System Verification and Validation Plan for TTE RecSys

Yinying Huo

March 2, 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

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

Huo (2025) Software requirements specification for this project.

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 MSI 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

The following tools will be used for automated testing and verification:

- **Unit Testing**:
  - **Pytest**: For testing individual components (e.g., embedding functions, ANN search, ranking logic).

- **Continuous Integration (CI)**:

- **GitHub Actions**: To automate testing and deployment workflows.
- **CML (Continuous Machine Learning)**: Automatically generates performance reports and emails metrics when changes are pushed to GitHub.
- The CI workflow will run all unit tests.

3.7 Software Validation Plan

The system will be validated using a 20% testing dataset split from the original dataset. This dataset will be used to evaluate the performance of the two embedding functions (user and item towers) by measuring metrics such as recall and precision.

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 embedding functions.
   Input: Dataset
   Output: A verified dataset where each user-item pair in the verified dataset has an associated reward and no missing values.
   Test Case Derivation: Ensures that the system handles valid inputs as specified in the SRS, which also includes the training dataset used for this project.

   How test will be performed: Automated test using GitHub Actions.

4.1.2   Test for performance

1. test-id2

   Control: Automatic
   Initial State: After training of the embedding functions
   Input: The testing dataset and embedding functions
   Output: The performance of the user embedding and item embedding
   functions is acceptable (accuracy > 80%)
   Test Case Derivation: Ensures the correctness of the output as speci-
   fied in SRS.
   How the test will be performed: This test will be performed automat-
   ically using GitHub Actions once the training is finished.

2. test-id3

   Control: Automatic
   Initial State: System initialized with pre-computed item embeddings
   and trained ANN index.
   Input: user embeddings
   Output: The ranking for the top 5 items should generally be the same
   for identical user embeddings. Non-consistent rankings are allowed (up
   to 5%) due to the inherent randomness in machine learning.
   Test Case Derivation: Ensures that identical user embeddings produce
   identical item rankings, as stated in SRS.
   How the test will be performed: This test will be performed automati-
   cally using GitHub Actions. It will generate 100 lists of recommended
   items for a random user.

4.1.3   Area of Testing 3: Model Storage

1. test-id4:

   Control: Automatic
   Initial State: After model training is complete
   Input: Trained model to be saved
   Output: Model successfully stored and can be retrieved with identical

parameters

Test Case Derivation: Ensures R3 (model storage) is properly implemented How test will be performed: Automated test that saves the model, loads it back, and verifies parameter integrity

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-id5

   Type: Manual

   Initial State: None

   Input/Condition: None

   Output/Result: The project should pass all the tests for functional requirements and run without any errors.

   How test will be performed: Potential users will install the project on their computers (Windows, macOS, or Linux) and execute a sample workflow.

4.2.3 Scalability

1. test-id6

   Type: Manual

   Initial State: System initialized with a model trained on 70% of the dataset.

   Input/Condition: Remaining 30% of the dataset added as new data.

   Output/Result: Model updates complete within 1 minute.

   How test will be performed: Because there is no external data for updates, the system will first train the model on 70% of the training set.

After training is done and all the functional requirements have passed, the remaining 30% will be added as new data to simulate incremental updates.

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	test-id6
R1	X					
R2		X				
R3				X		
R4			X			
R5		X	X			
R6		X	X			
NFR1						X
NFR2	X	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

All modules developed for this project will be tested. External libraries (PyTorch, FAISS) are considered outside the scope of unit testing.

5.2 Tests for Functional Requirements

5.2.1 System Interface Module (M1)

1. test-M1-1

   Type: Automatic, Functional

   Initial State: Empty file system storage location

   Input: A trained model object with known parameters

   Output: Boolean value 'True' indicating successful save operation, and file exists at the specified location with correct format

   Test Case Derivation: R3 requires the system to store trained embedding functions. This test verifies the save model functionality works correctly by checking both the return value and the actual file creation.

   How test will be performed: Create a small test model, save it to a temporary location, verify the return value is True, and check that the file exists and has non-zero size.

2. test-M1-2

   Type: Automatic, Functional

   Initial State: File system with a previously saved model

   Output: A model object with identical parameters to the one that was saved

   Test Case Derivation: R3 requires stored models to be retrievable. This test verifies that a model can be loaded without corruption by comparing parameters before and after the save-load cycle.

   How test will be performed: Save a model with known architecture and parameter values, load it back, and verify the architecture and parameters match the original.

3. test-M1-3

   Type: Automatic, Functional, Exception Handling

   Initial State: File system with no access permissions

   Input: A model to be saved to a protected location

   Output: IOError exception with appropriate error message

8

Test Case Derivation: The system should handle file system errors gracefully. This test verifies that the appropriate exception is raised when file operations fail.

How test will be performed: Attempt to save a model to a location without write permissions and verify the correct exception is raised.

4. test-M1-4
Type: Automatic, Functional

Initial State: File system with a previously saved batch of item embeddings

Input: Path to the saved embeddings

Output: Array of embeddings identical to the ones that were saved

Test Case Derivation: R3 requires precomputed item embeddings to be stored and retrieved. This test verifies that embeddings can be loaded correctly.

How test will be performed: Save a batch of embeddings with known values, load them back, and verify the values match the original.

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

Initial State:

Input/Condition:

Output/Result:

How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.3.2   Module ?

...

5.4   Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

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.

6   Appendix

This is where you can place additional information.

6.1   Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2   Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning.

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?