# A Personalized AI Tool Recommendation System Using Semantic Embeddings and LLMs

Sanskar S. Kotalwar[1], Ganesh D. Bhutkar[2], Neel R. Kote[3], Rishikesh B. Kothavade[4], and Abhishek G. Kotalwar[5]

[1,2,3,4,5]*Department of Computer Science, Vishwakarma Institute of Technology*

*Abstract*— **The fast development of artificial intelligence tools in various domains leads to information overload for the user. Consequently, it is not easy for the individual to identify tools that will suit their needs. Currently, most directories of AI tools utilize either keyword searching or manual organization methods. Such an approach fails to understand the query posed by the user and rarely updates recommendations based on the latter's behavior. In this paper, we introduce a real-time artificial intelligence tool recommendation system that can be used to solve the above-mentioned problems. There are three major stages in our system design. First, we use a combination of sentence transformer models and vector similarity search for semantic retrieval purposes. Second, we leverage large language models hosted on-premises via the Ollama framework to boost the effectiveness of ranking. Third, we offer personalized recommendations by means of a hybrid recommendation algorithm based on content-based and collaborative filtering and additional semantic similarity and engagement signals. Our approach is tested on a dataset of more than five hundred artificial intelligence tools that the authors collected and annotated manually. We present promising results for both Quality and relevance of the tools discovered. The outcome shows that the use of vector-based search along with reasoning by a large language model and personalized adaptability will help discover AI tools better.**

*Index Terms*—**AI Tool Discovery, Collaborative Filtering, FAISS, Hybrid Recommendation, Large Language Models, Personalized Ranking, Semantic Search, Sentence Transformers.**

## I. INTRODUCTION

AI (Artificial Intelligence) tools' ecosystem is rapidly expanding. It contains many specialized AI tools for such uses as image generation, code completion, data analysis, and natural language processing. By 2024, directories including Futurepedia and There's an AI for that have collected over 10,000 tools in their directories, which makes tool discovery difficult. Often users need help finding the right tool, without wasting too much time on evaluating possible options. Rapid access to the appropriate tool increases efficiency, improves the quality of research and provides AI accessibility. For non-tech-savvy people, fast tool search will allow them to implement AI in their work process. For developers and researchers, relevant recommendations minimize context switching time and improve prototyping speed. Existing solutions have multiple drawbacks. Firstly, search by keywords alone struggles with synonyms or alternative wording of requests (for instance, "text summarization" or "text condensing"). Secondly, static filters ignore changes in user needs, previous interactions or even collective intelligence. Thirdly, plain LLM-based (Large Language Model) responses may generate fictional tool titles or outdated recommendations without using databases. Most directories also lack a way to personalize feedback since they do not adjust to individual user interactions like dwell time, Click-Through Rate (CTR) or bookmarking. While recommendation systems are well studied in areas like e-commerce and content streaming, their use in AI tool discovery is mostly uncharted. Existing approaches typically focus on retrieval-only methods (like TF-IDF (Term Frequency – Inverse Document Frequency) or BM25) or LLM prompting without a structured dataset, and no previous system combines vector-space semantic search, LLM contextual re-ranking, and multiple personalization signals into a single real-time architecture for AI tool recommendations. This paper introduces a system that bridges this gap by merging semantic retrieval, LLM reasoning, and adaptable user modeling into a production-ready full-stack application. The main contributions of this paper are a three-stage recommendation pipeline, which combines FAISS(Facebook AI Similarity Search)-

based semantic search, Ollama LLM re-ranking, and adaptive hybrid scoring within a real-time system that can respond within 2 seconds for standard queries, an adaptive hybrid scoring engine, a four-signal personalization module that merges content-based, collaborative, semantic, and engagement scores, adjusting weights based on user feedback density, outperforming any single-signal alternative, and a curated AI tool dataset with a preprocessing pipeline, a structured, normalized dataset of over 500 AI tools with 18 features, along with a reproducible data cleaning and embedding generation pipeline available for future research.

## II. LITERATURE REVIEW

### *A. Semantic Search and Dense Retrieval*

Reimers & Gurevych (2019) introduced SBERT (Sentence Bidirectional Encoder Representations from Transformers), modifying the BERT (Bidirectional Encoder Representations from Transformers) architecture with siamese and triplet networks to create semantically meaningful sentence embeddings. Their model achieved top results on Semantic Textual Similarity(STS) tasks. Limitation: SBERT was not used for structured product/tool catalogs with multi-attribute metadata and was not integrated with LLM-based ranking.

Johnson et al. (2019) presented FAISS, a library for efficient similarity search over dense vectors. They showed billion-scale nearest-neighbor retrieval with sub-linear time complexity using inverted file structures and product quantization. Limitation: FAISS retrieval does not consider contextual query understanding and lacks personalization.

### *B. LLM-Based Recommendation*

Hou et al. (2023) examined Large Language Models as zero-shot recommendation systems, using GPT-4 (Generative Pre-trained Transformer 4) with item descriptions and query context. They reported strong zero-shot rankings on movie recommendation benchmarks but pointed out the lack of real-time performance and high API (Application Programming Interface) costs. Limitation: Cloud-based LLMs introduce latency and privacy issues; responses do not reference a curated, structured tool catalog.

Wang et al. (2023) introduced InstructRec, which fine-tunes instruction-following LLMs for personalized recommendation by encoding user preferences in natural language prompts. Limitation: This approach requires costly fine-tuning and does not integrate with vector-store retrieval for factual accuracy.

### *C. Hybrid Recommendation Systems*

He et al. (2017) proposed Neural Collaborative Filtering (NCF), which merges matrix factorization with multi-layer perceptrons to model user-item interactions. NCF outperformed traditional matrix factorization on MovieLens and Pinterest datasets by 3.5–7.5% in hit rate@10. Limitation: NCF needs extensive historical interaction data, leading to a cold-start issue, and does not incorporate natural language queries or semantic similarity.

Koren et al. (2009) introduced foundational matrix factorization techniques used in the Netflix Prize, achieving a 10% improvement over Netflix's benchmark. Limitation: These methods are solely interaction-based and cannot address the textual and semantic nature of tool discovery queries.

### *D. How Our Work Is Different*

Our system differs from previous efforts by: (a) grounding LLM responses in a curated, structured dataset instead of depending on LLM memory; (b) operating entirely on local, privacy-preserving LLMs via Ollama, bypassing cloud API dependence and costs; (c) incorporating four different personalization signals — explicit feedback, collaborative filtering, semantic similarity, and implicit engagement — into a single adaptive scoring model; and (d) being designed for real-time use with efficient FAISS indexing and asynchronous API design, which has not been tackled in previous tool recommendation work.

## III. RESEARCH METHODOLOGY

### *A. System Flow*

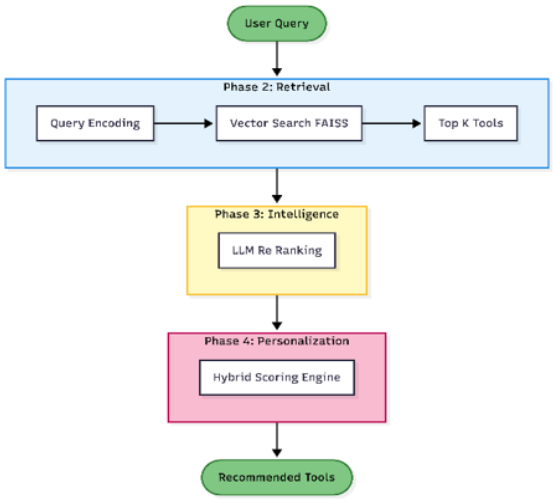


FIG.1 SYSTEM FLOW

As shown in the diagram, it is a multi-stage pipeline of recommendation generation of AI tools. The initial step is that the user will input a query in the form of natural language. The second phase, known as Retrieval, is when the query is first translated into a vector representation to be passed into the Faiss-based vector search model for finding out the top-K similar tools based on semantics. In the third phase, Intelligence, the LLM re-ranking will further enhance the result by focusing on context-based relevancy of the retrieved tools. Finally, Personalization is carried out using the hybrid ranking method to score the tools according to the preferences and engagement of the user.

*B. Dataset*

The database has been curated and validated by using public resources such as directories of AI technologies that include Futurepedia, There's an AI for That and the respective websites of each tool individually. There are more than 500 AI tools included in the database, categorized meticulously under 15 main groups, such as image generation, coding assistance, summarizing text, analyzing data, and speech synthesis, among others.

TABLE I. FEATURES (18 DIMENSIONS)

| Feature | Type | Description |
|---|---|---|
| Name | Text | Tool name |
| Company | Text | Developer/company |
| Category | Categorical | Primary use category |
| Subcategory | Categorical | Secondary use case |
| Task Description | Text | Natural language description |
| Input Type | Categorical | Text, Image, Audio, etc. |
| Output Type | Categorical | Text, Image, Code, etc. |
| Cost | Categorical | Free / Freemium / Paid / Open-Source |
| Ease of Use | Ordinal | Beginner=1, Developer=3, Enterprise=4 |
| Rating | Continuous | Normalized 0–1 |
| Popularity | Ordinal | Low=0.3, Medium=0.6, High=1.0 |
| Accuracy | Continuous | Normalized 0–1 |
| Speed | Ordinal | Fast=1.0, Medium=0.5, Slow=0.2 |
| Integration | Text | Supported platforms / APIs |
| Languages | Text | Supported natural languages |
| Training Domain | Text | Domain of training data |
| Link | URL | Official product URL (Uniform Resource Locator) |
| ai_id | Integer | Unique identifier |

Table I provides the complete set of attributes used to represent each AI technology in the dataset. The dataset contains a total of 18 attributes encompassing text data, categorical data, ordinal data, and numeric data. Text attributes like name, company, and function provide descriptive information about each AI technology. Categorical attributes like category, input type, and cost help with classifying and filtering the technologies. Numeric attributes like rating, accuracy, and popularity have been normalized for uniformity across the dataset.

*C. Recommendation Model and Algorithm*

*a. Semantic Retrieval (Stage 1)*

The semantic search module embeds the user input q into vector v_q ∈ R^384 through the use of the all-MiniLM-L6-v2 model. The FAISS index comprises pre-normalized embedding vectors of tools {v_i}.Similarity is calculated as cosine similarity, which uses the inner product on L2-normalized vectors. The top-K results are returned in sub-millisecond time after a one-time index load. Why selected: The all-MiniLM-L6-v2 model was chosen for its strong performance in semantic textual similarity, achieving a 77.4 STS-B (Semantic Textual Similarity Benchmark) Spearman correlation, its small size of 80MB, and its fast speed of about 1400 sentences per second on CPU (Central Processing Unit). Fallback chain: If the FAISS search fails or gives no results, the system falls back to (1) raw embedding cosine similarity over the loaded NumPy matrix, followed by (2) TF-IDF cosine similarity, and then (3) LLM-only generation.

*b. LLM Re-Ranking (Stage 2)*

The semantic candidates are organized into a structured prompt that clearly lists each tool's ID, name, and description. This prompt then instructs the LLM to return a JSON (JavaScript Object Notation) array containing the ranked results, including fields like ai_id, score(ranging from 0.0 to 1.0), and a short reasoning for each selection. This will ensure that the output is both consistent and easier to manipulate for further processing.

For the implementation, we have used two different models on our local machine for the inference process. We have chosen Qwen 2.5 (3B) as the main model; however, we have a backup model called Mistral 7B-Instruct. This way, we can ensure that there won't be any hanging issues with the models as they are being deployed on Ollama. Additionally, there will be no need for a cloud service that can take time; thus, privacy is maintained during the entire process.

The best part about using Qwen 2.5 (3B) is that it's quite efficient and can perform well in scenarios with limited resources. It can reason effectively with only 4GB of Random Access Memory(RAM) available to it.

We have added a timeout of 120 seconds to avoid any hanging issues. Moreover, the subprocesses add a level of abstraction and allow for greater modularity in the code. We have also provided a fallback function in case of any non-perfect JSON output.

*c. Adaptive Hybrid Scoring (Stage 3)*

For authenticated users, the final score for each tool i is computed as:

score(i) = w_content × P(i) + w_collab × CF(i) + w_semantic × S(i) + w_engagement × E(i)

Where:

P(i) – Personal content score: derived from personal feedback (+1/-1 for Like/Dislike), rating score (0, 1), whether it is bookmarked (1 for yes, 0 otherwise) and sentiment of the comment (from -1 to +1), normalized on [-1, +1] scale.

CF(i) – Collaborative Filtering Score: the global average ratio of likes = likes/(likes+dislikes) for all the users, on [0, 1] scale.

S(i) – Semantic Score: calculated based on the decay function based on Faiss ranking position, i.e., 1-(position/K).

E(i) – Engagement score: derived from CTR smoothing (clicks+0.5)/(impressions+α), normalized dwell time (not exceeding 300 s) and log-normalized repeat visits less the cost of search refinement.

Dynamic Weights: Initial weights are {content: 0.45, collaborative: 0.30, semantic: 0.15, engagement: 0.10}. For users with fewer than 5 known signal points, content weight is increased by 1.2, while collaborative weight is set to 0.8 (bias towards content-based recommendation in case of cold start). Otherwise, multipliers work inversely (1.1* collaborative, 0.9* content).

*d. Implementation*

This system was developed and tested using macOS operating system with Apple silicon (M-series) and Python version 3.10 or above. The vectors indexing using FAISS is carried out through the use of CPU, and that guarantees speed in the process of similarity search despite the fact that there is no specialized Graphics Processing Unit (GPU) hardware. The re-ranking is carried out by an Ollama Large Language Model running locally and that has the ability to run models of 3B parameters or more. The backend of the application is developed using FastAPI and has RESTful APIs for different purposes. /api/recommend API is the combination of semantic search and scoring, while /api/deep_search API uses an LLM for more complex re-ranking. It is possible to monitor user interactions through other APIs, such as /api/feedback API, which gathers explicit user feedback like likes/dislikes/ratings/saved queries, or /api/log_event API, which gathers implicit user interaction data. Authentication in the system is supported by using JWT (JSON Web Token) authentication.

TABLE II. TOOLS AND TECHNOLOGIES

| Component | Technology |
|---|---|
| Backend Framework | FastAPI 0.115.2 + Uvicorn 0.30.1 |
| Database | MySQL 8.0+ with connection pooling |
| Vector Search | FAISS 1.8.0 (CPU) |
| Embeddings | sentence-transformers 3.2.1 (all-MiniLM-L6-v2) |
| LLM Runtime | Ollama (Qwen 2.5:3B, Mistral:7B-Instruct) |
| Authentication | PyJWT (HS256 (HMAC with SHA-256) JWT (JSON Web Token) tokens) |
| Data Processing | Pandas 2.2.3, NumPy 1.26.4, scikit-learn 1.5.2 |
| Frontend | Vanilla HTML5 / CSS3 / JavaScript (SPA) |

The implementation technologies for each component of the system are listed in Table II above. The back-end architecture, database, vector search engine, embeddings, LLM inference engine, authentication scheme, and processing tools are all covered. Each technology is selected based on performance, scalability, and ability to work in real time. The

following table demonstrates how each component of the proposed system can be implemented.

## IV. RESULTS AND DISCUSSION

### *A. System Functional Observations*

This system was tested qualitatively using functional testing for all three phases of the process. Several test questions were raised in order to observe how each stage responded to various inputs.

AI Tool Recommendation System's landing page is where users interact with the application initially. It consists of a simplistic yet very user-friendly interface which includes the ability to input a natural language question via the search bar. There are two types of searches: one called "Search," and the other one called "Deep Search." Results can be filtered based on such criteria as price, category, language, and so on. The latter will enable personalization when users are logged into the website. In general, landing page design is simple but powerful enough to give a user both a basic search option and a more complex one.

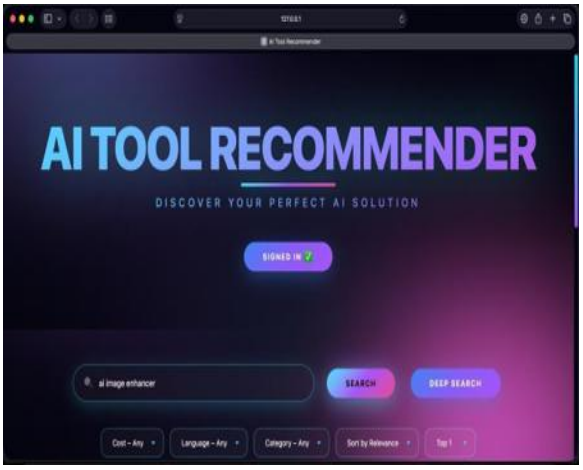


Fig.2 Landing Page of the System

The diagram below illustrates what appears on the screen when users enter their search query. It provides a list of suggested AI tools, which come with useful information such as the title of the application, its category, rating, and brief description. The search engine does not simply match keywords; rather, it analyzes the intention of the user in order to recommend tools that are suitable for his or her needs. For instance, a search query that involves terms such as "AI image enhancer" would result in suggestions for tools that can be used to enhance and upscale images.

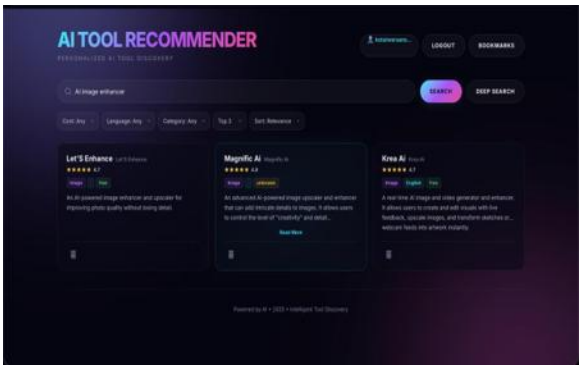


Fig.3 Result of User Query

### *B. Semantic Retrieval Observations*

The proposed semantic retrieval component based on the FAISS library was tested on different natural language queries for measuring its performance in terms of matching the intent of the user to suitable AI applications. Below is a summary of important findings made during the test:

TABLE III. EVALUATION OF FAISS-BASED SEMANTIC RETRIEVAL WITH SAMPLE QUERIES

| Sample Query | Top-3 Retrieved Tools | Observation |
|---|---|---|
| "Generate images from text" | DALL·E, Midjourney, Stable Diffusion | Correctly identified image generation tools without exact keyword matches with tool names |
| "Summarize long documents" | ChatGPT, Quillbot, Jasper AI | Retrieved text summarization tools and recognized "long documents" as a paraphrase for text input |
| "Code completion for Python" | GitHub Copilot, Codeium, Tabnine | Precise category match with code assistant tools |
| "Convert speech to text" | Whisper, Otter.ai, Deepgram | Understood the semantic intent of speech-to-text despite different tool descriptions |
| "Free tool for logo design" | Canva, Looka, Brandmark | Recognized both the task (logo design) and the constraint (free) from natural language |

Table III is an analysis of the semantic search algorithm based on example queries by the users. It shows the three best-retrieved items for each query and makes comments about the overall performance of the algorithm. The table demonstrates how the

algorithm understands the intentions of the user, handles synonyms and paraphrasing, and retrieves the required tools even without exact key-word matching. The semantic retrieval phase was able to address the problem of dealing with synonymous queries, paraphrased queries, as well as more complicated queries containing information about both types of tasks and restrictions on them.

### *C. LLM Re-Ranking Observations*

The deep search process (Stage 2) yielded better rankings than just semantic search. Notable observations include:

- Contextual reasoning: The LLM re-ranker was able to give reasoning for all recommendations, which explained how that tool was superior to other tools based on the query. Semantic search lacks this feature.
- Multi-aspect query processing: In cases where a query had various aspects, such as "a free image generator for social media posts that have a dark theme," the LLM was able to evaluate the query from different aspects to give a more relevant ranking than cosine similarity.
- JSON output format: The prompt engineering enabled the LLM to return JSON outputs with ai_id, Score, and Reasoning.
- Fallback robustness: Whenever Qwen 2.5 generated erroneous JSON output, the system effectively used the backup model Mistral 7B-Instruct and, if both models failed, a regular expression-based JSON extractor. This strategy avoided any failures during the experiment.
- Latency measurement: During the re-ranking step of the LLM, there was a notable delay of about 8-45 seconds. It seems that using it in the deep search scenario is better suited compared to recommendation generation tasks.

### *D. Hybrid Personalization Observations*

For assessing the performance of the hybrid recommendation engine, the authors simulated authentic users with various amounts of interaction history data. The researchers studied both new users with no past data and experienced users with plenty of previous feedback data. The experiments showed that combining four specific signals, such as content-based filtering, collaborative filtering, semantic processing, and engagement metrics, helped produce better results than applying only a single signal. Every signal was useful in its own way: content-based filtering took user preferences into account, collaborative filtering followed patterns observed among similar users, semantic analysis brought contextual awareness, and engagement metrics demonstrated actual user behavior.

Moreover, the model was able to tackle the cold start problem efficiently. If the amount of interaction history for a certain user was low or non-existent at all, the algorithm put emphasis on the content-based and semantic signals, generating recommendations for users accordingly. With more interactions performed by users, however, the weight of collaborative and engagement-based signals became higher.

TABLE IV. SYSTEM BEHAVIOR UNDER DIFFERENT USER FEEDBACK SCENARIOS

| **Scenario** | **Observed Behavior** |
|---|---|
| New user (0 feedback events) | The system relies entirely on semantic similarity since collaborative and engagement scores are zero. Content weight gets boosted (1.2×) according to the adaptive formula. |
| User with 3 likes and 1 dislike | Liked tools and similar-category tools are ranked higher. The disliked tool is significantly lowered (score suppressed to ≤ −0.95). |
| User with 10+ feedback events | Collaborative signals become meaningful. Tools liked by similar users appear higher. Adaptive weights shift to favor collaborative filtering (1.1× multiplier). |
| User with bookmarked tools | Bookmarked items receive a score bonus, reappearing in personalized recommendations across sessions. |

Table IV evaluates the behavior of the system based on various degrees of user interactions and feedback. In this respect, the table considers cases where the users have no prior history, where there is minimal feedback from the users, and cases where the users interact frequently with the system. The observations reveal the flexibility of the hybrid recommendation

system that assigns different weights for each approach.

*D. Discussion*

*a. System strengths:*

The three-stage architecture clearly separates retrieval, reasoning, and personalization. The fallback chain, which includes FAISS, raw embeddings, TF-IDF, and LLM-only, ensures the system never returns an empty response and responds smoothly across different failure modes. The locally hosted LLM design reduces cloud API costs and protects user privacy, which is essential for enterprise use.

*b. Observed limitations:*

The system finds it hard to cater to highly specialized or newly introduced types of tools, which are not represented in the data set, known as the cold item problem. The deep search process is hindered by large waiting times during the LLM re-ranking phase, which makes deep search unfeasible in applications where time constraints are a priority.

*c. Scalability considerations:*

The existing design utilizes FAISS with a flat index, where a full search is performed via comparison of the input vector with all the stored vectors. Such an approach ensures high precision, but its scalability is linear to the size of the data set, that is, the more vectors (in this case, tools) are present, the longer it takes to perform the search. In other words, scalability becomes a problem once the number of tools exceeds 10,000.

## V. CONCLUSION

A novel recommendation system was designed to suggest AI tools for people in need at the present time based on the following three mechanisms required for its efficient functioning. First, it analyzes the semantic meaning of the search queries, second, it takes into account the context of the search queries, and third, it introduces the mechanism for combining several signals to provide the most appropriate recommendations for AI tools that will be preferred by users. The creators of this system gathered information concerning more than 500 AI tools and assessed the effectiveness of their performance in each step of the procedure. These AI tools were tested to verify the consistency of accurate recommendations provided by the system.

*A. Achievements:*

The semantic retrieval module handles similar, reworded queries well and returns tool candidates across 15 different categories. The LLM reranking stage adds reasoning and explanations, providing justifications for each recommendation. The adaptive hybrid scoring engine uses four signals: content preference, collaborative filtering, similarity, and implicit engagement. This approach gives recommendations that improve as users interact more. The semantic retrieval module manages queries effectively and provides tool candidates, while the adaptive hybrid scoring engine delivers recommendations based on user interaction.

*B. Real-World Usefulness:*

The entire system is built using purely local resources without any dependence on any third-party APIs. It is both secure from privacy issues as well as cost-effective due to being completely free. The FastAPI backend enables instant feedback for simple semantic suggestions, while LLM-driven searches are enabled for more experimental scenarios. A powerful fail-over sequence of FAISS, embeddings, TF-IDF, and LLM guarantees consistent functionality in all scenarios.

*C. Limitations:*

The main limitations are: (1) cold-start issues for new users with little interaction history; (2) LLM deep search latency of 8 to 45 seconds, which limits real-time applications; (3) the dataset, though comprehensive, is not exhaustive and may miss new AI tools; (4) sentiment analysis is based on rule-based normalization rather than a learned model.

*D. Future Improvements:*

Future work will look into: (1) automated dataset expansion through web scraping and periodic re-indexing; (2) fine-tuning the sentence-transformer model on AI tool descriptions for specific embeddings; (3) replacing the rule-based sentiment approach with a learned BERT-based sentiment classifier; (4) quantized LLM inference to lower deep search latency to under 2 seconds; (5) a graph-based collaborative filtering extension to manage sparse interaction matrices more effectively.

REFERENCES

[1] N. Reimers and I. Gurevych, ―Sentence-BERT: Sentence embeddings using Siamese BERT networks, ‖ in Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP), Hong Kong, 2019, pp. 3982-3992.

[2] J. Johnson, M. Douze, and H. Jégou, ―Billion-scale similarity search with GPUs, ‖ IEEE Trans. Big Data, vol. 7, no. 3, pp. 535-547, 2021.

[3] Y. Hou, J. Zhang, Z. Lin, H. Lu, R. Xie, J. McAuley, and W. X. Zhao, ―Large language models are zero-shot rankers for recommender systems, ‖ in Proc. 46th European Conf. Information Retrieval (ECIR), 2024.

[4] Z. Wang, L. Lin, Z. An, and X. He, ―InstructRec: Instruction tuning for personalized recommendation, ‖ arXiv preprint arXiv:2312.16424, 2023.

[5] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, ―Neural collaborative filtering, ‖ in Proc. 26th Int. Conf. World Wide Web (WWW), 2017, pp. 173-182.]

[6] Y. Koren, R. Bell, and C. Volinsky, ―Matrix factorization techniques for recommender systems, ‖ IEEE Computer, vol. 42, no. 8, pp. 30-37, 2009.