new features) does not require to change the model training workflow

Merlin simplifies the dependency by providing a schema file which connects feature engineering and model training

Close integration between feature engineering and training simplifies workflow

Focus

Feature Engineering

NVTabular

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

Model

Merlin Models

High quality implementations from classical machine learning models to more advanced deep learning models in PyTorch and TensorFlow.

Transformers4Rec

Flexible and efficient library for sequential and session-based recommendations using Transformer architectures in PyTorch and TensorFlow.

HugeCTR

Highly efficient custom deep learning framework written in CUDA C++ to maximize throughput for CTR models and scale to multi-node systems.

Components can be used independently

NVTabular

NVTabular

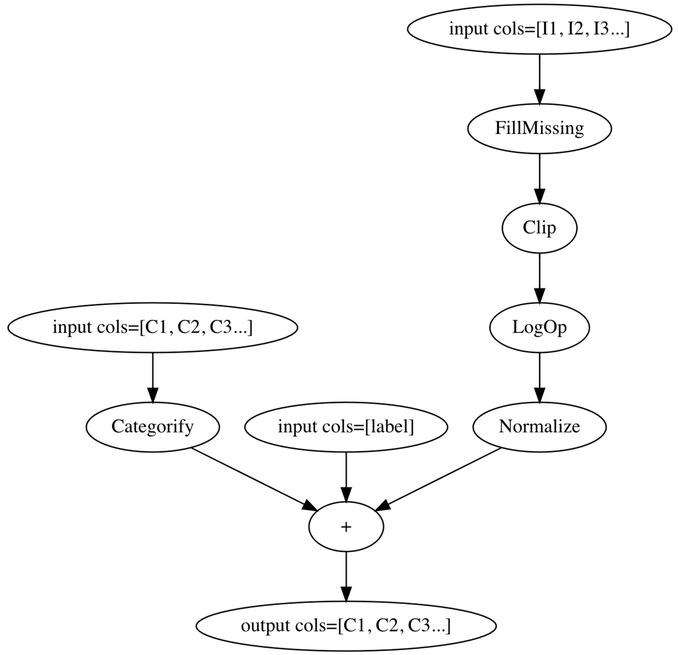
What it is:

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

What it's capable of:

- Pipelines can be executed on CPU or GPU, GPU acceleration shows speed-up to 10x.
- NVTabular scales to larger than memory datasets by streaming data from disk and transforming them in chunks.
- High level API creates complicated ETL workflows with only a few API calls - common operations are integrated into the library
- NVTabular is designed to be easy integrated with Merlin Models, HugeCTR, PyTorch and TensorFlow.

NVTabular



Visualization of feature engineering and preprocessing pipeline for Criteo Click Ads Prediction dataset

NVTabular provides operators for common ETL workflows

Continuous

- **Clip**: This operation clips continuous values so that they are within a min/max bound.
- **FillMissing**: This operation replaces missing values with a constant pre-defined value
- **LogOp**: This operator calculates the log of continuous columns.
- **Normalize**: This operation can be added to the workflow to standardize the features.
- **NormalizeMinMax**: This operation can be added to the workflow to standardize the features.
- **Median**: This operation calculates median of features.
- **FillMedian**: This operation replaces missing values with the median value for the column.
- **DifferenceLag**: Calculates the difference between two consecutive rows of the dataset.

Categorical

- **Categorify**: Categorify operation can be added to the workflow to transform categorical features into unique integer values.
- **HashBucket**: This op maps categorical columns to a contiguous integer range by first hashing the column then modulating by the number of buckets as indicated by num_buckets.
- **ColumnSimilarity**: Calculates the similarity between two columns using tf-idf, cosine or inner product as the distance metric.
- **JoinGroupby**: This operator groups the data by the given categorical feature(s) and calculates the desired statistics of requested continuous features.

Both

- **TargetEncoding**: Target encoding is a common feature-engineering technique for categorical columns in tabular datasets.
- **Dropna**: This operation detects missing values, and filters out rows with null values.
- **Filter**: Filters rows from the dataset. This works by taking a callable that takes a dataframe, and returns a dataframe with unwanted rows filtered out.
- **JoinExternal**: Join each dataset partition to an external table.
- **LambdaOp**: LambdaOp allows you to apply row level functions to a NVTabular workflow.

Merlin Models

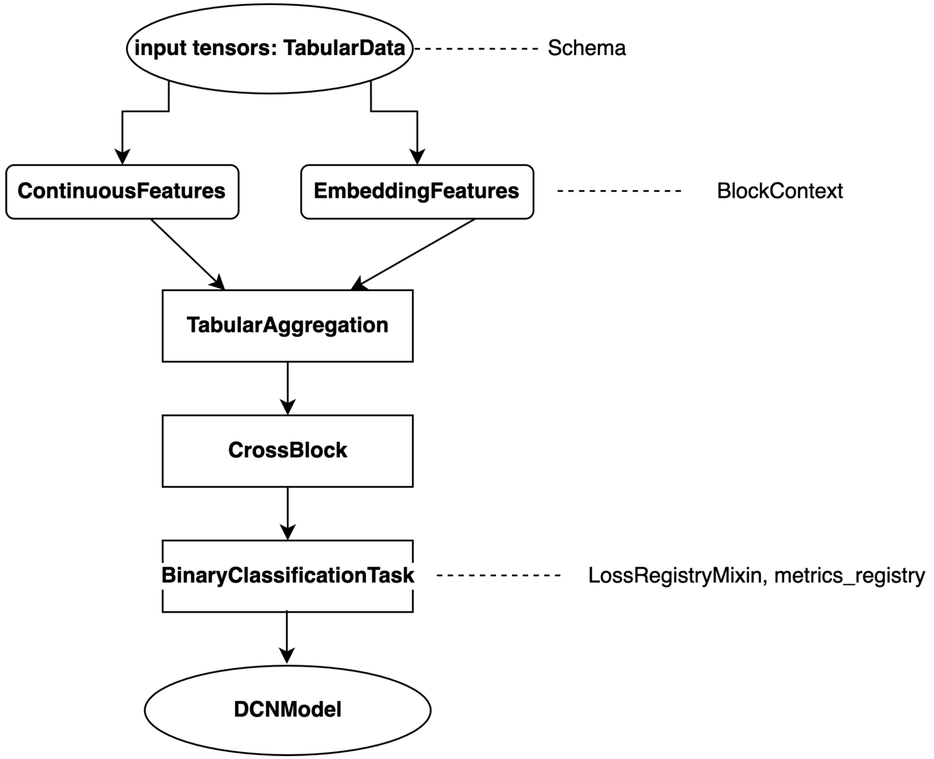
Merlin Models

What it is:

High quality implementations from classical machine learning models to more advanced deep learning models in TensorFlow and PyTorch.

What it's capable of:

- Unified API enables users to create models in TensorFlow or PyTorch (1) using same code
- Implementation of common architectures, loss functions and tasks (e.g. binary classification, multi-class classification, regression, next-item prediction, item retrieval)
- Flexible building blocks to design your own architecture based on common components
- GPU-optimized data-loading and model training
- Integration with NVTabular



Visualization of Merlin Models

(1) Coming until Merlin v2.0

Merlin Models provides a collection of building blocks for recommender systems

Classical Machine Learning:
- BPR-MF, ALS (Implicit)
- Trees ensembles based classification

…

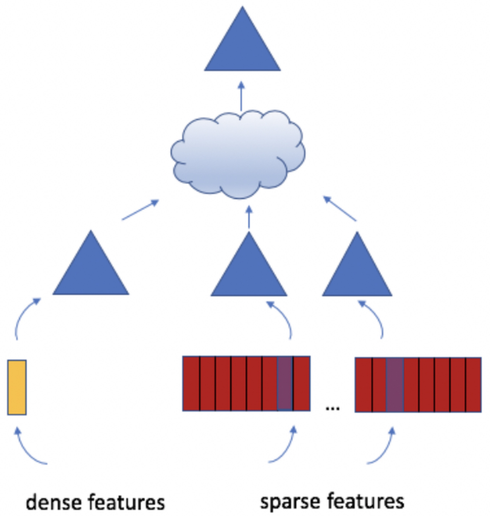
Deep Learning architectures:
- Matrix Factorization
- Neural Collaborative Filtering
- MLP
- DLRM
- DCN
- Two-Tower Retrieval
- YouTube DNN
- Mixture of Experts
- …

Loss functions:
- Binary Cross Entropy
- Categorical Cross Entropy
- BPR (Bayesian Personalized Ranking)
- TOP1
- Logistic
- Hinge
- Adaptive Hinge Loss
- …

Other Features:
- Optimized dataloader
- Multiple ranking metrics (e.g. Recall, Precision, NDCG, MRR, MAP)
- Multiple scalable negative sampling strategies (e.g. in-batch, cached cross-batch)
- Retrieval tasks
- Ranking tasks
- Multi-task learning

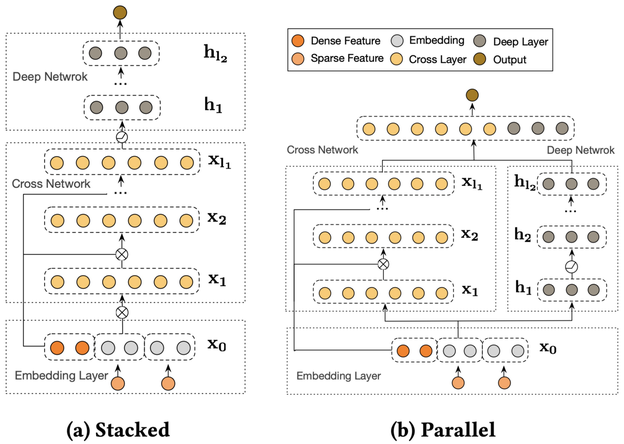
Examples to define architectures with Merlin Models in TensorFlow

Facebook DLRM




```
model = mm.DLRMModel(
    schema,
    embedding_dim=64,
    bottom_block=mm.MLPBlock([128, 64]),
    top_block=mm.MLPBlock([128, 64, 32]),
    prediction_tasks=mm.BinaryClassificationTask(
        target_column,
        metrics=[tf.keras.metrics.AUC()]
    )
)
```

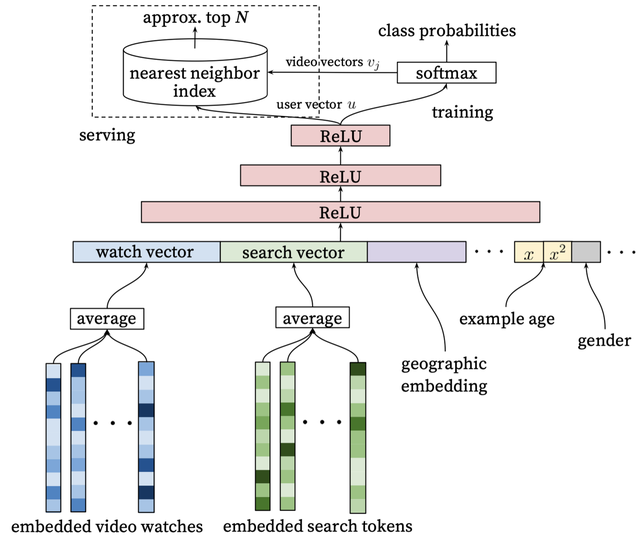
Google DCN



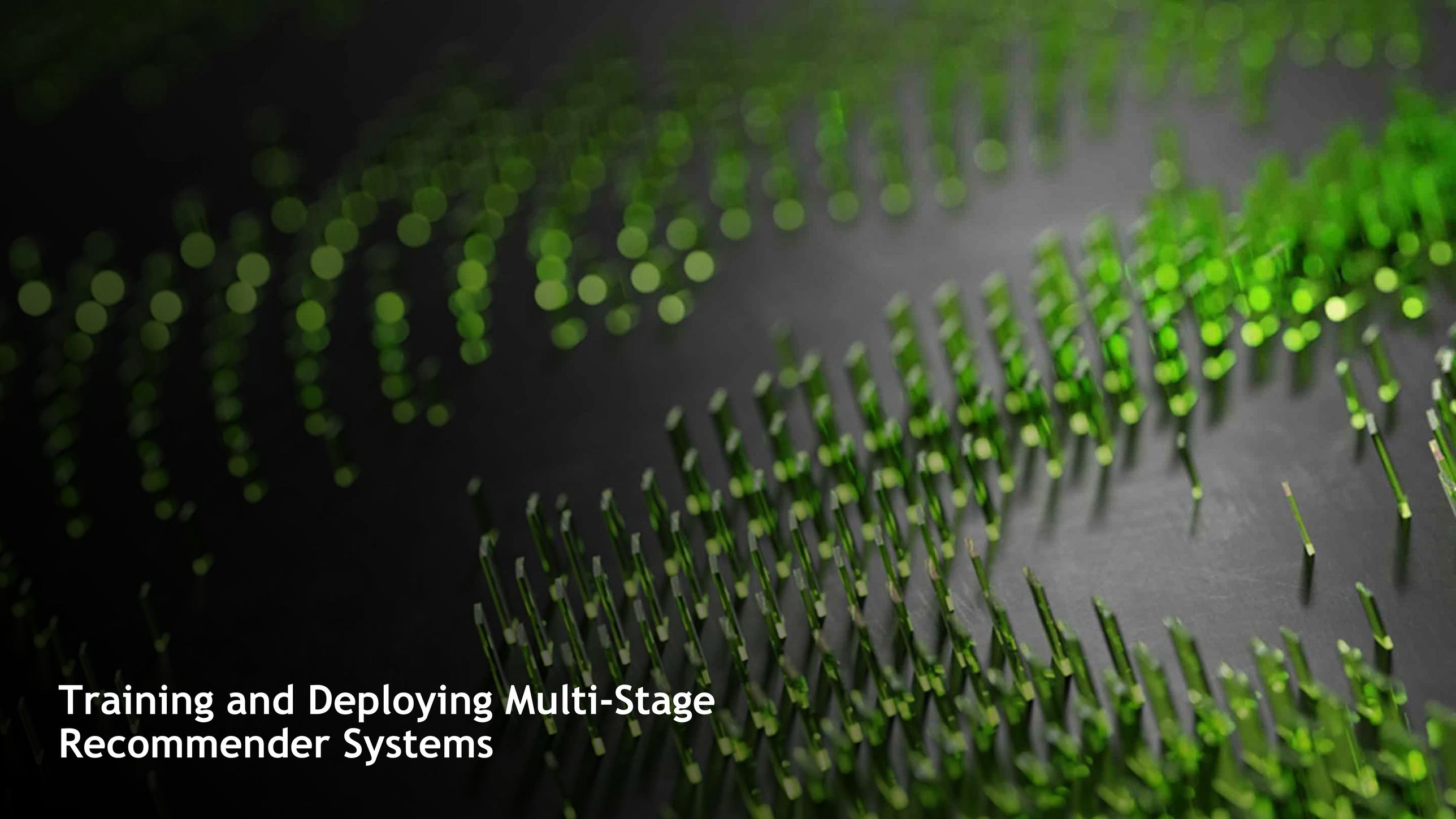
(a) Stacked (b) Parallel

```
model = mm.DCNModel(
    schema,
    depth=2,
    deep_block=mm.MLPBlock([128, 64]),
    prediction_tasks=mm.BinaryClassificationTask(
        target_column,
        metrics=[tf.keras.metrics.AUC()]
    )
)
```

YouTube DNN

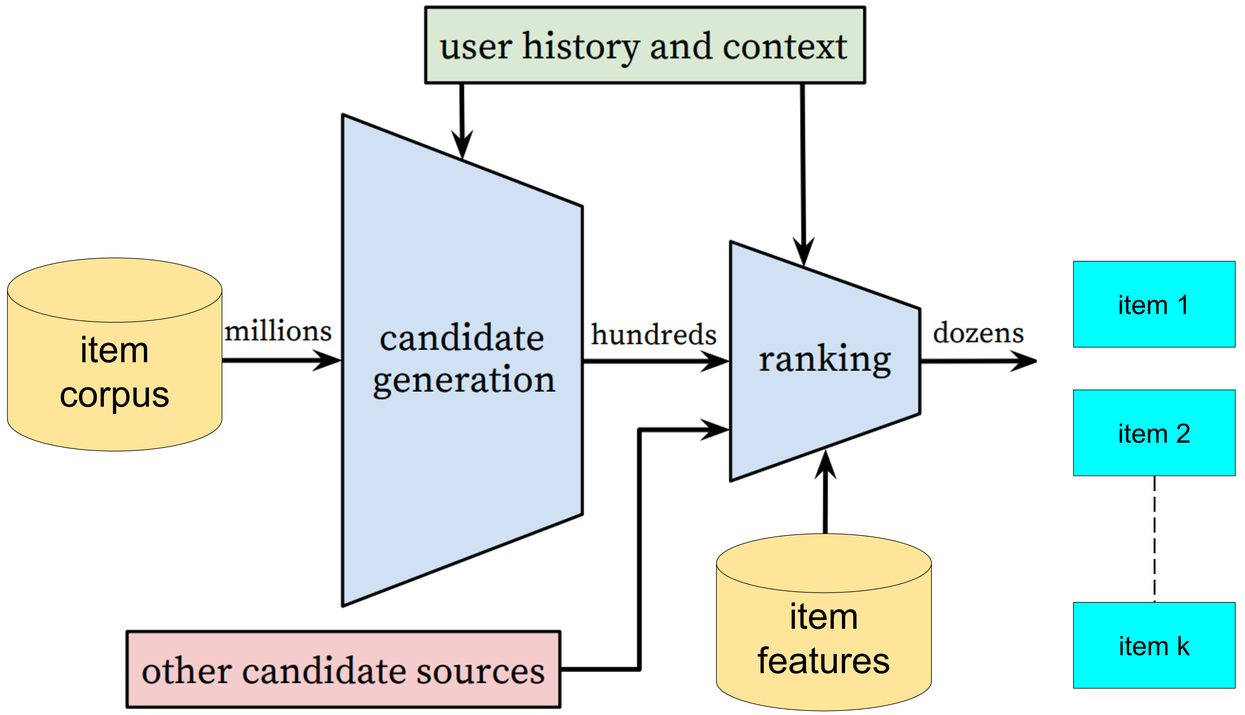



```
model = mm.YoutubeDNNRetrievalModel(
    schema,
    top_block=MLPBlock([128, 64]),
    num_sampled=100
)
```



Training and Deploying Multi-Stage Recommender Systems

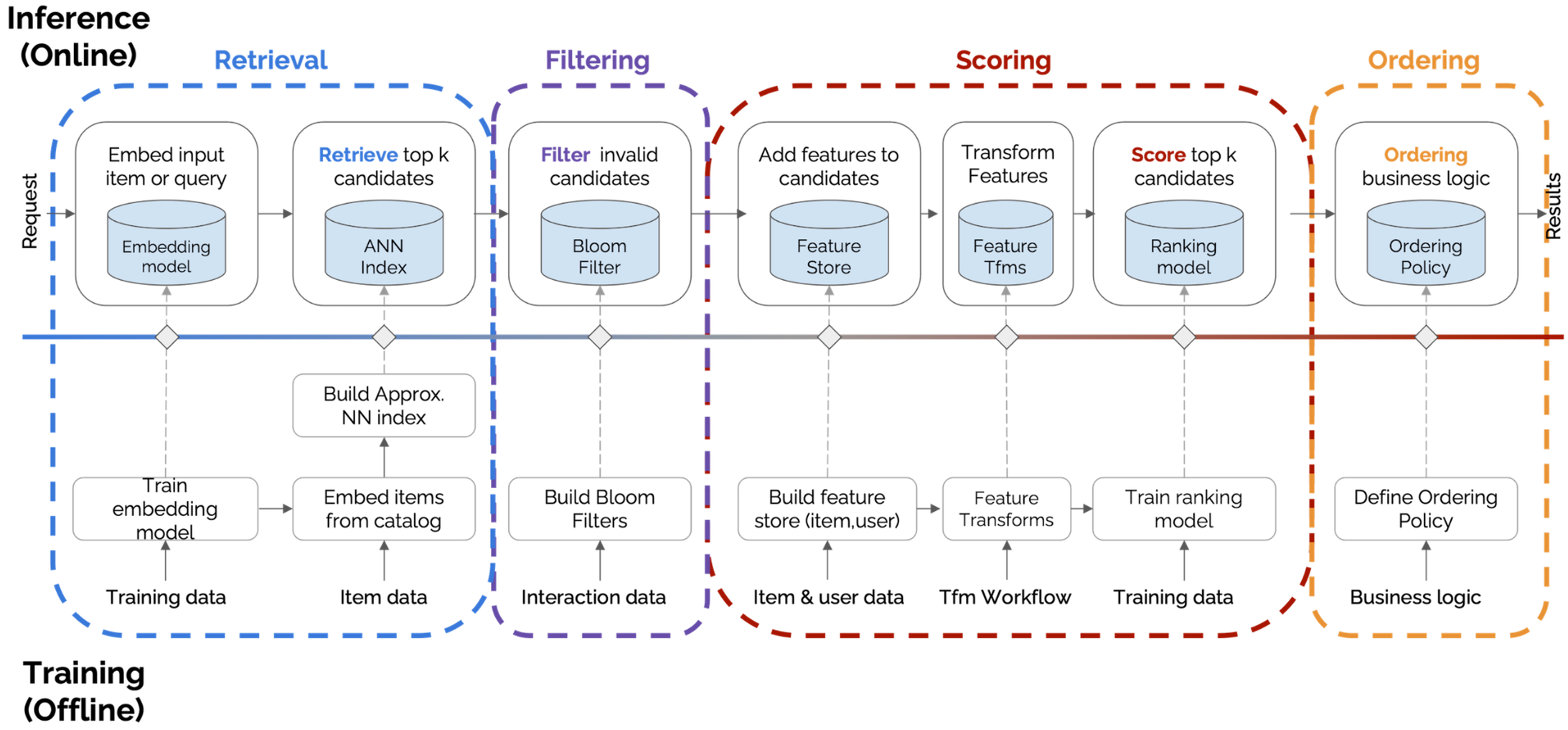
Two-Stage Recommender Systems



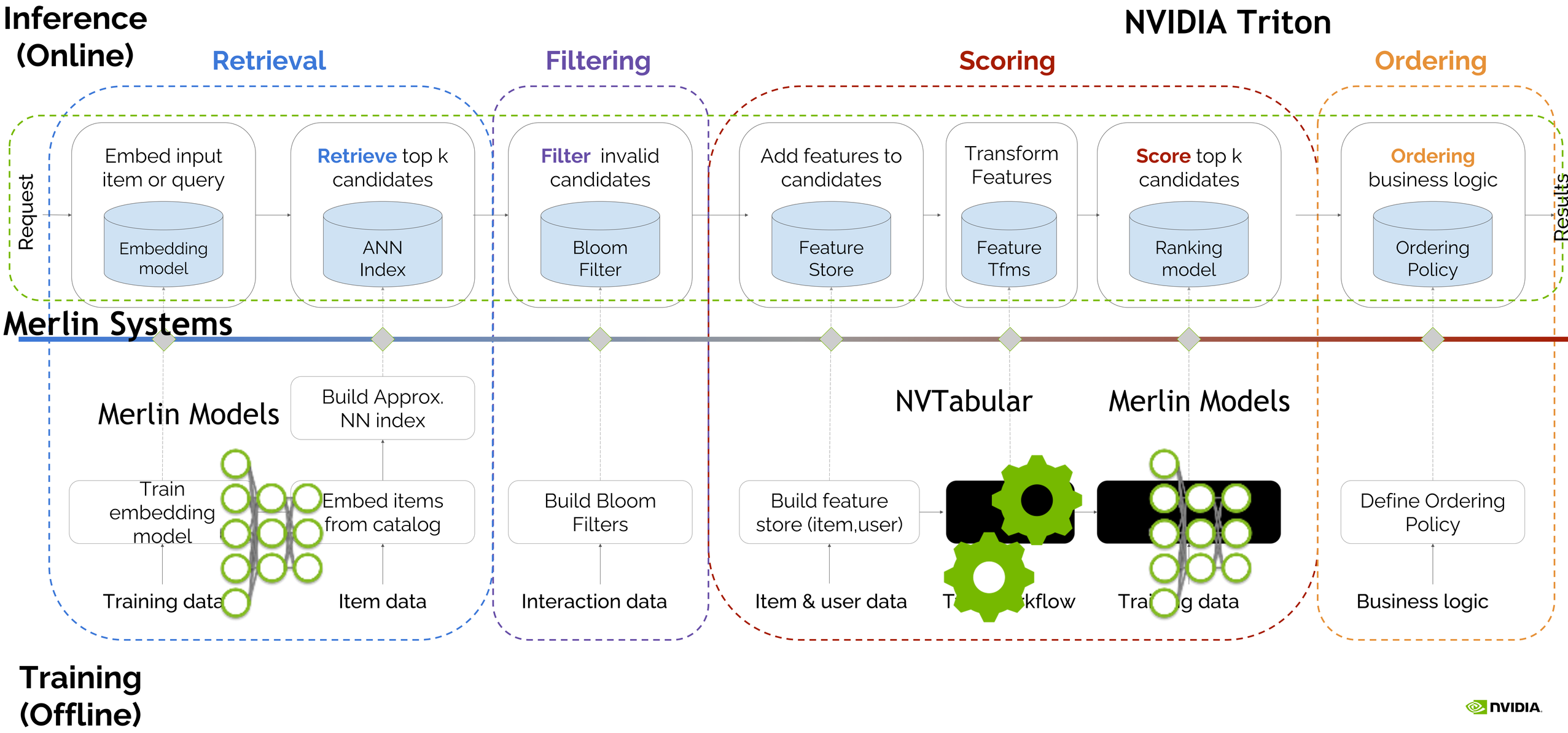
- Widely adopted in industry due to their scalability and maintainability.
- First stage - an efficient candidate generation model for item retrieval
- Second stage- more powerful ranker further narrows down the nominated (retrieved) items, and serves to the user.

Image adapted from Deep Neural Networks for YouTube Recommendations paper.

4-Stage Recommender Systems



End to End Recommender Systems in Production

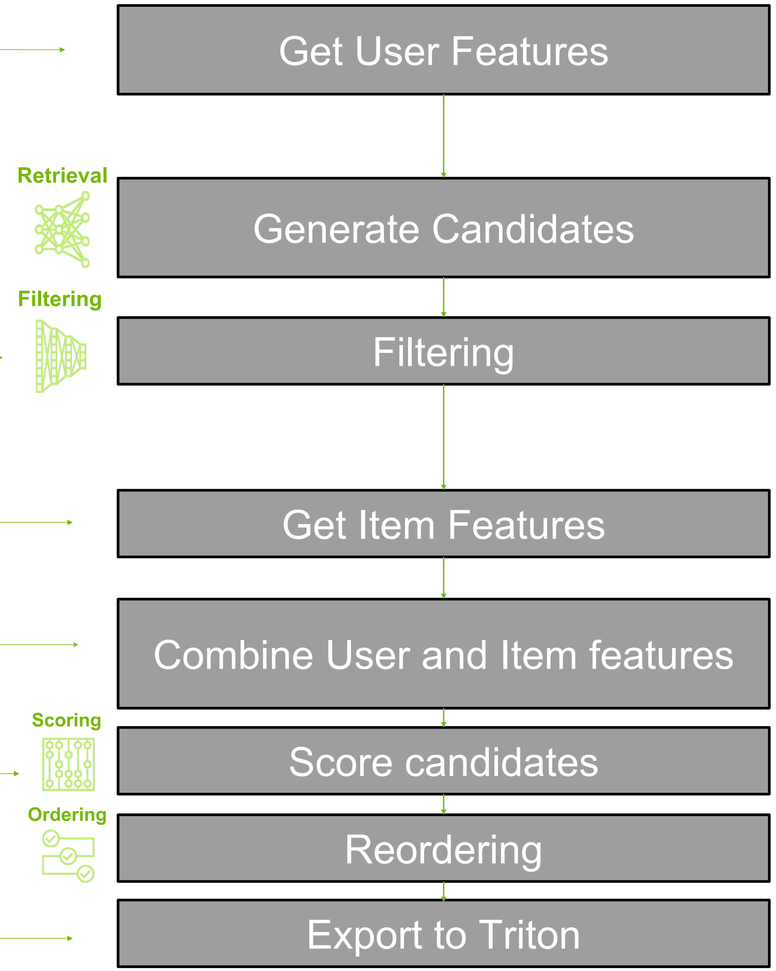


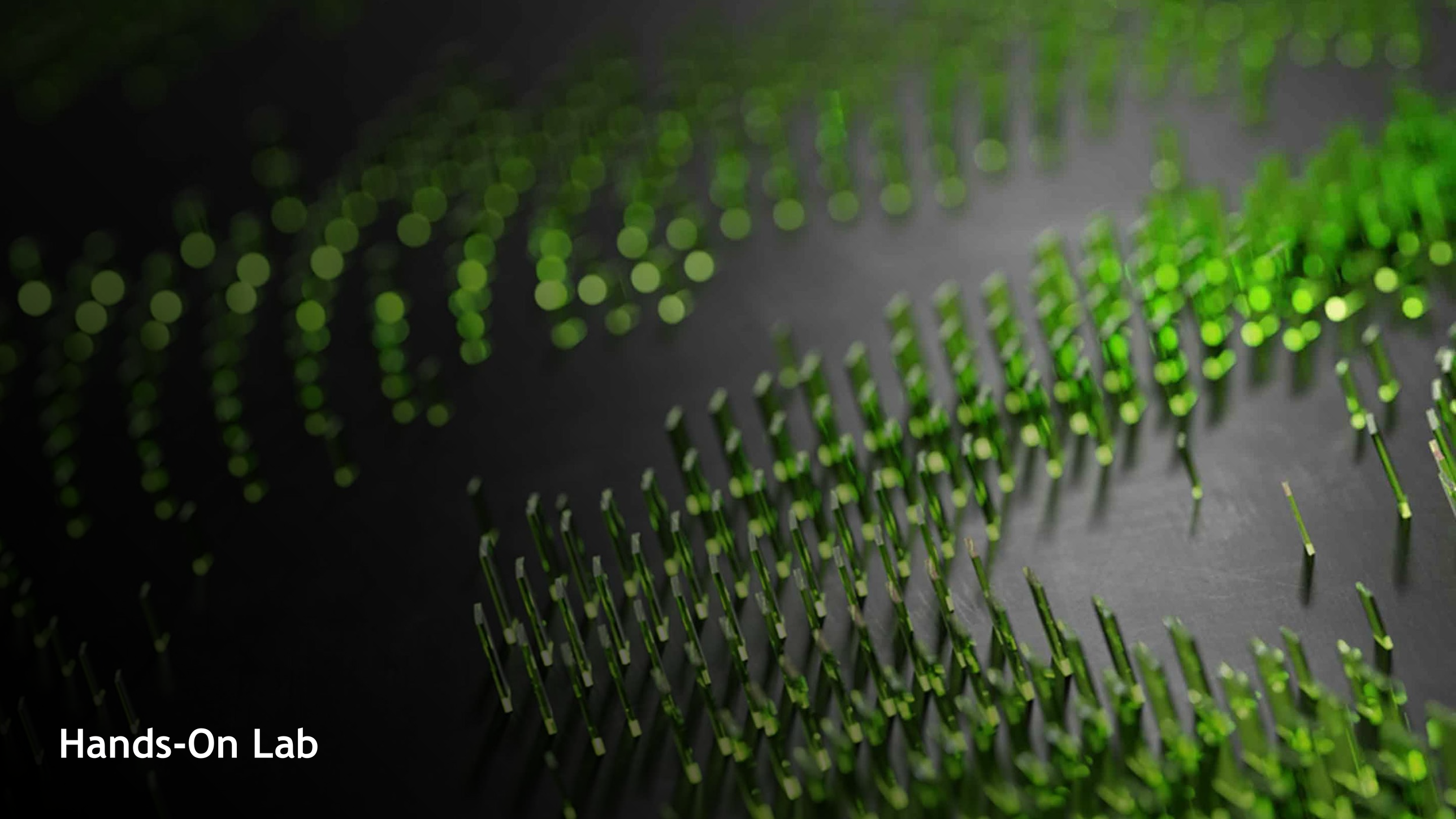
Merlin Systems simplifies complex Triton Pipeline

Merlin Systems Ensemble API (~50 lines)

```python
user_features = ["user_id"] >> QueryFeast(
    feast_repo_path,
    entity_view="user_features",
    entity_id="user_id",
    entity_column="user_id",
    features=["movie_id_count"],
    mh_features=["movie_ids", "genres", "search_terms"],
    input_schema=feast_user_in_schema,
    output_schema=feast_user_out_schema,
)

retrieval = (
    user_features
    >> PredictTensorflow(
        retrieval_model_path,
        custom_objects={"sampled_softmax_loss": sampled_softmax_loss},
    )
    >> QueryFaiss(faiss_index_path, query_vector_col="output_1", topk=100)
)

filtering = user_features["movie_ids_1"] + retrieval["candidate_ids"] >> FilterCandidates(
    candidate_col="candidate_ids", filter_col="movie_ids_1"
)

item_features = filtering >> QueryFeast(
    feast_repo_path,
    entity_view="movie_features",
    entity_id="movie_id",
    entity_column="filtered_ids",
    features=["tags_nunique"],
    mh_features=["genres", "tags_unique"],
    input_schema=Schema([ColumnSchema("filtered_ids", dtype=np.int32)]),
    output_schema=feast_item_out_schema,
    include_id=True,
    output_prefix="movie",
)

combined_features = user_features + item_features >> UnrollFeatures(
    "movie_id", feast_user_out_schema.column_names, unrolled_prefix="user"
)

ranking = combined_features >> PredictTensorflow(ranking_model_path)

ordering = (combined_features + ranking)["movie_id", "output_1"] >> SoftmaxSampling(
    "movie_id", relevance_col="output_1", topk=10, temperature=20.0
)

export_path = str("/nvtabular/test_poc/")

ensemble = Ensemble(ordering, request_schema)
ens_config, node_configs = ensemble.export(export_path)
```

Triton Pipeline




Hands-On Lab

Dataset of the Tutorial

Dataset: eCommerce behavior data from multi category store
Source: REES46 Marketing Platform
URL: https://www.kaggle.com/mkechinov/ecommerce-behavior-data-from-multi-category-store

Events: View, Cart, Purchase
Timeframe: Oct-2019 - April-2020

Positive target: Purchase
Negative target: AddToCart (removing AddToCarts of purchased items from the same session)

Dataset split:

Training: Oct-2019 - March-2020 (7.6 Mio samples after removing repeated interactions)
Validation: April-2020 (1.6 Mio samples after removing repeated interactions)

Baseline: 44.5% of events are purchases in the train set

Raw Features:
- user_id, user_session, product_id
- price
- category_code, category_id, brand
- event_time (timestamp)
- event_type (view, cart, remove_from_cart, purchase)

Resources

- Merlin GitHub repos:
  - https://github.com/NVIDIA-Merlin/NVTabular
  - https://github.com/NVIDIA-Merlin/models
  - https://github.com/NVIDIA-Merlin/systems

- Blog Posts:
  - Recommender Models: Reducing Friction with Merlin Models
  - Exploring Production Ready Recommender Systems with Merlin
  - Recommender Systems, Not Just Recommender Models

- **Winning challenges**: NVIDIA KGMON team and Merlin team won several recommendation competitions, specifically:
  - WSDM WebTour Workshop Challenge 2021, organized by Booking.com
  - SIGIR eCommerce Workshop Data Challenge 2021, organized by Coveo
  - ACM RecSys'21
  - ACM RecSys'22 (3rd place)
- Recordings:
  - Moving Beyond Recommender Models
  - NVIDIA Merlin: Recommender Systems at Scale on the GPU