System Verification and Validation Plan for TTE RecSys

Yinying Huo

February 24, 2025

Revision History

Date	Version	Notes
Feb 24, 2025	1.0	First draft – Unit tests will be added later.

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

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 a 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, gradient descent) 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 handle a large number of users or items with reasonable latency.

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

[Introduce this section. You can provide a roadmap of the sections to come. —SS]

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.

- **Static Analysis and Linting**:
  - **Black**: For automated code formatting.
  - **Flake8**: For enforcing Python coding standards.

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 Testing1: Input Validation

The following test will ensure that there is no missing data in the training and testing datasets, and that all input constraints are satisfied as specified in the data constraints table of the SRS.

Test for Valid Inputs

1. test-id1
   Control: Automatic
   Initial State: During the data cleaning, before traning the embedding function.
   Input: Valid user features (e.g., location, age), item features (e.g., book title, author, year of publication), and the corresponding rating for each user-item pair.
   Output: Each user-item pair have an accosiated reward and no missing features.
   Test Case Derivation: Ensures the system handles valid inputs as specified in SRS.
   How test will be performed: Automated test using Pytest.

4.1.2 Area of Testing 2: Performance of the embedding functions

Test for performance

1. test-id2

   Control: Automatic
   Initial State: After training of the embedding functions
   Input: The testing dataset
   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 specified in SRS.
   How the test will be performed: Use GitHub Actions to automatically evaluate the performance once the training is finished.

2. test-id3

   Control: Automatic
   Initial State: System initialized with pre-computed item embeddings.
   Input: user embeddings
   Output: The rankings for the top 10 items should generally be the same for identical user embeddings, but 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: Automated test using Pytest.

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 .

4.2.2 Portability

1. test-id4

   Type: Manual

   Initial State: None

   Input/Condition: None

   Output/Result: The system operates correctly on users' machines, with all functionalities working as expected.

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

4.2.3 Scalability

1. test-id5

   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. The remaining 30% will then be added as new data to simulate incremental updates. The time taken to update the model will be measured and verified to complete within the expected time.

4.3 Traceability Between Test Cases and Requirements

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

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

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

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

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

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

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

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

[This section is not required for CAS 741 —SS]

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?