Verification and Validation Report: TTE RecSys

Yinying Huo

April 11, 2025

1 Revision History

Date	**Version**	**Notes**
Apr. 2 2025	1.0	First draft
Apr. 11 2025	2.0	Revision 1

2 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

Contents

List of Tables

This document includes the results of VnV plan.

3 Functional Requirements Evaluation

The functional requirements are tested using both system tests and unit tests.

3.1 Dataset

To ensure the project receives a valid dataset for training:

Manually, a Jupyter Notebook (data_preprocessing.ipynb) is used to merge the raw data (three CSV files) into a single CSV file.

Then, the automated system test (test_data_validation.py) runs each time the dataset is updated before training starts to ensure that the input CSV file contains all the required columns.

Additionally, an automated unit test (test_data_processing.py) runs on each push to verify the correctness of the data processor, which generates new features for users and items.

All tests have passed successfully.

3.2 Model Training Convergence

The system test (test_model_convergence.py) checks whether the training loss decreases as training progresses.

This test has passed successfully.

3.3 Model storage

The system test (test_model_storage.py) runs each time model training is completed. The test ensures that the trained model and computed embeddings are correctly stored in the 'output' folder.

This test has passed successfully, confirming that:

- The user model (user_model.pth) is saved with the correct state dictionary format
- The item model (item_model.pth) is saved with the correct state dictionary format

- Item embeddings (item_embeddings.npy) are saved with the expected shape
- The ANN index is properly saved and can be loaded for inference

3.4 Embedding Generation

The unit test 'test_embedding_generation.py' verifies the correctness of embedding generation. This test has passed successfully, confirming that:

- The embedding generator correctly initializes with both user and item models
- Single user/item embeddings are generated with the expected dimensions
- All embeddings are properly normalized (unit norm)

4 Nonfunctional Requirements Evaluation

4.1 Usability

The usability of the system was evaluated using a survey as outlined in the VnV Plan.

4.2 Reliability

Reliability testing was conducted through both system tests and unit tests. The system demonstrated robust performance with the following results:

- All functional tests (test-id1, test-id2, test-id3) passed.
- Data validation tests showed 100% detection rate for improperly formatted inputs
- The system correctly handled edge cases such as:
  - Users with no location information
  - Extreme age values

- Recovery from invalid input was graceful, with appropriate error messages

The system meets the reliability requirement (NFR2) by performing as expected and handling edge cases appropriately.

4.3 Protability

The system's portability was tested across multiple operating systems as specified in NFR3:

5 Unit Testing

Unit tests were developed for five key modules. All unit tests were executed with pytest and integrated into the continuous integration pipeline.

5.1 Data Processing Module (M2)

The data processing module tests in 'test_data_processing.py' verify:

- Data loading functionality with both valid and invalid data files
- Data validation correctly identifies valid and invalid dataset formats
- Missing value handling ensures no NaN values
- Training data creation produces correctly structured input for the model
- Book mapping function correctly maps encoded IDs to book metadata

All tests passed successfully, indicating the data processing module functions as expected.

5.2 Embedding Generation Module (M4)

The embedding generation tests in 'test_embedding_generation.py' verify:

- Initialization with compatible and incompatible models
- User embedding generation for both single users

- Item embedding generation for both single items
- All generated embeddings are properly normalized
- Large batch handling for item embeddings

All tests passed successfully, confirming the embedding generation module meets its requirements.

5.3 Neural Network Module (M6)

The neural network tests in ‘test_neural_network.py‘ verify:

- Tower network initialization with correct architecture
- Forward pass produces normalized outputs of the expected dimension
- User tower creation function produces correct models
- Item tower creation function produces correct models
- Weight initialization produces non-zero weights with expected properties
- Different inputs produce different embeddings

All tests passed successfully, confirming the neural network module creates models with the expected behavior.

5.4 ANN Search Module (M7)

The ANN search tests in ‘test_ann_search.py‘ verify:

- Building a flat index with proper structure
- Exact match search correctly finds known vectors
- Approximate match search finds vectors similar to the query
- Multiple results retrieval with the correct number of candidates
- Error handling for empty or mismatched inputs
- Saving and loading indices preserves search functionality

All tests passed successfully, validating the ANN search module’s ability to perform efficient vector similarity search.

5.5 Vector Operations Module (M8)

The vector operations tests in 'test_vector_operations.py' verify:

- Basic dot product calculation with different vector formats
- Handling of empty vectors
- Orthogonal, same-direction, and opposite-direction vectors
- Handling of large and small values
- Error detection for dimension mismatches

All tests passed successfully, confirming the vector operations module correctly implements mathematical operations needed for similarity calculations.

6 Changes Due to Testing

Throughout the development process, many minor bugs and formatting issues in the document were fixed.

The major change was due to the poor performance (accuracy) of the DNN model, which resulted from the limited data. The quality of the recommendation results was poor because the dataset contained very few user features. Additionally, many items received ratings from only a single user, and the item features were also limited. This led to embedding collapse, meaning the model generated nearly identical embeddings for different inputs.

As a result, the system recommended very similar books to each different user.

After applying techniques such as feature engineering and negative sampling, the issue improved but still persisted. Due to time constraints, I was unable to implement more advanced techniques. Therefore, I had to remove the performance requirements.

7 Automated Testing

The project utilizes GitHub Actions for automated testing and continuous integration. The automated testing workflow follows a structured trigger-based approach:

- All unit tests and system tests run automatically whenever there is a change to any file in the `src` directory
- The dataset validation test (`test_data_validation.py`) executes whenever there are changes to the dataset files
- When dataset changes are detected and validation passes, the model retraining process is triggered automatically
- After model retraining completes, the model convergence test (`test_model_convergence.py`) and model storage test (`test_model_storage.py`) run sequentially to verify the new model

This automation pipeline ensures that code changes are continuously validated, data integrity is maintained, and the model quality is verified after each training cycle.

8 Trace to Requirements

The following table shows the traceability between tests and functional requirements:

The test coverage demonstrates that each functional requirement is verified by at least one test, ensuring comprehensive validation of system functionality.

For non-functional requirements:

- NFR1 (Usability): Verified through user surveys
- NFR2 (Reliability): Verified through system and unit tests.
- NFR3 (Portability): Verified by manually installing and running the system on different computers.

Test ID	R1	R2	R3	R4	R5	R6
test-id1 (Dataset Validation)	X					
test-id2 (Model Convergence)		X		X	X	
test-id3 (Model Storage)			X			
test-data-processing	X					
test-embedding-generation				X		
test-neural-network		X				
test-ann-search					X	X
test-vector-operations					X	X

Table 1: Traceability Matrix Between Tests and Requirements

9 Trace to Modules

The following table shows the traceability between tests and modules:

Test ID	M1	M2	M3	M4	M5	M6	M7	M8
test-id1 (Dataset Validation)		X						
test-id2 (Model Convergence)			X			X		
test-id3 (Model Storage)	X							
test-data-processing		X						
test-embedding-generation				X	X			
test-neural-network						X		
test-ann-search					X		X	
test-vector-operations					X			X

Table 2: Traceability Matrix Between Tests and Modules

Each module in the system is covered by at least one test, providing confidence in the correctness of individual components as well as their integration.

10 Code Coverage Metrics

Code coverage metrics were collected using pytest-cov during the automated testing process. The following table summarizes the coverage results for unit tests:

Module	**statements**	**missing**	**excluded**	**coverage**
modules/data_processing.py	119	17	0	86%
modules/embedding_generation.py	58	0	0	100%
modules/neural_network.py	28	0	0	100%
modules/ann_search.py	82	17	0	79%
modules/vector_operations.py	14	5	0	64%
Overall	301	39	0	87%

Table 3: Code Coverage

There is no unit testing for `main.py`, `model_training.py`, `recommendation.py`, `system_interface.py`, and `user_interface.py` due to limited time.

However, the modules `model_training.py`, `recommendation.py`, and `system_interface.py` are inherently tested through system tests and by users running the system using `user_interface.py`.

The code coverage report is also available as an artifact for download in GitHub Actions after each run of all unit tests.

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. Ȧgerfalk, 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.

Appendix — Reflection

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

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. Which parts of this document stemmed from speaking to your client(s) or a proxy (e.g. your peers)? Which ones were not, and why?
4. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)