

# EasyRec Online - Real-time Recommendation System with REST API

---

API Version: 1.1.0

Note: If the underlying recommendation model is not yet loaded, prediction endpoints (`POST /predict`, `POST /recommend`) will return HTTP 503 with a JSON body `{ "success": false, "status": "model_unavailable", "error": "..." }`. Clients should implement retry/backoff or surface an appropriate loading state.

This project provides an **online learning extension** for Alibaba's EasyRec framework, adding REST API capabilities and real-time model updates for production recommendation systems.

## Understanding Recommendation Systems (For Beginners)

Before diving into the technical details, let's understand what recommendation systems do and the key concepts involved:

### What is a Recommendation System?

A recommendation system is like a smart assistant that suggests items you might like based on your preferences and behavior. Think of:

- **Netflix** suggesting movies you might enjoy
- **Amazon** recommending products you might want to buy
- **Spotify** creating playlists based on your music taste
- **YouTube** showing videos you're likely to watch

### Key Concepts Explained

#### Users

- **Who:** People using your app/website (customers, viewers, listeners)
- **What we know:** Demographics (age, location), past behavior (clicks, purchases), preferences
- **Example:** User #123 is a 25-year-old from NYC who loves sci-fi movies

#### Items

- **What:** Things being recommended (products, movies, songs, articles)
- **What we know:** Categories, prices, descriptions, popularity, ratings
- **Example:** Item #456 is "The Matrix" - a sci-fi movie from 1999, rated 8.7/10

#### Features

- **Definition:** Pieces of information about users and items that help make predictions
- **User Features:** Age, gender, location, purchase history, browsing patterns

- **Item Features:** Category, price, brand, ratings, description keywords
- **Interaction Features:** Time of day, device used, season, context
- **Example:** "25-year-old user from NYC viewing sci-fi movies on mobile at 8PM"

## Interactions (User Actions)

- **What:** Actions users take with items - these are the core signals that teach the system about preferences
- **Where they go:** All actions are recorded as training data with labels and parameters

### Types of Actions:

- **Explicit Feedback:** User directly tells us their preference
  - Examples: Ratings (1-5 stars), thumbs up/down, like/dislike/super-like, reviews
- **Implicit Feedback:** User behavior that suggests preference
  - Examples: Views, clicks, time spent, purchases, downloads, shares

### Action Parameters (the details that make actions more meaningful):

- **Intensity:** How much they liked it (1-5 stars, like/dislike/super-like)
- **Context:** When, where, how (time of day, device, season, mood)
- **Duration:** How long they engaged (watched 10 minutes vs 2 hours)
- **Frequency:** How often they repeat the action
- **Outcome:** What happened next (bought after viewing, shared with friends)

### Real Examples:

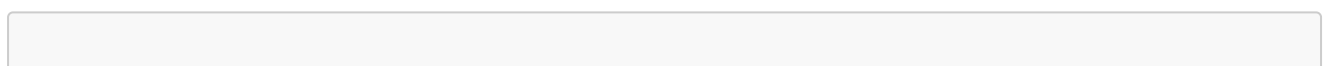
- Basic: "User #123 clicked on Movie A" → `{user_id: 123, item_id: MovieA, action: "click", label: 1}`
- With Parameters: "User #123 gave Movie A a 5-star rating on Friday evening" → `{user_id: 123, item_id: MovieA, action: "rating", label: 1, parameters: {stars: 5, time: "Friday_evening"}}`
- Complex: "User #123 super-liked Movie A, watched it completely, then shared it" → `{user_id: 123, item_id: MovieA, action: "super_like", label: 1, parameters: {intensity: "super", completion: 1.0, shared: true}}`

## The Recommendation Process

1. **Collect Data:** Gather user features, item features, and interactions
2. **Train Model:** Learn patterns from historical data ("Users like User #123 tend to enjoy sci-fi")
3. **Make Predictions:** For each user-item pair, predict how likely they are to interact
4. **Rank & Filter:** Show top items with highest predicted scores
5. **Learn & Adapt:** Update the model as new interactions come in

## Where User Actions Go in the System

### Action Flow Pipeline:



User Action → API Endpoint → Data Processing → Model Training → Updated Recommendations

1. User performs action (like, view, purchase, etc.)  
↓
2. Frontend/App sends action to API: POST /online/data/add  
↓
3. Action gets processed and stored with parameters:  
{user\_id, item\_id, action\_type, label, parameters, timestamp}  
↓
4. System decides: Immediate learning OR batch learning  
↓
5. Model updates and new recommendations become available

### Data Storage & Processing:

- **Immediate Storage:** Actions are stored instantly for future training
- **Real-time Learning:** High-priority actions (purchases, explicit ratings) trigger immediate model updates
- **Batch Learning:** Lower-priority actions (views, clicks) are processed in batches
- **Feature Engineering:** Action parameters become input features for the model
- **A/B Testing:** Some actions may be used to test different recommendation strategies

### Action Parameter Processing:

```
# Example of how action parameters become model features
raw_action = {
    "user_id": 123,
    "item_id": "movie_456",
    "action": "rating",
    "parameters": {"stars": 5, "time": "friday_evening", "device":
"mobile"}
}

# Processed into model features:
features = {
    "user_123_movie_456_interaction": 1,
    "rating_value": 5,
    "time_of_week": "weekend",
    "device_type": "mobile",
    "interaction_strength": "strong" # derived from 5-star rating
}
```

### Real-World Examples

#### Example 1: Movie Streaming Service

#### User Profile:

- User ID: 123
- Age: 25, Location: NYC
- Previously watched: Sci-fi movies, Action movies
- Rating pattern: Likes complex plots, dislikes romantic comedies

#### Available Movies:

- Movie A: "Blade Runner 2049" (Sci-fi, 8.0 rating, 2017)
- Movie B: "The Notebook" (Romance, 7.8 rating, 2004)
- Movie C: "John Wick" (Action, 7.4 rating, 2014)

#### User Actions & Parameters:

```
{user_id: 123, item_id: MovieA, action: "watch", parameters: {duration: 120_min, completion: 0.95, rating: 5}}  
{user_id: 123, item_id: MovieC, action: "watch", parameters: {duration: 45_min, completion: 0.4, abandoned: true}}
```

#### Recommendation System Logic:

1. User likes sci-fi + action ✓, completed MovieA fully ✓
2. User abandoned MovieC halfway → lower preference signal
3. Predictions: Movie A (95%), Movie C (78%), Movie B (23%)
4. Recommendation: Show "Blade Runner 2049" first!

### Example 2: Movie Theatre Chain

#### User Profile:

- User ID: 456
- Demographics: 30yo, family with kids
- Location: Suburban mall
- Past behavior: Weekend family movies, avoids late shows

#### Available Showtimes:

- Movie A: "Frozen 3" (Animation, 2:00 PM Saturday, Family-friendly)
- Movie B: "John Wick 5" (Action, 10:30 PM Friday, R-rated)
- Movie C: "Spider-Man" (Action, 4:00 PM Sunday, PG-13)

#### User Actions & Parameters:

```
{user_id: 456, item_id: ShowtimeA, action: "book_tickets", parameters: {tickets: 4, seats: "family_section", snacks: true, advanced_booking: 3_days}}  
{user_id: 456, item_id: ShowtimeB, action: "view_details", parameters: {time_spent: 10_sec, bounced: true, reason: "too_late"}}
```

#### System Learning:

- Family bookings → High preference for family movies
- Bounced from late shows → Prefers afternoon/evening times
- Books in advance → Plans family outings ahead

Recommendation: Prioritize family-friendly afternoon shows



### Example 3: Furniture Shopping Platform

#### User Profile:

- User ID: 789
- Demographics: 28yo, first apartment
- Budget signals: Views items \$200-800 range
- Style preference: Modern, minimalist

#### Available Items:

- Item A: IKEA Sofa (\$450, Modern, 4.2★, Quick delivery)
- Item B: West Elm Chair (\$750, Mid-century, 4.5★, 6-week delivery)
- Item C: Vintage Dresser (\$1200, Antique, 3.8★, Custom order)

#### User Actions & Parameters:

```
{user_id: 789, item_id: ItemA, action: "add_to_cart", parameters:
{quantity: 1, color: "gray", delivery: "express"}}
{user_id: 789, item_id: ItemA, action: "view_reviews", parameters:
{time_spent: 5_min, filter: "recent_reviews"}}
{user_id: 789, item_id: ItemB, action: "save_wishlist", parameters:
{list_name: "future_purchases", notes: "when_I_get_raise"}}
{user_id: 789, item_id: ItemC, action: "view", parameters: {time_spent:
15_sec, bounced: true, reason: "price_too_high"}}
```

#### System Learning:

- Price sensitivity → Avoid items >\$800
- Research behavior → User reads reviews carefully
- Wishlist saving → Interested but budget-conscious
- Quick bounce on expensive items → Strong price filtering

Recommendation: Show modern furniture under \$600 with good reviews



### Example 4: Smart Agriculture Platform

#### Farmer Profile:

- User ID: 101
- Farm: 50 acres, Midwest USA
- Crops: Corn, soybeans, wheat rotation
- Experience: 10 years, tech-adopter

#### Available Agricultural Actions:

- Action A: Watering (Item: Corn Field #3)
- Action B: Fertilizing (Item: Soybean Field #1)
- Action C: Harvesting (Item: Wheat Field #2)

#### User Actions & Parameters:

```
{user_id: 101, item_id: CornField3, action: "watering", parameters:
{amount: 2_inches, method: "drip_irrigation", soil_moisture: 0.3,
weather_forecast: "dry_week", timing: "early_morning"}}
{user_id: 101, item_id: SoybeanField1, action: "fertilizing",
```

```
parameters: {type: "nitrogen", amount: "150_lbs_per_acre", application:
"broadcast", soil_test: {pH: 6.8, N: "low", P: "medium"}, growth_stage:
"V6"}}
```

```
{user_id: 101, item_id: WheatField2, action: "harvesting", parameters:
{method: "combine_harvester", moisture_content: 0.14, yield:
"65_bushels_per_acre", quality_grade: "premium", weather: "sunny_dry"}}
```

System Learning:

- Watering patterns → Prefers early morning, uses drip irrigation
- Fertilizer choices → Data-driven based on soil tests
- Harvest timing → Waits for optimal moisture/quality
- Tech adoption → Uses precision agriculture tools

AI Recommendations:

- "Field #4 corn shows 0.25 soil moisture, recommend watering 1.5 inches tonight"
- "Soybean field #2 at V4 stage, soil test shows low phosphorus, recommend P fertilizer"
- "Wheat field #1 at 15% moisture, wait 2 days for optimal harvest conditions"

## Architecture Overview

**EasyRec Online** = **Alibaba EasyRec** (Core Framework) + **Online Learning Extensions** (This Project)




What is Alibaba EasyRec?

[Alibaba EasyRec](#) is a production-ready framework for recommendation systems that implements state-of-the-art deep learning models used in:

- **Candidate generation (matching)** - DSSM, MIND, etc.
- **Scoring (ranking)** - DeepFM, Wide&Deep, DCN, etc.
- **Multi-task learning** - MMoE, ESMM, PLE, etc.

What This Project Adds

**EasyRec Online** extends the original framework with production-ready features:

-  **REST API Server** - Easy-to-use web interface for getting recommendations
  - *What it means:* Instead of writing complex code, just send HTTP requests to get recommendations
  - *Example:* `curl -X POST /recommend` to get movie suggestions for a user
-  **Real-time Learning** - The system gets smarter as users interact with it
  - *What it means:* When users click, buy, or rate items, the model learns immediately
  - *Example:* User likes a new sci-fi movie → System instantly learns this preference
-  **Online Training** - Continuous model updates with streaming data

- *What it means:* No need to retrain the entire model - just add new data as it comes
- *Example:* New user interactions flow in via Kafka and update the model in real-time
- 🚀 **Easy Deployment** - Ready-to-use Docker containers and monitoring
  - *What it means:* Run the entire system with one command, monitor performance easily
  - *Example:* `docker-compose up` and you have a full recommendation system running
- 🛠️ **Continuous Training** - Automatic model improvement and management
  - *What it means:* The system handles model updates, versioning, and rollbacks automatically
  - *Example:* Model performance drops → System automatically trains a new version

## Features

- **Multiple Models:** DeepFM, Wide&Deep, DSSM, MIND, DCN, AutoInt, etc.
- **Easy Configuration:** Simple config files to define models and features
- **Scalable:** Supports large-scale embeddings and online learning
- **Multiple Platforms:** Local, MaxCompute, EMR-DataScience, PAI-DSW
- **Easy Deployment:** Automatic scaling and monitoring with EAS

## Project Structure

```

easyrec_online/
├── README.md                # This project documentation
├── requirements.txt         # Python dependencies (includes real
EasyRec)
├── setup.py                 # Package configuration
├── config/
│   └── deepfm_config.prototxt # EasyRec model configuration (original
format)
├── data/
│   └── process_data.py       # Sample data generation (this project)
├── models/
│   ├── __init__.py
│   └── recommendation_model.py # Model wrapper with online features
(this project)
├── api/                     # 🆕 REST API Layer (this project)
│   ├── __init__.py
│   ├── app.py               # Flask API server
│   └── routes_online.py     # Online learning endpoints
├── streaming/               # 🆕 Real-time Learning (this project)
│   ├── __init__.py
│   ├── kafka_consumer.py    # Kafka streaming input
│   └── online_trainer.py    # Incremental training
├── scripts/
│   ├── train.py             # Training script (uses EasyRec)
│   └── serve.py             # Production server (this project)
├── tests/
│   └── __init__.py

```

```
|   └─ test_api.py           # API tests (this project)
|   └─ setup.sh              # Setup script (this project)
|   └─ Dockerfile            # Docker configuration (this project)
|   └─ docker-compose.yml    # Docker Compose setup (this project)
|   └─ config.ini            # Configuration file (this project)
|   └─ .env.example          # Environment variables (this project)
```

## Component Attribution

### From Alibaba EasyRec (Original):

- Core training/evaluation engine (`easy_rec.python.train_eval`)
- Model implementations (DeepFM, Wide&Deep, DSSM, etc.)
- Configuration format (`.prototxt` files)
- Online training framework (ODL - Online Deep Learning)
- Kafka/DataHub streaming input support

### Added by EasyRec Online (This Project):

- REST API server and endpoints
- Real-time model serving infrastructure
- Incremental update API endpoints
- Docker deployment and orchestration
- Monitoring and health checks
- Client libraries and examples

## Installation

### Option 1: Local Installation

```
# Create conda environment
conda create -n easyrec_env python=3.6.8
conda activate easyrec_env

# Install dependencies
pip install -r requirements.txt

# Clone and install EasyRec
git clone https://github.com/alibaba/EasyRec.git
cd EasyRec
bash scripts/init.sh
python setup.py install
cd ..
```

### Option 2: Docker Installation



```
# Pull pre-built image
docker pull mybigpai-public-registry.cn-
beijing.cr.aliyuncs.com/easyrec/easyrec:py36-tf1.15-0.8.5

# Run container
docker run -td --network host -v $(pwd):/workspace mybigpai-public-
registry.cn-beijing.cr.aliyuncs.com/easyrec/easyrec:py36-tf1.15-0.8.5
```

## Quick Start

Let's get your recommendation system running in 4 simple steps:

### 1. Setup the project:

```
chmod +x setup.sh
./setup.sh
```

*This installs all dependencies and prepares your environment*

### 2. Train the exmple model and start the API server, following the instructions printed by setup.sh

*This starts your recommendation service on <http://localhost:5000>*

### 3. Test basic recommendations:

```
curl -X POST http://localhost:5000/recommend \
-H "Content-Type: application/json" \
-d '{"user_id": 123, "candidate_items": [1,2,3,4,5], "top_k": 3}'
```

*Translation: "For user #123, rank these 5 items and give me the top 3 recommendations"*

### Expected Response:

```
{
  "user_id": 123,
  "recommendations": [
    {"item_id": 2, "score": 0.95},
    {"item_id": 4, "score": 0.87},
    {"item_id": 1, "score": 0.73}
  ]
}
```

#### 4. Test online learning with action parameters:

##### Basic Action (Simple like/dislike):

```
curl -X POST http://localhost:5000/online/data/add \  
-H "Content-Type: application/json" \  
-d '{"samples": [{"user_id": 123, "item_id": 6, "label": 1}]}'
```

*Translation: "User #123 liked item #6 (basic positive interaction)"*

##### Action with Parameters (Rating with context):

```
curl -X POST http://localhost:5000/online/data/add \  
-H "Content-Type: application/json" \  
-d '{  
  "samples": [{  
    "user_id": 123,  
    "item_id": 6,  
    "label": 1,  
    "action_type": "rating",  
    "parameters": {  
      "stars": 5,  
      "time_of_day": "evening",  
      "device": "mobile",  
      "completion_rate": 0.95  
    }  
  }]  
'
```

*Translation: "User #123 gave item #6 a 5-star rating on mobile in the evening, watched 95% of it"*

##### Complex Action (Agriculture example):

```
curl -X POST http://localhost:5000/online/data/add \  
-H "Content-Type: application/json" \  
-d '{  
  "samples": [{  
    "user_id": 101,  
    "item_id": "corn_field_3",  
    "label": 1,  
    "action_type": "watering",  
    "parameters": {  
      "amount_inches": 2.0,  
      "method": "drip_irrigation",  
      "soil_moisture": 0.3,  
      "weather_context": "dry_week_forecast",  
      "timing": "early_morning",  
    }  
  }]  
'
```

```
    "efficiency_score": 0.88
  }
}]
}'
```

*Translation: "Farmer #101 successfully watered corn field #3 with 2 inches using drip irrigation at optimal timing"*

```
# Check training status
curl -X GET http://localhost:5000/online/training/status
```

*Translation: "Is the system currently learning from new action data?"*

```
# Start incremental training
curl -X POST http://localhost:5000/online/training/start
```

*Translation: "Start learning from all the new interaction data with parameters I just added"*

## API Endpoints

Your recommendation system provides these easy-to-use endpoints:

### Core Recommendation APIs

- **POST /recommend** - Get personalized recommendations for a user
  - *Use case:* "Show me the top 5 products this user might like"
  - *Input:* User ID, candidate items, number of recommendations needed
  - *Output:* Ranked list of items with confidence scores
- **POST /predict** - Predict interaction probability for specific user-item pairs
  - *Use case:* "How likely is this user to click/buy this specific item?"
  - *Input:* User ID, item ID (or multiple pairs)
  - *Output:* Probability scores (0-1, higher = more likely to interact)

### System Health & Info

- **GET /health** - Check if the system is running properly (includes **version** field)
- **GET /model/info** - Aggregated model info (now also includes online incremental details if trainer active)
- **POST /model/export** - Export current model (replaces former **/online/model/export**)

### Online Learning APIs

- **POST /online/data/add** - Add new user interaction data

- `GET /online/training/status` - Check if the model is currently learning
- `POST /online/training/start` - Start incremental training
- `POST /online/training/stop` - Stop incremental training
- `PATCH /online/training/restart-policy` - Update restart policy parameters
- `GET /online/training/logs` - Tail training logs
- `GET /online/updates/list` - List incremental update artifacts (may be merged into `/model/info` in future)



### Streaming (Kafka) Utilities

- `GET /online/streaming/status` - Kafka consumer status
- `POST /online/streaming/consume` - Manually consume a batch (debug/testing)

## References

- [EasyRec GitHub Repository](#)
- [EasyRec Documentation](#)