ARTIFICIAL INTELLIGENCE SYSTEMS

RentSense: User-Pattern Based Renting Recommendation System

Group 6

Chen Sigen A0326351L

Qi Yilin A0328843W

Ru Yanjie A0295237W

Yao Yiyang A0294873L

RentSense Project Report

Table of Contents

Executive Summary

Project Background

The RentSense project represents the second phase of our AI-powered rental recommendation system designed for international students in Singapore. Building upon the foundation established in the IRS Practice Module Project(hereafter referred to as Project 1), which focused on developing an end-to-end platform for property search and multi-objective optimization, RentSense introduces adaptive intelligence through user behavior modeling and advanced re-ranking algorithms.

Previous Project

In Project 1, the system provided personalized rental suggestions by filtering available properties according to user-specified preferences (such as rent range, proximity to schools, and flat type) and ranking them using a weighted multi-objective optimization model. This framework effectively supported both structured form input and natural language queries through an integrated NLU module.

Enhancements in RentSense Project

RentSense extends this framework by incorporating behavioral feedback signals—including dwell time and property bookmarking—to enhance recommendation quality. After the initial filtering stage, the candidate properties are re-ranked through a hybrid scoring approach that combines content-based similarity (via property and query embeddings) and collaborative filtering. Additionally, a Long Short-Term Memory (LSTM) model is employed to predict the dynamic weight of user preferences, allowing the multi-objective optimization process to adapt to each individual's evolving interests.

System Architecture and AI Integration

The enhanced RentSense architecture preserves the three-tier design of the previous system—frontend interface, backend services, and data infrastructure—while integrating new AI modules for behavioral analytics and adaptive preference modeling. This evolution transforms the system from a rule-driven recommender into a learning-based, user-centric platform capable of continuously improving its recommendation accuracy and interpretability.

Impact and Significance

Overall, RentSense demonstrates how user behavior understanding and machine learning can be leveraged to provide more accurate, personalized, and context-aware rental recommendations for students navigating Singapore's housing market.

System Design

Scope

The RentSense project extends the foundational system developed in Project 1 by introducing user-centric intelligence and adaptive learning mechanisms into the rental recommendation workflow.

While the initial version focused on **end-to-end property filtering and static multi-objective optimization**, RentSense broadens the system scope in three key dimensions:

1. **Behavioral Data Integration:**
The system now continuously captures user interaction signals such as dwell time and property bookmarking. These behavioral metrics are used to infer implicit preferences and provide a richer understanding of user intent.

2. **Dynamic Recommendation Framework:**
RentSense introduces a *re-ranking layer* that combines **content-based similarity** and **collaborative filtering** to refine the initial shortlist of properties generated during the filtering phase.

3. **Adaptive Optimization via Machine Learning:**
A **Long Short-Term Memory (LSTM)** model dynamically adjusts the weights of multiple objectives (e.g., rent, commute time, neighbourhood) based on predicted user preference shifts over time, transforming the system from a rule-based engine into a learning-based recommender.

Overall, RentSense redefines the scope from a “filter-and-rank” pipeline to an **intelligent, feedback-driven recommendation ecosystem** capable of personalization and continuous improvement.

Objectives

The primary objectives of the RentSense project are as follows:

1. **Enhance Personalization Accuracy**
Incorporate user behavioral data and historical preferences to provide recommendations that more accurately align with individual priorities.

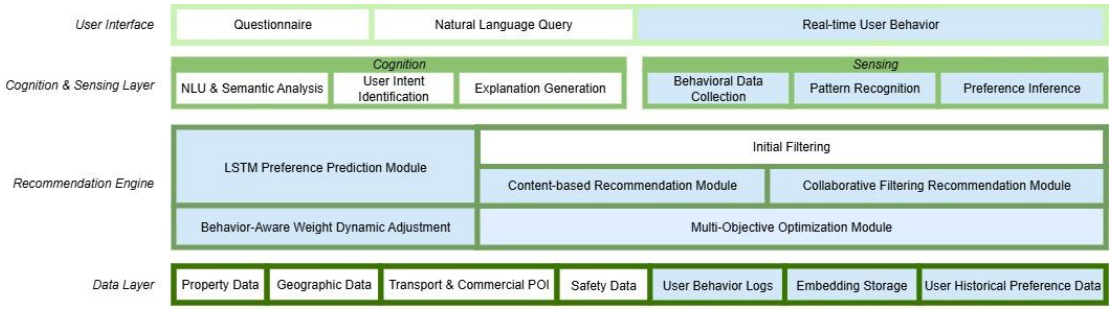
2. **Integrate Multi-Source Recommendation Signals**
Combine *content-based embeddings* and *collaborative filtering* within a unified re-ranking framework to leverage both item similarity and user–item interaction patterns.

3. **Enable Dynamic Preference Modeling**
Utilize LSTM-based modeling to capture temporal changes in user behavior and automatically adjust the weighting of objectives in the optimization process.

4. **Maintain End-to-End System Integrity**
Ensure that new modules are seamlessly integrated into the existing architecture without disrupting the performance or scalability of the backend infrastructure.

These objectives collectively aim to transform the recommendation process from a **static optimization task** into a **responsive, data-driven feedback loop**.

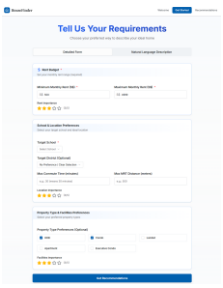
Architecture

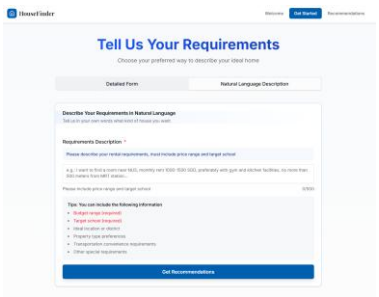


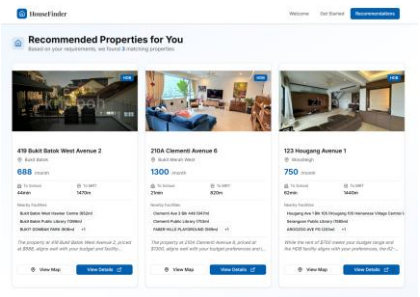
Overall architecture, filled parts are newly introduced or further implemented in this project

RentSense retains the three-tier structure from Project 1 — comprising the frontend interface, backend services, and data layer — but introduces **additional modules that enable adaptive learning and personalized reranking**.

User Interface







The frontend continues to provide dual-mode user interaction through:

- A structured form interface that allows users to specify quantitative filters (e.g., price range, commute time to school, district, etc.)
- A natural language interface powered by a NLU module that parses intent and entities from user queries.

In this project, the frontend also includes **behavior tracking components**, such as event listeners for dwell time and “favorite” actions. These signals are sent asynchronously to the backend for analysis.

Cognition & Sensing Layer

1. Cognition Module

Project 1 originally implemented the fundamental natural language understanding and

recommendation explanation components, including:

- **NLU & Semantic Analysis** – Driven by a Large Language Model (LLM) to parse user's natural language queries.
- **User Intent Identification** – Extracts rental requirements and preferences from parsed input.
- **Explanation Generation** – Utilizes Retrieval-Augmented Generation (RAG) for generating natural language explanations.

In RentSense Project, these components have been expanded to support **adaptive context reasoning** and **user profile–driven semantic interpretation**, enabling the system to:

- Integrate user behavioral context and dialogue history into intent recognition.
- Generate more personalized and contextually relevant explanations by aligning semantic interpretation with inferred preferences.

2. **Sensing Module**

This project introduces Sensing Module focused on behavioral data collection and sequential pattern recognition. It is for providing **real-time perception** and **adaptive preference learning**, including:

- **Behavioral Data Fusion** – Combines dwell time and interaction trajectory data for a more comprehensive behavioral understanding.
- **Dynamic Pattern Recognition** – Utilizes advanced sequence models (e.g., LSTM) for real-time user state estimation.

Recommendation Engine

The recommendation engine is expanded from a multi-objective optimization engine into a multi-stage recommendation engine with the following modules:

1. **Filtering Module (Base Layer):**
Performs initial candidate retrieval based on user-specified criteria and constraints from the property database (e.g., location, price, flat type).

2. **Reranking Module (Enhancement Layer):**
Combines two AI models:

   a. *Content-based model:* Computes cosine similarity between property embeddings and user query embeddings.

   b. *Collaborative filtering model:* Generates user–item interaction scores based on aggregated behavioral patterns from multiple users.

   The final rerank score is a weighted sum of both signals and user input preference(if applicable).

3. **Adaptive Optimization Module:**
Uses an **LSTM network** trained on sequential behavioral data to predict how user

preferences evolve. The output weights dynamically adjust the multi-objective optimization model (e.g., balancing affordability, commute time, and neighbourhood quality).

Data Layer

The database schema from Project 1 is extended to support new entities and logs:

- **User Behavior Logs:** Tracks dwell time and favorites events.
- **Embedding Storage:** Maintains vector representations for properties to support similarity computations.
- **User Historical Preference Data:** Enables the LSTM model to learn sequential preference patterns.

System Implementation

Overview

The implementation of the system builds upon the existing framework established in Project 1, extending it into a more intelligent and adaptive recommendation ecosystem.
While Project 1 implemented the core end-to-end pipeline — consisting of frontend user interface, NLU-based intent understanding, PostgreSQL knowledge base, filtering module, and multi-objective optimization — **this project** introduces several new components that enhance personalization, learning capability, and behavioral understanding.

Specifically, the extended system integrates **five new core modules**:

1. **User Behavior Tracking and Logging** – capturing implicit feedback such as dwell time and property bookmarking from the frontend interface.

2. **Content-Based Recommendation Model** – utilizing property and query embeddings to measure semantic similarity.

3. **Collaborative Filtering Model** – leveraging user–item interaction patterns to infer preferences from similar users.

4. **LSTM Preference Prediction Module** – modeling temporal sequences of user behavior to predict dynamic preference weights.

5. **Reranking and Multi-Objective Integration Layer** – merging all signals (content, collaborative, and behavioral predictions) into a unified scoring framework.

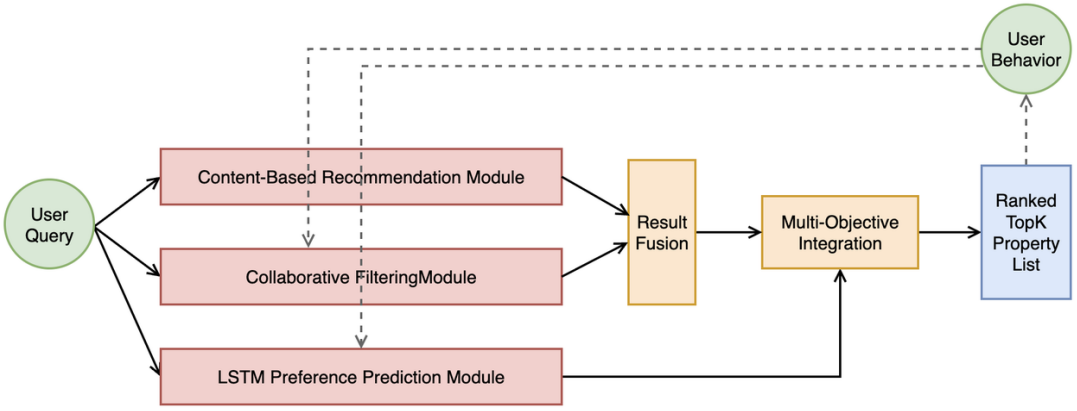
In addition to these algorithmic modules, several **infrastructure enhancements** were made:

- The frontend was instrumented with event listeners to collect user interaction data.
- The PostgreSQL database was extended with new tables for user behavior logs and historical preference records.
- A Neo4j graph database was introduced to store property embeddings and enable efficient vector similarity queries.
- A synthetic user dataset was generated to simulate real-world interactions for model training and testing.

System Integration Overview

All new modules are designed as extensions to the existing backend framework, ensuring compatibility and modular deployment.
The implementation follows a **layered integration flow**, illustrated conceptually below:

User Input → Initial Filtering → Candidate Set → Content + Collaborative Models → LSTM Preference Prediction → Multi-Objective Optimization → Final Recommendations



1. **Initial Filtering:**
The system first applies constraint-based filtering (price, distance, flat type, etc.) to narrow the search space to top-k candidate properties.

2. **Reranking Stage:**
These candidates are passed to the reranking layer, which computes:

   - *Content similarity scores* between the user's query vector and each property vector (stored in Neo4j).
   - *Collaborative filtering scores* derived from aggregated behavioral interactions.

3. **Adaptive Weight Adjustment:**
The user behavior also contributes to LSTM model, which analyzes user's recent browsing and selection patterns to predict updated preference weights (e.g., shifting emphasis between affordability and proximity).

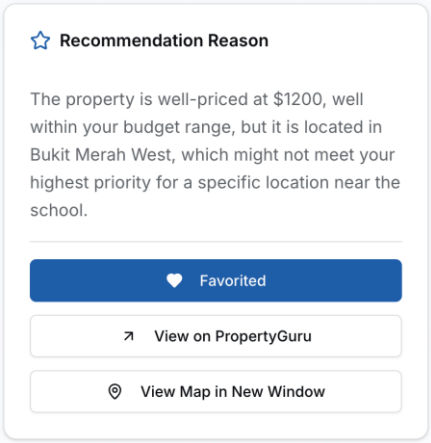
4. **Multi-Objective Optimization:**
The reranking stage filters top 50 listings to multi-objective optimization stage.

5. **Final Recommendations:**
The system returns the Top-10 optimized recommendations along with explainable justifications.

System Components

User Behavior Tracking & Data Collection

To enable personalized recommendations and continuously optimize model performance, the system implements a dedicated User Behavior Tracking and Data Collection module. This module is responsible for capturing real-time user interactions with the recommended property listings, providing critical input for subsequent recommendation algorithms.



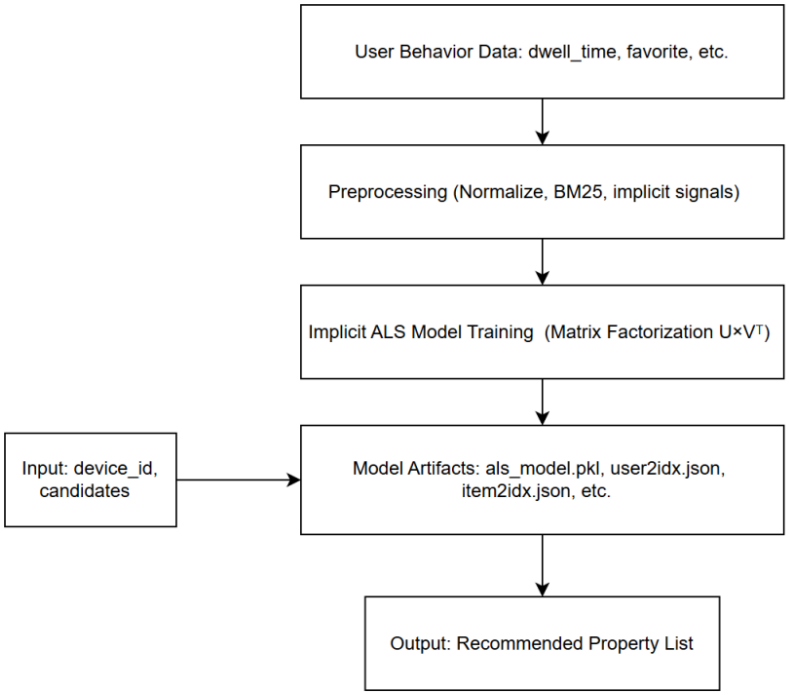
We primarily collect and process two categories of user feedback:

- **Explicit Feedback:** Actions where the user actively expresses preference. In this system, this primarily refers to a user "Liking" or "Favoriting" a property.
- **Implicit Feedback:** Behavioral data generated unconsciously by the user. This includes "Click-throughs" on recommended listings, "Dwell Time" (time spent viewing a property), and "Skips" (ignoring a recommendation).

All collected behavioral data is structurally logged. This data is then used to dynamically update user profiles and iteratively refine our recommendation models (e.g., Collaborative Filtering, Content-Based, and the LSTM preference model), ensuring that recommendations more accurately match the user's potential needs.

Collaborative Filtering Module

To improve the recommendation performance of the rental system, a collaborative filtering (CF) module is developed based on the Implicit Alternating Least Squares (ALS) algorithm. Unlike traditional models that rely on explicit user ratings, this method learns from implicit feedback signals (such as user browsing time and preference actions), which more realistically reflect user engagement. Each interaction between a user and a property is transformed into a preference strength, representing the user's level of interest. To better model the confidence of these interactions, BM25 weights are applied and an α scaling parameter to effectively balance the impact of frequent and infrequent actions. The ALS algorithm then decomposes the user-item interaction matrix into two low-dimensional embedding matrices: one representing the user and the other representing the property. During the recommendation phase, the dot product of each user embedding and each property embedding is calculated to estimate the preference score. The properties with the highest scores are ranked as the most relevant recommendations. This design enables the system to discover users' latent behavioral patterns, providing more accurate and personalized property recommendations even without explicit user ratings.



Content-Based Recommendation Module

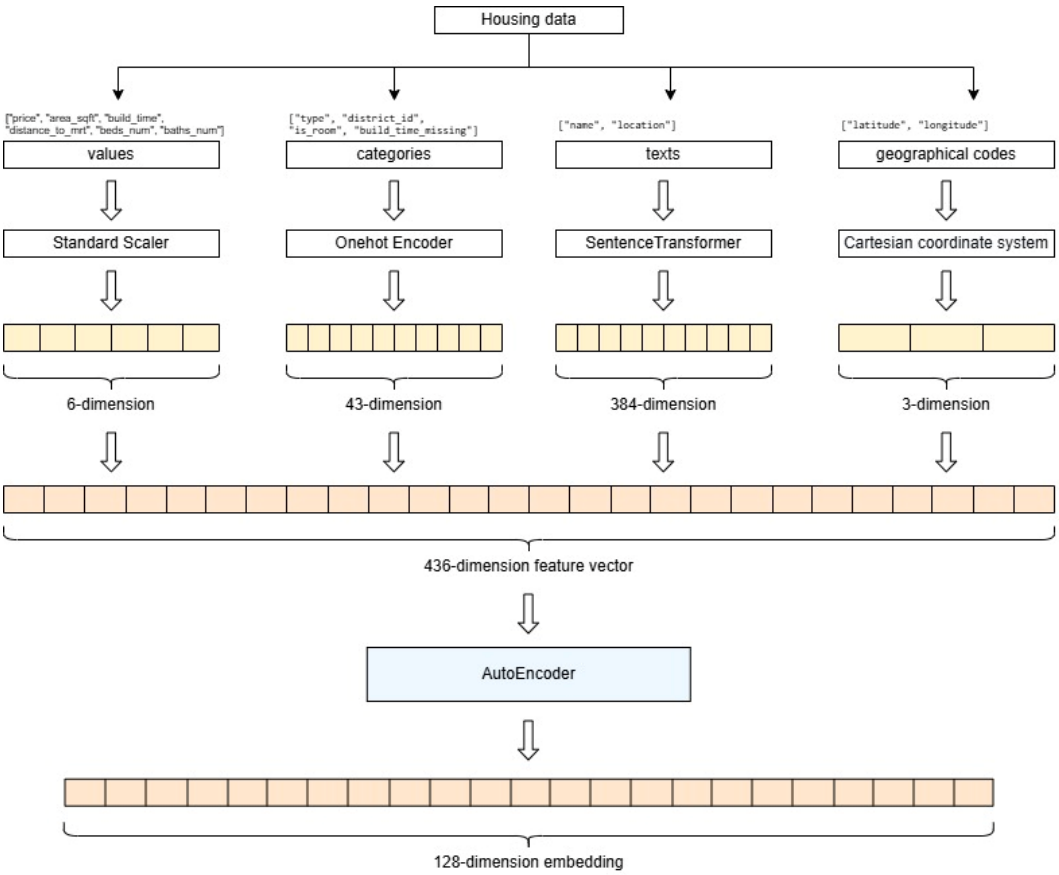
Module Overview

The **Content-Based Recommendation Module** is designed to capture the intrinsic similarity between properties and user preferences by representing each listing as a multi-dimensional embedding vector.

This module complements the rule-based filtering stage by providing a semantic understanding of property features—allowing the system to recommend listings that are not only constraint-compliant but also *contextually similar* to the user's desired attributes.

All embeddings are precomputed and stored in a Neo4j graph database for efficient similarity retrieval during runtime.

Feature Construction and Embedding Process



Overview of embedding process

The embedding process integrates four categories of features from the property database, each representing different aspects of a rental listing:

1. **Numerical Features**

Key quantitative attributes are normalized using `StandardScaler` to ensure consistent scale and prevent dominance of high-magnitude features.

2. **Categorical Features**

Categorical variables, such as flat type and district, are encoded using **One-Hot Encoding** to preserve their non-ordinal nature.

3. **Textual Features**

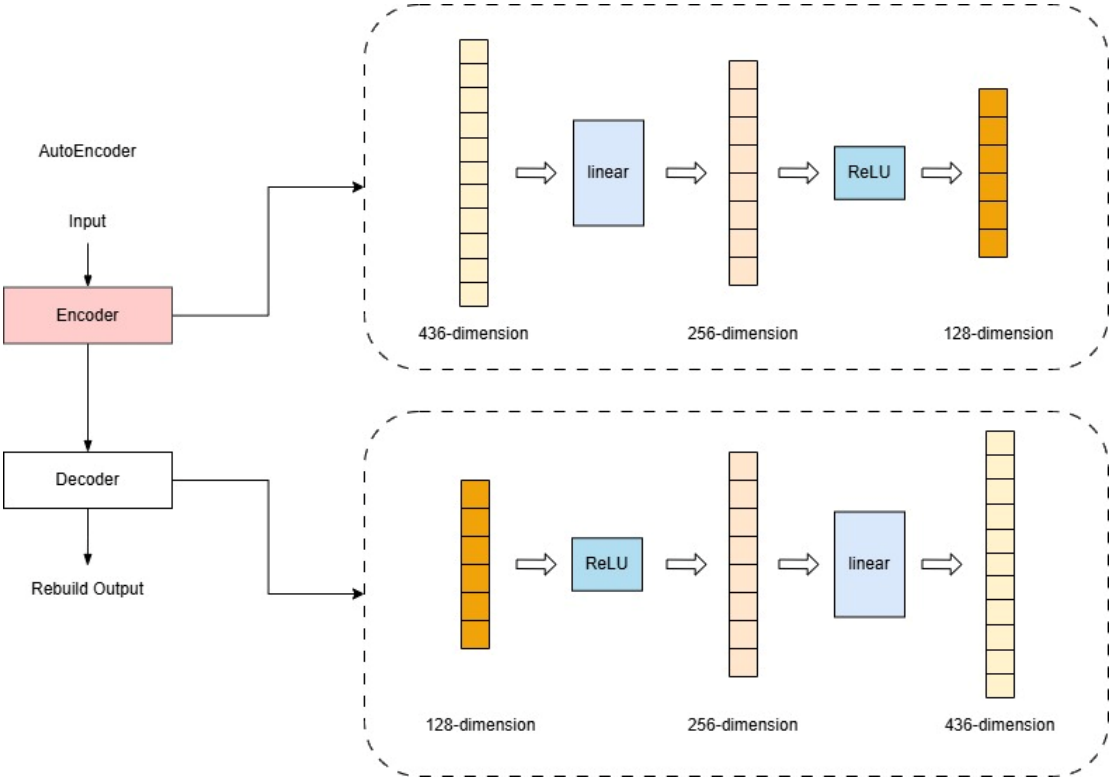
The property's textual information—comprising its `name` and `location`—is transformed into semantic embeddings using the **SentenceTransformer** model `all-MiniLM-L6-v2`, which produces 384-dimensional normalized vectors.

4. **Geospatial Features**

Latitude and longitude are converted into 3D Cartesian coordinates to maintain spatial relationships while avoiding discontinuities near geographic poles.

After processing, all feature groups are concatenated into a unified feature vector.

Dimensionality Reduction via Autoencoder



Autoencoder structure

To obtain a compact and expressive representation, an **AutoEncoder** neural network is trained to compress the 436-dimensional feature vectors into a 128-dimensional latent space. The MSE loss is used to evaluate the model's performance.

```
Epoch 1: train=0.0253, val=0.0137
Epoch 2: train=0.0154, val=0.0097
Epoch 3: train=0.0106, val=0.0068
Epoch 4: train=0.0081, val=0.0054
Epoch 5: train=0.0062, val=0.0045
Epoch 6: train=0.0053, val=0.0040
Epoch 7: train=0.0047, val=0.0036
Epoch 8: train=0.0043, val=0.0034
Epoch 9: train=0.0040, val=0.0032
Epoch 10: train=0.0037, val=0.0030
Epoch 11: train=0.0035, val=0.0029
Epoch 12: train=0.0034, val=0.0027
Epoch 13: train=0.0032, val=0.0026
Epoch 14: train=0.0030, val=0.0024
Epoch 15: train=0.0027, val=0.0022
Epoch 16: train=0.0025, val=0.0021
Epoch 17: train=0.0023, val=0.0019
Epoch 18: train=0.0021, val=0.0018
Epoch 19: train=0.0020, val=0.0017
Epoch 20: train=0.0019, val=0.0016
```

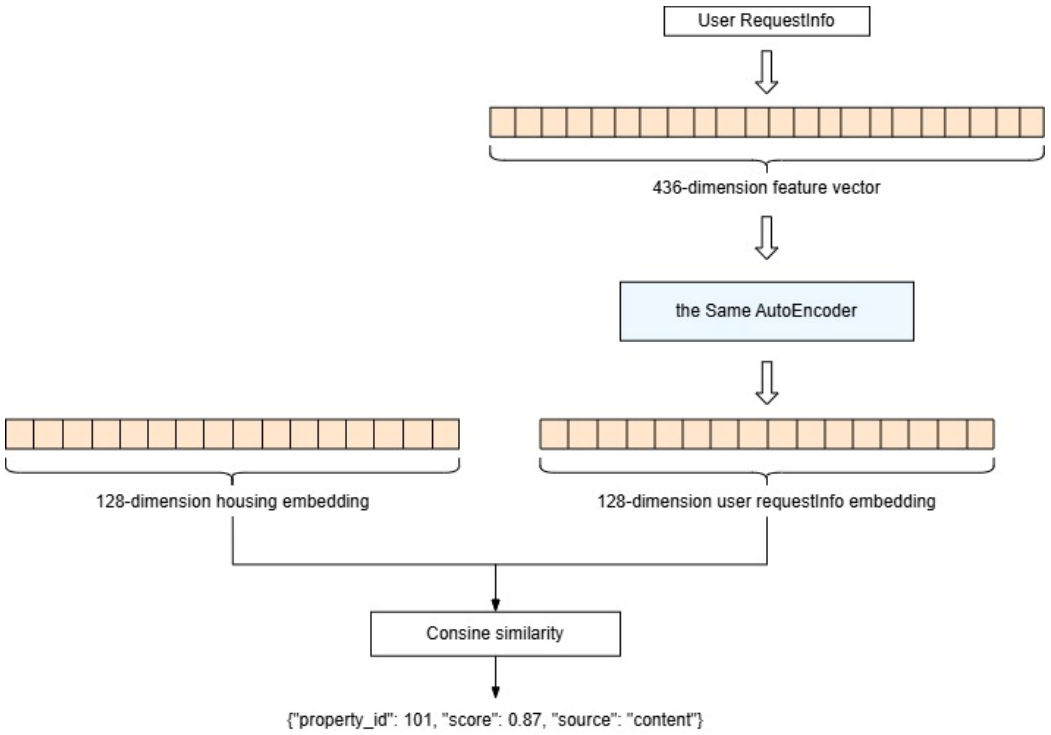
The loss in training stage and validation stage

This representation retains the most informative variations while reducing noise and redundancy.

Once trained, only the encoder part is used to transform each property into a 128-dimensional **content embedding**.

These embeddings, along with their corresponding property IDs, are stored as nodes in **Neo4j** for fast vector-based retrieval and similarity computation.

User Query Embedding and Similarity Scoring



Overview of scoring process

When a user submits a search request, the input data (`RequestInfo`) is transformed into a feature vector following the **same preprocessing pipeline** (scaling, encoding, text embedding, and coordinate conversion). The resulting vector is passed through the trained encoder to obtain a **user embedding** in the same 128-dimensional space.

To compute the similarity between the user query and each candidate property (from the initial filtering stage), the system performs cosine similarity scoring:

$$Score(u, p) = \frac{u \cdot p}{|u||p|}$$

The output is a ranked list of candidate properties, each with its similarity score and content-based source tag:

```
[
    {"property_id": 101, "score": 0.87, "source": "content"},
    {"property_id": 102, "score": 0.72, "source": "content"},
    ...
]
```

Integration and Output

- The content-based scores are forwarded to the Fusion Layer, where they are combined with collaborative filtering results and rerank the listings.
- By leveraging both structured and semantic features, this module ensures that recommendations align not only with the user's explicit filters but also with *latent contextual similarities* across listings.
- The pre-computed embedding structure in Neo4j also allows for efficient future scaling, supporting similarity-based retrieval beyond direct property matching.

Summary of Technical Contributions

Component	Description	Purpose
Feature Engineering	Combined numerical, categorical, textual, and geospatial features	Capture multi-faceted property characteristics
SentenceTransformer	Generates contextual text embeddings	Understand property semantics
AutoEncoder	Reduces 436D → 128D	Compresses high-dimensional features
Neo4j Graph Storage	Stores property embeddings	Supports fast similarity queries
Cosine Similarity Scoring	Compares user & property embeddings	Computes content-based relevance

Reranking and Score Integration

Module Overview

The **Reranking and Score Integration Module** acts as the central fusion layer of the *RentSense* recommendation pipeline.

After receiving preliminary candidate properties from the filtering stage, this module consolidates three distinct scoring sources — **content-based similarity**, **collaborative filtering**, and **feature-based importance weighting** — into a unified ranking score for each property.

This integration allows the system to balance different aspects of recommendation intelligence:

- *Content-based model* captures semantic similarity between user queries and property attributes.
- *Collaborative filtering model* reflects user–item interaction patterns learned from behavioral data.
- *Feature-importance scoring* aligns recommendations with the user's stated priorities (e.g., rent affordability, commute convenience, or neighborhood quality).

The output of this module is a sorted list of properties with composite scores, ready for downstream processing by the LSTM-based adaptive weighting and multi-objective optimization layers.

Weighted Score Fusion Mechanism

The integration follows a linear weighted aggregation formula, defined as:

$$S_{final}(i) = w_{imp} \times S_{imp}(i) + w_{content} \times S_{content}(i) + w_{cf} \times S_{cf}(i)$$

where:

- $S_{imp}(i)$: feature-based importance score derived from user input preferences (e.g., rent, commute, neighborhood quality).
- $S_{content}(i)$: cosine similarity score from the content-based module.
- $S_{cf}(i)$: collaborative filtering prediction score.
- $w_{imp}, w_{content}, w_{cf}$: corresponding fusion weights.

The prototype implementation uses static weights:

$$w_{imp} = 0.4, w_{content} = 0.4, w_{cf} = 0.2$$

Dynamic Weight Adaptation (Future Integration)

While the current prototype employs static weights, the design anticipates **dynamic weight adjustment** guided by user profile maturity:

- **Cold-start users:**

With no behavioral data, the system increases the influence of content-based and importance-driven signals (`w_cf ↓`, `w_content ↑`, `w_importance ↑`).

- **Active users with interaction history:**

As collaborative data accumulates, `w_cf` gradually increases to emphasize peer behavior similarity, allowing more personalized and data-driven recommendations.

This adaptive mechanism is to be realized through integration with the LSTM Preference Prediction Module, which dynamically predicts the optimal weight distribution based on user behavior sequences.

After the fusion step, all candidate properties are sorted by their final score in descending order.

The top-50 ranked results are forwarded to the Multi-Objective Optimization layers for further fine-tuning and adaptive scoring.

LSTM Preference Prediction Module

Core Concept & Objectives

This module is a deep learning-based system designed to dynamically predict user preferences in a rental property context. By analyzing a user's historical browsing data, the system infers their latent priority weights across three key decision-making dimensions:

- **Cost Sensitivity**
- **Commuting Convenience**
- **Neighborhood Quality**

The core objective is to move beyond static profiling and enable personalized recommendations by modeling the temporal patterns in user behavior using a specialized Recurrent Neural Network (RNN) known as a Long Short-Term Memory (LSTM) network.

Theoretical Foundation and Modeling Approach

Preference Quantification Framework

The system is grounded in utility theory, framing a user's rental choice as a multi-objective decision. A user's latent preference is represented as a 3-dimensional weight vector `[w_cost, w_commute, w_neighborhood]`, where the weights sum to 1. The final choice is assumed to be the property that maximizes the weighted utility across these dimensions. Our task is an inverse optimization problem: to deduce these hidden weights from observed historical behavior.

Temporal Modeling with LSTM

User browsing behavior is sequential; earlier interactions influence later ones. LSTM networks are uniquely suited for this task due to their gating mechanisms (input, forget, and output gates), which allow them to learn long-range dependencies in sequence data. A sliding window approach is used to transform a user's continuous browsing history into fixed-length sequential samples for model training.

Data Processing Pipeline

Feature Engineering

The system ingests raw property data and engineers a comprehensive set of **20 feature** dimensions. These features are categorized into:

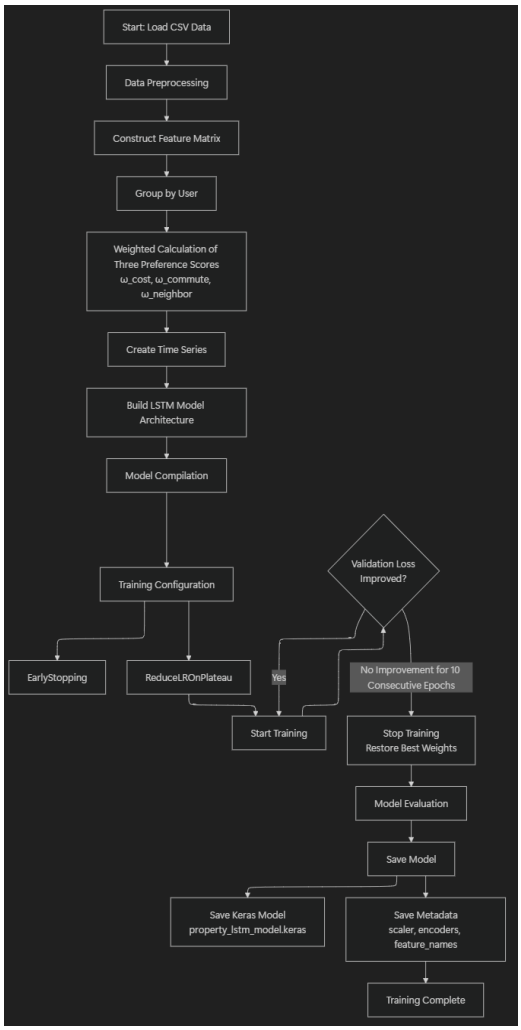
- Basic Attributes
- Geographic Information
- User Behavior Tags
- Temporal Features

A key engineering step involves parsing complex nested data (e.g., a list of nearby amenities) into concise statistical features, effectively balancing information richness with manageable dimensionality. Categorical variables are processed using label encoding.

Self-Supervised Label Generation

To train the model without manual labeling, a **self-supervised** approach is used to generate ground-truth preference vectors from user behavior:

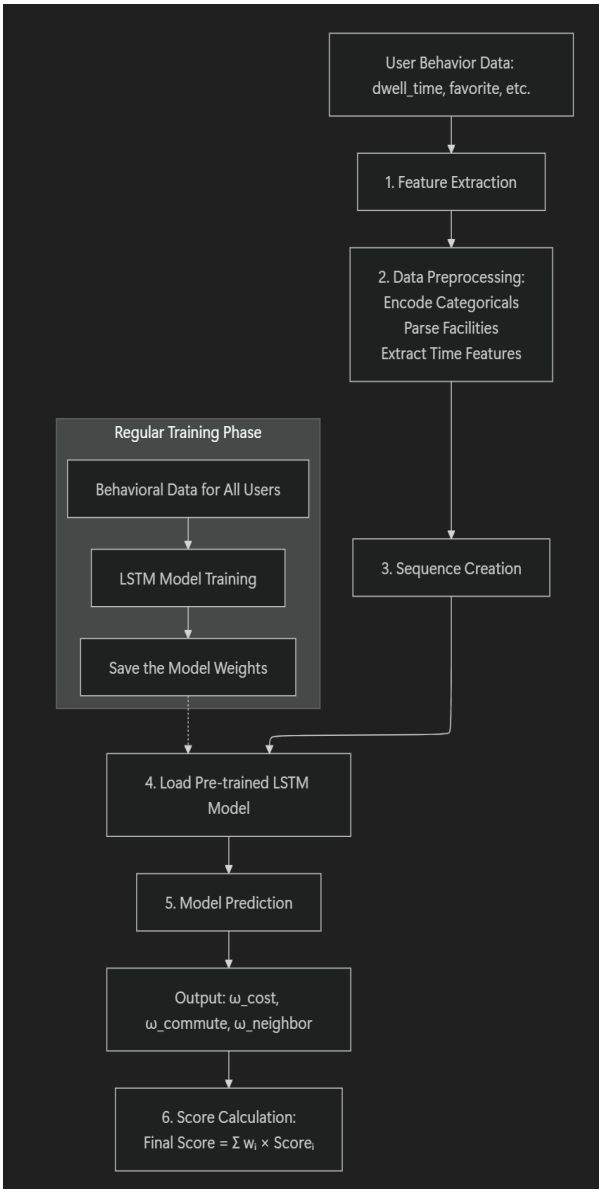
1. **Behavioral Weighting:** Each property in a user's history is assigned an important weight based on:
    - **Dwell Time:** Implicit indicator of interest level.
    - **Favorite:** Explicit signal of preference.
2. **Weighted Averaging:** These weights are used to calculate a weighted average of the property's scores on the three core dimensions.
3. **Normalization:** The resulting vector is normalized to produce the final 3D preference weight vector used as the training label.



Neural Network Architecture Design

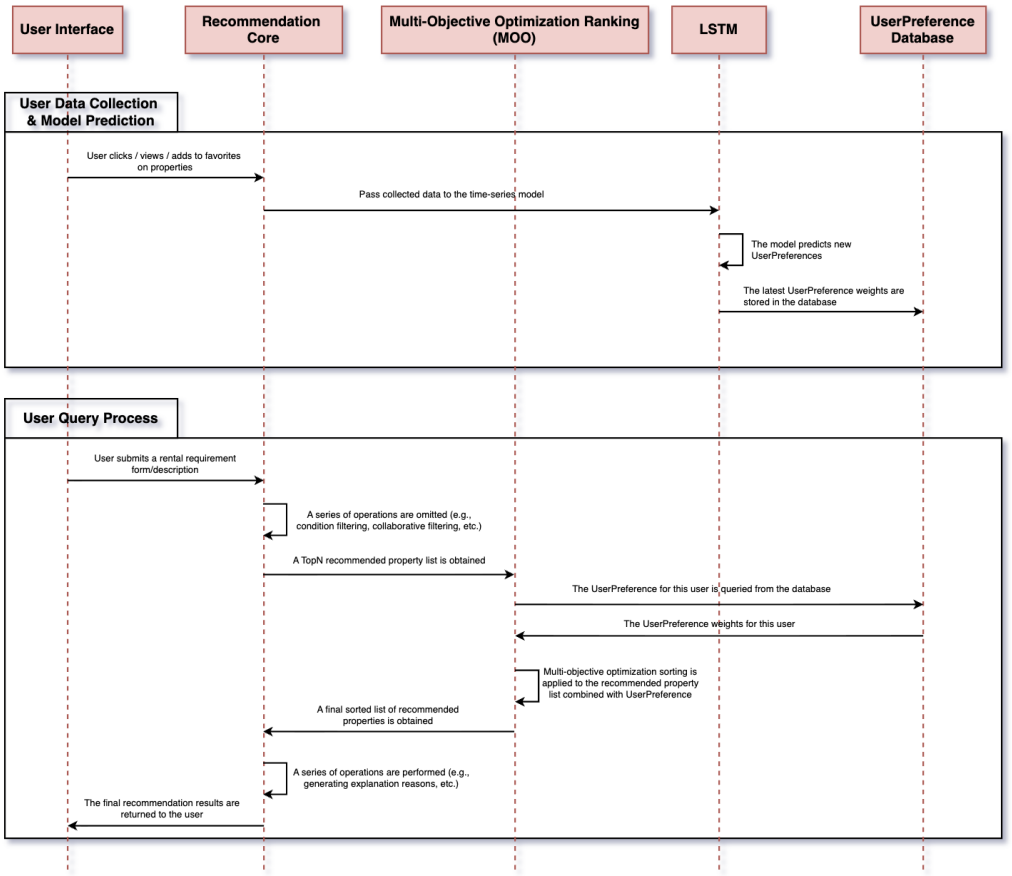
The model adopts a three-layer cascaded LSTM architecture with decreasing neuron counts across layers, forming an encoder-like structure. The first layer captures shallow temporal patterns, while subsequent layers progressively abstract higher-level semantic features. Batch normalization is applied at each layer to stabilize training, and dropout mechanisms are introduced to prevent overfitting. A fully connected layer further integrates temporal features, and a softmax activation function outputs a normalized three-dimensional weight vector.

The system uses Mean Squared Error (MSE) as the loss function to measure the Euclidean distance between predicted weights and ground-truth preferences. Compared to cross-entropy, MSE is more suitable for continuous-valued regression tasks. The optimizer of choice is the Adam algorithm with adaptive learning rates, combined with learning rate decay and early stopping strategies to balance convergence speed and generalization performance.



Multi-Objective Integration

A user's final rental choice balances multiple objectives (e.g., price, location), which basic filters often overlook. We implemented a Multi-Objective Optimization (MOO) module as a critical Reranking layer. The sequence diagram is shown in the figure below:



The module's core function is to apply personalized preference weights to the candidate property list generated by preceding filtering steps (such as collaborative and content-based filtering).

Crucially, these preference weights are not static. They are the live output of our LSTM time-series model, which is trained on historical user behavior data (clicks, likes, dwell time) to dynamically understand a user's *true* priorities.

The MOO algorithm integrates the "relevance" of the candidate list with these "behavior-driven preference weights," outputting a final, balanced list to the user.

User Behavior Data Simulation

To enable the training of behavioral and predictive models in the absence of real-world user data, a **synthetic user behavior simulation module** was developed.

This component generates artificial yet realistic user interaction records, including dwell time and property bookmarking actions.

The simulated dataset serves as the foundation for training the **Collaborative Filtering** and **LSTM Preference Prediction** modules, providing them with structured, consistent, and diverse input data.

Motivation and Design Rationale

Since the project is developed as an academic project without access to an operational user base, genuine behavioral data are unavailable.

However, models like collaborative filtering and sequential preference prediction require sufficient user–item interaction logs to learn latent behavioral patterns.

To bridge this gap, the simulation module was designed to:

1. Emulate realistic user requests and browsing behaviors.
2. Generate consistent data across multiple users and property listings.
3. Introduce a degree of stochasticity to reflect natural variation in user decisions.

Simulation Process

The data generation pipeline follows a multi-stage process:

1. **User Input Generation**
    - Randomly simulate 500 users, each assigned a unique set of input preferences, including rent range, preferred districts, proximity to schools, and housing type, etc.
    - Each simulated input is sent to the initial filtering module, which retrieves at least 50 candidate properties satisfying (or approximately satisfying) the given criteria.
2. **Property Interaction Simulation**
    - For each user, a random subset of 5–10 properties from the filtered results is selected to represent the user's browsing session.
    - For every viewed property, a relevance score is computed based on the alignment between the property's attributes and the simulated user's preferences.
    - Higher-scoring properties correspond to longer simulated viewing durations and a higher probability of being bookmarked.
3. **Behavioral Response Modeling**

    User interactions were modeled through linear correlations with the relevance score.

- Browsing duration was generated between 5–60 seconds, scaled proportionally to the normalized score, so higher-scoring properties received longer simulated viewing times.

- The favorite probability also increased linearly with the score, ranging from 0.1 to 0.4, introducing mild randomness while reflecting user interest levels.

This approach produces realistic engagement patterns without relying on complex nonlinear functions.

Resulting Dataset

- **Simulated Users:** 500
- **Interaction Records:** 3,802

Future Improvement

Automated Model Update Pipeline

At present, the model training process in our system is executed manually — data are collected, cleaned, and used to retrain models such as the AutoEncoder (for embeddings), Collaborative Filtering, and LSTM Preference Prediction modules.

In a production-grade system, this workflow can be automated through a **scheduled retraining pipeline**.

A background scheduler (e.g., hourly or daily cron jobs) could periodically:

1. Retrieve the latest behavioral and property data from the database,
2. Retrain or fine-tune the relevant models,
3. Validate performance using historical benchmarks, and
4. Automatically deploy updated model parameters to the running system.

Such an automated loop would enable *continuous learning*, ensuring that recommendation quality improves as new user behavior accumulates — turning the system into a self-adaptive, data-driven platform.

Supervised Fine-Tuning with User Behavior

Another future direction is to leverage accumulated user behavior data (e.g., dwell time, favorites) as supervised learning signals to fine-tune the embedding models.

Instead of training embeddings purely in an unsupervised manner, the model could be refined such that properties frequently viewed or bookmarked together become closer in the latent space.

This supervised fine-tuning would align the embedding structure with real user preferences, improving recommendation relevance and personal adaptation over time.

Extending the Graph-Based Representation

Currently, Neo4j is primarily used to store property embeddings for similarity search.

Future work can expand the graph structure to include entities such as districts, universities, and facilities, and explicitly model their relationships with properties.

Embedding these heterogeneous entities into a shared latent space constitutes a form of knowledge graph embedding (KGE).

This approach allows the system to learn relational patterns such as *“properties near university X”* or *“districts similar in amenities”*, enabling richer, context-aware recommendations.

It transforms the graph from a simple vector store into a relational reasoning layer,

supporting queries that combine spatial, semantic, and behavioral knowledge.

References

Report of Project1(IntelligentRentingRecommendationSystem):

https://github.com/csgen/IRS-PM-2025-10-01-ISY5001FT-GRP14-IntelligentRentingRecommendationSystem/blob/master/ProjectReport/Group14_Project%20Report.pdf

Link of Project1:

https://github.com/csgen/IRS-PM-2025-10-01-ISY5001FT-GRP14-IntelligentRentingRecommendationSystem

Project1 Video:

https://youtu.be/lcBed8bz-TI

https://youtu.be/keiCTkPs3p8