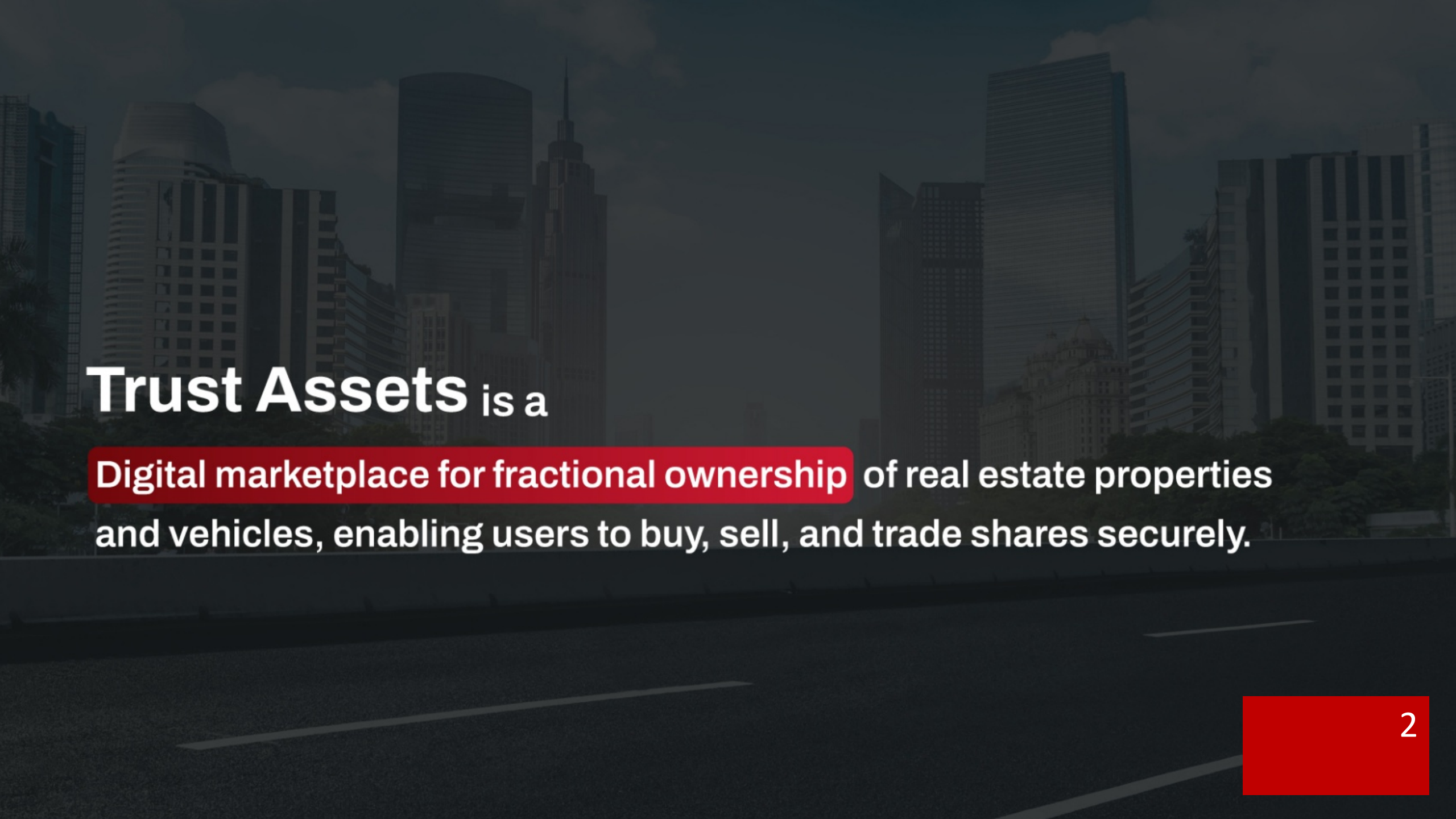

TRUST
ASSETS


Trust Assets is a
Digital marketplace for fractional ownership of real estate properties
and vehicles, enabling users to buy, sell, and trade shares securely.

# Recommendation System

| Reference | Year | Data | Preprocessing | Model | Result |
|---|---|---|---|---|---|
| (Muñoz, Meza, & Ventura, 2025) | 2025 | Spatial distances (health, education, security, etc.) + housing characteristics (price, size, age) | Normalization, logarithmic transformation, factor analysis, dimensionality reduction (PCA) | Fuzzy C-Means (FCM) Clustering | Identified 4 clusters (Jambu's elbow method); improved segmentation accuracy. |
| | | Geospatial data (QGIS) + web-scraped housing features | Spatial distance calculation, variable transformation | Bat Algorithm (Metaheuristic Optimization) | Execution time: 1.38–1.42 seconds (fastest); reduced search time by 15% vs. PSO/GA. |
| | | User preferences (sliders) + housing clusters | Cosine similarity, graph-based memory | Hybrid Recommender System (Content-Based + Collaborative Filtering) | Precision: 81.6% (improved relevance), Recall: 89% (retrieved 89% relevant items). |
| | | Spatial factors (latent variables) + user ratings | Fuzzy membership assignment, ANOVA validation | Spatial Fuzzy Logic Model | User Satisfaction: 58% (2.9/5), highlighting need for collaborative filtering |

| Reference | Year | Data | Preprocessing | Model | Result |
|---|---|---|---|---|---|
| **(C, Oberoi, Goyal, & Sikka, 2024)** | 2024 | Real-world data (3M users, 1.1M properties) | Cohort creation (locality, price/area bins), scoring (leads, conversions). | Rule-Based Engine | Latency <40 ms (Ensured real-time cold-start recommendations) |
| | | | Binning (price, area), activity weighting (Table 1), binary encoding. | Content Filtering | MAP@6 = 0.881 (Buy) (Improved short-term precision by 3.2%) |
| | | | Interaction scoring (CRF=10, scrolling=1), ALS training (daily). | Collaborative Filtering (ALS) | NDCG = 0.685 (Buy) (Optimized ranking for long-term preferences) |
| | | | Combined preprocessing (content + collaborative steps). | Hybrid Model | Latency = 29.3 ms (Balanced accuracy + speed for hybrid users) |

| Reference | Year | Data | Preprocessing | Model | Result |
|---|---|---|---|---|---|
| **(Arif, Muhtarom, & Nurhayati, 2023)** | 2023 | 50 questionnaire entries (14 criteria) | | Cosine-based similarity | 63.8% |
| | | | | Adjusted Cosine similarity | 70.4% |
| | | | | Pearson correlation | 88.7% |
| | | | | Spearman correlation | 75.57% |
| **(Boteju & Munasinghe, 2020)** | 2020 | Hybrid Recommender:<br>- User data (1,873 survey records)<br>- Item data (98 vehicle models) | - Null value removal<br>- Categorical-to-numeric conversion<br>- Feature engineering | Multiclass Neural Network (1 hidden layer, 1,000 nodes, 160 iterations) | 96% accuracy (Improved recommendation accuracy by capturing linear/non-linear data patterns) |
| | | NLP Sentiment Analysis:<br>- 14,000 labeled Twitter reviews | - Tokenization<br>- Lemmatization<br>- Stop-word removal<br>- Text cleaning | Naive Bayes Classifier | 92.73% test accuracy (Improved sentiment classification via labeled tweet analysis) |

| Reference | Year | Data | Preprocessing | Model | Result |
|---|---|---|---|---|---|
| **(Zhang, et al., 2019)** | 2019 | Data1: 1-week sale data (Sydney) | Filtered items with <2 interactions | Stage 1 (Content-based) | Precision@5: 1.65%<br>Improved baseline (1.03%) by 60% |
| | | | One-hot encoding for categorical/numerical features | | |
| | | | Imputed missing prices | | |
| | | Data2: 2-week sale data (Sydney) Scenario 1/2 | Same as Data1 | | Precision@5: 3.75%<br>Improved baseline (2.28%) by 64.5% |
| | | | Time-based split for Scenario 2 | | |
| | | Data2: 2-week sale data (Sydney) Scenario 3 (Cold-start) | Training/test split to exclude new items in training | Full two-stage model | Recall@50: 4.71%<br>Baseline failed (0% recall) |
| | | Data3: 3-month data (user attributes) | Accumulated user features (e.g., persona, engagement level) | Stage 2 (XGBoost spam filtering) | Precision@5: Improved from 3.75% to 4.2%<br>Reduced spam notifications |
| | | | One-hot encoding | | |

# Intelligent Real Estate Recommendations

A Content-Based System Powered by Deep Learning Embeddings

# 1 The Challenge: Finding the Perfect Property

## Our Goal

To create a smart system that understands the nuanced features of a property and recommends truly similar listings, saving users time and effort.

# 2 Our Two-Phase System Architecture

## Offline Training

This is where we process data and train our model periodically. This is the "learning" phase.

## Online Serving

This is the live API that uses the pre-trained model to deliver real-time recommendations to our web and mobile apps.

# 3 The Offline Engine - Building the Recommendation Intelligence

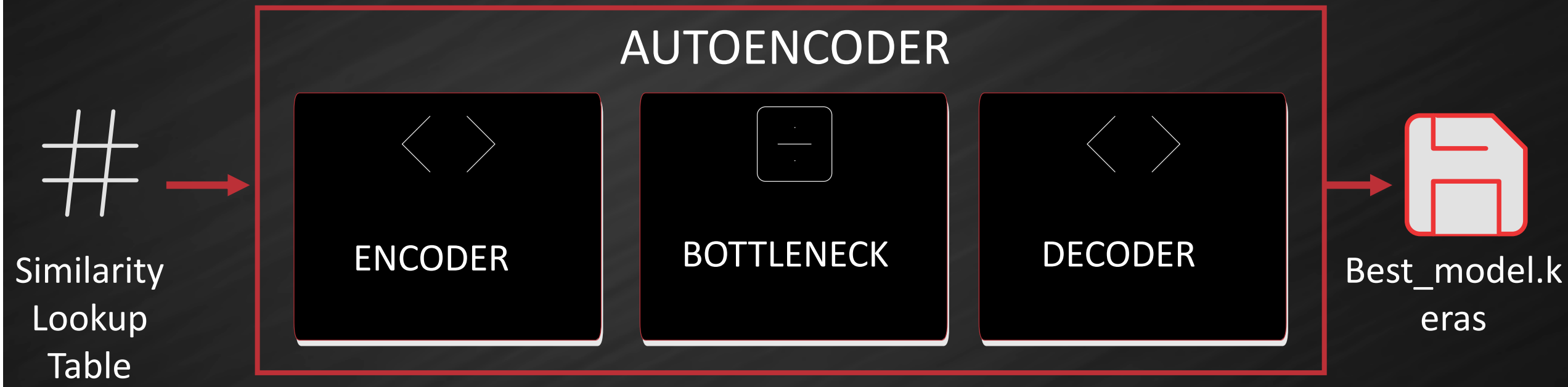
# 4 Step 1 - Preprocessing Pipeline

1. Initial Data Loading & Correction
2. Feature Engineering
3. Missing Data Imputation
4. Skewness Correction and Outlier Treatment
5. Categorical Encoding and Feature Scaling

## 5 Step 2: Learning Property "DNA" with an Autoencoder

Our system uses an Autoencoder neural network to create a unique "fingerprint" for each property. This fingerprint, called an embedding, captures the property's most important features by compressing them into a dense vector.

# 5 Step 2: Learning Property "DNA" with an Autoencoder



# 5 Model Optimization & Experiments

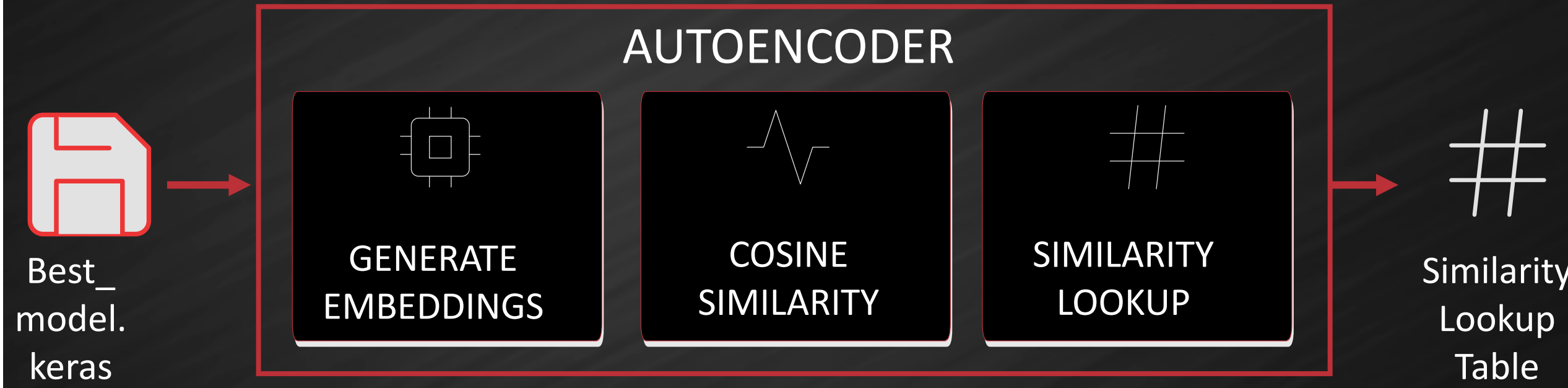
We conducted several experiments to find the optimal model architecture.

| Experiment | Embedding Dim | Embedding Activation | Hidden Layers (Neurons) | Dropout Rate | Best Val MSE |
|---|---|---|---|---|---|
| Baseline | 10 | 'relu' | (256, 128) | 0.3 | 0.0064 |
| Exp 1 | 16 | 'relu' | (256, 128) | 0.3 | 0.0067 |
| Exp 2 | 10 | 'linear' | (256, 128) | 0.3 | 0.0048 |
| Exp 3 | 10 | 'linear' | (512, 256) | 0.3 | 0.0016 |
| **Exp 4 (Final)** | **10** | **'linear'** | **(512, 256)** | **0.2** | **0.0009** |

# 7 Creating the Similarity Cache

1. Generate property "fingerprints" (embeddings).
2. Calculate similarity between all fingerprints.
3. Store top 50 similar properties for quick lookup.

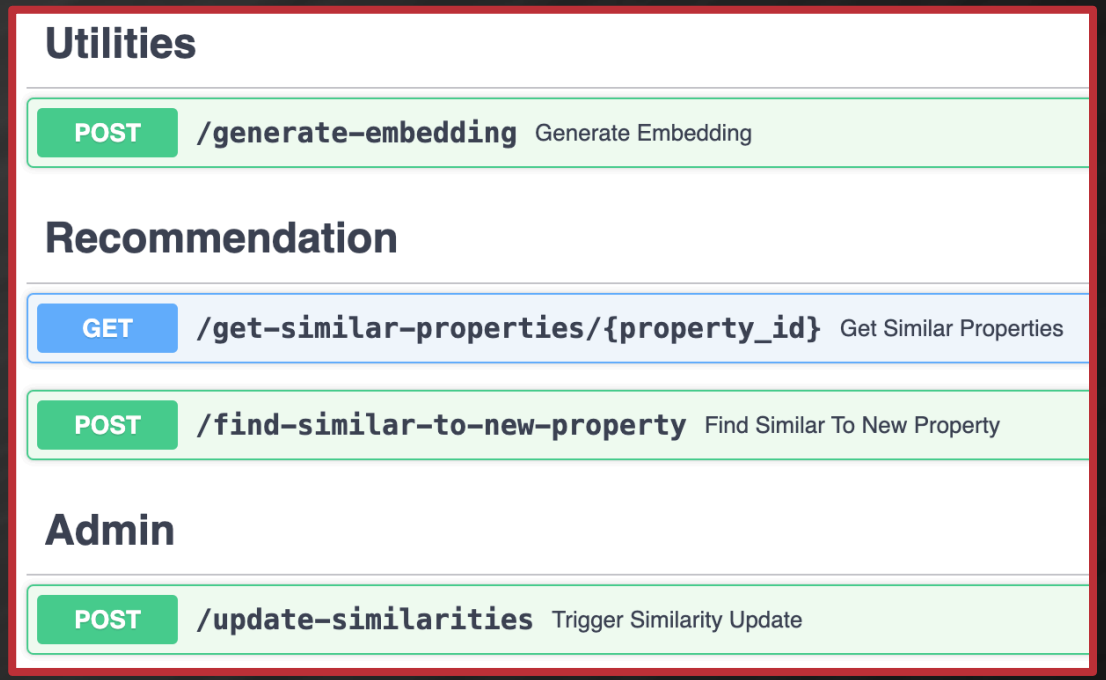
# 7 Creating the Similarity Cache



# 8 Phase 2: The Live Recommendation API (FastAPI)

On startup, the API loads all offline artifacts (model, pipeline, similarity lookup) into memory for maximum speed.

Exposes simple, well-defined endpoints for getting recommendations.

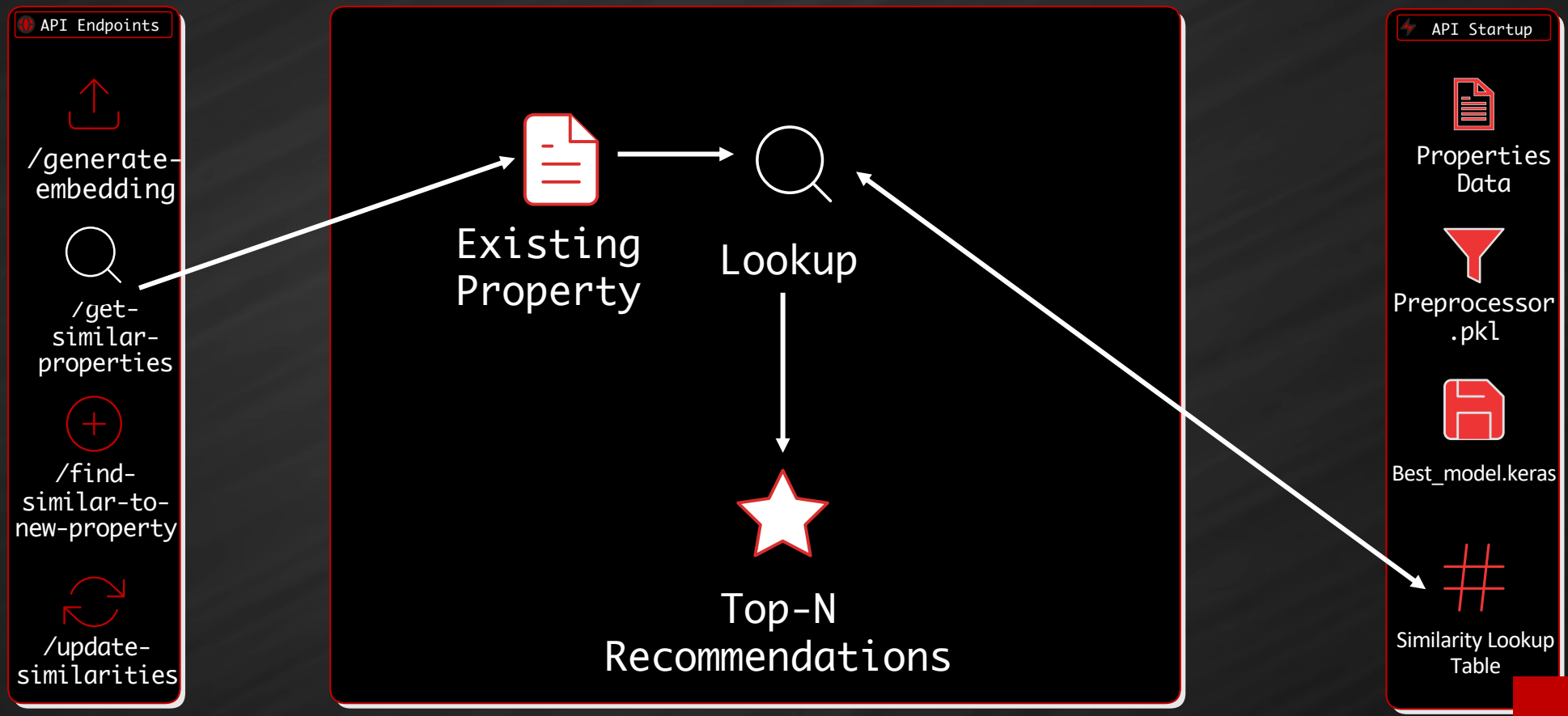


## 8 How the API Delivers Recommendations

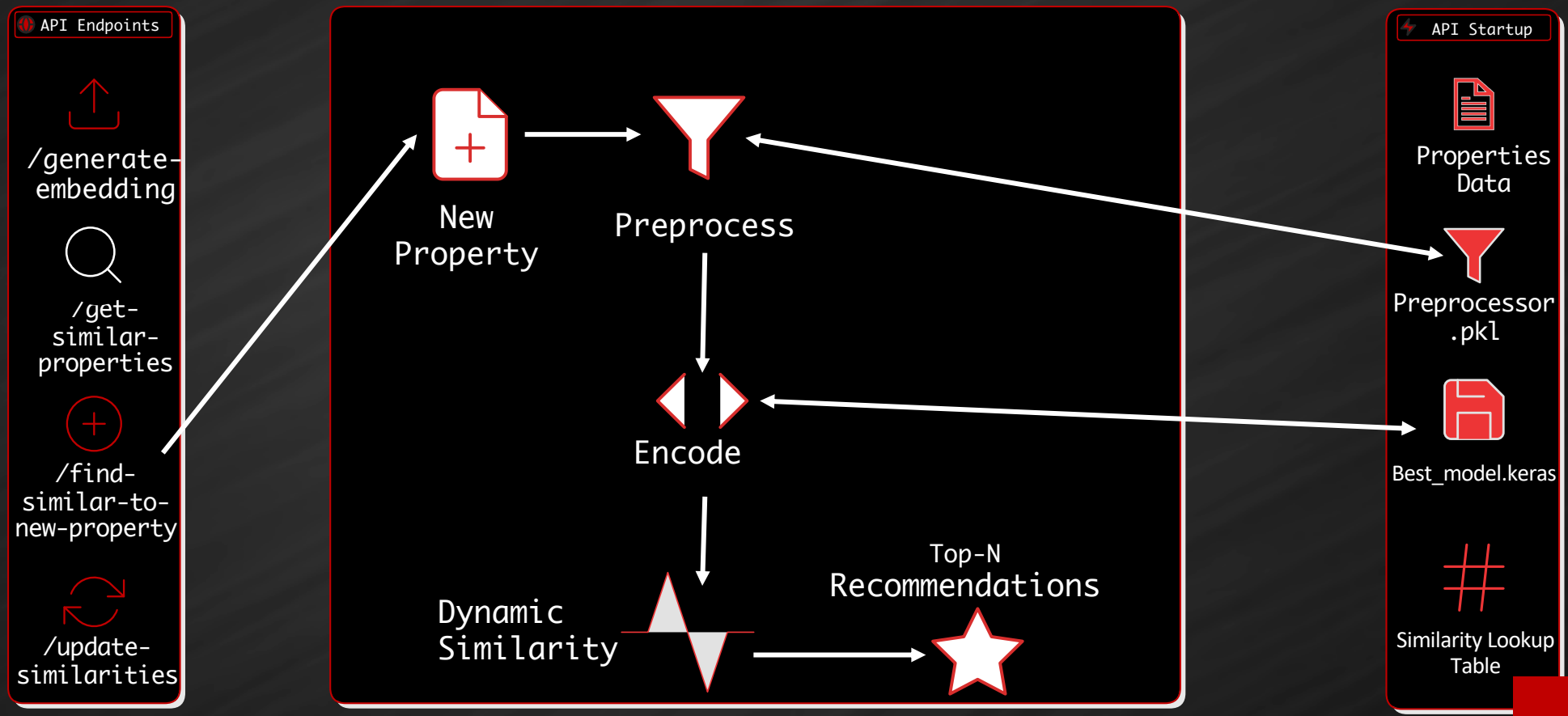
**Scenario 1: Existing Property**

1. Receives a property_id.
2. Performs an instant lookup in the cached **Similarity Table**.
3. Returns the pre-calculated top N similar items

# 8 Scenario 1: Existing Property



# 8 Scenario 2: New Property



# 9 Result

We built an end-to-end system that uses deep learning embeddings to provide highly relevant, content-based property recommendations.

# Vehicle Recommendation System

Powering Personalized Discovery with Collaborative and Content-Based Filtering

## 1 The Goal: Increase Sales by Enhancing User Experience

**Problem:** Our massive inventory is overwhelming for users.

**Goal:** Guide users to the right car, boosting satisfaction and sales.

# 2 Our Solution: A Smart Recommendation Engine

## Initial Methodology

- Item-Item **Collaborative Filtering**

## Why

- It's a powerful, industry-standard technique that leverages community behavior (user ratings) to find similar items.

## The Plan

- Analyze user ratings to understand which vehicles are perceived as similar by the community.

# 3 Data: The Fuel for Our Engine

## Data Sources:

| Domain | Name | Description | Source | Size | Year |
|---|---|---|---|---|---|
| **Vehicle** | Vehicle Sales Data | US used vehicle sales data including price, condition, seller, and key vehicle attributes (make, model, etc.). | Kaggle | 558,837 records | 2024 |
| **Vehicle** | Vehicle Reviews Data | Consolidated user reviews and ratings for various vehicle models. | Kaggle | 284,715 records | 2018 |

# 4 Preprocessing Steps

Loaded and consolidated all data.

Handled missing values by dropping incomplete user-item-rating records.

Parsed unstructured text (Vehicle_Title) to create a standardized item_id.

**Result** A clean, model-ready dataset of user-item interactions

# 5 Model V1 & The Sparsity Challenge

| Initial Model | The Problem | Root Cause: Extreme Data Sparsity (99.989%) |
| --- | --- | --- |
| A k-Nearest Neighbors (k-NN) model trained on a user-item rating matrix. | Initial recommendations were irrelevant and illogical. | The model couldn't find meaningful patterns because too few users had rated the same set of cars. |

# 6 Model V2: Addressing Sparsity by Filtering

**Filter by Item Popularity**

Kept only vehicles with 10+ ratings

**Filter by User Activity**

Kept only users with 10+ ratings.

A much smaller, but significantly denser and higher-quality dataset for model training

# 7 The Pivot: Evolving to a Hybrid System

| | |
|---|---|
| The Limitation | • Pure collaborative filtering fails for new items with no ratings (the "cold start" problem). |
| The Solution | • Evolve the system into a **Hybrid Model.** |
| New Component | • A **Content-Based** recommendation engine was added to the API. |
| Capability | • The new engine can recommend similar vehicles based on their features (price, make, year) alone, solving the cold start problem. |

# 8 API Endpoints Usage

## GET /recommend?item_id=<ITEM_ID>

- **Method:** Collaborative Filtering
- **Description:** Retrieves pre-calculated recommendations for a known vehicle from the similarity cache.

## POST /recommend/unseen

- **Method:** Content-Based Filtering
- **Description:** Generates recommendations dynamically for a new, unseen vehicle based on its features.
- **Request Body:** JSON object with vehicle features (Year, Make, Model, Trim, HorsePower, Price.).

# 9 Result

We successfully built and deployed a secure, memory-optimized, hybrid recommendation API that solves the cold-start problem.