

# Module Interface Specification for TTE RecSys

Yinying Huo

April 1, 2025

# 1 Revision History

Date	Version	Notes
March 2 2025	1.0	First Draft

## 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

<b>1</b>	<b>Revision History</b>	<b>i</b>
<b>2</b>	<b>Symbols, Abbreviations and Acronyms</b>	<b>ii</b>
<b>3</b>	<b>Introduction</b>	<b>1</b>
<b>4</b>	<b>Notation</b>	<b>1</b>
<b>5</b>	<b>Module Decomposition</b>	<b>2</b>
<b>6</b>	<b>MIS of Hardware-Hiding Module</b>	<b>3</b>
6.1	Module . . . . .	3
6.2	Uses . . . . .	3
6.3	Syntax . . . . .	3
6.3.1	Exported Constants . . . . .	3
6.3.2	Exported Access Programs . . . . .	3
6.4	Semantics . . . . .	3
6.4.1	State Variables . . . . .	3
6.4.2	Environment Variables . . . . .	3
6.4.3	Assumptions . . . . .	4
6.4.4	Access Routine Semantics . . . . .	4
<b>7</b>	<b>MIS of Data Processing Module</b>	<b>4</b>
7.1	Module . . . . .	4
7.2	Uses . . . . .	5
7.3	Syntax . . . . .	5
7.3.1	Exported Constants . . . . .	5
7.3.2	Exported Access Programs . . . . .	5
7.4	Semantics . . . . .	5
7.4.1	State Variables . . . . .	5
7.4.2	Environment Variables . . . . .	5
7.4.3	Assumptions . . . . .	5
7.4.4	Access Routine Semantics . . . . .	5
<b>8</b>	<b>MIS of Model Training Module</b>	<b>6</b>
8.1	Module . . . . .	6
8.2	Uses . . . . .	6
8.3	Syntax . . . . .	6
8.3.1	Exported Constants . . . . .	6
8.3.2	Exported Access Programs . . . . .	7
8.4	Semantics . . . . .	7
8.4.1	State Variables . . . . .	7

8.4.2	Environment Variables . . . . .	7
8.4.3	Assumptions . . . . .	7
8.4.4	Access Routine Semantics . . . . .	7
8.4.5	Local Functions . . . . .	8
<b>9</b>	<b>MIS of Embedding Generation Module</b>	<b>8</b>
9.1	Module . . . . .	8
9.2	Uses . . . . .	9
9.3	Syntax . . . . .	9
9.3.1	Exported Constants . . . . .	9
9.3.2	Exported Access Programs . . . . .	9
9.4	Semantics . . . . .	9
9.4.1	State Variables . . . . .	9
9.4.2	Environment Variables . . . . .	9
9.4.3	Assumptions . . . . .	9
9.4.4	Access Routine Semantics . . . . .	10
<b>10</b>	<b>MIS of Recommendation Module</b>	<b>10</b>
10.1	Module . . . . .	10
10.2	Uses . . . . .	10
10.3	Syntax . . . . .	10
10.3.1	Exported Constants . . . . .	10
10.3.2	Exported Access Programs . . . . .	11
10.4	Semantics . . . . .	11
10.4.1	State Variables . . . . .	11
10.4.2	Environment Variables . . . . .	11
10.4.3	Assumptions . . . . .	11
10.4.4	Access Routine Semantics . . . . .	11
10.4.5	Local Functions . . . . .	12
<b>11</b>	<b>MIS of Neural Network Architecture Module</b>	<b>12</b>
11.1	Module . . . . .	12
11.2	Uses . . . . .	12
11.3	Syntax . . . . .	12
11.3.1	Exported Constants . . . . .	12
11.3.2	Exported Access Programs . . . . .	12
11.4	Semantics . . . . .	12
11.4.1	State Variables . . . . .	12
11.4.2	Environment Variables . . . . .	13
11.4.3	Assumptions . . . . .	13
11.4.4	Access Routine Semantics . . . . .	13

<b>12 MIS of ANN Search Module</b>	<b>13</b>
12.1 Module . . . . .	13
12.2 Uses . . . . .	13
12.3 Syntax . . . . .	13
12.3.1 Exported Constants . . . . .	13
12.3.2 Exported Access Programs . . . . .	14
12.4 Semantics . . . . .	14
12.4.1 State Variables . . . . .	14
12.4.2 Environment Variables . . . . .	14
12.4.3 Assumptions . . . . .	14
12.4.4 Access Routine Semantics . . . . .	14
<b>13 MIS of Vector Operations Module</b>	<b>15</b>
13.1 Module . . . . .	15
13.2 Uses . . . . .	15
13.3 Syntax . . . . .	15
13.3.1 Exported Constants . . . . .	15
13.3.2 Exported Access Programs . . . . .	15
13.4 Semantics . . . . .	15
13.4.1 State Variables . . . . .	15
13.4.2 Environment Variables . . . . .	15
13.4.3 Assumptions . . . . .	16
13.4.4 Access Routine Semantics . . . . .	16

### 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
Array	$[T]$	A sequence of elements of type $T$
Dictionary/Map	dict	A collection of key-value pairs
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 $(-\infty, \infty)$
real	$\mathbb{R}$	Any number in $(-\infty, \infty)$
Tuple	$(T_1, T_2, \dots)$	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 vector of real numbers
- Tensors: Multi-dimensional arrays

- User: A type representing user features
- Item: A type representing item features
- DataSet: A collection of data records containing user, item, and interaction information
- Model: A neural network model
- ANNIndex: An index structure for approximate nearest neighbor search
- TrainingConfig: A dictionary containing configuration parameters for model training (learning rate, batch size, etc.)
- LayerConfig: A dictionary describing the configuration of neural network layers
- EvaluationMetrics: A dictionary of evaluation metric names and values

## 5 Module Decomposition

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

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

Table 1: Module Hierarchy



## 6 MIS of Hardware-Hiding 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	success: $\mathbb{B}$	IOError
load_model	path: String	model: Model	IOError, FormatError
save_emds	embeddings: [Embedding], path: String	success: $\mathbb{B}$	IOError
load_emds	path: String	embeddings: [Embedding]	IOError
save_training_history	history: dict, path: String	success: $\mathbb{B}$	IOError
load_training_history	path: String	history: dict	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: `success = true` if operation succeeds
- 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_embeddings(embeddings, path):`

- output: `success = true` if operation succeeds
- exception: `IOError` if file cannot be written

`load_embeddings(path):`

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

`save_training_history(history, path):`

- output: `success = true` if operation succeeds
- exception: `IOError` if file cannot be written

`load_training_history(path):`

- output: `history = training metrics dictionary`
- 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	data: DataSet	IOError, FormatError
validate_data	data: DataSet	is_valid: $\mathbb{B}$	-
prep_data	data: DataSet	processed_dataset: DataSet	-
split_data	data: DataSet, train_ratio: $\mathbb{R}$	train_data: DataSet, test_data: DataSet	ValueError
create_training_data	data: DataSet	dataset: dict	ValueError
get_book_mapping	data: DataSet	mapping: dict	-

## 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: data = data from file at path
- exception: IOError if file cannot be read, FormatError if file format is invalid

validate\_data(data)::

- output: `is_valid = true` if data meets all validation criteria

`preprocess_data(data):`

- output: `processed_data` = dataset after applying feature engineering transformations, including normalization, frequency encoding, and derived feature creation.

`split_data(data, train_ratio):`

- output: (`train_data`, `test_data`) where:
  - `train_data` = subset of data for training (size  $\approx \text{train\_ratio} * |data|$ )
  - `test_data` = subset of data for training (size  $\approx (1 - \text{train\_ratio}) * |data|$ )
- exception: `ValueError` if `train_ratio` is not in  $(0, 1)$

`create_training_data(data):`

- output: `dataset` = dictionary containing `user_ids`, `item_ids`, `ratings`, `user_features`, and `item_features`
- exception: `ValueError` if features are missing

`get_book_mapping(data):`

- output: `mapping` = dictionary mapping encoded book IDs to book details (title, author, year, publisher)

## 8 MIS of Model Training Module

### 8.1 Module

`ModelTrainer`

### 8.2 Uses

`DataProcessor`, `NeuralNetworkArchitecture`, `VectorOperations`

### 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: DataSet, epochs: $\mathbb{Z}$	model: dict	RuntimeError
evaluate	test_data: DataSet,	metrics: EvaluationMetrics	RuntimeError
get_user_model	-	user_model: Model	RuntimeError
get_item_model	-	item_model: Model	RuntimeError

## 8.4 Semantics

### 8.4.1 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
- 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']
  - Config = config

- Optimizer = initialize optimization algorithm
- IsInitialized = true

- exception: ValueError if config contains invalid parameters

train(train\_data, epochs):

- transition:
  - Update UserModel and ItemModel parameters through training
- output: model = dictionary containing user\_model, item\_model, and training history
- exception: RuntimeError if IsInitialized is false

evaluate(test\_data):

- output: metrics = evaluation metrics on test data
- exception: RuntimeError if IsInitialized is false

get\_user\_model():

- output: user\_model = UserModel
- exception: RuntimeError if IsInitialized is false

get\_item\_model():

- output: item\_model = ItemModel
- exception: RuntimeError if IsInitialized is false

#### 8.4.5 Local Functions

compute\_loss(user\_embeddings, item\_embeddings, ratings):

- Type:  $[\mathbb{R}^k] \times [\mathbb{R}^k] \times [\mathbb{R}] \rightarrow \mathbb{R}$
- Description: Computes 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: [ProcessedUser]	embeddings: [Embed- ding]	RuntimeError
generate_item_embedding	items: [Pro- cessedItem]	embeddings: [Embed- ding]	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 = detection of available hardware (CPU or GPU)`
- exception: `ValueError` if models are incompatible

`generate_user_embedding(users):`

- output: `embeddings = embeddings` for the provided users
- exception: `RuntimeError` if `IsInitialized` is false

`generate_item_embedding(items):`

- output: `embeddings = embeddings` for the provided items
- 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: dict	-	ValueError
get_recommendations	user: ProcessedUser, num_results: $\mathbb{Z}$	recommendations: [dict]	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 = ann\_index
  - EmbeddingGenerator = embedding\_generator
  - BookLookup = book\_lookup
  - IsInitialized = true
- exception: ValueError if parameters are invalid

get\_recommendations(user, num\_results):

- output: recommendations = list of dictionaries containing item details and similarity scores
- 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 candidates by similarity score (dot product)

## 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: Model	ValueError
create_item_tower	input_dim: $\mathbb{Z}$ , hidden_layers: $[\mathbb{Z}]$ , embedding_dim: $\mathbb{Z}$	model: 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 = neural network model for user tower
- exception: `ValueError` if dimensions are invalid

`create_item_tower(input_dim, hidden_layers, embedding_dim):`

- output: model = neural network 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: [Z], index_type: String	index: ANNIndex	ValueError
two_stage_search	index: ANNIndex, query: Embedding, candidates: Z, final_k: Z	results: [(Z, R)]	ValueError
save_index	index: ANNIndex, path: String	success: B	IOError
load_index	path: String	index: 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: index = ANN index built from embeddings and associated item IDs
- exception: ValueError if parameters are invalid

two\_stage\_search(index, query, candidates, final\_k):

- output: results = list of (item\_id, similarity\_score) tuples for k nearest neighbors

- exception: ValueError if parameters are invalid

save\_index(index, path):

- output: success := true if operation succeeds
- exception: IOError if file cannot be written

load\_index(path):

- output: index := ANNIndex loaded from file
- exception: IOError if file cannot be read, FormatError if 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: $[\mathbb{R}]$	result: $\mathbb{R}$	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:  $\text{result} = \sum_i^{\text{len}(v1)} v1[i] * v2[i]$
- exception: `DimensionMismatchError` if  $\text{len}(v1) \neq \text{len}(v2)$

## 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>.