System Verification and Validation Plan for TTE RecSys

Yinying Huo

April 1, 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 update after VnV presentation

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).
- Accuracy: Validate that the system achieves acceptable prediction accuracy on a held-out test set.
- 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, ranking consistency, and output correctness. These tests ensure 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: Dataset
   Output: A verified training dataset where each user-item pair has an associated reward.
   Test Case Derivation: Ensures that the system receives a valid dataset for training, as specified in the SRS.

   How test will be performed: This test will be performed automatically using GitHub Actions every time before training the DNN.

4.1.2 Area of Testing 2: Model Training Convergence

1. test-id2

   Control: Automatic
   Initial State: After training of the DNN
   Input: The loss record during training
   Output: A boolean value, "True" if the loss of the DNN decreases over

iterations, and “False” otherwise.
Test Case Derivation: Ensures the correctness of the output as specified in SRS.
How the test will be performed: This test will be performed automatically using GitHub Actions once training is complete.

4.1.3 Area of Testing 3: Model Storage

1. test-id3:

   Control: Automatic
   Initial State: After model training is complete
   Input: Path to the model and pre-computed item embeddings
   Output: bA boolean variable – “True” iif the model and pre-computed item embeddings are stored in the specified location, and “False” otherwise
   Test Case Derivation: Ensures R3 (model storage) is properly implemented
   How test will be performed: The test will be performed automatically using GitHub Actions once model training convergence is complete.

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.

4.2.3 Usability

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 ask to filfiled the servey after using this software. The survey can be find at 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 four core modules of the system: Data Processing, Vector Operations, Neural Network Architecture, and Recommendation. Other modules will be tested indirectly through system tests or as part of the testing of these core modules. External libraries (PyTorch, FAISS) are considered outside the scope of unit testing.

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 test dataset

   Output: Boolean value True if the dataset meets all validation criteria

   Test Case Derivation: R1 requires the system to accept valid input data.

   How test will be performed: Load the dataset and run validation with the module's validate_data method.

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

   Initial State: None

   Input: Path to CSV file

   Output: Dataset containing loaded data

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

How test will be performed: Load data from a test CSV and verify the shape and columns match expected values.

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

   Initial State: None

   Input: Dataset

   Output: Processed dataset with derived features

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

   How test will be performed: Process sample data and verify all expected derived features are created correctly.

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

   Type: Automatic, Functional

   Initial State: None

   Input: Dataset with missing values

   Output: Processed Dataset without missing values

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

   How test will be performed: Create a DataFrame with missing values, process it, and verify no NaN values remain.

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

   Initial State: None

   Input: dataset

   Output: Dictionary containing training data arrays

   Test Case Derivation: The system needs to convert dataset to training format.

   How test will be performed: Create training data from processed sample and verify the structure and shapes of the output.

5.2.2 Embedding Generation Module (M4)

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

   Initial State: Initialized EmbeddingGenerator

   Input: A random user data from the dataset

   Output: User embedding vector of the expected dimension.

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

   How test will be performed: Generate an embedding for a user from the dataset and verify its dimension and normalization.

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

   Type: Automatic, Functional

   Initial State: Initialized EmbeddingGenerator

   Input: A random item data from the dataset

   Output: Item embedding vector of the expected dimension.

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

   How test will be performed: Generate an embedding for an item from the dataset and verify its dimension and normalization.

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

   Initial State: None

   Input: User and item models

   Output: Initialized EmbeddingGenerator

   Test Case Derivation: The system must initialize embedding generators with proper models.

   How test will be performed: Initialize the generator with models and verify models are properly set.

5.2.3 Neural Network Architecture Module (M6)

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

   Initial State: None

   Input: Configuration parameters for input dimension, hidden layers, and embedding dimension

   Output: Neural network model with the specified architecture

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

   How test will be performed: Create a model with the architecture and verify the layer structure matches the specification.

2. test-M6-2: Forward Pass Test

   Type: Automatic, Functional

   Initial State: Initialized neural network

   Input: Batch of user/item features

   Output: Embeddings with expected shape and normalization

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

   How test will be performed: Run a forward pass on sample data and verify output dimensions and normalization.

3. test-M6-3: Network Initialization Test

   Type: Automatic, Functional

   Initial State: None

   Input: Configuration parameters

   Output: Network with properly initialized weights

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

   How test will be performed: Create a network and verify weights are initialized according to the specified scheme.

4. test-M6-4: Batch Processing Test

   Type: Automatic, Functional

   Initial State: Initialized neural network

   Input: Batch of features and individual features

   Output: Identical embeddings for batch and individual processing

   Test Case Derivation: Batch processing should behave the same as individual processing.

   How test will be performed: Compare embeddings generated by batch processing with those from individual processing.

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 from the production model and a query embedding

   Output: Array of (item_id, similarity_score) tuples with the expected length

   Test Case Derivation: R6 requires efficient retrieval of nearest items.

   How test will be performed: Build an index with item embeddings, search for an exact match, and verify the result.

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

   Initial State: None

   Input: Array of item embeddings and a query embedding that is similar to one of the items

   Output: Array of (item_id, similarity_score) tuples where the top results include the similar item

   Test Case Derivation: R6 requires the system to find similar items.

   How test will be performed: Build an index, search with a similar query, and verify the results include the expected item.

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

   Initial State: None

   Input: ANN index and path to save/load

   Output: Loaded index matching the saved index

   Test Case Derivation: R3 requires model storage and loading.

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

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

   Initial State: None

   Input: Array of item embeddings, query embedding, and number of results

   Output: Array of (item_id, similarity_score) tuples with the

5.2.5 Vector Operations Module (M8)

1. test-M8-1
   Type: Automatic, Functional

   Initial State: None

   Input: Two embedding vectors

   Output: Scalar dot product value

   Test Case Derivation: The system relies on dot product for similarity calculation.

   How test will be performed: Calculate the dot product of two vectors and compare it with the expected value.

5.3 Tests for Nonfunctional Requirements

Unit testing the nonfunctional requirements is beyond the scope.

5.4 Traceability Between Test Cases and Modules

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

Test ID	M1	M2	M3	M4	M5	M6	M7	M8
test-M2-1		X						
test-M3-1			X					
test-M4-1				X				
test-M5-1					X			
test-M6-1						X		
test-M7-1							X	
test-M8-1								X

Table 3: Traceability Matrix Between Test Cases and Modules

References

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

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?