Red Hat AI Quickstart: Implementing AI-Driven Product Recommendations and Semantic Search

Challenges and Solutions for Modern Recommender Systems

Red Hat

AI-Driven Recommendations Are Pervasive

Red Hat

Recommendation systems are part of many digital platforms consumers, businesses and enterprises interact with daily

Retail: Amazon, Chewy, Home Depot, Target, Wayfair	Suggest products based on search and purchase history and common patterns shared across customers
Entertainment: Amazon Prime Video, Pandora, Tubi, Pluto TV, YouTube	Recommend content customers will likely enjoy based on preferences, reviews and viewing history
Web, Search and Social Platforms: Google Search, Facebook, LinkedIn	Leverage online activity for autocompletion, news topic suggestions, job recommendations and “people you may know”
Targeted Marketing Platforms: Klaviyo, HubSpot, SalesForce	These platforms rely on fine-tuned recommendation systems and CRM-integration to drive sales across businesses and enterprises

The Logic of Recommendations

While there are many types of recommendation and search systems, they share a common goal and a common set of technological requirements.

Goal

Match potential customers with relevant products and services a business offers.

Requirements

- Models that predict the products and services people want.
- Powerful semantic search engines
- Model training pipeline management
- Efficient storage and use of vector embeddings: AI’s lingua franca
- Inference servers: Middleware makes it possible to scale ML models over thousands of concurrent users
- Modern platform for deploying, running and scaling today’s containerized and microservices-based applications

We’ll dive in to these requirements and show how the Red Hat Product Recommender Quickstart leverages **OpenShift AI** to satisfy these requirements.

Capturing the Signal …

The key idea behind building an effective Product Recommender is to capture and leverage the implicit and explicit signals customers provide on your products and services.

Web & App Activity (Implicit)

Capturing **product clicks**, browser history, session duration, and navigation patterns from all websites and microservices.

Transactional Data (Implicit)

Tracking **purchases** and **returns** from backend databases to understand what customers value and what they reject.

Direct Feedback (Explicit)

Collecting explicit signals from **product reviews**, star **rankings**, and **support tickets** in issue management systems.

Unpacking the Signal

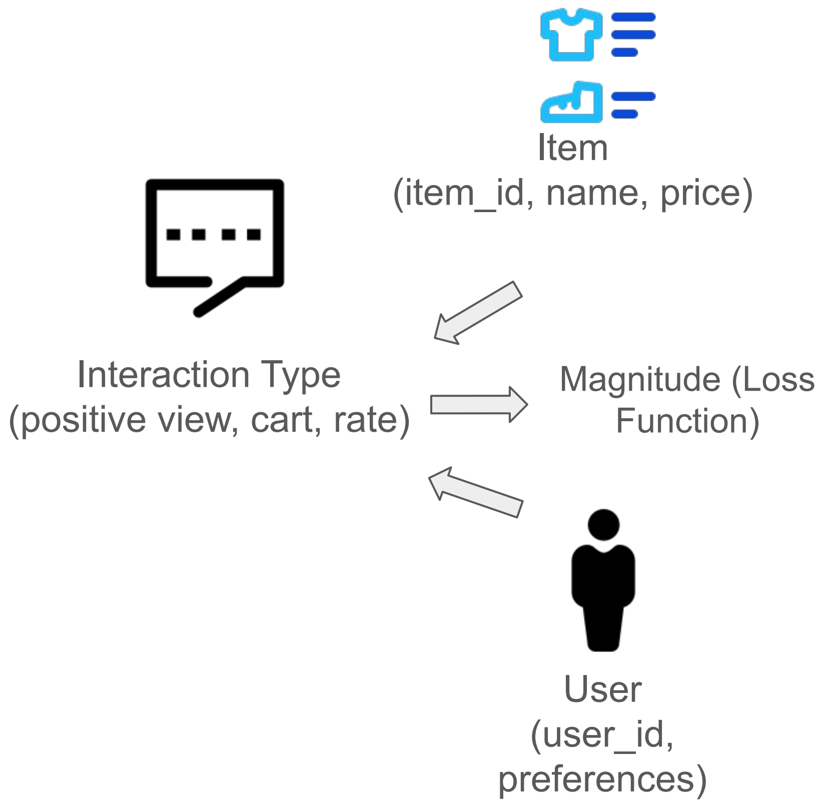
To create an accurate predictor of which products and services customers want, we’ll also need to unpack these signals and interpret them semantically in the context of a product or service catalog.

The signals the Red Hat Product Recommender quickstart examines focus on three types of information: (1) User information (preferences); (2) Products (names, descriptions, etc.); and (3) Interactions between these two entities.

Loss Function

Neural networks work on a simple principle: Modify the network weights until it’s output minimizes some loss function. In the next slide, we’ll see how Product Recommender defines a loss function that will help us match users with the right products. Later, we’ll see how embeddings work and how we can apply this loss function to cluster users together with the products they may like.

User-Product Interactions



Adjustment Type	Adjustment
Interaction Type	Pos. View: x / factor Neg. View: x * factor Cart: x / (factor * 3) Purchase: x / (factor * 10) Rating: No type modification
Rating	Rating = 3: No modification Rating <= 2: x * (factor * (3 - rating)) Rating > 2: x / (factor * (r - 2))
Quantity	Quantity > 1: x / (factor * (quantity - 1))

Example: A purchase of 10 units of an item reduces our loss by dividing x by (factor * 10). Here, both x and factor are initialized with default values (we can think of these as learned hyperparameters or knobs that affect our loss function). However, a negative rating multiplies our loss.

Recommendation Predictions as a Search Problem

What if we could map users and products in some high-dimensional geometric space that clustered the two types of entities together if their interactions were strong and positive (i.e., our loss function computed lower scores for them)?

If we can do that, than we can use simple techniques from Euclidean geometry to search for items that are nearby a given user in this geometric space and consider these items as effective recommendations!

To do this accurately though, we’ll need to learn how LLMs and embeddings work in the next few slides.

LLMs and other Machine Learning Models

Let’s face it – the headliners these days are the LLMs and machine learning models. These models are applied across a range of recommender tasks:

Semantic Search and Recommendations

Review Summarization

Dynamic Faceted Search

Re-Ranking

Let’s take a deeper dive into embeddings: The backbone of search and recommendations and a key aspect behind the power of LLMs.

Feature Embeddings - Brief Primer

Computers really only know physical states. It's easy to map numbers to these states, but how do we represent semantic objects like product descriptions and user preferences as numbers.

We map them as *vectors* in an *embedding space*.

A vector is just an ordered list of numbers; e.g., <1.2, 0.1, 2.2>.

An embedding space uses these vectors in a coordinated and geometric way such that the distance between similar words is small and the addition or subtraction of vectors are conceptually meaningful (see example below).

Vector('Trail Hiker Backpack') + Vector('Upgrade') = Vector('Mountaineers Backpack')

Feature Embeddings - Static Embeddings

Up to as recent as 2019, static embeddings, like Word2Vec, were still widely used to map text tokens to vectors.

These models tried to capture each word's meaning by the 'company it kept' (i.e., the words before and after it).

Static embeddings are trained by gradually tweaking the weights of a neural network so it can predict a word's neighbors (or vice versa). Once trained, these weight matrices form an embedding space.

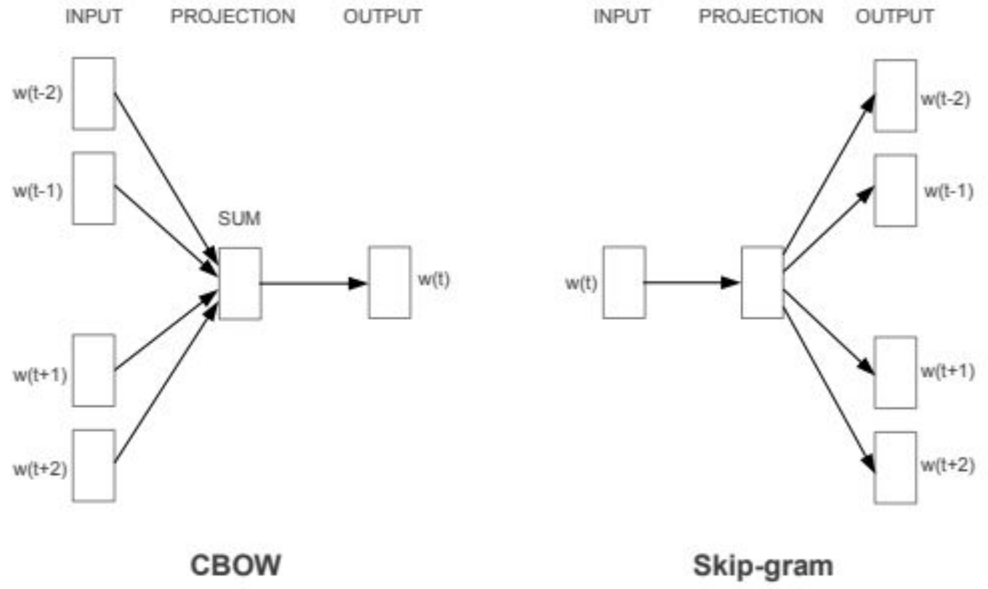


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Efficient Estimation of Word Representations in Vector Space (Mikolov, et. al. 2013)

Pachinko Machines

Pachinko machines teach us most of what we need to understand to appreciate static embeddings. These machines were basically vertical pinball machine that allowed players to shoot metal balls to the top using a spring-loaded handle. The balls would bounce around against the pins and, if they landed in the right place, you would win … what? Well more metal balls of course to keep you playing!

An embedding is like this, except instead of controlling the spring-loaded handle to influence the ball's trajectory, you move the pins ever so slightly. Now picture metal balls of 50k different sizes (these are the words in your vocabulary) and billions of moveable pins in 3 dimensions (these are your weights) and you've understood how static embeddings work.

A 1970s style Pachinko machine (https://en.wikipedia.org/wiki/Pachinko)

Feature Embeddings - Current (Complex) Landscape

One problem: Static emeddings don't handle polysemy well.

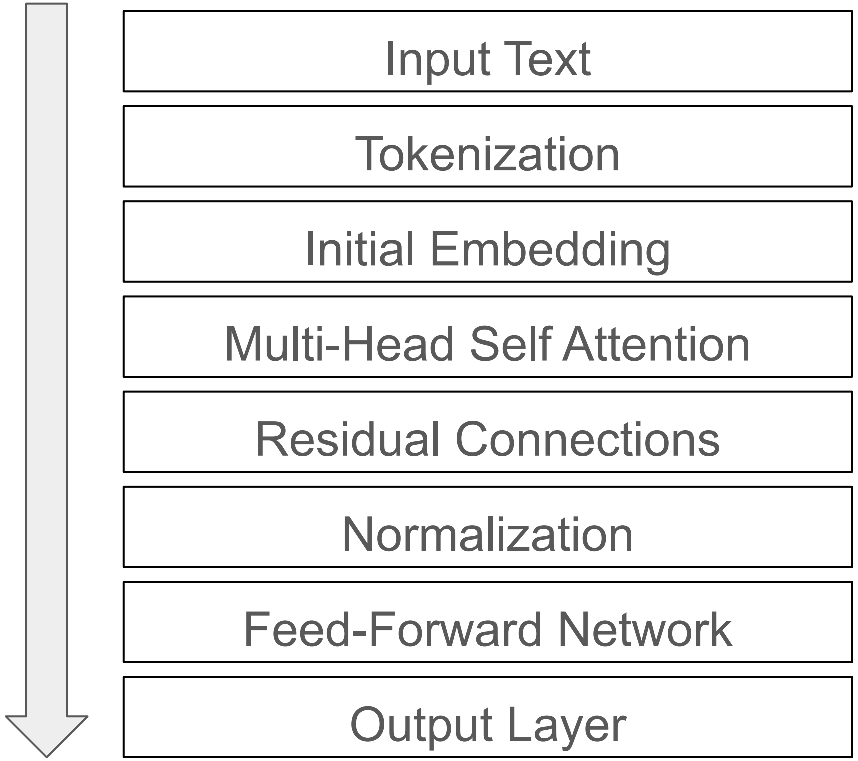
"I had to crane my neck to see the crane perched on the crane."

Their training regime was also focused on semantics vs. useful tasks (like information retrieval, summarization, etc. – which are more important for product recommenders).

The **BAAI/bge-small-en-v1.5** model used by the product recommender takes things to the next level.

Encoder-only LLM: Instead of a static matrix, a transformer architecture is used to generate contextual embeddings which (given an input sequence) take a weighted sum of every token's contribution to every other token's meaning.

Contrastive Learning – These weights are learned using contrastive learning: A sophisticated pipeline of supervised positive/negative input pairs, including natural language inference examples (SNLI), question/answers (SQuAD), etc.. The model is trained to push embeddings for positive pairs closer and those for negative pairs further apart.



BAAI/bge-small-en-v1.5 - Is It Right For Your Specific Case?

Red Hat

Consider these factors when choosing an embedding model.

Output Dimension: 384 (relatively low)
- Minimizes storage, speeds vector comparisons (especially with Approximate Nearest Neighbor)
- Not good for capturing fine shades of meaning or representing large documents.

Max Input Tokens: 512 (relatively small, will have issues with large inputs)
Embedding Quality: 62.17 Average MTEB across multiple tasks/datasets
- This is a strong score for a general embedding model

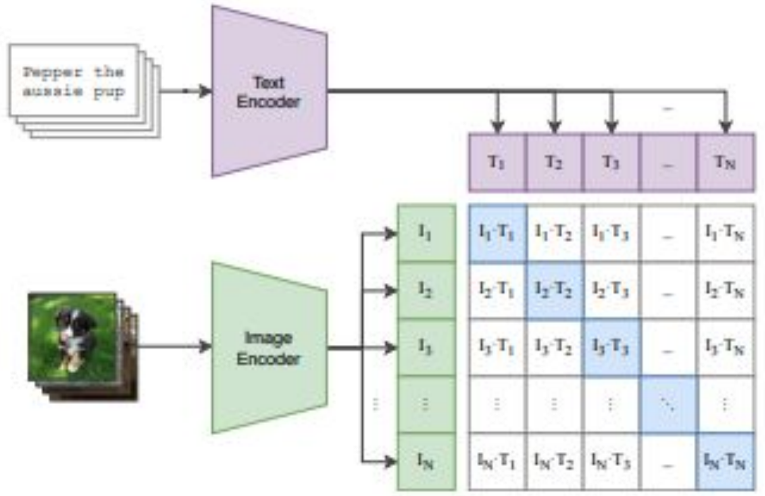
Multi-lingual: No (use BGE-M3 instead)
Multimodal: No (text only)
Domain Specific: No (fine-tune using FlagEmbedding toolkit to handle jargon better (https://github.com/FlagOpen/FlagEmbedding/tree/master)
Purpose: Tailored for asymmetric, information retrieval tasks (short query to long results) versus tasks that require strict semantic similarity.
Speed: Tokens Per Second (TPS)/Latency (unknown/check with your hardware)
Compliance Issues: May not be right for US security applications (model's origin is China)

Image Embeddings - openai/clip-vit-base-patch32

Product Recommender uses a separate embedding model for images

Here again we see an evolution in technology from traditional Convolutional Neural Networks (CNNs) to transformers for learning image features and embedding representations.

- This model is a standard dual encoder (or two-tower model) that uses two transformers: One to process text inputs and the other images, creating two vector outputs.
- Contrastive training: For known similar items (an image of a remote control and the text ‘remote control’), backpropagation is used to update the weights of each tower so they project vectors closer in n-dimensional space.


Pepper the aussie pup
Text Encoder
Image Encoder
T_1 T_2 T_3 … T_N
I_1 I_2 I_3 ⋮ I_N
$I_1 \cdot T_1$ $I_1 \cdot T_2$ $I_1 \cdot T_3$ … $I_1 \cdot T_N$
$I_2 \cdot T_1$ $I_2 \cdot T_2$ $I_2 \cdot T_3$ … $I_2 \cdot T_N$
$I_3 \cdot T_1$ $I_3 \cdot T_2$ $I_3 \cdot T_3$ … $I_3 \cdot T_N$
$I_N \cdot T_1$ $I_N \cdot T_2$ $I_N \cdot T_3$ … $I_N \cdot T_N$

Learning Transferable Visual Models From Natural Language Supervision (Radford et. al. 2021)

openai/clip-vit-base-patch32 - Is It Right For Your Specific Case

Consider these various factors when choosing an embedding model.

Output Dimension: 512
Max Input Tokens: 76 (image descriptions must be brief)
Expected Image Resolution: 224 x 224
Embedding Quality: Multiple benchmarks (see Huggingface)
Multi-lingual: No
Multimodal: Yes (text and image)
Domain Specific: No
Purpose: Tailored for image-to-image, text-to-image, and image-to-text tasks
Speed: ?
Compliance Issues: This is a **research-only model** with known bias and fairness issues (see paper)

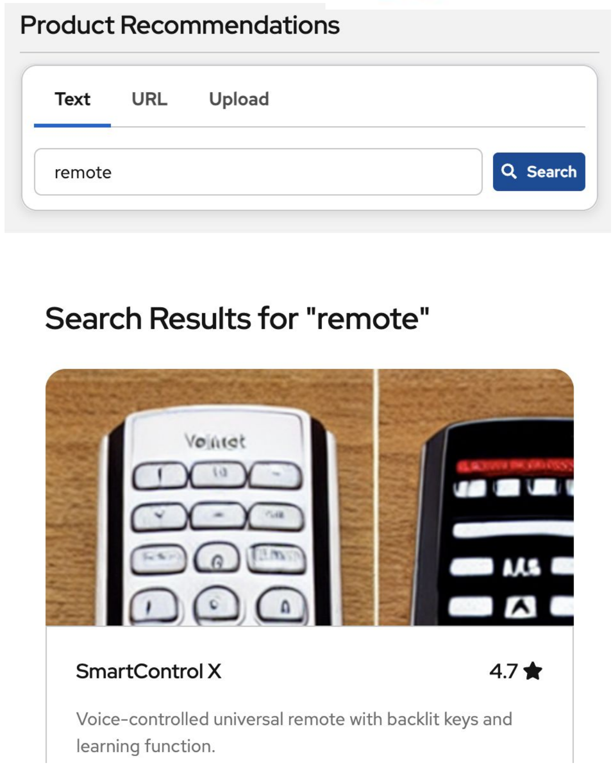
How Product Recommender Uses Embeddings

Product Recommender uses text and image embeddings to support hybrid semantic search.

The models are used to generate embeddings for the product catalog and stored in FEAST (a feature store which we'll discuss later). At runtime, queries are converted into embeddings using the same models and passed to FEAST to return similar products based on the cosine similarity of their embeddings.

Hybrid Search

Employed to combine symbolic (SQL/regex) and semantic techniques. Addresses the sometimes counterintuitive results that occur with semantic search (where items with hard text matches are ranked lower than those with semantic matches).



Search Results for "remote"

SmartControl X 4.7 ★

Voice-controlled universal remote with backlit keys and learning function.

Product Search Screen

So Where do the Recommendations Come In?

Product and user embeddings are great, but unless a model is somehow jointly trained to create a common embedding space, the two entities can’t be compared semantically and the system can’t make recommendations.

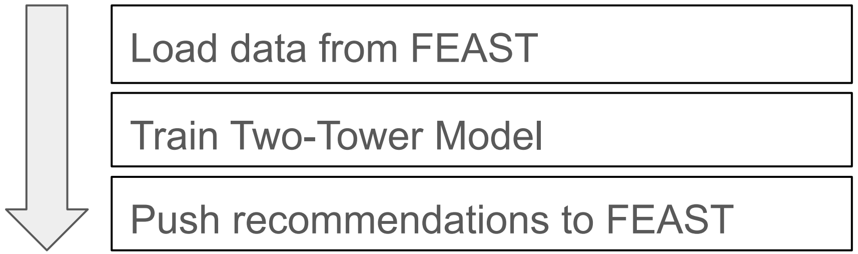
Fortunately, we’ve already covered the basic concepts to understand a solution to this problem:

- Dual Encoders (Two-Tower Model)
- Contrastive Training (Using our Heuristic Magnitude Loss Function)

Similar to the **openai/clip-vit-base-patch32** model, the Product Recommender creates a custom dual encoder to ensure the vectors for products and users are pushed closer if they have low magnitude scores. Given a user, we can now return nearby items as recommendations.

Kubeflow - ML Training Pipeline

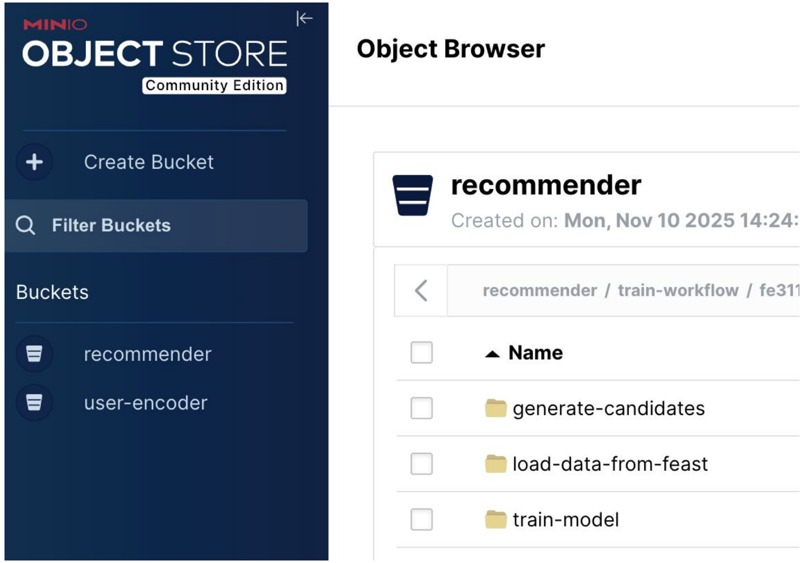
The Product Recommender leverages OpenShift’s integrated Kubeflow Pipeline (KFP) SDK to orchestrate training for its Two-Tower Recommendation Engine.



Benefits

- Python decorators are used to containerize Python functions that implement each pipeline stage. Containerization helps distribute work efficiently across pods.
- KFP manages shared data across pipeline stages using injected input/output function arguments and MinIO object storage.
- The OpenShift AI Dashboard makes it easy to work with this data and other artifacts.

Kubeflow - ML Training Pipeline


MINIO OBJECT STORE Community Edition

Create Bucket

Filter Buckets

Buckets

recommender

user-encoder

Object Browser

recommender

Created on: Mon, Nov 10 2025 14:24:

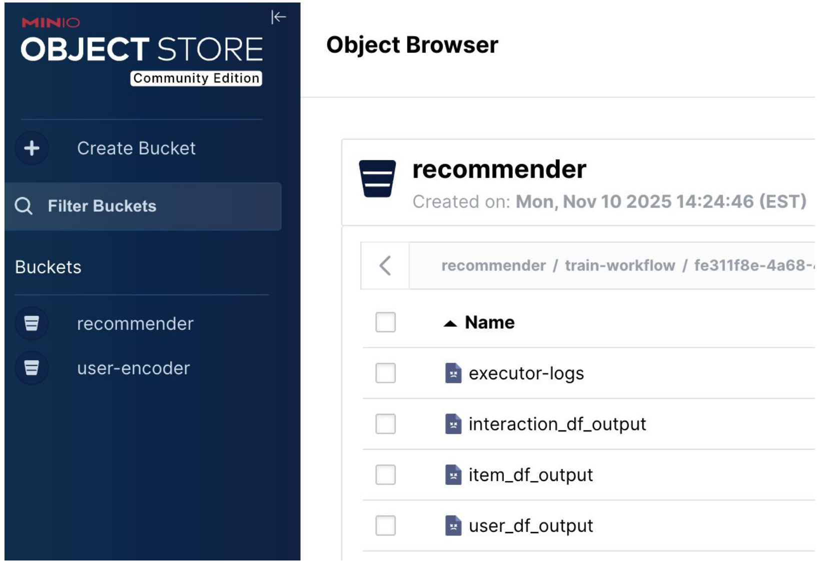
recommender / train-workflow / fe311

Name

generate-candidates

load-data-from-feast

train-model


MINIO OBJECT STORE Community Edition

Create Bucket

Filter Buckets

Buckets

recommender

user-encoder

Object Browser

recommender

Created on: Mon, Nov 10 2025 14:24:46 (EST)

recommender / train-workflow / fe311f8e-4a68-

Name

executor-logs

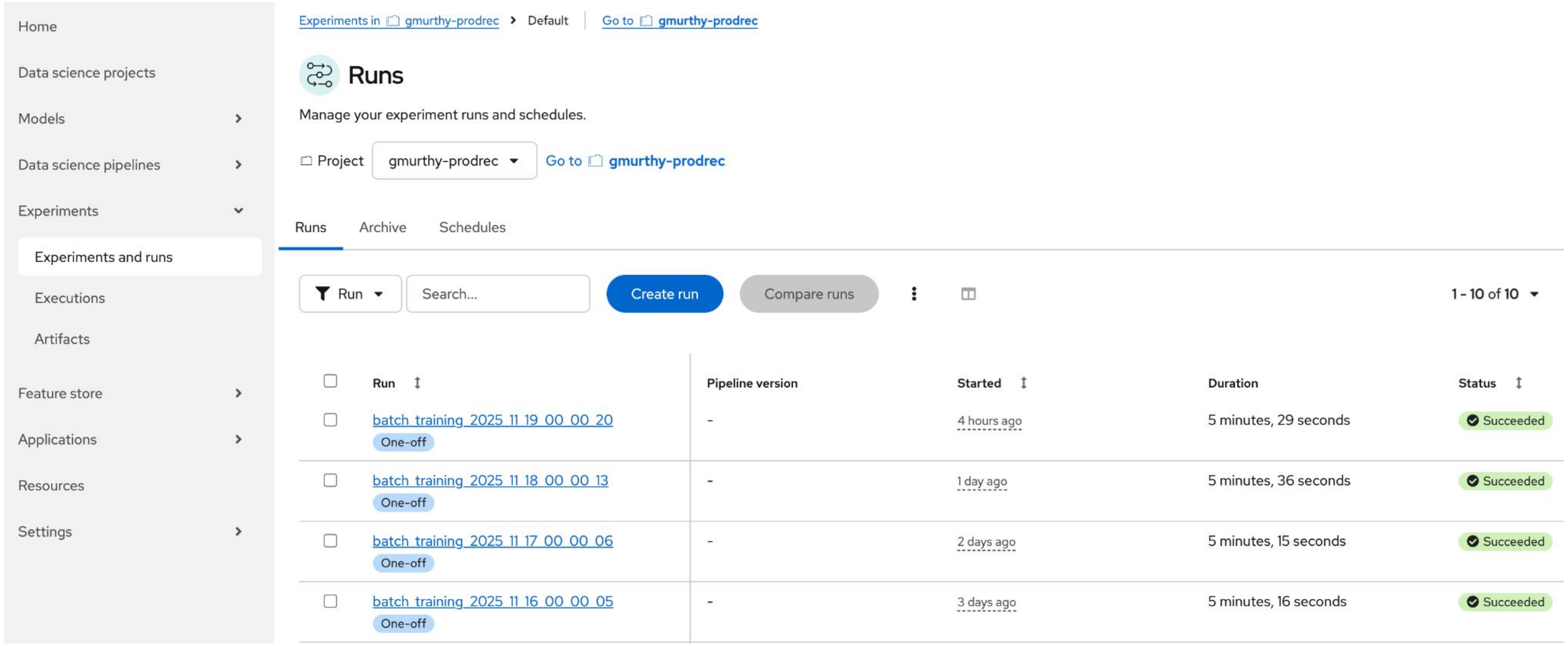
interaction_df_output

item_df_output

user_df_output

KFP Artifacts (logs, data from each stage, etc.) in MinIO

Kubeflow OpenShift AI Integration

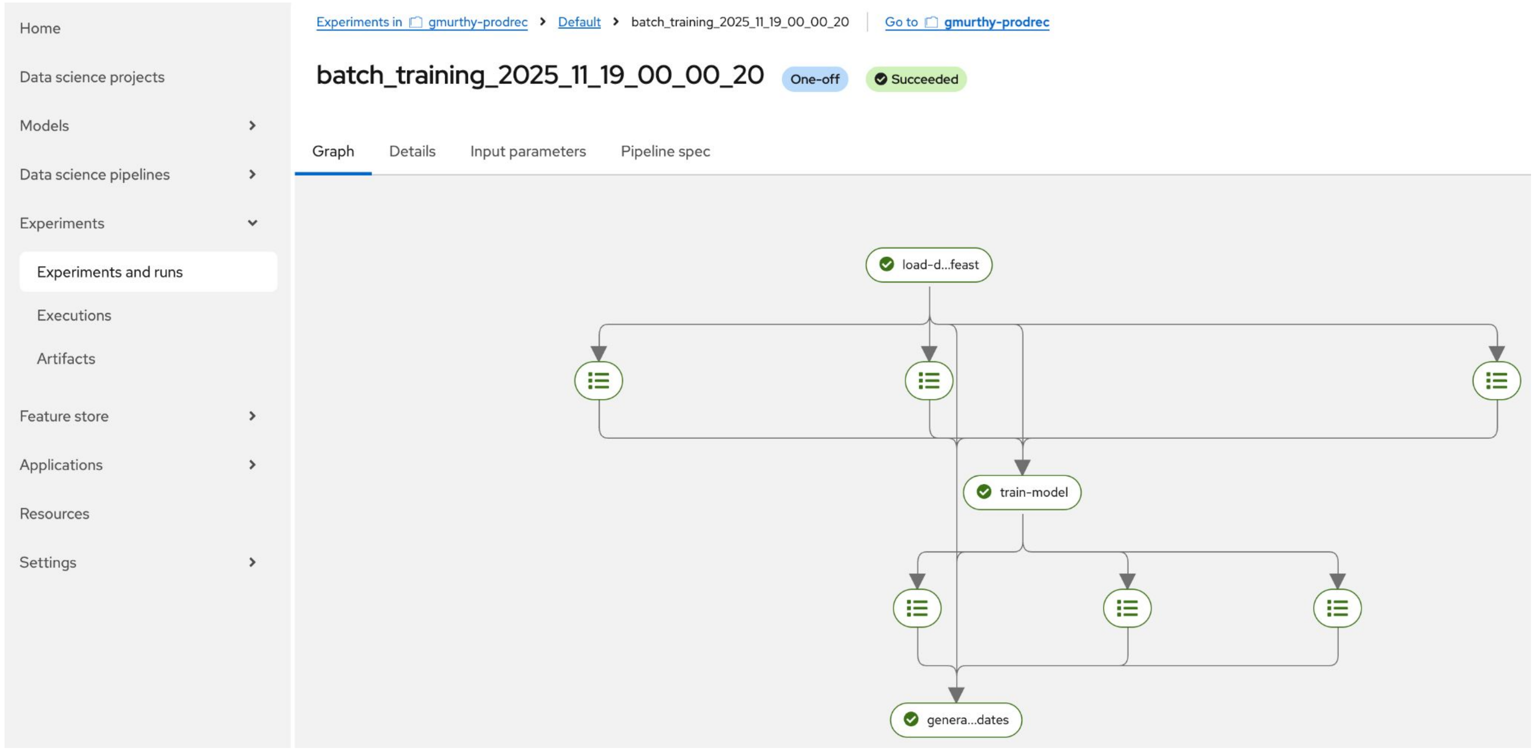


Manage your experiment runs and schedules.

Run	Pipeline version	Started	Duration	Status
batch_training_2025_11_19_00_00_20 One-off	-	4 hours ago	5 minutes, 29 seconds	Succeeded
batch_training_2025_11_18_00_00_13 One-off	-	1 day ago	5 minutes, 36 seconds	Succeeded
batch_training_2025_11_17_00_00_06 One-off	-	2 days ago	5 minutes, 15 seconds	Succeeded
batch_training_2025_11_16_00_00_05 One-off	-	3 days ago	5 minutes, 16 seconds	Succeeded

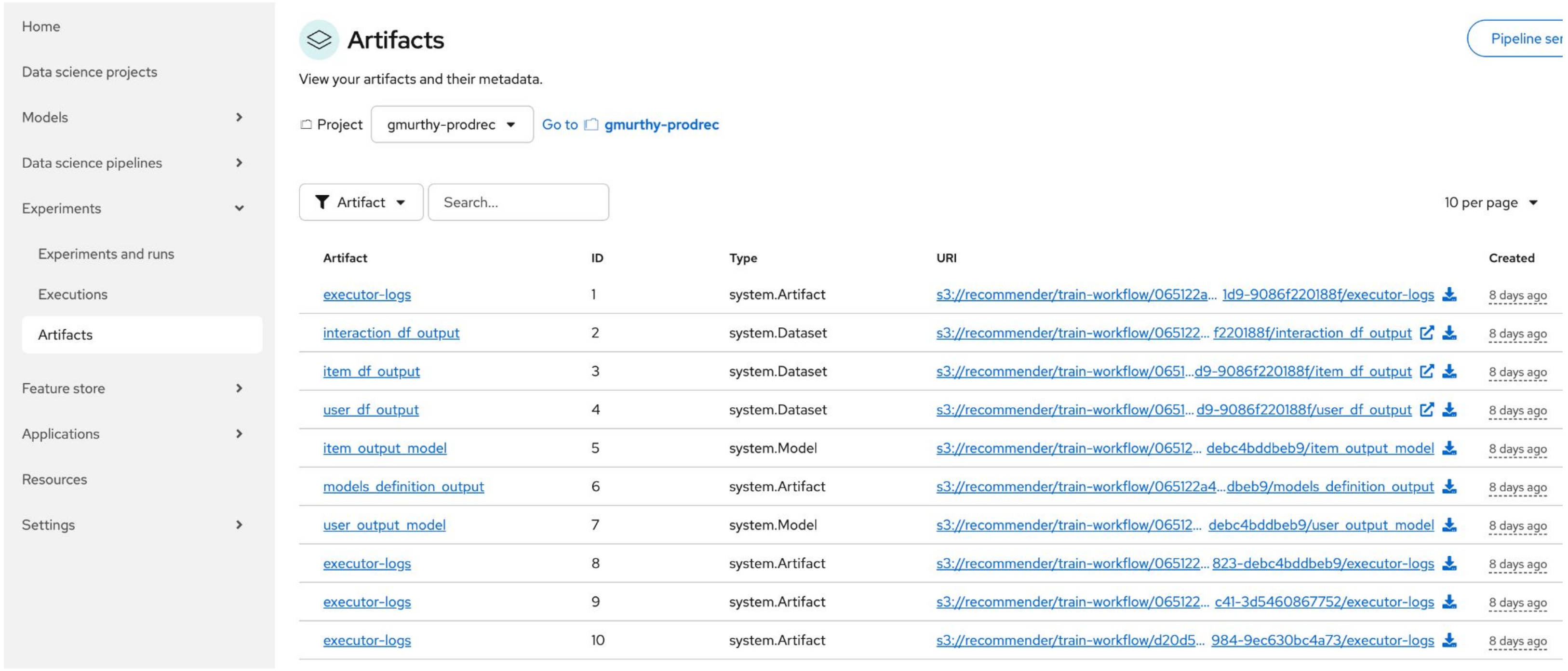
OpenShift AI Dashboard Integration Simplifies Job Troubleshooting and Visibility

Kubeflow OpenShift AI Integration



OpenShift AI Dashboard Visualization of Training Pipeline

Kubeflow OpenShift AI Integration



Artifacts

View your artifacts and their metadata.

Project: gmurthy-prodrec — Go to gmurthy-prodrec

Artifact | Search... | 10 per page

Artifact	ID	Type	URI	Created
executor-logs	1	system.Artifact	s3://recommender/train-workflow/065122a... 1d9-9086f220188f/executor-logs	8 days ago
interaction_df_output	2	system.Dataset	s3://recommender/train-workflow/065122... f220188f/interaction_df_output	8 days ago
item_df_output	3	system.Dataset	s3://recommender/train-workflow/0651...d9-9086f220188f/item_df_output	8 days ago
user_df_output	4	system.Dataset	s3://recommender/train-workflow/0651... d9-9086f220188f/user_df_output	8 days ago
item_output_model	5	system.Model	s3://recommender/train-workflow/06512... debc4bddbeb9/item_output_model	8 days ago
models_definition_output	6	system.Artifact	s3://recommender/train-workflow/065122a4...dbeb9/models_definition_output	8 days ago
user_output_model	7	system.Model	s3://recommender/train-workflow/06512... debc4bddbeb9/user_output_model	8 days ago
executor-logs	8	system.Artifact	s3://recommender/train-workflow/065122... 823-debc4bddbeb9/executor-logs	8 days ago
executor-logs	9	system.Artifact	s3://recommender/train-workflow/065122... c41-3d5460867752/executor-logs	8 days ago
executor-logs	10	system.Artifact	s3://recommender/train-workflow/d20d5... 984-9ec630bc4a73/executor-logs	8 days ago

OpenShift AI Dashboard to Manage Training Artifacts

Embeddings: Storage and Comparisons

LLMs are great for generating embeddings, but to use them effectively, we’ll need a solution that provides storage and query access.

In the next few slides we’ll see how OpenShift’s integrated FEAST feature store addresses these requirements.

FEAST Core Components

Data Sources

Let FEAST know the location and type of source data (FILE or PUSH_SOURCE).

```
users_source = FileSource(
    file_format=ParquetFormat(),
    path=os.path.join(data_path,
"recommendation_users.parquet"),
    timestamp_field="signup_date",
)


item_embed_push_source =
PushSource(
    name="item_embed_push_source",
batch_source=items_embed_dummy_sour
ce)
```

Feature Views

Views form a single-source-of-truth for feature definitions.

```
user_feature_view = FeatureView(
    name="user_features",
    entities=[user_entity],
    ttl=timedelta(days=365 * 6),
    schema=[
        Field(name="user_id",
dtype=String),
        Field(name="user_name",
dtype=String),
        Field(name="preferences",
dtype=String),
    ],
    source=users_source,
    online=False,
)
```

Feature Service

Built on one or more joined feature views, feature services provide a stable API layer for applications.

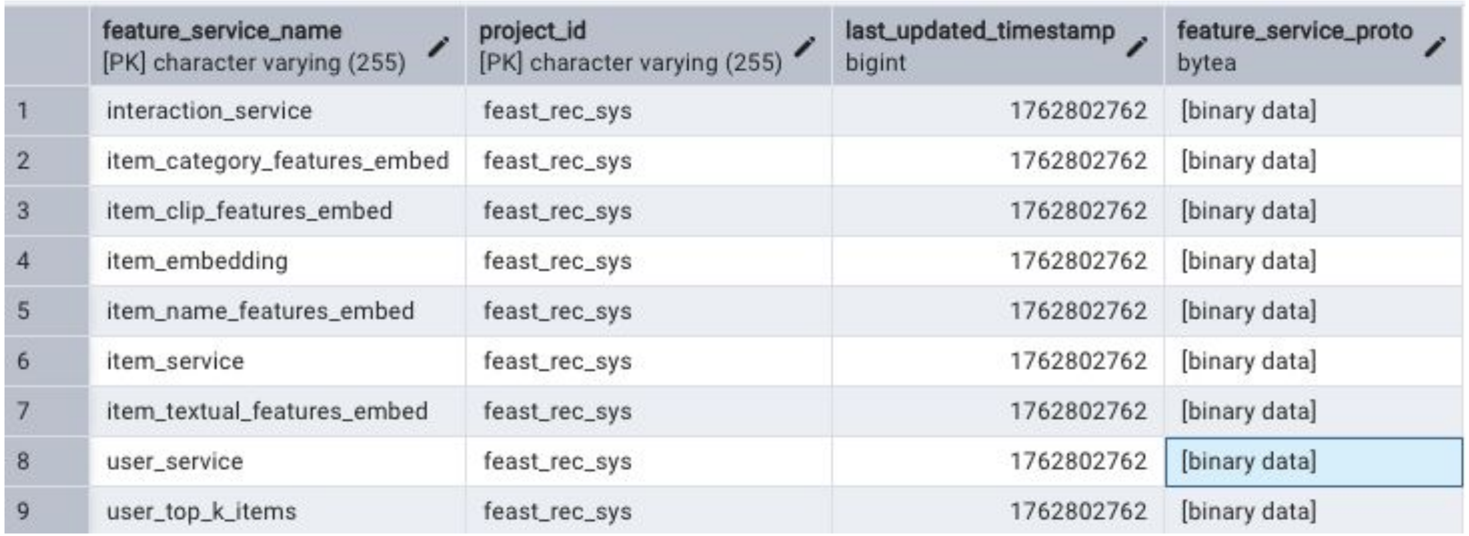
```
from feast import FeatureService
user_feature_service =
FeatureService(name="user_service",
features=[user_feature_view])
```

Client Code

Retrieve data from FEAST.

```
from feast import FeatureStore
self._user_df =
store.get_historical_features(
            entity_df=user_entity_df,
features=user_service).to_df()
```

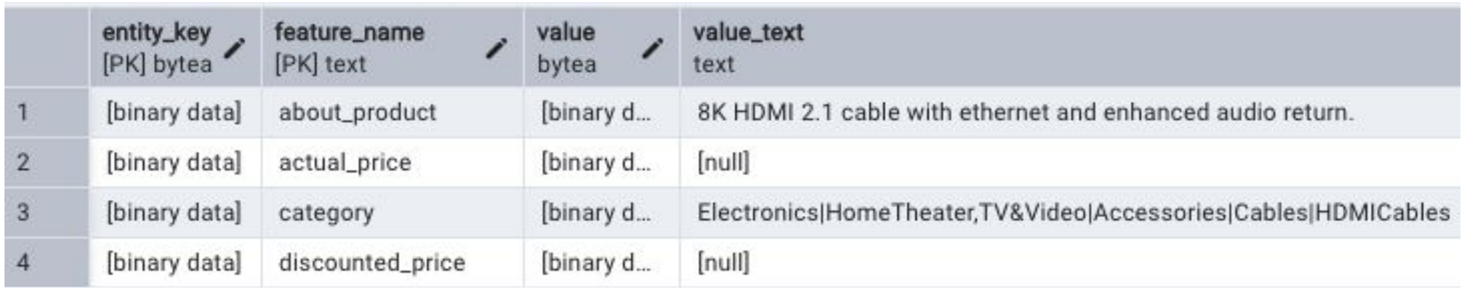
FEAST Backend (Postgres)



	feature_service_name [PK] character varying (255)	project_id [PK] character varying (255)	last_updated_timestamp bigint	feature_service_proto bytea
1	interaction_service	feast_rec_sys	1762802762	[binary data]
2	item_category_features_embed	feast_rec_sys	1762802762	[binary data]
3	item_clip_features_embed	feast_rec_sys	1762802762	[binary data]
4	item_embedding	feast_rec_sys	1762802762	[binary data]
5	item_name_features_embed	feast_rec_sys	1762802762	[binary data]
6	item_service	feast_rec_sys	1762802762	[binary data]
7	item_textual_features_embed	feast_rec_sys	1762802762	[binary data]
8	user_service	feast_rec_sys	1762802762	[binary data]
9	user_top_k_items	feast_rec_sys	1762802762	[binary data]

feature_services table keeps track of all service definitions.

FEAST Backend (Postgres)



	entity_key [PK] bytea	feature_name [PK] text	value bytea	value_text text
1	[binary data]	about_product	[binary d...	8K HDMI 2.1 cable with ethernet and enhanced audio return.
2	[binary data]	actual_price	[binary d...	[null]
3	[binary data]	category	[binary d...	Electronics|HomeTheater,TV&Video|Accessories|Cables|HDMICables
4	[binary data]	discounted_price	[binary d...	[null]

EAV table for Product Features

Online features (e.g,. PUSH_SOURCE views) are stored in dedicated tables in an efficient Entity-Attribute-Value (EAV) format, which is suitable for sparse columns or dynamic column definitions (ability to add/remove features without schema changes)

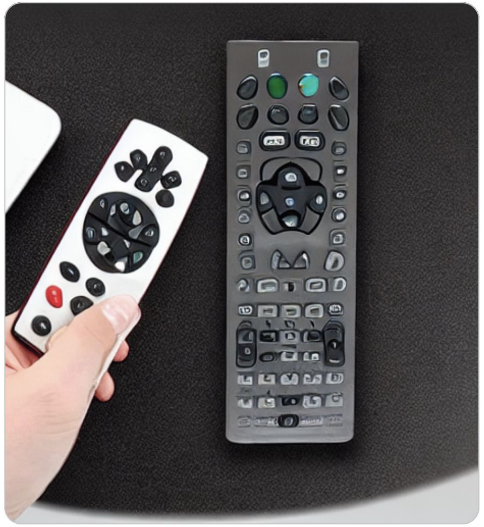
FEAST Benefits

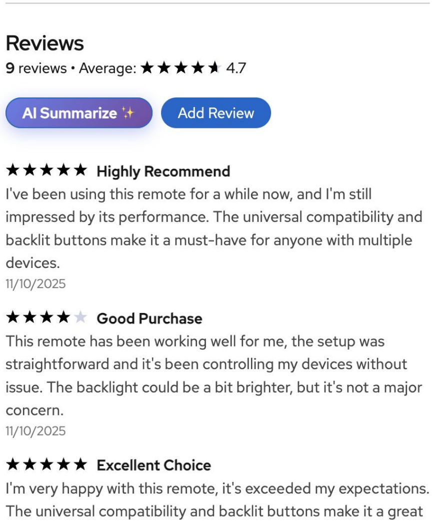
What are the benefits of FEAST’s architecture:

- Helps prevent feature mismatch between training and inference time (Training-Serving-Skew) by creating a single source of truth for feature definitions and enabling developers to “version” feature sets. For example, as a feature set evolves over time to accommodate new clients, new feature services and views can be created without disrupting existing clients using existing services and views.
- Provides the unified client API we saw earlier to work with feature data.
- Supports temporal queries (e.g., what was the state of the database when an event occurred), helping to prevent data leakage.
- Provides some built-in capabilities storing and comparing vectors.

Next, let’s take a look at some generative capabilities in Product Recommender

Product Review Summaries




Reviews
9 reviews • Average: ★★★★★ 4.7

★★★★★ Highly Recommend
I've been using this remote for a while now, and I'm still impressed by its performance. The universal compatibility and backlit buttons make it a must-have for anyone with multiple devices.
11/10/2025

★★★★☆ Good Purchase
This remote has been working well for me, the setup was straightforward and it's been controlling my devices without issue. The backlight could be a bit brighter, but it's not a major concern.
11/10/2025

★★★★★ Excellent Choice
I'm very happy with this remote, it's exceeded my expectations. The universal compatibility and backlit buttons make it a great

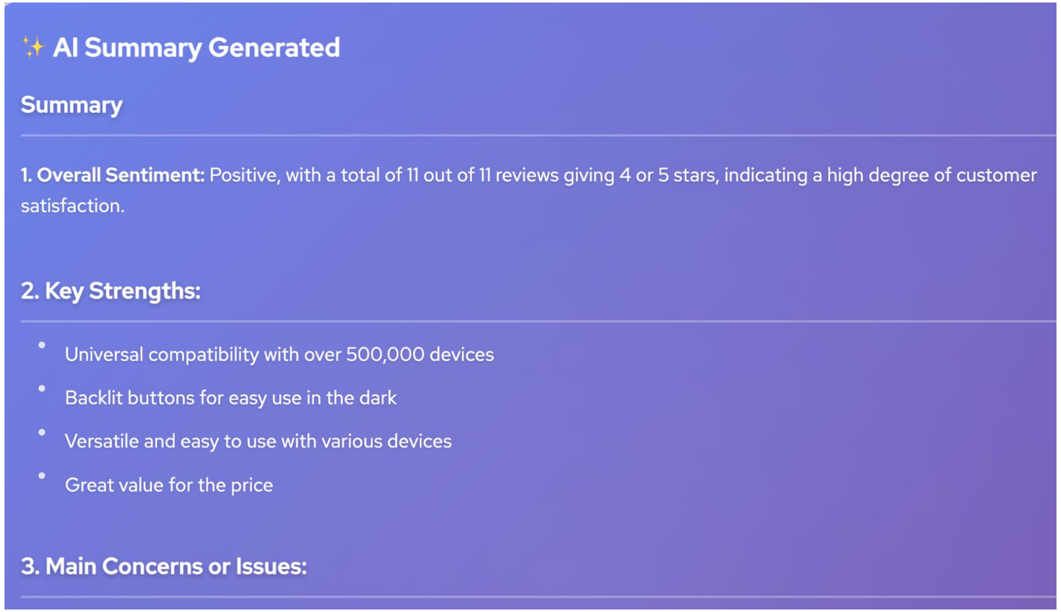
Product Review Page for an Item

The Product Recommender quickstart enables users to create reviews (like those shown to the left).

On most retail sites, however, there are usually more reviews per product than users typically have time to consider.

LLMs tailored for chat completion, like llama-3.1-8b-Instruct, can help!

Product Review Summaries


✨ AI Summary Generated

Summary

1. **Overall Sentiment:** Positive, with a total of 11 out of 11 reviews giving 4 or 5 stars, indicating a high degree of customer satisfaction.

2. **Key Strengths:**

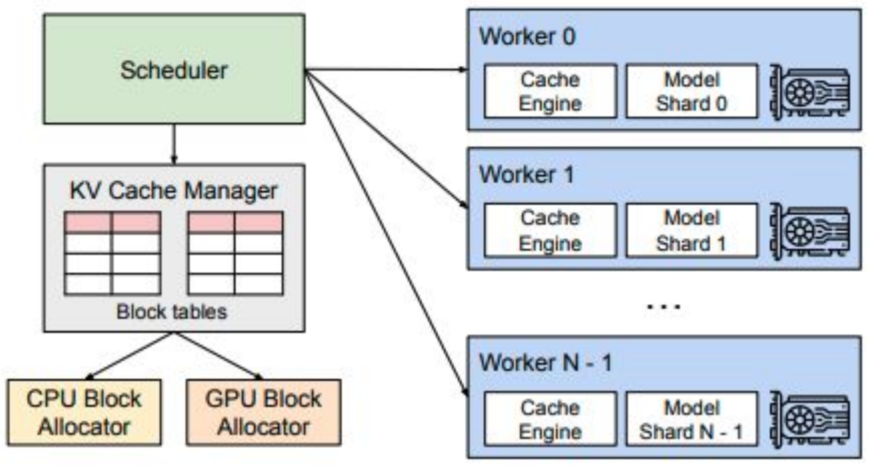
- Universal compatibility with over 500,000 devices
- Backlit buttons for easy use in the dark
- Versatile and easy to use with various devices
- Great value for the price

3. **Main Concerns or Issues:**

Product Review Summary for an Item

Users can click on the AI Summarize link in the item review page to ingest and summarize all the product reviews on the fly. This feature is dynamic so as reviews are added, the summaries reflect the new information. But how would this scale with thousands of reviews per product?

Red Hat Open Shift AI Inference Server



System Overview of OpenShift AI's vLLM Inference Server (Efficient Memory Management for Large Language Model Serving with Paged Attention, Kwon et. al., 2023)

OpenShift AI's inference server boosts performance in several ways, including intelligent continuous batching, paged attention and KV cache management.

Efficient Utilization of Accelerators

Since Product Recommender's inference server is integrated within the OpenShift cluster, sharing the model and its hardware resources across clients is straightforward.

Stratified Sampling

Llama-3.1-8B-Instruct can support up to a 128k context window size (if the underlying hardware accelerator has adequate memory). This is roughly 100k tokens and is very large, but would easily be exceeded in a production online retail system.

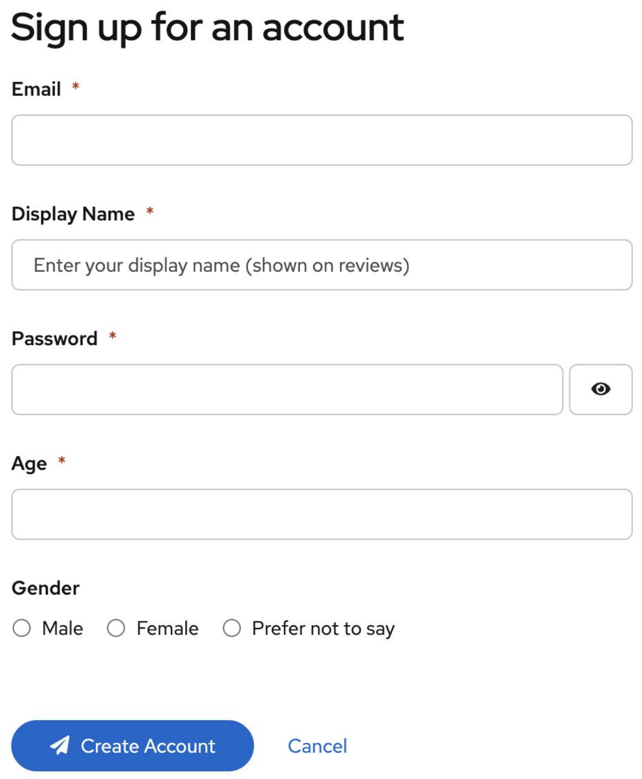
To avoid unbounded reviews that would exceed the LLM's context length, the Product Recommender employs several techniques:

- The number of reviews is capped using an environment variable.
- Ratings are first sorted by date to ensure the LLM receives the most recent reviews.
- Stratified sampling is used over ranking scores (1-5) to ensure even coverage across review sentiments.

User Registration and Onboarding

Before the Product Recommender can build a model of a user's preferences, it needs to build a sequence of user-product interactions. Moreover, the recommendation models are built periodically offline. Hence, when a user first registers, the system won't have any recommendations for him.

The Product Recommender employs a smooth onboarding workflow together with data from other users' interactions to address this challenge.


Sign up for an account

Email *

Display Name *

Enter your display name (shown on reviews)

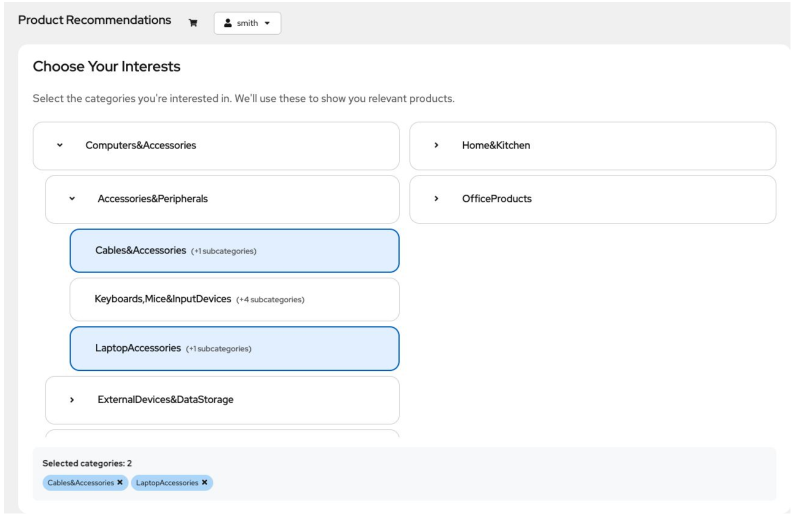
Password *

Age *

Gender

○ Male ○ Female ○ Prefer not to say

Product Category Preferences


Product Recommendations smith

Choose Your Interests

Select the categories you're interested in. We'll use these to show you relevant products.

- Computers&Accessories
  - Accessories&Peripherals
    - Cables&Accessories (+1 subcategories)
    - Keyboards,Mice&InputDevices (+4 subcategories)
    - LaptopAccessories (+1 subcategories)
  - ExternalDevices&DataStorage
- Home&Kitchen
- OfficeProducts

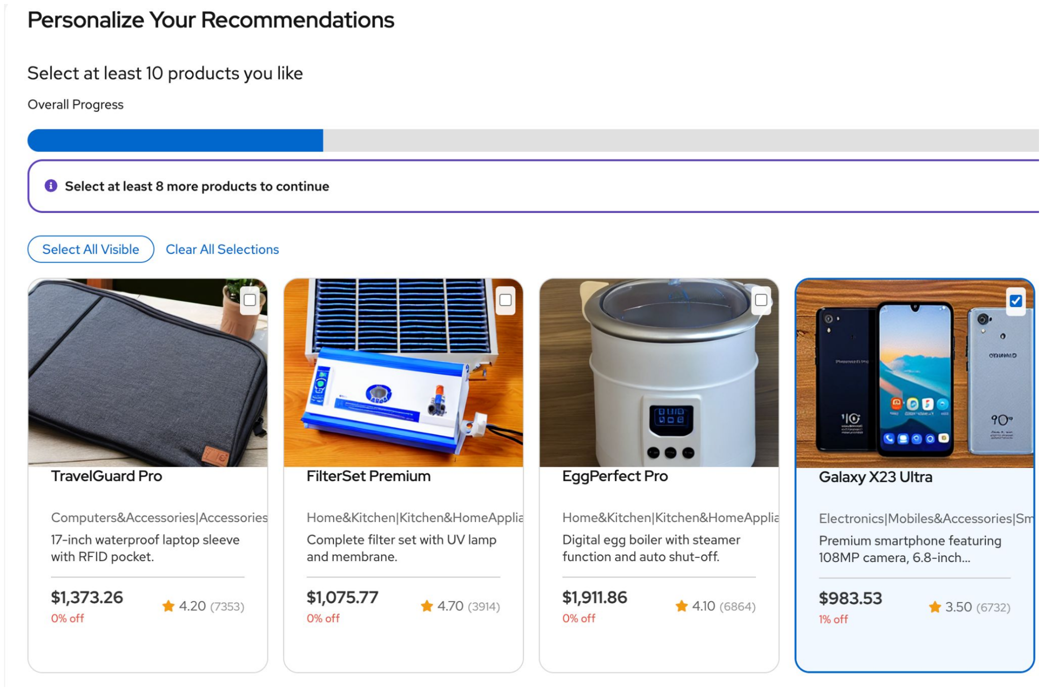
Selected categories: 2

Cables&Accessories ✕ LaptopAccessories ✕

Onboarding begins by asking the new user which categories of products he is interested in. These categories are used in two ways:

- To filter the list of products shown to the user in the next step (see next slide)
- To show the user an initial list of recommendations based on the most popular products within the selected categories.

User Registration and Onboarding



Personalize Your Recommendations

Select at least 10 products you like

Overall Progress

Select at least 8 more products to continue

TravelGuard Pro
Computers&Accessories|Accessories
17-inch waterproof laptop sleeve with RFID pocket.
$1,373.26 0% off ★ 4.20 (7353)

FilterSet Premium
Home&Kitchen|Kitchen&HomeApplia
Complete filter set with UV lamp and membrane.
$1,075.77 0% off ★ 4.70 (3914)

EggPerfect Pro
Home&Kitchen|Kitchen&HomeApplia
Digital egg boiler with steamer function and auto shut-off.
$1,911.86 0% off ★ 4.10 (6864)

Galaxy X23 Ultra
Electronics|Mobiles&Accessories|Sm
Premium smartphone featuring 108MP camera, 6.8-inch...
$983.53 1% off ★ 3.50 (6732)

Users are then guided to select at least 10 products within the selected categories.

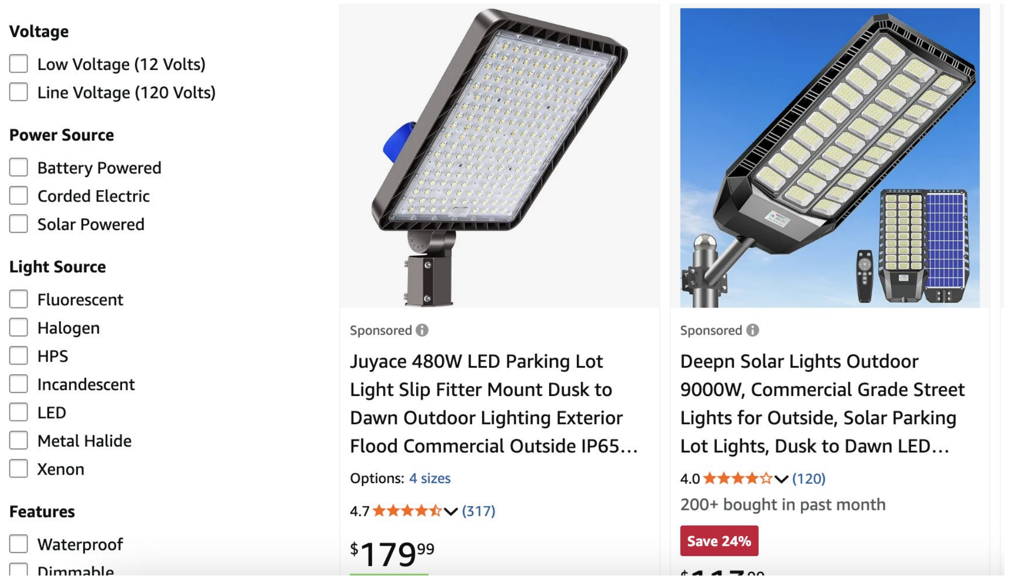
The system uses these interactions in the next scheduled build of the recommendation model.

Conclusions and Next Steps

- We’ve seen how OpenShift AI and the Product Recommender provide a quickstart on creating a recommendation system.
  - Red Hat OpenShift AI’s Inference server is used to serve LLMs for product summarization.
  - Red Hat OpenShift AI’s integrated Kubeflow Pipeline orchestrates the training of the recommender’s two-tower model.
  - Red Hat OpenShift’s FEAST capability provides a solid data access API for working with the product catalog and its semantic vector counterparts.
  - Finally, OpenShift itself provides a high-performance platform to run all the application’s microservices, including hybrid search, KFP jobs, and user onboarding – centralizing and leveraging an organization’s infrastructure investment
- We’ve reviewed some of the technical considerations for choosing embedding models, the backbone of a recommendation system.
- We’ve delved in to more advanced generative features, like product review summarization.

Conclusions and Next Steps

- Expand on the generative capabilities to dynamically create a faceted search for product families and search results. Manually creating a faceted search for product families is very time consuming; applying an LLM to create these facets automatically for any set of search results would greatly reduce development costs and improve customer experience.
- Employ re-ranking models to sort search results and recommendations more intelligently based on the user's preferences and prior interactions.


Voltage
- ☐ Low Voltage (12 Volts)
- ☐ Line Voltage (120 Volts)

Power Source
- ☐ Battery Powered
- ☐ Corded Electric
- ☐ Solar Powered

Light Source
- ☐ Fluorescent
- ☐ Halogen
- ☐ HPS
- ☐ Incandescent
- ☐ LED
- ☐ Metal Halide
- ☐ Xenon

Features
- ☐ Waterproof
- ☐ Dimmable

Sponsored
Juyace 480W LED Parking Lot Light Slip Fitter Mount Dusk to Dawn Outdoor Lighting Exterior Flood Commercial Outside IP65…
Options: 4 sizes
4.7 (317)
$179.99

Sponsored
Deepn Solar Lights Outdoor 9000W, Commercial Grade Street Lights for Outside, Solar Parking Lot Lights, Dusk to Dawn LED…
4.0 (120)
200+ bought in past month
Save 24%

Example of faceted search (Amazon)

More Information and Source Code

Red Hat

Source code: https://github.com/rh-ai-quickstart/product-recommender-system

FEAST

https://www.redhat.com/en/blog/feast-open-source-feature-store-ai

https://docs.feast.dev/

Kubeflow

https://www.redhat.com/en/topics/cloud-computing/what-is-kubeflow

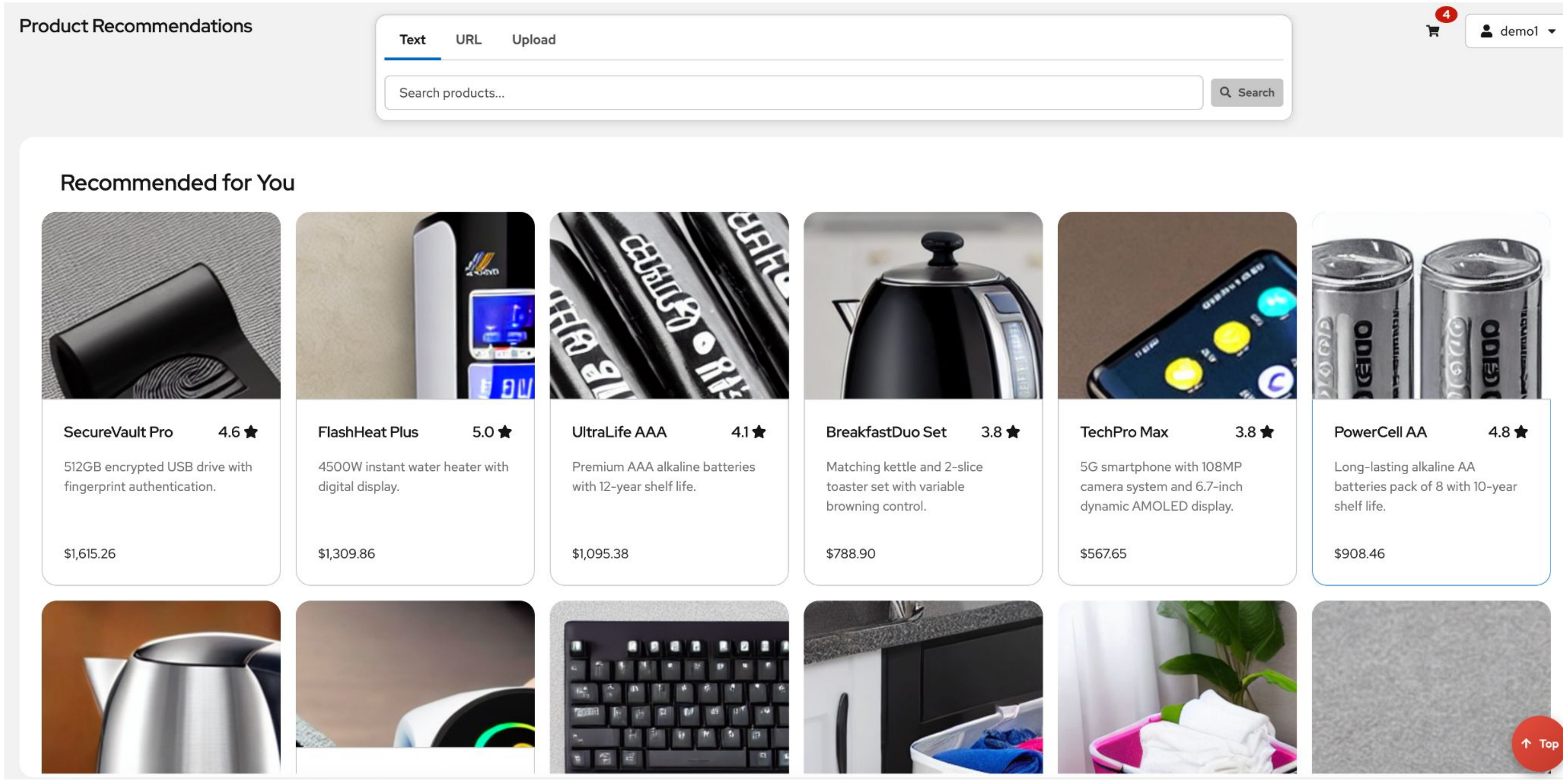
https://www.kubeflow.org/docs/

OpenShift AI Inference Server

https://www.redhat.com/en/topics/ai/what-is-vllm

https://docs.vllm.ai/en/latest/

Demo



Product Recommendations

Recommended for You

SecureVault Pro 4.6 ★
512GB encrypted USB drive with fingerprint authentication.
$1,615.26

FlashHeat Plus 5.0 ★
4500W instant water heater with digital display.
$1,309.86

UltraLife AAA 4.1 ★
Premium AAA alkaline batteries with 12-year shelf life.
$1,095.38

BreakfastDuo Set 3.8 ★
Matching kettle and 2-slice toaster set with variable browning control.
$788.90

TechPro Max 3.8 ★
5G smartphone with 108MP camera system and 6.7-inch dynamic AMOLED display.
$567.65

PowerCell AA 4.8 ★
Long-lasting alkaline AA batteries pack of 8 with 10-year shelf life.
$908.46