Module Interface Specification for TTE RecSys

Yinying Huo

April 12, 2025

1 Revision History

Date	Version	Notes
Mar. 2 2025	1.0	First Draft
Apr. 11 2025	2.0	Revision 1

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/V-AS/Two-tower-recommender-system/blob/main/docs/SRS/SRS.pdf

Contents

3 Introduction

The following document details the Module Interface Specifications for TTE RecSys, a two-tower recommendation system. The system leverages deep learning to create both the user tower and item tower, mapping inputs to a shared embedding space. Then, an effective algorithm is used to select a large number of candidate items. Finally, the dot product is applied for a refined ranking of the candidate items, returning the final recommendations accordingly.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/V-AS/Two-tower-recommender-system

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003).

The following table summarizes the primitive data types used by TTE RecSys.

Data Type	Notation	Description
Character	char	A single character
String	string	A sequence of characters representing text
List	$[T]$	A sequence of elements of type T
Dictionary	dict	A Python dictionary
Vector	$\mathbb{R}^n$	An ordered collection of n real numbers
Matrix	$[T]^{m\times n}$	A 2D array of type T with m rows and n columns
Boolean	$\mathbb{B}$	True or False value
Integer	$\mathbb{Z}$	A number without a fractional component in (-∞, ∞)
Real	$\mathbb{R}$	Any number in (-∞, ∞)
Tuple	$(T_1, T_2, ...)$	An ordered collection of elements with possibly different types

TTE RecSys uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

The specification also uses derived data types:

- **Embedding**: A list of real numbers representing learned features.
- **Tensor**: A multi-dimensional list, used for numerical computations.

- **DataFrame**: A two-dimensional list with labeled axes (rows and columns), typically used for storing tabular data.
- **User Feature**: A dictionary where the key is a string and the value is the corresponding feature value for the user.
- **Item Feature**: A dictionary where the key is a string and the value is the corresponding feature value for the item.
- **Model**: A PyTorch neural network (nn.Module) used for learning user and item representations.
- **ANNIndex**: A data structure used by FAISS for approximate nearest neighbor search in embedding space.
- **EvaluationMetrics**: A dictionary mapping evaluation metric names (strings) to their computed values (e.g., accuracy, RMSE).
- **TrainingConfig**: A dictionary containing configuration parameters for model training.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	System Interface Module Data Processing Module Model Training Module Embedding Generation Module Recommendation Module
Software Decision Module	Neural Network Architecture Module ANN Search Module Vector Operations Module

Table 1: Module Hierarchy

6 MIS of System Interface Module

6.1 Module

SystemInterface

6.2 Uses

None

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
save_model	model: Model, path: String	$\mathbb{B}$	IOError
load_model	path: String	Model	IOError, FormatError
save_emds	embeddings: [Embedding], path: String	$\mathbb{B}$	IOError
load_emds	path: String	[Embedding]	IOError
save_training_history	history: Dictionary, path: String	$\mathbb{B}$	IOError
load_training_history	path: String	Dictionary	IOError, FormatError

6.4 Semantics

6.4.1 State Variables

None

6.4.2 Environment Variables

FileSystem: The file system where models and embeddings are stored

6.4.3 Assumptions

- The file system is accessible and has sufficient space
- The paths provided are valid

6.4.4 Access Routine Semantics

`save_model(model, path)`:

- output: True if success, False otherwise
- exception: IOError if file cannot be written

`load_model(path)`:

- output: Model
- exception: IOError if file cannot be read, FormatError if file format is invalid

`save_emds(embeddings, path)`:

- output: True if success, False otherwise
- exception: IOError if file cannot be written

`load_embeddings(path)`:

- output: [Embedding]
- exception: IOError if file cannot be read, FormatError if file format is invalid

`save_training_history(history, path)`:

- output: True if success, False otherwise
- exception: IOError if file cannot be written

`load_training_history(path)`:

- output: Dictionary where keys are strings representing metrics ('loss', 'accuracy', etc.) and values are lists of corresponding numeric values for each training epoch
- exception: IOError if file cannot be read, FormatError if file format is invalid

7 MIS of Data Processing Module

7.1 Module

DataProcessor

7.2 Uses

SystemInterface

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
load_data	path: String	DataFrame	IOError, FormatError
validate_data	data: DataFrame	$\mathbb{B}$	-
prep_data	data: DataFrame	DataFrame	-
split_data	data: DataFrame, train_ratio: $\mathbb{R}$	(DataFrame, DataFrame)	ValueError
create_training_data	data: DataFrame	Dictionary	ValueError
get_book_mapping	data: DataFrame	Dictionary	-

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

None

7.4.3 Assumptions

- Input data follows the expected schema

7.4.4 Access Routine Semantics

`load_data(path)`:

- output: DataFrame containing the data from the file at the specified path
- exception: `IOError` if the file cannot be read; `FormatError` if the file format is invalid

`validate_data(data)`:

- output: True if the data meets all validation criteria, False otherwise

`preprocess_data(data)`:

- output: DataFrame containing the input data after applying normalization and feature creation

`split_data(data, train_ratio)`:

- output: (DataFrame, DataFrame) representing training data and testing data
- exception: `ValueError` if `train_ratio` is not in $(0, 1)$

`create_training_data(data)`:

- output: Dictionary where keys are strings 'user_ids', 'item_ids', 'ratings', 'user_features', and 'item_features', with values being arrays of user identifiers, arrays of item identifiers, arrays of numerical rating values, lists of User Feature dictionaries, and lists of Item Feature dictionaries, respectively
- exception: `ValueError` if any key did not have an associtaed value.

`get_book_mapping(data)`:

- output: Dictionary where each key is a book ID ($\mathbb{Z}$) and the value is a tuple of (String, String, $\mathbb{Z}$, String) representing the associated title, author, year, and publisher

8 MIS of Model Training Module

8.1 Module

ModelTrainer

8.2 Uses

DataProcessor, NeuralNetworkArchitecture

8.3 Syntax

8.3.1 Exported Constants

DEFAULT_LEARNING_RATE = 0.001
DEFAULT_BATCH_SIZE = 64
DEFAULT_REGULARIZATION = 0.0001

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
initialize	config: TrainingConfig	-	ValueError
train	train_data: Dictionary, epochs: $\mathbb{Z}$	Dictionary	RuntimeError
evaluate	test_data: Dictionary	EvaluationMetrics	RuntimeError
get_user_model	-	Model	RuntimeError
get_item_model	-	Model	RuntimeError

8.4 Semantics

8.4.1 State Variables

- UserModel: The neural network model for the user
- ItemModel: The neural network model for the item
- IsInitialized: Boolean indicating if the module has been initialized
- Config: Training configuration parameters
- Optimizer: Optimization algorithm
- Device: Computation device (CPU/GPU)

8.4.2 Environment Variables

None

8.4.3 Assumptions

- The training data is preprocessed and valid
- The model configuration is valid

8.4.4 Access Routine Semantics

`initialize(config)`:

- transition:
  - `UserModel` ← `config['user_architecture']`
  - `ItemModel` ← `config['item_architecture']`
  - `Optimizer` ← initialize optimization algorithm
  - `IsInitialized` ← `True`
- exception: `ValueError` if `config` contains invalid parameters

`train(train_data, epochs)`:

- transition:
  - Use `Optimizer` to optimize the loss function of the user and item models
  - The loss is computed using the local function `compute_loss`
- output: Dictionary where keys are strings 'loss', 'training_loss', and 'validation_loss', and values are list of real number.
- exception: `RuntimeError` if `IsInitialized` is `false`

`evaluate(test_data)`:

- output: EvaluationMetrics computed on the test data
- exception: `RuntimeError` if `IsInitialized` is `false`

`get_user_model()`:

- output: Model
- exception: `RuntimeError` if `IsInitialized` is `false`

`get_item_model()`:

- output: Model
- exception: `RuntimeError` if `IsInitialized` is `false`

8.4.5 Local Functions

`compute_loss(user_embeddings, item_embeddings, ratings)`:

- Type: $[\mathbb{R}^k] \times [\mathbb{R}^k] \times [\mathbb{R}] \to \mathbb{R}$
- Description: Computes the mean squared error (MSE) loss between predicted and actual ratings

9 MIS of Embedding Generation Module

9.1 Module

EmbeddingGenerator

9.2 Uses

NeuralNetworkArchitecture, VectorOperations

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
initialize	user_model: Model,item_model: Model	-	ValueError
generate_user_embedding	users: [User Feature]	[Embedding]	RuntimeError
generate_item_embedding	items: [Item Feature]	[Embedding]	RuntimeError

9.4 Semantics

9.4.1 State Variables

- UserModel: The neural network model for the user tower
- ItemModel: The neural network model for the item tower
- IsInitialized: Boolean indicating if the module has been initialized
- Device: Computation device (CPU/GPU)

9.4.2 Environment Variables

None

9.4.3 Assumptions

- The models have been trained
- User and item inputs have same dimensions

9.4.4 Access Routine Semantics

`initialize(user_model, item_model)`:

- transition:
  - `UserModel` ← `user_model`
  - `ItemModel` ← `item_model`
  - `IsInitialized` ← `true`
  - `Device` ← detected available hardware (CPU or GPU)
- exception: `ValueError` if the models are incompatible

`generate_user_embedding(users)`:

- output: [Embedding] for the provided user(s)
- exception: `RuntimeError` if `IsInitialized` is `false`

`generate_item_embedding(items)`:

- output: [Embedding] for the provided item(s)
- exception: `RuntimeError` if `IsInitialized` is `false`

10 MIS of Recommendation Module

10.1 Module

Recommender

10.2 Uses

EmbeddingGenerator, ANNSearch, VectorOperations

10.3 Syntax

10.3.1 Exported Constants

DEFAULT_NUM_RECOMMENDATIONS = 10
SIMILARITY_THRESHOLD = 0.5

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
initialize	ann_index: ANNIndex, embedding_generator: EmbeddingGenerator, book_lookup: Dictionary	-	ValueError
get_recommendations	user: User Feature, num_results: $\mathbb{Z}$	[($\mathbb{Z}$, String, $\mathbb{R}$)]	RuntimeError

10.4 Semantics

10.4.1 State Variables

ANNIndex: The index for approximate nearest neighbor search
EmbeddingGenerator: Reference to the embedding generator
BookLookup: Dictionary mapping item IDs to book details
IsInitialized: Boolean indicating if the module has been initialized

10.4.2 Environment Variables

None

10.4.3 Assumptions

- The ANN index has been built with item embeddings
- The embedding generator has been initialized with trained models
- The book lookup dictionary contains valid mappings

10.4.4 Access Routine Semantics

`initialize(ann_index, embedding_generator, book_lookup)`:

- transition:
  - `ANNIndex` $\leftarrow$ `ann_index`
  - `EmbeddingGenerator` $\leftarrow$ `embedding_generator`
  - `BookLookup` $\leftarrow$ `book_lookup`
  - `IsInitialized` $\leftarrow$ `true`
- exception: `ValueError` if any parameter is invalid

`get_recommendations(user, num_results)`:

- output: $[(\mathbb{Z}, \text{String}, \mathbb{R})]$ representing a ranked list of (item_id, item_title, similarity_score) tuples. The ranks are calculated using the local function `rank_candidates`.
- exception: `RuntimeError` if `IsInitialized` is `false`

10.4.5 Local Functions

`rank_candidates(user_embedding, candidate_embeddings)`:

- Type: $\mathbb{R}^k \times [\mathbb{R}^k] \rightarrow [(\mathbb{Z}, \mathbb{R})]$
- Description: Ranks candidate items based on similarity scores computed using the dot product; returns a list of item indices with associated scores.

11 MIS of Neural Network Architecture Module

11.1 Module

NeuralNetworkArchitecture

11.2 Uses

VectorOperations

11.3 Syntax

11.3.1 Exported Constants

DEFAULT_HIDDEN_LAYERS = [256, 128]
DEFAULT_ACTIVATION = "relu"

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
create_user_tower	input_dim: $\mathbb{Z}$, hidden_layers: [$\mathbb{Z}$] , embedding_dim: $\mathbb{Z}$	Model	ValueError
create_item_tower	input_dim: $\mathbb{Z}$, hidden_layers: [$\mathbb{Z}$] , embedding_dim: $\mathbb{Z}$	Model	ValueError

11.4 Semantics

11.4.1 State Variables

None

11.4.2 Environment Variables

None

11.4.3 Assumptions

- Input dimensions are valid positive integers
- Hidden layers and embedding dimensions are compatible

11.4.4 Access Routine Semantics

create_user_tower(input_dim, hidden_layers, embedding_dim):

- output: Model for user tower
- exception: ValueError if dimensions are invalid

create_item_tower(input_dim, hidden_layers, embedding_dim):

- output: Model for item tower
- exception: ValueError if dimensions are invalid

12 MIS of ANN Search Module

12.1 Module

ANNSearch

12.2 Uses

VectorOperations

12.3 Syntax

12.3.1 Exported Constants

DEFAULT_SEARCH_NPROBE := 10
DEFAULT_INDEX_TYPE := "Flat"

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
build_index	embeddings: [Embedding] , item_ids: [$\mathbb{Z}$], index_type: String	ANNIndex	ValueError
two_stage_search	index: ANNIndex, query: Embedding, candidates: $\mathbb{Z}$, final_k: $\mathbb{Z}$	$[(\mathbb{Z}, \mathbb{R})]$	ValueError
save_index	index: ANNIndex, path: String	$\mathbb{B}$	IOError
load_index	path: String	ANNIndex	IOError, FormatError

12.4 Semantics

12.4.1 State Variables

None

12.4.2 Environment Variables

None

12.4.3 Assumptions

- Embeddings are of consistent dimension
- Query vector is of same dimension as indexed vectors
- FAISS library is available

12.4.4 Access Routine Semantics

`build_index(embeddings, item_ids, index_type)`:

- output: ANNIndex
- exception: `ValueError` if any parameter is invalid

`two_stage_search(index, query, candidates, final_k)`:

- output: $[(\mathbb{Z}, \mathbb{R})]$ representing a list of (item_id, similarity_score) pairs for the top-k nearest neighbors
- exception: `ValueError` if any parameter is invalid

`save_index(index, path)`:

- output: True if success, False otherwise
- exception: `IOError` if the file cannot be written

`load_index(path)`:

- output: ANNIndex
- exception: `IOError` if the file cannot be read; `FormatError` if the file format is invalid

13 MIS of Vector Operations Module

13.1 Module

VectorOperations

13.2 Uses

None

13.3 Syntax

13.3.1 Exported Constants

EPSILON := 1e-8 (small value to prevent division by zero)

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
dot_product	v1: [$\mathbb{R}$], v2: [$\mathbb{R}$]	$\mathbb{R}$	DimensionMismatchError

13.4 Semantics

13.4.1 State Variables

None

13.4.2 Environment Variables

None

13.4.3 Assumptions

None

13.4.4 Access Routine Semantics

`dot_product(v1, v2)`:

- output: $\mathbb{R}$ representing $\sum_{i=1}^{n} v1_i \cdot v2_i$, where n is the number of elements in each vector
- exception: `DimensionMismatchError` if the input vectors do not have the same number of elements

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.