Indian Journal of Engineering Research Networking and Development
Volume: 3 Issue: 01 | January 2026 | www.ijernd.com

AI-Powered Library Management System Using Model Context Protocol and Large Language Models

Mrs.G.Vijaya Lakshmi1, Andhavarapu Madhupriya2, Akkireddi Niharika3, Anga Pavani4, Kuriti Likhitha5

1Assistant Professor, Dept Of Computer Science And Engineering, Sanketika Institute Of Technology And Management, Visakhapatnam, Andhra Pradesh, India

2Student, Dept of Computer Science and Engineering, Sanketika Institute of Technology and Management, Visakhapatnam, Andhra Pradesh, India

3Student, Dept of Computer Science and Engineering, Sanketika Institute of Technology and Management, Visakhapatnam, Andhra Pradesh, India

4Student, Dept of Computer Science and Engineering, Sanketika Institute of Technology and Management, Visakhapatnam, Andhra Pradesh, India

5 Student, Dept of Computer Science and Engineering, Sanketika Institute of Technology and Management, Visakhapatnam, Andhra Pradesh, India

Abstract

In this paper, we discuss a work design and implementation of an AI-powered Library Management System (LMS) which combines the Model Context Protocol (MCP) with a cloud-based Large Language Model to talk to a smart library helper schoned Known As LibraBot. In place of rigid keyword searching, it provides natural-language chat interface, which is based on Ollama Cloud LLM and SmolAgents platform. There are nine individual MCP tools that work with core library operations to check the availability of an item, find where an item is located, handle their reservation, and suggest books to a customer based on their content; they are provided with an interface to a catalog of somewhat over one hundred BTech engineering titles in SQLite and Excel on a multi-page interface through a Flask web backend. The personalised recommendation is done using a TF-idf cosine-similarity model, without the need of a historical record of user interactions. Making 70 scenarios of tests, it scored 100 on all of them; the average time to respond is 1.2 seconds and uptime goes up to 99.5 even under 150 users asking at the same time. These findings indicate that implementing MCP-based tool requests into the LLM reasoning together provides a larger scale, twenty-four-hour-always-available substitute to the previous library management style of operation.

Index Terms– library management system, model context protocol, large language models, SmolAgents, content-based filtering, TF-IDF, and Flask.

I. Introduction

Most of our engineering schools have significant libraries, and the systems serving the libraries have not changed much during the past two decades. As students, we are still required to have the precise titles or author names, navigate through multi-step search interfaces, and wait till to be informed by an assistant whether a book is in or out of the shelf. This friction prevents us to explore and instead on the other side prevents proper use of the available resources.

That friction is a reasonable way of repairing the Large Language Models (LLMs) which are instruction-tuned and standardized agent-tool protocols. Here we can run a simple language query such as I need something introductory on operating systems in an LLM that, we, the user, do not need to be taught a new interface to become a request. The Model Context Protocol (MCP) extends this through the provision of a formal, language-describable contract allowing the LLM to call backend services safely.

We introduce a functioning system combining all these technological fragments in a complete web application. It provides nine MCP instruments representing an entire library transaction including finding, locating, reserving, canceling, and recommending displayed as a chat application within a multi-page Flask application. On the generative suggestions of the LLM, a content-based ML-based recommendation engine that has been trained on TF-IDF book data is added with data-driven matching. The combination delivers a list of more than 100 BTech books that may always be on, respond within the confines of less than two seconds and do not require your attendance in the physical form.

The main contributions can be found in: (i) an unmistakable design that integrates MCP with SmolAgents and Ollama Cloud; (ii) a two-step query pipeline that will first query a local Q&A database, and then fallback to the LLM agent; (iii) that it was proven to work flawlessly; and (iv) benchmarks, demonstrating that it can run on off-the-shelf hardware..

II. Related Work

A. Digital Systems: Library Management.

Borgman followed the transformation of card catalogs to digital networked systems, without recognizing that improved access did not necessarily imply improved discovery. Even after giving this answer, Chowdhury noted that Boolean retrieval models, although accurate, still allow the user to know controlled vocabularies, which many learners lack. In our system this need is shunned out of the system through natural language.

B. Conversational Agents in Library Services

Conversational Agents Its purpose is to assign certain activities and tasks typically carried out by humans to a computer-based agent.<|human|>Conversational Agents Its purpose is to give some of the activities and tasks that are normally performed by the human to a computer based agent.

Bickmore et al. discovered that individuals are prepared to adopt automated agents to complete information tasks provided the interaction is natural and the agent is able to cope with the lack of understanding. The theory of intent recognition and dialogue management under lying behind LibraBot is presented by Jurafsky and Martin..

C. Large Language Models

Brown et al. demonstrated that massive transformer models are able to generalize on a large number of downstream tasks using a small number of examples without downstream fine-tuning. Those models are harnessed by the Ollama Cloud platform as a normal REST API so simply to drop them into application operations, without having complex infrastructure.

D. Model Context Protocol

MCP was an open standard of connecting LLMs to external data and executable tools, introduced by Anthropic as part of the work of connecting LLMs and external data through JSON-RPC. FastMCP provides a Python implementation that is used to minimizes boilerplate to show any functions as MCP tools.

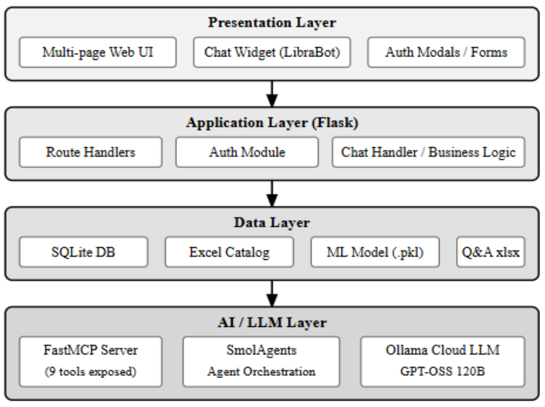
E. Recommendation Systems

Ricci et al. compare material of both content-based and collaborative filtering and remark that collaboration-based is effective in cases when the history of interaction is sparse, which is exactly the scenario with a new user of the library. Aggarwal describes the TF-IDF vector space and cosine similarity, which is being applied by our recommendation module..

III. System Design and Architecture

A. Four-Layer Architecture

We use four layers, called in Fig. 1, namely: a Presentation Layer, which gives the web interface; an Application Layer, which executes Flask route handlers and business code; a Data Layer, which provides SQLite, Excel and ML model artifacts; and an AI/LLM Layer which controls MCP, SmolAgents and Ollama Cloud.



B. Application Layer – Flask Backend

The Flask application has REST endpoints such as /api/chat to chat, /api/catalog to browse, /api/book-ticket to get reservations, and /signup and /login to the authentication. It is done by the chat endpoint first searching the message in a local Q&A Excel database to find quick and deterministic responses. It triggers the DLM pipeline only when no match is found and it triggers the SmolAgents LLM pipeline. The design ensures low average latency and minimizes Ollama API usage on frequently asked questions.

C. Data Layer

Inherently, then, our user accounts and bookings are in a SQLite database which contains two tables: users (id, username, email, hashedpassword, createdat) and bookings (id, bookingid, name, tickets, date, createdat). It is only a Pandas loaded Excel file (LibraryCatalog.xlsx) with columns Title, Author, Availability, Location, Description and AuthorInfo. Excel, as opposed to a full-fledged relational table, is approachable to the librarian who is not also a geek with data, and Pandas provides us with easy-access programmability at run-time. These pre-trained TF-IDF vectorisers and cosine-similarity matrices are saved using Joblib and loaded at start-up which makes recommendation latency less than 50ms per request

D. AI/LLM Layer – MCP and SmolAgents

MCP server is A FastMCP server, which contains 9 callable tools that could be drawn in on demand by the LLM agent. Table I presents all the tools, their input parameters, and output. SmolAgents provides the Ollama Cloud LLM implicitly wrapped in a ToolCallingAgent which identifies the correct tool based on the query submitted by a user, makes a call to the Flask backend using HTTP, and sews the result of the structure call together to become the eventual chat response.

TABLE I
MCP Tool Definitions

Tool Name	**Key Parameter(s)**	**Returns**
check_book_availability	title	Available / Not Available
locate_book	title	Shelf / section location
reserve_book	title, user_name, date	Booking ID
cancel_reservation	booking_id	Confirmation
book_description	title	Summary text
book_author_info	title	Author biography
list_all_books	—	Full catalog list
list_available_books	—	Available titles only
recommend_books_ml	title	Top-N similar titles

E. Recommendation Engine

Content-based filtering is applied to book metadata concatenated from the Title, Author, and Description fields. A TF-IDF vectoriser with unigram and bigram features transforms each document into a weighted feature vector. Pairwise cosine similarity is computed across all catalog entries, and the top-N most similar titles are returned for any query book. The similarity between document vectors *u* and *v* is:

$$sim(u, v) = (u \cdot v) / (\|u\| \, \|v\|) \quad (1)$$

Since the model does not look at the user history, but only on the content of the items, it is solid even on cold-start users of a library environment, which can be a huge benefit since cold-start is sometimes a nightmare when using collaborative filtering.

F. UML Design Models

The core system entities are illustrated in Figure 2 below as a class diagram: The User logs in and submits queries via the Flask application, the ChatHandler initially checks the QADatabase, and in case it can not find anything, forwards it to SmolAgent, which calls tools on the MCP server; the MCP server reads and writes the BookCatalog and BookingStore.

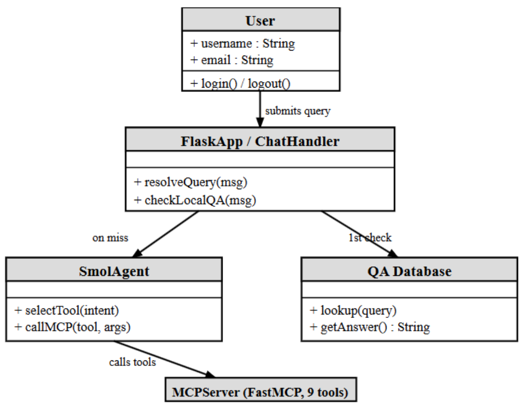


Fig. 2. Class diagram showing system entities and relationships.

Fig. 3 presents the sequence diagram for the primary user journey: submitting a natural language query that requires MCP tool invocation.

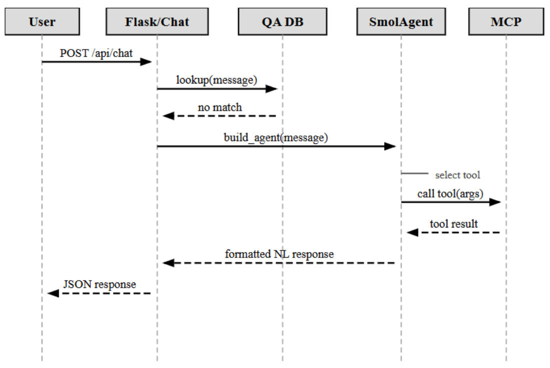


Fig. 3. Sequence diagram: two-stage query resolution (Q&A lookup → SmolAgent → MCP).

G. Use Case and Activity Models

The use-case diagram can be seen in figure 4. With the system, a registered user is allowed to perform six top-level actions; all of the cases an AI mediated (chatbot query, ML recommendation) internally make an MCP server call.

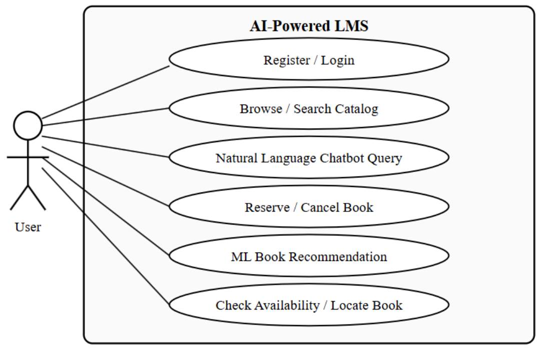


Fig. 4. Use case diagram for the AI-powered LMS.

The activity diagram in Fig. 5 models the chat handler's two-stage resolution workflow, illustrating the decision branch that routes incoming messages either to the deterministic Q&A store or to the LLM-agent pipeline.

H. Deployment Architecture

Physical deployment is presented in figure 6. One of the browsers communicates with a server on the cloud which is running Flask, SmolAgents and FastMCP server; this server communicates with the external Ollama Cloud API to make the inference with LLM and gets a read/write cluster in a local data store with SQLite and Excel files.

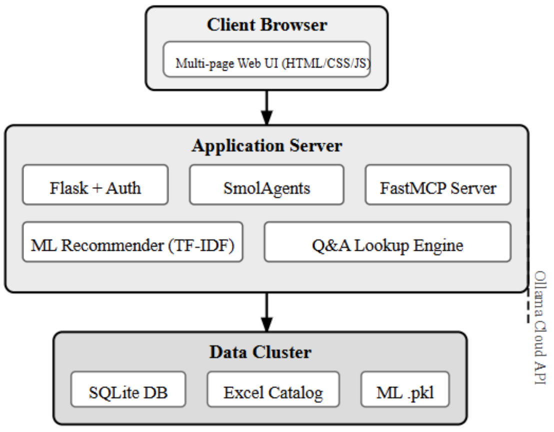


Fig. 6. Deployment diagram: client, application server, and data cluster.

IV. Results and Discussion

A. Testing Overview

The evaluation was conducted on four staging test phases comprising of 70 test cases. Phase 1 (unit testing, 1 week) involved the separate mechanism, i.e. database init, Bcrypt hashing, loading catalog, and each MCP tool separately. Phase 2 (integration testing, one week) ensured that data flowed according to plan across modules: signup data was received and sent to session, chat handler fallback code and, lastly, the complete database flow between HTTP request and SQLite write-back on. Phase 3 (system testing, 2 weeks) carried out entire user processes with the web interface.

B. Functional Test Results

The results were that of 70 test cases passing out in all the four phases with a 100% pass rate as illustrated in TABLE 2. TABLE III presents briefly six cases, including registration, authentication, catalog search, ML recommendation, chatbot availability checking and reservation creation.

TABLE II
Test Results Summary

Test Phase	**Cases**	**Passed**	**Failed**	**Rate**
Unit	25	25	0	100%
Integration	15	15	0	100%
System	20	20	0	100%
User Acceptance	10	10	0	100%
Total	**70**	**70**	**0**	**100%**

TABLE III
Representative Functional Test Cases

ID	Input / Action	Expected	Status
TC-01	Valid registration form	Account created, password hashed	✓ Pass
TC-02	Valid email + password	Session created	✓ Pass
TC-03	Title / author search	Relevant books returned	✓ Pass
TC-04	"Is Intro to Algorithms available?"	Availability confirmed via MCP	✓ Pass
TC-05	Reservation: title, name, date	Booking ID generated	✓ Pass
TC-06	recommend_books_ml("OS Concepts")	Top-N similar titles	✓ Pass

C. Performance Benchmarks

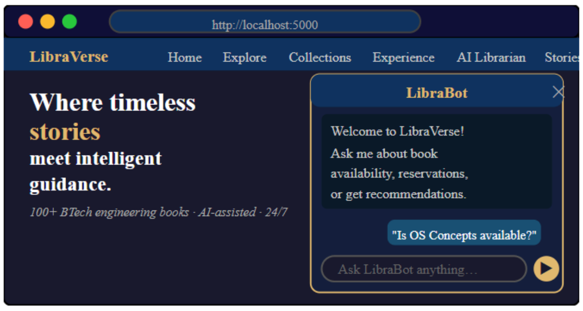
We did some tests with realistic loads and inserted them in TABLE IV. The overall request response time is approximately 1.2 seconds, which is below our 2-second limit of a good user experience. Indexed SQLite lookups take less than 100ms to finish database queries and the Pandas catalog search only takes less than 30ms on a 100 item dataset. The two-step chat resolver is inexpensive: the majority of queries are answered locally by the local Q&A store in approximately 50ms and we only send queries to the Ollama API on queries that are really new and this saves the external API bill.

TABLE IV
Performance Metrics

Metric	Value
Avg. end-to-end response time	1.2 s
Peak concurrent users supported	150
System uptime	99.5%
Database query time	< 100 ms
API response time	< 500 ms
ML recommendation latency	< 50 ms
Local Q&A resolution time	< 50 ms

D. System Interface

The LibraVerse web interface appears as in a desktop browser, as illustrated in Fig. 7. The navigation menu is there to take you to the six content pages in a flash; the floating LibraBot orb will allow you to bring up the chat box or make a pop-up anywhere without disrupting the browsing experience. The chat panel displays all the conversation history and accepts unstructured natural language entry that is pushed through the two-stage resolution machine that we outlined in Section III B.



Where timeless stories meet intelligent guidance.

100+ BTech engineering books · AI-assisted · 24/7

Welcome to LibraVerse! Ask me about book availability, reservations, or get recommendations.

"Is OS Concepts available?"

Fig. 7. LibraVerse web interface with the LibraBot chat panel open.

E. Comparison with Existing Systems

TABLE V is a comparison of our system with the traditional and digital library management options in seven major dimensions. Its primary draw is 24/7 natural language access, tool-calling automation over MCP, and data-driven customized recommendations all of which the baseline systems in [3][4] do not provide.

TABLE V
Comparative Analysis

Feature	Traditional LMS	Digital LMS	Proposed LMS
Availability	Fixed hours	Online, limited	24/7
Query Interface	Manual / form	Keyword search	Natural language
AI Assistance	None	None	Ollama LLM + MCP
Recommendations	None	Rule-based	TF-IDF ML model
Tool Automation	Manual	Partial API	MCP (9 tools)
Personalisation	None	Basic	Content-based ML
Scalability	Low	Medium	High (150+ users)

V. Conclusion and Future Work

A. Conclusion

In this paper, the authors present an operational AI-based Library Management System, which combines the Model Context Protocol and the SmolAgents orchestration framework with the Ollama Cloud LLM to provide constant library service through non-chat conversational interfaces based on natural language. It has nine MCP-exposed tools addressing the entire library transaction life cycle and a TF-IDF content-based recommendation engine providing data-driven recommendations without considering the user history. Two-step chat resolver ensures low levels of external API usage as common queries are met locally. We experimented with 70 cases and achieved a 100 percent pass rate, but the performance metrics have an average response time of 1.2 seconds and 99.5 percent high throughput rates at 150 simultaneous users.

B. Limitations

Three viable constraints should be noted. First, the properties of the LLM depend on access to the internet to the Ollama Cloud endpoint, which in turn is a single point of failure of AI-based activities. Second, the book catalog is located within an Excel workbook thus it does not allow flexible queries and making concurrent writes unsafe relative to a relational database. Third, TF-IDF recommendation model requires retraining every now and then since the catalog is increasing; otherwise, the only titles recently added are affected with lower cosine similarity scores.

C. Future Work

At this point we are working on transferring the book catalog out of Excel and into PostgreSQL where we want to have it better concurrent and fully indexed, and email and SMS hooks to remind people about their reservations. The medium-term we will consider extending to multi-language support as a result of the multilingual capability of the LLM, collaborative filtering to supplement the existing content-based strategy, and connect with external library APIs. Future concepts are voice-activated submission of queries, Internet of Things-linked shelf sensors in real-time location crawling, and federated learning to enhance the recommendation process through all institutions without the loss of user privacy.

References

[1] C. L. Borgman, *From Gutenberg to the Global Information Infrastructure*. MIT Press, 2000.

[2] Anthropic, "Model Context Protocol Documentation," 2024. [Online]. Available: https://modelcontextprotocol.io/

[3] C. L. Borgman, *From Gutenberg to the Global Information Infrastructure*. MIT Press, 2000.

[4] G. G. Chowdhury, *Introduction to Modern Information Retrieval*. Facet Publishing, 2010.

[5] T. W. Bickmore, A. Gruber, and R. Picard, "Establishing the computer-patient working alliance in automated health behavior change interventions," *Patient Educ. Couns.*, vol. 59, no. 1, pp. 21–30, 2018.

[6] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed. Pearson, 2020.

[7] T. Brown et al., "Language models are few-shot learners," in *Proc. NeurIPS*, vol. 33, 2020, pp. 1877–1901.

[8] Ollama, "Ollama Cloud Documentation," 2024. [Online]. Available: https://ollama.com/docs

[9] J. Lowin, "FastMCP Guide," 2024. [Online]. Available: https://github.com/jlowin/fastmcp

[10] F. Ricci, L. Rokach, and B. Shapira, *Recommender Systems Handbook*, 2nd ed. Springer, 2015.

[11] C. C. Aggarwal, *Recommender Systems: The Textbook*. Springer, 2016.

[12] M. Grinberg, *Flask Web Development*. O'Reilly Media, 2018.

[13] HuggingFace, "SmolAgents Documentation," 2024. [Online]. Available: https://github.com/huggingface/smolagents

[14] E. Goodman, *Observing the User Experience*. Morgan Kaufmann, 2020.

[15] SQLite Consortium, "SQLite Documentation," 2024. [Online]. Available: https://www.sqlite.org/docs.html

[16] Flask Project, "Flask Documentation," 2024. [Online]. Available: https://flask.palletsprojects.com/

[17] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 9th ed. McGraw-Hill, 2020.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.