System Verification and Validation Plan for TTE RecSys

Yinying Huo

April 3, 2025

Revision History

Date	Version	Notes
Feb. 24, 2025	1.0	First draft - Unit tests will be added after the MIS has been completed..
Feb. 25, 2025	1.1	Minor updates after VnV presentation
Mar. 27, 2025	1.2	Addressed comments from Dr. Smith.
Apr. 2, 2025	1.3	Added unit tests.

Contents

List of Tables

1 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test
R	Requirement
NFR	Nonfunctional Requirement
TTE	Two Tower Embedding
RecSys	Recommendation System
ANN	Approximate Nearest Neighbor
SRS	Software Requirements Specification
FAISS	Facebook AI Similarity Search
DNN	Deep Neural Network

This document outlines the verification and validation plan for the Two-Tower Embeddings Recommendation System (TTE RecSys) to ensure compliance with the requirements and objectives specified in the Software Requirements Specification (SRS). It is structured to first present general information and verification strategies, followed by detailed descriptions of system and unit testing for both functional and non-functional requirements.

2 General Information

2.1 Summary

The software under test is the Two-Tower Embedding Recommendation System, which generates personalized recommendations using user and item embeddings. The system consists of two main components:

- Training Phase: Learns user and item embedding functions using a deep neural network architecture, optimized via Stochastic Gradient Descent (SGD).
- Inference Phase: Retrieves candidate items using Approximate Nearest Neighbor (ANN) search and ranks them by dot product similarity.

The system is implemented in Python, leveraging libraries such as PyTorch for model training and FAISS for ANN search.

2.2 Objectives

The primary objectives of this VnV plan are:

- Correctness: Verify that the system correctly implements the mathematical models for training (e.g., MSE loss) and inference (e.g., ANN search, dot product ranking).
- Scalability: Demonstrate that the system can support incremental update when new data available.

Out-of-Scope Objectives

- External Library Verification: Libraries such as PyTorch and FAISS are assumed to be correct and are not verified as part of this plan.

2.3 Challenge Level and Extras

This is a non-research project. The extra component of this project will be a user manual.

2.4 Relevant Documentation

The following documents are available for this project: Software Requirements Specification, Module Guide, and Module Interface Specification

3 Plan

The VnV plan starts with an introduction to the verification and validation team, followed by verification plans for the SRS and design. Next, it covers verification plans for the VnV Plan and implementation. Finally, it includes sections on automated testing and verification tools as well as the software validation plan .

3.1 Verification and Validation Team

Name	Document	Role	Description
Yinying Huo	All	Author	Prepare all documentation, develop the software, and validate the implementation accoridng to the VnV plan.
Dr. Spencer Smith	All	Instructor/ Reviewer	Review all the documents.
Yuanqi Xue	All	Domain Expert	Review all the documents.

Table 1: Verification and Validation Team

3.2 SRS Verification Plan

The Software Requirements Specification (SRS) will be reviewed by domain expert Yuanqi Xue and Dr. Smith. Feedback from reviewers will be provided on GitHub, and the author will need to address it.

There is a SRS checklist designed by Dr. Spencer Smith available to use.

3.3 Design Verification Plan

The design verification, including the Module Guide (MG) and Module Interface Specification (MIS), will be reviewed by domain expert Yuanqi Xue and Dr. Smith. Feedback from reviewers will be provided on GitHub, and the author will need to address it.
Dr. Spencer Smith has created a MG checklist and MIS checklist, both of which are available for use.

3.4 Verification and Validation Plan Verification Plan

The Verification and Validation (VnV) Plan will be reviewed by domain expert Yuanqi Xue and Dr. Smith. Feedback from reviewers will be provided on GitHub, and the author will need to address it.
There is a VnV checklist designed by Dr. Spencer Smith available to use.

3.5 Implementation Verification Plan

The implementation will be verified by testing both the functional and non-functional requirements outlined in section 4. Unit tests, as described in section 5, will also be performed. Additionally, a code walkthrough will be conducted with the class during the final presentation.

3.6 Automated Testing and Verification Tools

All system tests and unit tests will be performed using Python scripts. GitHub Actions is used for continuous integration, and the workflow will run all unit tests.

3.7 Software Validation Plan

The software validation plan is beyond the scope of TTE RecSys, as it requires additional time and data that are not available within the scope of the project.

4 System Tests

This section covers the system tests that will be applied to both the functional and non-functional requirements.

4.1 Tests for Functional Requirements

The functional requirements are tested in the following areas: input validation, model training convergence, and model storage. These tests ensure that the system behaves as expected under various conditions.

4.1.1 Area of Testing 1: Dataset

1. test-id1
   Control: Automatic
   Initial State: Before training the DNN.
   Input: A CSV file (data/processed/recommender_data.csv) containing user-item interactions with expected columns: 'User-ID', 'Book-Rating', 'Book-Title', 'Book-Author', 'Year-Of-Publication', 'Publisher', and 'Age'.
   Output: A boolean value: True if the dataset meets all validation criteria (contains required columns and each user-item pair has an associated reward), False otherwise.
   Test Case Derivation: Verifies R1 which requires that all user-item pairs in the training dataset should have an associated reward as specified in the SRS.

   How test will be performed: This test will be performed automatically using GitHub Actions via `test_data_validation.py`, which runs before model training to verify the dataset meets all requirements.

4.1.2 Area of Testing 2: Model Training Convergence

1. test-id2

   Control: Automatic
   Initial State: After training of the DNN
   Input: The training_history.json file containing loss values recorded during training

Output: A boolean value: True if the loss decreases over time (final loss $\leq$ initial loss), False otherwise.
Test Case Derivation: Verifies R2 which requires successful training of embedding functions as specified in the SRS.
How the test will be performed: This test will be performed automatically using GitHub Actions via `test_model_convergence.py`, which analyzes the training history after model training completes.

4.1.3 Area of Testing 3: Model Storage

1. test-id3:

   Control: Automatic
   Initial State: After model training is complete
   Input: Path to output directory containing trained models
   Output: A boolean value: True if all expected files exist and can be loaded with proper structure, False otherwise.
   Test Case Derivation: Verifies R3 which requires proper model storage and persistence.
   How test will be performed: The test will be performed automatically using GitHub Actions via `test_model_storage.py`, which checks for the existence and loadability of all output files.

4.2 Tests for Nonfunctional Requirements

4.2.1 Reliability

The reliability of the software is tested through the tests for functional requirements in section 4.1 and 5.2.

4.2.2 Portability

1. test-id4

   Type: Automatic and manual

   Initial State: None

   Input/Condition: None

Output/Result: The results of all automatic tests and feedback from users.

How test will be performed: All automatic tests will be conducted during the continuous integration workflow. Potential users will install the project on their computers (Windows, macOS, or Linux) and follow the instructions in README.md to run the software.

1. test-id5

   Type: Manual

   Initial State: The software is setup and ready to use.

   Input/Condition: None

   Output/Result: Survey result from the user

   How test will be performed: The user will be asked to fill out the survey after using this software. The survey can be found in appendix 6.1.

4.3 Traceability Between Test Cases and Requirements

The table 4.3 shows the traceability between test cases and requirements

	test-id1	test-id2	test-id3	test-id4	test-id5
R1	X				
R2		X			
R3			X		
R4		X			
R5		X			
R6		X			X
NFR1					X
NFR2	X	X	X		
NFR3				X	

Table 2: Traceability Matrix Showing the Connections Between Test Cases and Requirements

5 Unit Test Description

The unit tests for this system will follow a hierarchical approach based on the module decomposition in the Module Guide. The testing philosophy focuses on:

1. Black-box testing of module interfaces according to their specifications 2. White-box testing for complex algorithms and edge cases 3. Mock objects for isolating modules from their dependencies

5.1 Unit Testing Scope

The unit testing will focus on the modules with direct user interaction and core algorithmic functionality. Some modules will be tested indirectly through system tests or as part of the testing of other modules. The following modules will be tested directly with unit tests:

1. Data Processing Module (M2)
2. Embedding Generation Module (M4)
3. Neural Network Architecture Module (M6)
4. ANN Search Module (M7)
5. Vector Operations Module (M8)

The following modules are not directly tested with unit tests but are verified through system tests or user interaction:

Hardware System Interface (M1)

The Hardware System Interface module is tested indirectly through:

- System test-id3 (Model Storage), which verifies the file I/O capabilities of this module
- Usage within other modules, which depend on its functionality for loading and saving models and embeddings

This approach is sufficient because the module contains straightforward file I/O operations with well-defined error handling.

Model Training Module (M3)

The Model Training module is tested indirectly through:

- System test-id2 (Model Training Convergence), which verifies the training process produces properly converging loss values

Recommendation Module (M5)
The Recommendation module is tested indirectly through:

- Usability testing (test-id5), where users evaluate the quality of recommendations
- System-level evaluation that uses this module to generate final recommendations

This module integrates the outputs of other tested modules (embedding generation and ANN search), so thorough testing of those components provides confidence in this module's correct operation.

5.2 Tests for Functional Requirements

5.2.1 Data Processing Module (M2)

1. test-M2-1: Dataset Validation Test
   Type: Automatic, Functional

   Initial State: None

   Input: Path to CSV file

   Output: Boolean value (True if the dataset meets all validation criteria, False otherwise)

   Test Case Derivation: R1 requires the system to accept valid input data with each user-item pair having an associated reward.

   How test will be performed: Load the dataset using `load_data` method, then apply `validate_data` method to verify the dataset contains all required columns.

2. test-M2-2 Data Loading Test
   Type: Automatic, Functional

   Initial State: None

   Input: Path to a test CSV file

   Output: Pandas DataFrame containing loaded data with expected columns

Test Case Derivation: The system needs to correctly load data from CSV files.

How test will be performed: Call `load_data` with a test CSV file path and verify the DataFrame shape and column names match expected values.

3. test-M2-3 Data Preprocessing Test
   Type: Automatic, Functional

   Initial State: None

   Input: Pandas DataFrame with raw user-book interaction data

   Output: Processed Pandas DataFrame with additional derived features

   Test Case Derivation: R1 requires the system to preprocess data for model training.

   How test will be performed: Pass a sample DataFrame to `preprocess_data` and verify all expected derived features are created, including 'Author-Frequency', 'Publisher-Frequency', 'Age-Normalized', etc.

4. test-M2-4: Missing Value Handling Test

   Type: Automatic, Functional

   Initial State: None

   Input: Pandas DataFrame with missing values in various columns

   Output: Processed DataFrame with no NaN values

   Test Case Derivation: The system must handle incomplete data gracefully for production usage.

   How test will be performed: Create a DataFrame with deliberately missing values in 'Age', 'State', and 'Country' columns, process it with `preprocess_data`, and verify all NaN values are replaced with appropriate defaults.

5. test-M2-5: Training Data Creation Test
   Type: Automatic, Functional

   Initial State: None

   Input: Processed Pandas DataFrame with all required derived features

Output: Dictionary containing training data arrays: 'user_ids', 'item_ids', 'ratings', 'user_features', and 'item_features'

Test Case Derivation: The system needs to convert preprocessed DataFrame to a format suitable for neural network training.

How test will be performed: Pass a processed sample DataFrame to `create_training_data` and verify the returned dictionary contains all required keys with arrays of correct shapes.

5.2.2 Embedding Generation Module (M4)

1. test-M4-1: User Embedding Generation Test
   Type: Automatic, Functional

   Initial State: Initialized EmbeddingGenerator with trained models

   Input: NumPy array of user feature

   Output: NumPy array of user embedding vectors

   Test Case Derivation: R4 requires the system to generate user embeddings based on user features.

   How test will be performed: Generate embeddings for sample user features and verify embedding dimension of each embedding vector.

2. test-M4-2: Item Embedding Generation Test

   Type: Automatic, Functional

   Initial State: Initialized EmbeddingGenerator with trained models

   Input: NumPy array of item feature vectors

   Output: NumPy array of item embedding vectors

   Test Case Derivation: R4 requires the system to generate embeddings based on item features.

   How test will be performed: Generate embeddings for sample item features and verify embedding dimension of each embedding vector.

3. test-M4-3: Embedding Generator Initialization Test Type: Automatic, Functional

   Initial State: None

Input: Trained DNN for users and items

Output: Initialized EmbeddingGenerator instance

Test Case Derivation: The system must initialize embedding generators with compatible models that produce embeddings of the same dimension.

How test will be performed: Initialize the generator with models of compatible and incompatible dimensions, verify successful initialization for compatible models and error raising for incompatible ones.

5.2.3 Neural Network Architecture Module (M6)

1. test-M6-1: Network Architecture Test
   Type: Automatic, Functional

   Initial State: None

   Input: input_dim (int), embedding_dim (int)

   Output: DNN with the expected layer structure

   Test Case Derivation: R2 requires the system to create models for embedding generation with appropriate architecture.

   How test will be performed: Create a DNN with specified parameters and verify the network structure (input size, hidden layer size, output size) matches the expected configuration.

2. test-M6-2: Forward Pass Test

   Type: Automatic, Functional

   Initial State: Initialized DNN instance

   Input: PyTorch tensor of feature vectors

   Output: PyTorch tensor of embedding vectors

   Test Case Derivation: The network must produce properly normalized embeddings for similarity calculations.

   How test will be performed: Run a forward pass on a batch of sample data and verify output dimensions and L2-normalization property of each embedding.

3. test-M6-3: Network Initialization Test

   Type: Automatic, Functional

   Initial State: None

   Input: input_dim (int), embedding_dim (int)

   Output: DNN with properly initialized weights (non-zero)

   Test Case Derivation: Proper weight initialization is essential for training convergence and embedding diversity.

   How test will be performed: Create a network and verify weights are non-zero.

4. test-M6-4: Batch Processing Test

   Type: Automatic, Functional

   Initial State: Initialized DNN in evaluation mode

   Input: Single feature vector and batch of identical feature vectors

   Output: Identical embeddings for both inputs (when in evaluation mode)

   Test Case Derivation: Batch processing should produce the same embeddings as individual processing for the same inputs when not in training mode.

   How test will be performed: Compare embeddings generated by batch processing with those from individual processing, verifying consistent outputs in evaluation mode.

5.2.4 ANN Search Module (M7)

1. test-M7-1: Exact Match Search Test
   Type: Automatic, Functional

   Initial State: None

   Input: Array of item embeddings and a query embedding identical to one of the items in the array

   Output: Search results containing the exact matching item ID at the top position

Test Case Derivation: R6 requires efficient retrieval of nearest items, including exact matches.

How test will be performed: Build an ANN index with item embeddings including a known vector, search using that same vector as query, and verify the matching item is returned as the top result.

2. test-M7-2: Approximate Match Search Test
   Type: Automatic, Functional

   Initial State: None

   Input: Array of item embeddings and a query embedding similar but not identical to one of the items in the array

   Output: Search results containing the most similar item ID among the top results

   Test Case Derivation: R6 requires the system to find similar items even without exact matches.

   How test will be performed: Build an index, search with a query vector that is close to (but not identical to) a vector in the index, and verify the results include the expected similar item.

3. test-M7-3: Index Save/Load Test Type: Automatic, Functional

   Initial State: None

   Input: ANN index object and path to save/load location

   Output: Loaded index that produces identical search results to the original saved index

   Test Case Derivation: R3 requires model and index storage and loading.

   How test will be performed: Save an index to disk, load it back, and verify the loaded index produces the same search results for identical queries.

4. test-M7-4: Multiple Results Test
   Type: Automatic, Functional

   Initial State: None

   Input: Array of item embeddings, query embedding, and number of results to return (k)

Output: Array of exactly k (item_id, similarity_score) tuples sorted by descending similarity

Test Case Derivation: R6 requires the system to return a customizable number of recommendations.

How test will be performed: Build an index with multiple vectors, search with a query requesting k results, and verify exactly k results are returned in descending order of similarity.

5.2.5 Vector Operations Module (M8)

1. test-M8-1: Dot Product Calculation Test
   Type: Automatic, Functional

   Initial State: None

   Input: Two vectors (NumPy arrays or lists) of the same dimension

   Output: Scalar dot product value calculated as the sum of element-wise products

   Test Case Derivation: R5 requires consistent ranking based on dot product similarity.

   How test will be performed: Calculate the dot product of various vector pairs (identical, orthogonal, similar, opposite direction) and verify the results match expected values.

5.3 Tests for Nonfunctional Requirements

Unit testing the nonfunctional requirements is beyond the scope.

The nonfunctional requirements will be tested primarily through the usability survey.

5.4 Traceability Between Test Cases and Modules

The table 5.4 shows the traceability between test cases and modules.

References

Author Author. System requirements specification. https://github.com/..., 2019.

Test ID	M1	M2	M3	M4	M5	M6	M7	M8
test-M2-1		X						
test-M2-2		X						
test-M2-3		X						
test-M2-4		X						
test-M2-5		X						
test-M4-1				X				
test-M4-2				X				
test-M4-3				X				
test-M6-1						X		
test-M6-2						X		
test-M6-3						X		
test-M6-4						X		
test-M7-1							X	
test-M7-2							X	
test-M7-3							X	
test-M7-4							X	
test-M8-1								X

Table 3: Traceability Matrix Between Test Cases and Modules

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library, 2024. URL https://arxiv.org/abs/2401.08281.

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.

Yinying Huo. Software requirements specification for tte recsys, 2025. URL https://github.com/V-AS/Two-tower-recommender-system/blob/main/docs/SRS/SRS.pdf. Accessed: February 18, 2025.

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

David L. Parnas and P.C. Clements. A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering*, 12(2):251–257, February 1986.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.

James Robertson and Suzanne Robertson. *Volere Requirements Specification Template*. Atlantic Systems Guild Limited, 16 edition, 2012.

W. Spencer Smith. Systematic development of requirements documentation for general purpose scientific computing software. In *Proceedings of the 14th IEEE International Requirements Engineering Conference, RE 2006*, pages 209–218, Minneapolis / St. Paul, Minnesota, 2006. URL http://www.ifi.unizh.ch/req/events/RE06/.

W. Spencer Smith and Nirmitha Koothoor. A document-driven method for certifying scientific computing software for use in nuclear safety analysis. *Nuclear Engineering and Technology*, 48(2):404–418, April 2016. ISSN 1738-5733. doi: http://dx.doi.org/10.1016/j.net.2015.11.008. URL http://www.sciencedirect.com/science/article/pii/S1738573315002582.

W. Spencer Smith and Lei Lai. A new requirements template for scientific computing. In J. Ralyté, P. Ågerfalk, and N. Kraiem, editors, *Proceedings of the First International Workshop on Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes, SREP'05*, pages 107–121,

Paris, France, 2005. In conjunction with 13th IEEE International Requirements Engineering Conference.

W. Spencer Smith, Lei Lai, and Ridha Khedri. Requirements analysis for engineering computation: A systematic approach for improving software reliability. *Reliable Computing, Special Issue on Reliable Engineering Computation*, 13(1):83–107, February 2007.

W. Spencer Smith, John McCutchan, and Jacques Carette. Commonality analysis of families of physical models for use in scientific computing. In *Proceedings of the First International Workshop on Software Engineering for Computational Science and Engineering (SECSE 2008)*, Leipzig, Germany, May 2008. In conjunction with the 30th International Conference on Software Engineering (ICSE). URL http://www.cse.msstate.edu/~SECSE08/schedule.htm. 8 pp.

W. Spencer Smith, John McCutchan, and Jacques Carette. Commonality analysis for a family of material models. Technical Report CAS-17-01-SS, McMaster University, Department of Computing and Software, 2017.

Wikipedia. Dot product — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Dot%20product&oldid=1268352915, 2025a. [Online; accessed 17-February-2025].

Wikipedia. Gradient descent — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Gradient%20descent&oldid=1274464503, 2025b. [Online; accessed 17-February-2025].

Wikipedia. Mean squared error — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Mean%20squared%20error&oldid=1276072434, 2025c. [Online; accessed 17-February-2025].

Wikipedia. Stochastic gradient descent — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Stochastic%20gradient%20descent&oldid=1265558819, 2025d. [Online; accessed 17-February-2025].

Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Ajit Kumthekar, Zhe Zhao, Li Wei, and Ed Chi, editors. *Sampling-Bias-Corrected Neural Modeling for Large Corpus Item Recommendations*, 2019.

6 Appendix

6.1 Usability Survey

- On a scale from 1 to 5, with 5 being the most satisfied, how would you rate your overall experience using the software?
- Were the installation and setup process straightforward? If not, what difficulties did you encounter?
- Did you find the user interface intuitive and easy to navigate? If not, what improvements would you suggest?
- Did the software meet your expectations? If not, what features or improvements would you like to see?