Annals of Library and Information Studies
Vol. 73, September 2026, pp. 297-306
DOI: 10.56042/alis.v73i3.32444

Implementing Retrieval-Augmented Generation (RAG) Pipeline for a Virtual Library Assistant (ViLA): An In-house Approach

Kaustav Kumar Khanikar[a], Niraj Barua[b] and Gitanjali Gogoi[c]

[a]Research Scholar, Department of Library and Information Science, Gauhati University, Guwahati, 781014, India. E-mail: kaustavkhanikar99@gmail.com

[b]Assistant Professor, Department of Library and Information Science, Gauhati University, Guwahati, 781014, India. E-mail: nirajbarua@gauhati.ac.in

[c]Research Scholar, Department of Library and Information Science, Gauhati University, Guwahati, 781014, India. E-mail: gitanjaligogoi008@gmail.com

Received: 08 June 2026; Accepted: 06 August 2026

This paper demonstrates the application of technique in AI called Retrieval-Augmented Generation (RAG) for integration into a Virtual Library Assistant (ViLA). The RAG technique helps in the retrieval of relevant information from a database, which is then used by an LLM to generate accurate responses to user queries. The paper shows the entire process of setting up a RAG pipeline to LLM inferencing. The RAG pipeline employs an open-source embedding model alongside a vector database to function as the knowledge base. Various LLMs, typically ranging from approximately 150 million to 8 billion parameters in size were tested with the RAG output, using BERTScore, ROUGE-L and AI evaluation techniques, to identify the most suitable model for ViLA. The knowledge base consists of a collection of frequently asked questions (FAQs) from a library setting. The resulting system serves as a concrete example of AI integration into the daily workflows of information professionals and managers.

Keywords: Artificial Intelligence, AI Integration into Libraries, Retrieval Augmented Generation, Virtual Library Assistant, Library Chatbot, Ask a Librarian

1. INTRODUCTION

Libraries are service institutions, and they have long stood and will continue to stand by their promise of providing excellent service to users. The developments in the field of AI present significant opportunities for the information professionals to extend their service capabilities and accommodate a diverse array of users. By embracing these technological advancements and integrating them into the daily operations of Library and Information Centres (LICs), professionals can better utilize their time and become more effective in their work.

Library users are already familiar with chatbots from OpenAI, Google, Microsoft, and others. Even a standard Google search now returns answers generated by Gemini, often eliminating the need for users to click through search results to obtain information. This reality presents both a unique challenge and an opportunity for LICs: either to evolve with the changing times or risk becoming prestigious warehouses for physical assets, where the digital community no longer knows how to find their way.

2 LITERATURE REVIEW

In the 1950s, A.M. Turing proposed a new criterion for machine intelligence called the 'imitation game', also known as the Turing Test, to replace the ambiguous question of "Can machines think?".[1] ELIZA, developed during the 1960s, can be considered as an early model of AI that worked by analyzing the keywords appearing in the input text and generating responses by reassembly rules.[2] Following the success of ELIZA, various models, such as Parry and Alice, have been developed.[3,4] The main idea behind all these systems was more or less the same, i.e., identifying keywords in the input sequence and responding with pre-programmed matches. Also, these models were rarely advanced due to a lack of computing power, model architecture and data availability.[5]

Recent years have witnessed significant advancements in Large Language Models (LLMs), which are advanced models trained on massive text data to generate human-like responses.[6] A key development was the transformer architecture and its underlying attention mechanism, which uses self-

attention to identify relevant parts of the input regardless of word position, enabling the model to understand relationships within sentences.[7,8,9] Pre-trained transformer models, trained on large corpora with substantial computational resources include notable examples like BERT and GPT.[5] For instance, GPT-4 was trained on 300 billion words using a neural network with 175 billion parameters.[6] Research indicates that pre-training with sufficient data and computing allows models to acquire basic language comprehension, enabling them to perform well in few-shot or zero-shot scenarios.[10] The training process is recursive, with capabilities improving through exposure to new data; subsequently, LLMs undergo fine-tuning for specialized tasks.[11] Fine-tuning adapts pre-trained models to custom datasets for domain-specific applications.[12] ChatGPT, a revolutionary LLM-based chatbot released by OpenAI in November 2022, represents a fine-tuned GPT model optimized for interactive human engagement.[13]

While LLMs continue to grow in scale, smaller models with 2B to 7B parameters are gaining popularity in technical communities due to their efficiency and minimal resource consumption.[14] Compared to larger counterparts, fine-tuning and deploying small models is faster, less costly, and achievable on consumer-grade hardware, while delivering comparable or superior performance on domain-specific tasks.[15] Low-Rank Adaptation (LoRA) is an innovative fine-tuning technique that reduces trainable parameters by 10,000 times and GPU memory requirements by threefold.[16] However, fine-tuning on domain-specific instruction datasets often leads to data memorization, necessitating costly and complex continuous retraining to maintain current information.[17] Retrieval Augmented Generation (RAG) addresses this limitation by enabling frequent knowledge base updates without model retraining, offering a cost-effective solution that enhances overall system efficiency.[18]

Related works on libraries.

Since the launch of ChatGPT, it has received significant attention and appreciation, sparking a wave of research into how LLMs can be integrated across academic disciplines.[5] The library profession has already started exploring literature on the role of ChatGPT in redefining academic landscape.[19] Early predictions show that these AI tools could become cohesive assistants, which will help users to navigate collections and complete tasks without needing a librarian's direct intervention for every routine inquiry.[20] In fact, chatbots have already started handling basic reference questions which has enabled librarians to focus on more complex tasks demanding human knowledge.[21]

A comparison between ChatGPT and older, conventional library chatbots revealed that ChatGPT could actually suggest specific databases, while conventional chatbots struggled even to understand the question.[22] Its ability to sustain natural, flexible conversations represents a significant leap over the rigid, menu-based systems of the past.[23] Recognizing the limitations of commercial chatbot solutions, the University of Manitoba Libraries (UML) decided to design and develop a custom in-house chatbot.[24] Similarly, Zayed University Library in the United Arab Emirates developed a custom chatbot named 'Aisha'.[25] These implementations make use of proprietary APIs (Application Programming Interfaces) to deliver a more tailored and reliable user experience.

3. RESEARCH GAP

Existing literature identifies a gap in the transparency of the chatbot implementation process in libraries, as most of the heavy lifting is performed by developers while only limited information is conveyed to librarians. Furthermore, although RAG is increasingly popular, libraries often rely on proprietary LLM APIs for generating final responses to user queries. While using an API may not pose an immediate threat, the transmission of sensitive information and the risk of leaking user data remain common concerns when the RAG system passes context along with a prompt to a proprietary LLM. Additionally, reliance on these APIs introduces recurring costs for the library, which may ultimately conflict with the fundamental mission of libraries as service institutions.

4. OBJECTIVE OF THE CURRENT STUDY

The study was conducted to achieve the following objectives:

1. To showcase the development of an RAG pipeline utilizing an open-source embedding model and a vector database to serve as the core knowledge retrieval mechanism for a Virtual Library Assistant (ViLA).
2. To curate a structured knowledge base (KB) of FAQs for serving as the authoritative source of information for the library assistant.

3. To evaluate and identify the most effective LLM within the 150M to 8B parameter range to generate most useful responses based on the retrieved information.
4. To demonstrate a practical implementation scenario to integrate AI-driven tools into a library's workflow.

5 METHODOLOGY

This study is based on design and development methodology for building a RAG pipeline for a ViLA. The approach consists of curating a KB from library FAQs, which are then embedded using an open-source embedding model, and finally stored into a vector database for retrieving efficiently. Multiple small-scale LLMs consisting of 300 million to 8 billion parameters are systematically evaluated for identifying the most suitable model based on accuracy and relevance of responses. The entire system is implemented on local hardware without proprietary APIs for ensuring data privacy, minimizing costs, and maintaining transparency for information professionals.

5.1 Experimental Setup

The experimental setup was conducted on an Apple MacBook Air (M4) with 16GB unified memory and Visual Studio Code as the development environment. The RAG pipeline was implemented in Python, utilizing the all-MiniLM-L12-v2 open-source embedding model to convert the KB into vector representations, which were indexed and stored in a local Chroma DB vector database. LLM inferencing was performed using LMStudio, which served the various models via a localhost server.

5.2 Dataset Preparation

The dataset for this study was prepared by extracting frequently asked questions from the FAQ sections of several eminent libraries in India. An LLM was employed to refine and standardize these questions for clarity and consistency. The corresponding answers were then manually curated and populated using data from the author's own library, "KKHL, Gauhati University: ASSAM". This process resulted in a structured, self-curated knowledge base of Q&A pairs tailored to the Indian library context.

6. RETRIEVAL AUGMENTED GENERATION (RAG)

Retrieval-augmented generation, or RAG, is a technique that enhances the ability of an AI model to answer questions to which the model was not exposed before. RAG system assists the LLMs by passing related context along with the user prompt, which is then used by the LLM to provide a relevant response.[18] Imagine a library patron asking an LLM, "Is the book 'X' available in library 'Y'?" In this scenario, the RAG system first searches the knowledge base for any information related to library 'Y' and book 'X', retrieving relevant records such as catalogue entries, shelf availability, or circulation status. This retrieved information is then passed to the LLM, which is then used by the LLM to generate a precise response grounded in the actual data.[18] This ensures that the response is based on valid facts and not just some random generalization of the LLMs.

7. EXPERIMENTATION

The first step in building a RAG framework is the creation of the knowledge base, which contains all domain-specific or application-orientated information. This knowledge base is stored in a searchable database, enabling the retriever to fetch documents relevant to a user's query. The retrieved information is then passed along with the original prompt to the LLM, which formulates a response grounded in the provided context. The two main steps involved in building an RAG framework are - (I) Data Ingestion Pipeline & (II) Query Retrieval Pipeline.

The Python packages used for the experiment are mentioned below. While conducting the experiment, GitHub Copilot (Free Version) was very helpful, from debugging code to providing helpful suggestions throughout the project.

```
### Install using pip or uv…
pip install langchain
pip install langchain-core
pip install langchain-community
pip install chromadb
pip install sentence-transformers
```

7.1 Data Ingestion Pipeline

7.1.1 Data Ingestion

In the data ingestion step, the RAG system is fed with a domain-related dataset. This dataset for forming the knowledge base can be of various formats, such as txt, pdf, csv, html, etc. For reading these documents or files, Langchain's document loader integration provide a standard interface, ensuring that data can be handled consistently regardless of source.

Since the experimental dataset is a CSV file containing the FAQs for a library, it is loaded as shown:

```
### Loading documents…
from langchain_community.document_loaders import CSVLoader
loader = CSVLoader("./LibraryFAQ.csv")
data = loader.load()
```

Like the CSVLoader class, a host of other classes are available for other document types, such as PyPDFLoader, WebBaseLoader, TextLoader, etc. In addition to these, entire directory with files can be loaded using the DirectoryLoader class.

Preprocessing the data to combine the question-and-answer pairs as one line of text.

```
for doc in data:
 lines=doc.page_content.split('\n')
 q=lines[0].replace('question: ', '')
 a =lines[1].replace('answer: ', '')
 doc.page_content = f"Q: {q},A: {a}"
```

7.1.2 Chunking

Chunking refers to the process of dividing large documents into smaller, manageable segments for efficient storage and retrieval. This technique is essential because LLMs operate within a limited context window**, allowing only a finite amount of information to be processed at once. Additionally, token* consumption directly influences computational costs. Supplying entire documents to the model may lead to context limit errors or incur unnecessary expenses.

One important component of chunking is the chunk size, and determination of chunk size will determine how efficient the retrieval results will be. If chunk sizes are too small, it will fail to capture the essence of information passed to the LLM; and if chunk size is too large, it will carry irrelevant information in every context, leading to a decrease in the efficiency of pay-per-token use cases.

When talking about chunk size, another important term is 'chunk overlap'. Consider the example shown below; if the chunk size is 50 characters, the sentences get stripped after 50 characters. In this scenario, if a faculty member enquire about "Do teachers have to pay late fees?": the retriever may find only the first chunk and ignore the second chunk. To overcome this scenario, chunk overlap is used, wherein the chunks take in additional information from the sentence after it and the one preceding it. A chunk overlap of 40 will incorporate 40 more characters from the cutoff point from both ends.

Students must pay a fine of 5 rupees per day. This rule does not apply to faculty members. However, for non-registered members of the institution issuing of books is not allowed in any case.

**token: Tokens are not exactly words, they are parts of words, punctuation marks, or even spaces represented in numbers for the LLMs to process upon.*

Below is an example of a sentence converted to tokens in accordance with gpt-4 model [Generated by Tiktokenizer]:

What is a green library?

3923, 374, 264, 6307, 6875, 30

***context window: The context window is the maximum no of tokens the LLM can fit for processing a prompt and generating the response.*

Chunking is achieved using the 'text_splitter' integration from the Langchain framework.

```
### Splitting documents into chunks…
From langchain.text_splitter import RecursiveCharacterTextSplitter
text_splitter =
RecursiveCharacterTextSplitter(
chunk_size=50, #Experiment uses 300
chunk_overlap=40 #Experiment uses 50
 )
chunks=text_splitter.split_documents(
 data)
```

7.1.3 Embedding

The embedding process converts the chunks into a list of numeric values called vectors. Converting the chunks to vectors helps them to be represented in a high-dimensional space, where chunks with semantically similar meanings are placed close to each other (Fig. 1). Storing these semantically similar chunks together helps the retriever to retrieve information efficiently.

The experiment applies an open-source embedding model called 'all-MiniLM-L12-v2' to convert the chunks to vectors.

```
### Generating embeddings…
import numpy as np
from sentence_transformers import SentenceTransformer
class Embedding:
def __init__(self, model_name: str = "all-MiniLM-L12-v2"):
self.model_name = model_name
self.model = None
self._load_model()
def _load_model(self):
self.model = SentenceTransformer(self.model_name, token=False)
def generate_embeddings(self, texts: List[str]) -> np.ndarray:
embeddings = self.model.encode(texts)
```

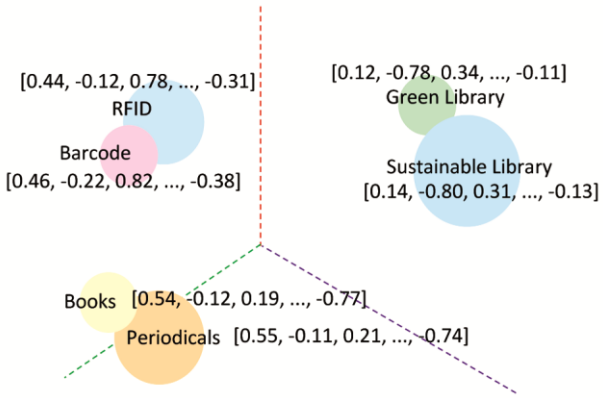

Fig. 1 — Clustering of similar embeddings

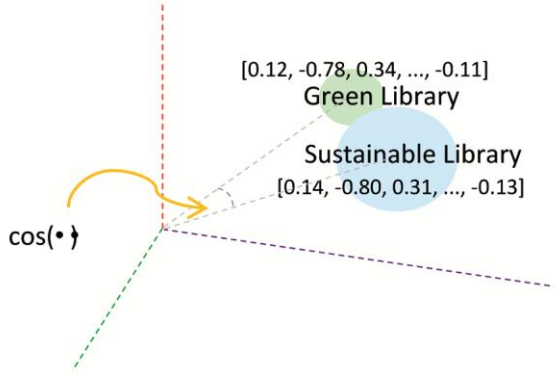

Fig. 2 — Similarity search in Vector DB

```
    return embeddings
    embedding = Embedding()
    texts = [doc.page_content for doc in chunks]
    embeddings = embedding.generate_embeddings(texts)
```

7.1.4 Storing in a Vector Database

After the chunks are converted to vector form, they are then stored in a database, but not in a traditional database; it is called a vector database. In traditional databases, like SQL, if someone searches for a book named 'Hamlet', the search query will try to find the exact match for the word 'Hamlet', and if this match fails, it will return none. In a vector database, the search is not performed using exact matches but rather with a method called cosine similarity search, where the resultant term is determined by the closeness of the angle between the searched vector and the resultant vector. For instance, if someone is searching for a 'green library', he will get access to results like a 'sustainable library' (refer to Fig. 2).

The experiment utilizes a popular open-source vector database called Chroma for storing embeddings.

```
    ### Storing in vector database…
    import chromadb, uuid, os
    from typing import List, Dict, Any
    import numpy as np
    class VectorStore:
     def __init__(self, collection_name="my_collection",
persist_dir="./chroma/vector_store"):
    os.makedirs(persist_dir, exist_ok=True)
     self.client =
chromadb.PersistentClient(path=persist_dir)
     self.collection = self.client.get_or_create_collection(
     name=collection_name,
     metadata={"description": "LibraryFAQ embeddings for
RAG"}
     )
     def add_documents(self, documents: List, embeddings:
np.ndarray):
    ids, metadatas, texts, embeds = [], [], [], []
     for i, (doc, emb) in enumerate(zip(documents,
embeddings)):
     ids.append(f"doc_{uuid.uuid4().hex[:8]}_{i}")
     meta = dict(doc.metadata)
     meta.update(doc_index=i,
content_length=len(doc.page_content))
     metadatas.append(meta)
     texts.append(doc.page_content)
     embeds.append(emb.tolist())
     try:
     self.collection.add(ids=ids, metadatas=metadatas,
embeddings=embeds, documents=texts)
     print(f"Successfully added {len(documents)}
documents")
     except Exception as e:
     print(f"Error adding documents: {e}")
     raise
    vector_store = VectorStore()
    vector_store.add_documents(chunks, embeddings)
```

7.2 Query Retrieval Pipeline

As the knowledge base has been generated with the data ingestion pipeline, the query retrieval pipeline can now perform its task. The query retrieval pipeline is responsible for retrieving chunks from the vector store that is similar to the user's query and passing it along with the prompt as context to the LLM for generating a response (Fig. 3).

7.2.1 Retrieval

The retrieval step is concerned with drawing out embeddings/chunks from the database (Fig. 3), which have semantic similarity with the user's query. Before passing the query to the knowledge base, the query needs to get converted to vectors, so that the similarity check can be performed in the vector database. An important parameter while retrieving

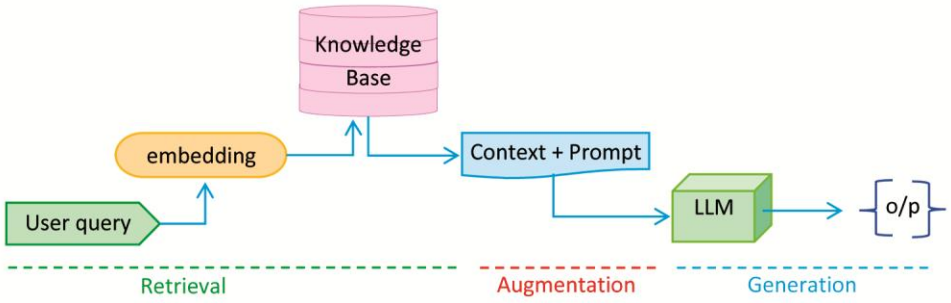

Fig. 3 — Query Retrieval Pipeline

similar embeddings from the vector database is 'top_k', which tells the retriever to retrieve only that many embedding[s] identified in the 'top_k'.

```
### Retrieving from the vector database…
class Retriever:
def __init__(self, vector_store: VectorStore, embedding:
Embedding):
 self.vector_store = vector_store
 self.embedding = embedding
def retrieve(self, query: str,
top_k: int=2,
score_threshold: float=0.0) -> List[Dict[str, Any]]:
 query_embed =
self.embedding.generate_embeddings([query])[0]
 try:
 res = self.vector_store.collection.query(
 query_embedding=[query_embed.tolist()],
 n_results=top_k
 )
 retrieved_docs = []
 if res['documents'] and res['documents'][0]:
 for i, (doc_id, document, metadata, distance) in
enumerate(zip(res['ids'][0], res['documents'][0],
res['metadatas'][0], res['distances'][0])):
 similarity_score = 1 - distance
 if similarity_score > score_threshold:
 retrieved_docs.append({'id': doc_id,
 'content': document,'metadata': metadata,
 'similarity_score': similarity_score,
 'distance': distance,'rank': i + 1
 })
 else:
 print("No documents retrieved from vector store.")
 return retrieved_docs
 except Exception as e:
 print(f"Error during retrieval: {e}")
 return []
retriever = Retriever(vector_store, embedding)
```

7.2.2 Augmentation

After the relevant chunk[s] are retrieved by the retriever, they then form the context to the LLM, passed along with the prompt (see Fig. 3). The prompt is an instruction to the LLM, instructing it to respond in a certain manner using the context. For example, the prompt used in the experiment was something like: *"Use the following pieces of context to answer the user's question. If you don't know the answer, just say that you don't know, don't try to make up an answer."*

```
### Generating the user prompt…
def augmentation(query, k=5):
retrieved_results = retriever.retrieve(query)
context = "\n\n".join([doc['content'] for doc in
retrieved_results])
prompt = f""" Use the following pieces of context to
answer the
 user's question.
If you don't know the answer, just say that you don't
know, don't try to make up an answer.
 \n Context:
\n {context}
 \n User Question: {query}
 \n Answer:"""
return prompt
```

7.2.3 Generation

The generation step is concerned with getting a response from an LLM using the context along with the prompt (Fig. 3). The experiment uses Ollama and LM Studio (both are free to use tools) as the backend server for interacting with different local LLMs. The code below shows the generation of responses using the LMStudio server. LM Studio makes it very easy to download and host LLMs locally with its easy-to-use user interface.

```
### Generating the response…
def llm_generation(query, temperature=0.7,
max_tokens=512):
prompt = augmentation(query)
payload = {
"messages": [{"role": "user", "content": prompt}],
"temperature": temperature,
"max_tokens": max_tokens,
"stream": False
}
response =
```

```
requests.post("http://localhost:1234/v1/chat/completions",
    json=payload)
    return
response.json()["choices"][0]["message"]["content"]
    print(llm_generation("What is a Green Library?"))
```

8. RESULTS

The RAG framework was tested with seven different LLMs, ranging from 135M to 8B parameters, after which their responses were compared using BERTScore metrics, ROGUE-L Score and a AI model evaluation across five criteria.

BERTScore:

The BERTScore metric compares the tokens of a candidate string *y* against a provided reference string *x*, by representing the tokens in the form of embeddings.[26] As such, the *m* reference text tokens are converted to a sequence of *m* vector embeddings $x=\{x_1, x_2, \ldots x_m\}$ and the *n* candidate text tokens to a sequence of *n* vector embeddings $y=\{y_1, y_2, \ldots y_n\}$;and the pairwise similarity scores between all *mn* possible (reference token, candidate token) pairs are then calculated.[8] The recall is computed by matching each token in *x* to a token in *y*, and precision is computed by matching each token in *y* to a token in *x*.[27] A technique called greedy matching is used where is token is matched to the most similar token in the other sentence.[28,29] The F1 measure is the combination of precision and recall; for a reference x and candidate y, the recall, precision and F1 scores are[27]:

$$R_{BERT} = \frac{1}{|x|}\sum_{x_i \in x} \max_{y_j \in y}(x_i^T y_j)$$

$$P_{BERT} = \frac{1}{y}\sum_{y_j \in y} \max_{x_i \in x}(x_i^T y_j)$$

$$F_{BERT} = \frac{2 \times P_{BERT} \times R_{BERT}}{P_{BERT} + R_{BERT}}$$

ROUGE-L: Longest Common Subsequence:

To apply Longest Common Sequence (LCS) in sentence evaluation, a sentence is viewed as a sequence of words. Longer the LCS of two sentences, more similar to their summaries are. If X denotes the reference sentence of length m and Y is the candidate summary of length n, then LCS-based F measure or ROUGE-L is given by[30]:

$$R_{lcs} = \frac{LCS(X,Y)}{m}$$

$$P_{lcs} = \frac{LCS(X,Y)}{n}$$

$$F_{lcs} = \frac{(1+\beta^2) \times R_{lcs} \times P_{lcs}}{\beta^2 \times P_{lcs} + R_{lcs}}$$

Where LCS (X, Y) is the length of a longest common subsequence of X & Y, and $\beta = P_{lcs}/R_{lcs}$.[30]

AI evaluation:

Even though the above metrics provide valuable insights, they do not necessarily take into account the nuances like tone, clarity and whether the answer truly addresses a library's patron's need. Hence, evaluating them on categories such as correctness, completeness, relevance, clarity and tone by DeepSeek as an AI judge on a scale of 1 to 5 for each query gives new insights into the model's capabilities.

Quantitative Evaluation:

Table 1 presents the BERTScore (Precision, Recall, & F1) and ROUGE-L scores for each model, along with their parameter size, memory requirements, and quantization formats.

The BERTScore F1 values show an upward trend with increasing model size, with the Granite-3.1 (8B) model achieving the highest score of 0.9675, followed by Gemma3 (1B) model at 0.9493 and Qwen3 (0.6B) at 0.9105. The smallest models (SmolLM2-135M and LFM2-350M) scored significantly lower at 0.8365 and 0.8402 respectively. A similar pattern is observed in ROUGE-L scores, where Granite-3.1 outperformed all other models with a score of 0.8996, followed by Gemma3 and Qwen3 at 0.7560 and 0.6216 respectively. Even though SmolLM-2 and Phi-4-mini were larger in size at 1.7B and 3.8B respectively, they were outperformed by Gemma3 and Qwen3 on all metrics.

Table 1 — BERTScore and ROUGE-L Evaluation Results

Model Name	SmolLM2-Instruct	LFM2	Qwen3***	Gemma3	SmolLM2	Phi-4-mini***	Granite- 3.1
Parameters	135M	350M	0.6B	1B	1.7B	3.8B	8B
Size	148 MB	362 MB	639 MB	772 MB	1056 MB	2.18 GB	4.94 GB
Quantization	8bit	Q8_0	Q8_0	4bit	Q4_K_M	4bit	Q4_K_M
ROUGE-L	0.379011	0.332937	0.621587	0.755955	0.563158	0.524829	0.899601
Precision	0.8366	0.8278	0.9311	0.9553	0.8830	0.8844	0.9643
Recall	0.8378	0.8551	0.8917	0.9437	0.9048	0.8796	0.9712
F1	0.8365	0.8402	0.9105	0.9493	0.8931	0.8813	0.9675

****Reasoning models*

Table 2 — AI Evaluation Scores

Evaluation Criterion	Model Name						
	SmolLM2-Instruct	LFM2	Qwen3***	Gemma3	SmolLM2	Phi-4mini ***	Granite3.1
	135M	350M	0.6B	1B	1.7B	3.8B	8B
Correctness	2.87	3.07	4.60	4.67	4.33	4.40	5.00
Completeness	2.87	3.13	4.60	4.67	4.33	4.47	5.00
Relevance	3.60	3.60	5.00	4.87	4.67	4.80	5.00
Clarity	3.80	3.60	4.87	4.87	4.73	4.87	5.00
Tone	3.60	3.60	3.93	4.00	4.07	4.07	4.27
Total	16.74	17	23	23.08	22.13	22.61	24.27

****Reasoning models*

Qualitative AI Evaluation:

Table 2 presents the average AI evaluation scores in a scale of 1-5 across five criteria for all the responses from all the models. Analysis of the individual responses revealed that larger models (≥0.6B) provided more accurate results across all question categories. Smaller models (≤350M) produced incomplete answers or failed to capture the nuanced information. Granite-3.1-8B model achieved perfect scores in correctness, completeness, relevance, and clarity, with the highest tone score of 4.27, which resulted in a total score of 24.27, the highest among all the models tested. Gemma3-1B and Qwen3-0.6B followed with total scores of 23.08 and 23.00 respectively. The smallest models (SmolLM2-135M and LFM2-350M) scored the lowest. Both SmolLM-1.7B and Phi-4-mini came after Gemma3 & Qwen3, despite having higher parameter counts. The same pattern was observed in both quantitative and qualitative measurements, where top performing models showed similar results followed by under performing models.

9. DISCUSSION

The results demonstrate a positive correlation between model size and quality of response. Granite-3.1 (8B) outperformed all other models across all the qualitative and quantitative performance metrics. This was expected behaviour, as larger models tend to perform better. However, the exceptional performance of the Gemma3 (1B) model, achieving scores comparable to those of larger models (in this case Granite-3.1-8B) while being eight times smaller in size, demands special attention. The Gemma3-1B model with a BERTScore F1 of 0.9493 and an AI evaluation total of 23.08 (out of 25), shows that small models can deliver performance near to state-of-the-art models for domain-specific applications.

A critical trade-off lies between model performance and resource consumption when considered for implementation into a library scenario. The Granite 3.1-8B offers superior performance, however, its 4.94 GB size requires substantial storage and memory. In comparison, Gemma3-1B with a space occupancy of 772 MB (4-bit quantization) achieves 98.1% of Granite's BERTScore F1 (0.9493 vs. 0.9675) while using only 15.6% of the memory. Similarly, Qwen3-0.6B at 639 MB delivers 94.1% of Granite's F1 score. Thus, for libraries with limited computational resources or with a wide user base, the efficiency of these smaller models proves to be compelling alternatives. Despite being heavily quantized, both Gemma3-1B (4-bit) and Qwen3-0.6B (Q8_0) managed to achieve very good scores, which suggests that modern quantization techniques boosts efficiency while reducing model footprints at the same time. This finding proves to be of great relevance for those libraries that are looking to deploy AI assistants on local hardware.

The locally run models, especially Granite-3.1(8B) and Gemma3(1B), achieved response qualities that are comparable to the proprietary API-based solutions mentioned in the prior studies. The important fact is, it was achieved without using external servers to process patron queries, which addresses the privacy concern identified in the research gap. This implementation also eliminates the recurring costs associated with proprietary API.

The qualitative analysis by an AI model revealed differences that automated metrics alone could not capture. While Granite-3.1 was able to achieve perfect scores in four categories, its tone score of 4.27 suggests room for improvement. Conversely, smaller models exhibited appropriate tone (3.60-4.07), but they lagged behind in factual accuracy and completeness in responses. This suggests room for exploration of fine-tuning a model

on library-specific data to combine factual accuracy with an appropriate professional tone.

This study successfully attends to the research gap by showcasing a reproducible methodology for deploying a library chatbot or ViLA, which operates without the use of proprietary APIs. The documented experimentation in this paper enables information professionals to develop similar systems on their own, without relying on external vendors.

10. LIMITATION

Although the study yielded promising results, the study is not without limitations. The dataset used in this study consisted of a limited number of samples. For a more robust evaluation, a larger and more diverse KB will provide better insights. The different models were evaluated based on 15 queries, which may be sufficient for comparative analysis, but a larger test set would reveal potential performance variation across different categories.

The experiments were conducted and tested through a single machine to demonstrate the feasibility on consumer hardware; however, it was not evaluated for production environments. The experimentation did not include actual interaction with the ViLA system by patrons. Real-world usage patterns and the system's ability to handle ambiguous queries remain unexplored. A pilot study with library patrons would provide valuable insights. The study did not systematically evaluate the models for potential bias or harmful content in the output. Future work must include comprehensive safety testing and bias mitigation tactics.

11. CONCLUSION

This study successfully demonstrates the implementation of a cost-effective and privacy-conscious ViLA using an in-house RAG pipeline with locally deployed small language models. The comparative analysis reveals that while large models like Granite-3.1 (8B) deliver superior performance, smaller models like Gemma3 (1B) achieve 98% of the quality with a fraction of the resource footprint, which offers libraries the opportunity to deploy them in cost effective hardware. By eliminating the need to rely on proprietary APIs to pass the retrieved information from the RAG framework as context, the proposed ViLA takes into account the concerns regarding data privacy and recurring costs. The transparent and reproducible methodology discussed through this paper offers libraries the ability to independently implement and customize AI assistants, aligning technological advancement with the core service mission of libraries. Future work may expand the knowledge base, include support for multilingual inputs and outputs, and perform real-world testing with patrons to further refine the system.

REFERENCES

1. Turing A M, I.—Computing machinery and intelligence, *Mind*, 59 (236) (1950) 433–60.
2. Weizenbaum J, ELIZA—a computer program for the study of natural language communication between man and machine, *Communications of the ACM*, 9 (1) (1966) 36–45.
3. Colby K M, Hilf F D, Weber S and Kraemer H C, Turing-like indistinguishability tests for the validation of a computer simulation of paranoid processes, *Artificial Intelligence*, 3 (1972) 199–221.
4. Wallace R S, The anatomy of A.L.I.C.E, In *Proceedings of the paper presented at the Conference on Parsing the Turing Test*, Dordrecht, 2009, p. 181–210. Available at http://link.springer.com/10.1007/978-1-4020-6710-5_13 (Accessed on 21 Feb 2026).
5. Zhou X, Jeong B and Chapple K, A large language model-based chatbot system framework for urban planners, *Expert Systems with Applications*, 300 (2026) 130307.
6. Kasneci E, Sessler K, Küchemann S, Bannert M, Dementieva D and Fischer F, et al., ChatGPT for good? On opportunities and challenges of large language models for education, *Learning and Individual Differences*, 103 (2023) 102274.
7. Devlin J, Chang M W, Lee K and Toutanova K, BERT: pre-training of deep bidirectional transformers for language understanding, In *Proceedings of the paper presented at the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Volume 1, Minneapolis, Minnesota, 2019, p. 4171–86. Available at https://aclanthology.org/N19-1423/ (Accessed on 28 Dec 2025).
8. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L and Gomez AN, et al., Attention is all you need. Available at http://arxiv.org/abs/1706.03762 (Accessed on 28 Dec 2025).
9. Acheampong F A, Nunoo-Mensah H and Chen W, Transformer models for text-based emotion detection: a review of BERT-based approaches, *Artificial Intelligence Rev.*, 54 (8) (2021) 5789–829.
10. Brown T, Mann B, Ryder N, Subbiah M, Kaplan J D and Dhariwal P, et al., Language models are few-shot learners, In *Proceedings of the paper presented at Advances in Neural Information Processing Systems*, 2020, p. 1877–901. Available at https://proceedings.neurips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html (Accessed on 21 Feb 2026).
11. Kalla D and Smith N, Study and analysis of Chat GPT and its impact on different fields of study, *International Journal of Innovative Science and Research Technology*, 8 (3) (2023).
12. Anisuzzaman D M, Malins J G, Friedman P A and Attia Z I, Fine-tuning large language models for specialized use cases, *Mayo Clinic Proceedings: Digital Health*, 3 (1) (2025) 100184.

13. Baker P, Introducing ChatGPT, In *ChatGPT For Dummies*, 1st edn (Wiley), 2023, p. 5–25. Available at https://ieeexplore.ieee.org/abstract/document/10950516 (Accessed on 21 Feb 2026).
14. RedHat, Llms vs slms. Available at https://www.redhat.com/en/topics/ai/llm-vs-slm (Accessed on 2024).
15. Pareja A, Nayak N S, Wang H, Killamsetty K, Sudalairaj S and Zhao W, et al., Unveiling the secret recipe: a guide for supervised fine-tuning Small LLMs. Available at http://arxiv.org/abs/2412.13337 (Accessed on 1 Jan 2026).
16. Hu E J, Shen Y, Wallis P, Allen-Zhu Z, Li Y and Wang S, et al., LoRA: Low-rank adaptation of large language models. Available at http://arxiv.org/abs/2106.09685 (Accessed on 28 Dec 2025).
17. Eliseev V, Maksimova A and Bondarenko V, Method of building rag-powered instruction dataset from raw corporate text data for llm fine-tuning, *In Proceedings of the paper presented at National Conference on Math AI*, Sirius, Russia, 9 March 2025. Available at https://openreview.net/pdf?id=rOvRWhZruk (Accessed on 21 Feb 2026).
18. Lewis P, Perez E, Piktus A, Petroni F, Karpukhin V and Goyal N, et al., Retrieval-augmented generation for knowledge-intensive NLP tasks. Available at http://arxiv.org/abs/2005.11401 (Accessed on 21 Feb 2026).
19. Lund B D and Wang T, Chatting about ChatGPT: how may AI and GPT impact academia and libraries?, *Library Hi Tech News*, 40 (3) (2023) 26–29.
20. Adetayo A J, Artificial intelligence chatbots in academic libraries: The rise of ChatGPT, *Library Hi Tech News*, 40 (3) (2023) 18–21.
21. Cox C and Tzoc E, ChatGPT: implications for academic libraries, *College & Research Libraries News*, 84 (3) (2023) 99–102. Available at https://crln.acrl.org/index.php/crlnews/article/view/25821 (Accessed on 20 Feb 2026).
22. Chen X, ChatGPT and its possible impact on library reference services, *Internet Reference Services Quarterly*, 27 (2) (2023) 121–129. Available at https://doi.org/10.1080/10875301.2023.2181262.
23. Panda S and Kaur N, Exploring the viability of ChatGPT as an alternative to traditional chatbot systems in library and information centers, *Library Hi Tech News*, 40 (3) (2023) 22–25.
24. Xuan W, An In-House Approach to AI: developing a custom chatbot for library services, *Internet Reference Services Quarterly*, 29 (3) (2025) 333–345. Available at https://www.tandfonline.com/doi/epdf/10.1080/10875301.2025.2495917?needAccess=true (Accessed on 16 Feb 2026).
25. Lappalainen Y and Narayanan N, Aisha: a custom AI library chatbot using the ChatGPT API, *Journal of Web Librarianship*, 17 (3) (2023) 37–58.
26. Zhang T, Kishore V, Wu F, Weinberger KQ and Artzi Y, BERTScore: evaluating text generation with BERT. Available at http://arxiv.org/abs/1904.09675 (Accessed on 3 Mar 2026).
27. Jaskowski S, Chava S and Shah A, BERTScoreVisualizer: a web tool for understanding simplified text evaluation with BERTScore. Available at http://arxiv.org/abs/2409.17160 (Accessed on 3 Mar 2026).
28. Papineni K, Roukos S, Ward T and Zhu W J, Bleu: a method for automatic evaluation of machine translation, In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, Pennsylvania, USA, 2002, p. 311–8. Available at https://aclanthology.org/P02-1040/ (Accessed on 3 Mar 2026).
29. Banerjee S and Lavie A, METEOR: an automatic metric for MT evaluation with improved correlation with human judgments, In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, Ann Arbor, Michigan, 2005, p. 65–72. Available at https://aclanthology.org/W05-0909/ (Accessed on 3 Mar 2026).
30. Lin C Y, ROUGE: a package for automatic evaluation of summaries, In *Proceedings of the Conference on Text Summarization Branches Out*, Barcelona, Spain, 2004, p. 74–81. Available at https://aclanthology.org/W04-1013/ (Accessed on 3 Mar 2026).