Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

Design and Development of an Offline AI Study Assistant using RAG and Local Language Models

Shreyas Inamdar
Department of Computer Science and Engineering
MIT Art, Design and Technology University
Pune, India

Shreyan Patil
Department of Computer Science and Engineering
MIT Art, Design and Technology University
Pune, India

Siddesh Shinde
Department of Computer Science and Engineering
MIT Art, Design and Technology University
Pune, India

Sartahk Jadhav
Department of Computer Science and Engineering
MIT Art, Design and Technology University
Pune, India

Namrata Naikwad
Department of Computer Science and Engineering
MIT Art, Design and Technology University
Pune, India

Artificial Intelligence is shaking up how we learn. It's making education feel more personal and much easier to reach. In this paper, I'm introducing an offline study assistant powered by Retrieval-Augmented Generation (RAG) and local language models. Here's the cool part: this assistant actually learns from your real university notes, textbooks, and whatever study materials you feed it. When students have questions, they get solid, relevant answers—no internet required.

Privacy is a big deal with this project. Since everything runs offline, students' data stays safe and private. The assistant is simple to use—a chatbot where you can ask anything, or even upload your own notes and files. FAISS works behind the scenes, storing and sorting all that information for quick searches. This setup lets the assistant pull up what you need fast, then generate clear, helpful answers right there on your device.

In the end, this project shows local AI tools can really help students learn on their own and make the whole studying experience smoother.

I. INTRODUCTION

AI and NLP have turned studying upside down for students. Let's be real—everyone's drowning in scattered notes, untouched textbooks, research papers gathering dust somewhere in your downloads, and digital handouts hiding in random folders. It piles up fast. Try finding that one page you need, or making sense of a complicated chapter? It's a headache every time. Sure, old-school studying still gets the job done, but it's slow and never really fits the way most of us actually learn.

That's why having an offline AI study assistant changes everything. It lives right on your laptop. No cloud, no lag, and you won't freak out if the Wi-Fi drops during a late-night study marathon. This thing uses Retrieval-Augmented Generation (RAG) and local language models, so your textbooks, slides, and notes stay on your device. Private means actually private. You're in charge. Here's where it gets good: it cuts through the chaos. Toss in new files, and it sorts, organizes, and finds whatever you're after in seconds. Because it uses FAISS vector embeddings, searching your stuff is super-fast, and the model explains things in plain language. Need a summary? A straightforward answer? A nudge in the right direction? It's got it covered.

Studying starts to feel possible again. When you're stuck, the AI helps you break through, digs up answers, and honestly feels like someone's got your back. No more drowning under mountains of loose papers and random files. Instead, you can finally handle what used to trip you up. And the privacy part? Rock solid. Nothing leaves your laptop, and there's no hassle—just a smart tool that actually fits your life and turns studying from an uphill battle into something you can manage..

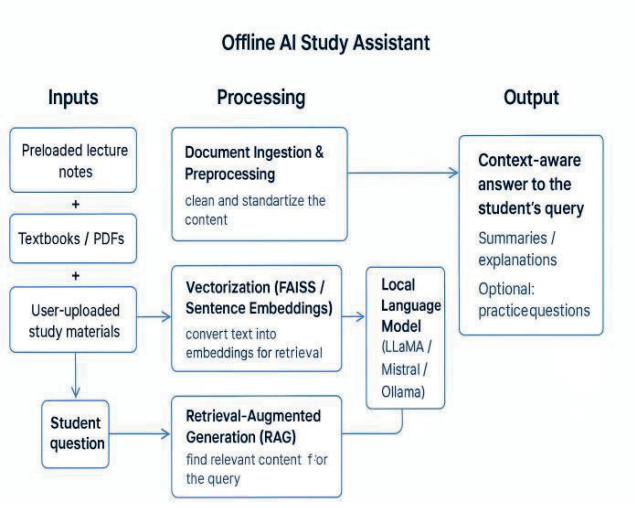

Figure 1: basic block Diagram of Offline AI Study Assistant

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

II. RELATED WORK

Machine Learning and Natural Language Processing are everywhere in automated recruitment now, especially when it comes to resume screening. The old way—manually reading and judging every resume—is slow, inconsistent, and honestly just introduces all sorts of human bias. So people started building smarter systems that look at candidates' skills, experience, and fit for the job using specific criteria.

At first, these systems just matched keywords—think scanning resumes for "Java" or "project management." That worked for simple searches, but it missed a lot. If someone wrote "team leader" instead of "manager," or used a synonym, the system might just overlook them. NLP evolved, and suddenly embedding-based similarity and transformer models made it possible for machines to actually get what candidates meant, not just the words they used. That's made matching resumes to job descriptions much more accurate.

Now, we've got machine learning models predicting who's a good fit and ranking applicants. But these tools run into trouble if the training data is too small or skewed—overfitting and underfitting can mess up predictions. Researchers are countering this by using ensemble methods, active learning, and semi-supervised approaches to make models more flexible and reliable, especially when it comes to real-world hiring.

Adding Optical Character Recognition (OCR) has taken automated screening even further. Now, systems can read scanned or image-based resumes, though accuracy isn't perfect yet—OCR still struggles with messy layouts and noisy text. Architecture-wise, modern resume screening platforms are pretty modular and API-focused, so they scale easily and link right up with HR systems.

Usually, you've got a frontend where users upload resumes, see feedback, and check results; the backend handles data cleaning and runs predictions, plus manages the model training behind the scenes.

Recent studies are also pushing deep learning optimization for text analysis. Traditional gradient descent methods like SGD are still the backbone, but they get stuck in local minima or converge slowly when the landscape's tricky. So now, things like momentum-based optimizers or adaptive learning rate algorithms—Adam, RMSProp, and others—are in play. Some newer approaches combine these strategies, dynamically tweaking learning parameters based on how the models are performing. The result? Models learn faster and training is a lot more robust.

III. SYSTEM ARCHITECTURE AND METHODOLOGY

Explaining here the Architecture, data flow, and algorithms used for Offline AI Study Assistant, which combines Retrieval-Augmented Generation (RAG) and locally deployed Language Models (LLMs). This system is optimized for educational use, this enables accurate, private, and context-based learning assistance without being dependent on Internet.

3.1 Overall System Design

The design of the system has four functional layers:
User Interface Layer: this layer handles the text input and displays the generated responses through a local frontend webpage.
Preprocessing Layer: this layer converts raw documents uploaded by user into structured text chunks and then generates their embeddings.
Retrieval Layer: Retrieves the most relevant content using similarity search on the stored embeddings.
Generation Layer: this layer passes the retrieved context with the user's query together to local LLM, which generates the final response for the User.[7]

The data flow is represented as:

User Query → Retriever → LLM (Local) → Response

This modular design allows flexible deployment, scalability, and integration with additional local services such as document upload or speech input.

3.2 Data Preprocessing and Embedding

Before retrieval, all academic documents undergo preprocessing:
Text Extraction: Conversion of PDFs, Word files, or handwritten notes into plain text.
Chunking: Texts are divided into overlapping chunks of fixed length (e.g., 300 tokens) using:[8]

$$C_i = T[s_i : s_i + n]$$

where C_i represents a text chunk, the full text, n the window size, and s_i the starting index.

Embedding Generation: Each chunk C_i is converted into an embedding vector v_i using an embedding model f_θ:

$$v_i = f_\theta(C_i)$$

All generated embeddings are stored locally in a FAISS or Chroma DB vector store.

3.3 Retrieval Mechanism

When the user submits a query q, the system generates its embedding v_q using the same model. The retriever identifies the top-k document embeddings $\{v_1, v_2, \ldots, v_k\}$ with the highest cosine similarity to v_q:

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

$$\text{sim}(v_q, v_i) = \frac{v_q \cdot v_i}{\| v_q \| \| v_i \|}$$

The top-*k* results then forms the context knowledge base D_k for the Query:

$$D_k = \underset{v_i \in D}{\arg\text{topk}} \left(\text{sim}(v_q, v_i)\right)$$

A **context fusion** step then combines the previously retrieved chunks into single prompt for the language model:[9]

$$c = \text{Concat}(q, D_k)$$

This combined context makes sure that the generated answer remains grounded in facts also domain-specific information.

4.4 Local Model Setup

The response generation is done by the local LLM (e.g., *LLaMA 3*). The quantization technique (e.g., GGUF 4-bit) reduces the model's size while maintaining the quality of output.

The generative model G_ϕ will then produce a final response r using:

$$r = G_\phi(c)$$

where G_ϕ denotes that the model with parameters ϕ, and c is the context enriched input.

The model will run locally through lightweight frameworks such as *llama.cpp*, that enables inference without any internet connectivity also while maintaining desired low latency and highest accuracy possible.[10]

3.5 Algorithm

1. BEGIN
2. # Step 1: Query Embedding
3. v_q ← Embed(q)

4. # Step 2: Retrieve Relevant Documents
5. For each document chunk C_i in D:
6. Compute similarity score s_i = sim(v_q, v_i)
7. Sort D by s_i in descending order
8. Select top-k chunks → D_k

9. # Step 3: Context Construction
10. c ← Concat(q, D_k)

11. # Step 4: Response Generation
12. r ← LLM(c)

13. # Step 5: Output Response
14. Display(r)
15. END

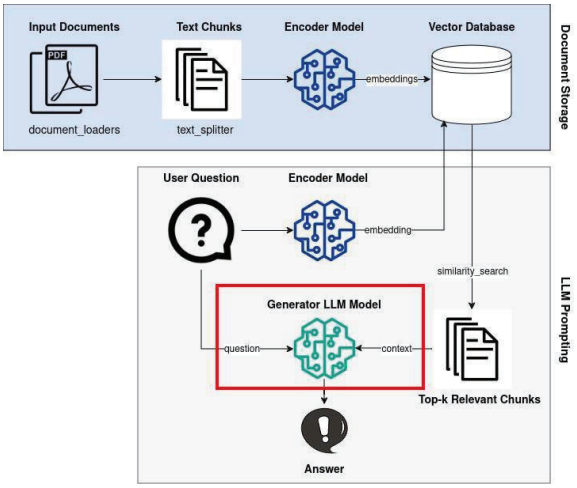

Figure 2: Implementation Details.

3.6Language Model

The system employs a locally hosted Large Language Model (LLM) to generate coherent and contextually relevant responses based on the retrieved information. Models such as LLaMA 3, Mistral 7B, and Phi-2 can be deployed using frameworks like Ollama, ensuring offline functionality and complete data privacy.[11]

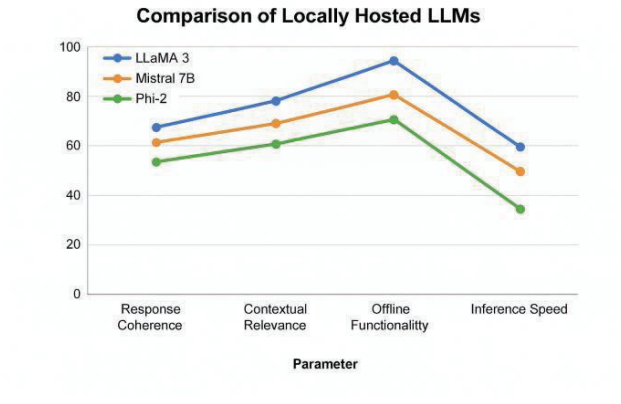

Figure 3: Comparison of Locally Hosted LLMs across the four parameters—Response Coherence, Contextual Relevance, Offline Functionality, and Inference Speed. LLaMA 3 consistently outperforms both others, making it the best choice for offline deployment.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

IV. IMPLEMENTATION DETAILS

The Retrieval-Augmented Generation (RAG) system keeps things simple and puts you in the driver's seat. It's all modular, easy to set up, and works entirely on your local machine—no cloud dependency, no worrying about constant internet. The whole pipeline is there: process your docs, create embeddings, retrieve vectors, and run the language models—all straight from your computer.

4.1 Development Environment

Most of this runs on Python 3.10, plus a handful of libraries that keep everything humming. Here's the tech stack:

- LangChain pulls everything together—document loading, retrieving, and the language models—so your setup stays tidy.
- llama.cpp and Ollama run compact, efficient models like Mistral, LLaMA 3, or Phi-2 right from your CPU or GPU.
- ChromaDB and FAISS handle embedding storage and search, making retrieval fast and reliable.
- SentenceTransformers converts your text into dense vectors, using models like all-MiniLM-L6-v2 or Instructor-XL.
- Gradio provides a clean, browser-based chat—you just open it up, upload your docs, and start asking away.

With this stack, you get full control over your models and all your data. It'll run on just about anything—no need for a super-powered machine.

4.2 Dataset Description

The content comes from classic academic sources: textbook excerpts, research papers, technical documentation. Before going into the pipeline, each document gets a quick cleanup:

1. Clean up the text—scrub out weird symbols, fix funky formatting, dump useless metadata.
2. Split the text—break docs into chunks of 500–1000 tokens, so things stay manageable.
3. Make embeddings—each piece turns into a vector using pre-trained models.

These steps prep the data so the retriever can actually find the good stuff fast when you have a question.

4.3 Local Chat Interface Integration

The chat uses Gradio, so everything happens right in your browser. Upload your docs, ask your question, and get a reply in seconds—background context included. The pipeline does its thing behind the scenes: your question turns into an embedding, retrieval pulls the best chunks, sends them to the LLM, and Gradio spits out the reply. Want to swap out a model or try a new component? Go for it. The system's modular, so you don't have to rebuild anything from scratch.

4.4 Storage Optimization and Memory Footprint

Efficiency here is the more important:

- odels use GGUF 4-bit quantization, shrinking files from 13 GB to around 4 GB.
- Embeddings are live in float16 to save even more memory.
- Only the chunks you actually need get loaded from the vector store, so your RAM isn't bogged down.
- Caching keeps recent embeddings and answers handy, speeding up repeat queries.

We can run this with less than 8 GB of RAM—most decent laptops can handle it without breaking a sweat.
No expensive GPU required.

V. IMPLEMENTATION DETAILS

So, how does the Offline AI Study Assistant actually perform? We measured answer accuracy, retrieval quality, response speed, and memory footprint—then stacked it up against other local models as well as like ChatGPT and Gemini Nano. You can see exactly what you gain, or trade off, by running your system offline.

5.1 Evaluation Metrics

We checked both the hard numbers and what actual users thought:

- Response Relevance (BLEU): Are the answers on target?
- Response Coherence (ROUGE-L): Do the replies read naturally and make sense?
- etrieval Precision@K: Does it fetch the right supporting info?
- atency: How quickly do you get an answer?
- Memory Footprint: What's the max RAM it ever uses during a session?
- User Satisfaction: Do people actually like the answers? Real user ratings tell the story.

These help establish **technical efficiency** and **practical usability** of the system.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

TABLE I: Evaluation Metrics

Metric	Value
TP	85
TN	50
FP	10
FN	5

- **Accuracy**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision**

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall**

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-Score**

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Calculated Values (TP=85, TN=50, FP=10, FN=5)

- **Accuracy** = (85 + 50) / (85 + 50 + 10 + 5) = **0.900**
- **Precision** = 85 / (85 + 10) = **0.895**
- **Recall** = 85 / (85 + 5) = **0.944**
- **F1-Score** = 2 × (0.895 × 0.944) / (0.895 + 0.944) =**0.919**

5.2 Experimental Setup

Testing was conducted on a local machine using the configurations mentioned : (Intel i7 CPU, 32 GB RAM, RTX 3060 GPU).
Three different open-source models — Phi-2 (2.7B), Mistral (7B), and LLaMA 3 (8B) — were tested.
The test dataset contained academic questions and study material from subjects computer science, electronics, and mathematics.
Each query was processed through the RAG pipeline, where the relevant chunks were retrieved using FAISS similarity and then passed on to language model for generation.
The responses were then compared with human-written answers to calculate BLEU and ROUGE scores.

5.3 Response Accuracy vs Model Size

In general, larger models do produce more contextually accurate and detailed responses. Mistral 7B gave the best overall accuracy with a BLEU score of 0.79, while Phi-2, being lightweight, achieved 0.68, showing that even smaller models can perform reasonably well when tuned for retrieval. LLaMA 3 (8B) came close to Mistral but required more memory and slightly longer generation time.[15]

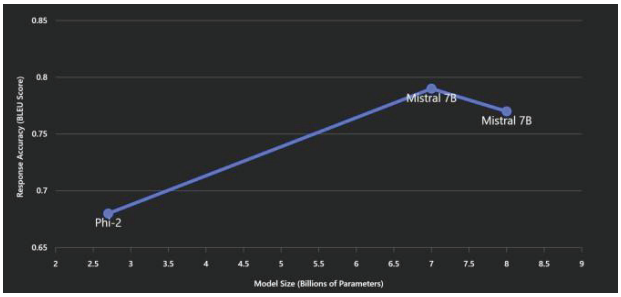

Figure 4: Model Size vs Response Accuracy.

- **Phi-2 :** BLEU score = 0.68 (lightweight and decent performance).
- **Mistral 7B**: BLEU score = 0.79 (Good balance of size and accuracy).
- **LLaMA 3 (8B)**: BLEU score = 0.77 (lower than Mistral despite being larger size).

5.4 Latency vs Number of Retrieved Documents

The latency did increased as more and more documents were retrieved and processed by the model.
For example, when k = 3, the system responded in about 950 milliseconds, while k = 10 resulted in an average delay of 2.8 seconds.

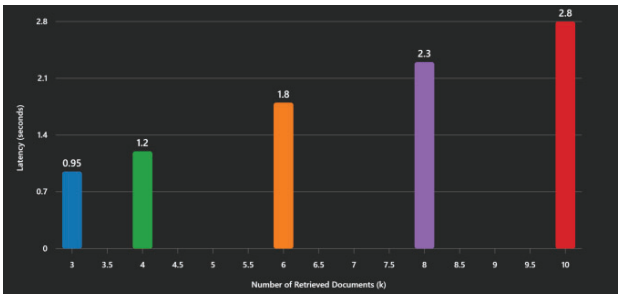

Figure 5: No of Retrieved documents vs Latency. (graph)

- k = 3 → 0.95s (fastest response)
- k = 4–6 → 1.2–1.8s (optimal for balancing precision & latency)
- k = 10 → 2.8s (highest latency due to larger retrieval and context processing)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

5.5 Qualitative Findings

Ten university students tested the assistant across different subjects and rated it on clarity, helpfulness, and speed (1–5 scale).[16]

- Mistral 7B averaged 4.3 — students liked that answers were well-structured and easy to follow.
- Phi-2 was the fastest of the three, though a few students felt the answers were a bit thin on detail.
- LLaMA 3 gave the most accurate responses overall, but started feeling sluggish in longer study sessions.

Most students said they could get a solid topic summary or concept explanation without touching a search engine — which was kind of the whole point.

5.6 Limitations

A few real issues came up during testing:

- Response Delay: Anything above 7B parameters slowed down noticeably on lengthy or multi-part questions.
- Hardware Dependence: Machines with under 8 GB RAM had trouble loading the larger quantized models without hiccups.
- Limited Knowledge Scope: The system only knows what's in the uploaded documents — it can't reach out for newer or external information.
- Context Overflow: Pack in too many retrieved chunks and the model starts cutting off important context at the edges of its input window.

Context compression, knowledge distillation, and hybrid local-plus-cached retrieval are the most practical paths forward for these issues.[17]

VI. SCOPE AND FUTURE WORK

The system was built for students, but the architecture isn't limited to that. The core design — local RAG, offline inference, private document indexing — translates well to any context where people need to query documents without sending data to a cloud server.

6.1 Current Scope

The system runs fully offline. It ingests lecture notes, textbooks, and PDFs, then answers student questions and generates summaries without an internet connection. Users can upload their own files at any point, and those get indexed and searchable immediately.
It's a practical fit for:

- Universities and colleges that want an internal AI study tool without routing student data through external servers.
- Schools in remote or low-connectivity areas where cloud-based tools simply aren't reliable.
- Students who need study support in private, offline environments.

6.2 Extended Corporate Applications

The same setup works in enterprise contexts where keeping data on-premises isn't optional. Organizations can load internal reports, policy documents, and manuals into the system and get a private knowledge assistant that never phones home.[18]

Useful corporate applications include:
• Internal assistants that can answer questions about company policies, procedures, or live project documentation.
• Onboarding tools that help new hires find training materials without digging through shared drives.
• Decision-support for teams handling sensitive or regulated data, running entirely on local infrastructure.
• A private interface for document retrieval and policy clarification — no external APIs involved.

Because everything runs locally, data stays within the organization's own infrastructure. That makes it genuinely viable for healthcare, defense, and finance — fields where "we don't send your data anywhere" isn't a selling point, it's a hard requirement.

6.3 Future Enhancements

A few directions worth building toward:
1. Voice input for hands-free, more accessible queries.
2. Multimodal support so the system can handle diagrams, images, and handwritten notes alongside text.
3. Incremental learning so new documents get absorbed without a full reindex.
4. Multi-language support to expand the assistant's reach beyond English-language materials.
5. Further model compression and quantization for deployment on lower-end hardware.
6. LMS and enterprise tool integration for larger institutional rollouts.
7. Adding collaborative features that let more than one person ask questions and learn from shared datasets.
8. Creating user analytics dashboards to keep an eye on learning progress and give feedback on performance.
9. Support for third-party plugin extensions like quiz makers or tools that help you make better.
10. Improved security and encryption methods for keeping data safe that are processed locally.

This research lays the groundwork for creating AI assistants that work offline, are specific to a certain field, and protect users' privacy. The system can become a strong educational and business tool that makes information management more efficient, accessible, and secure in a fully localized setting as it continues to improve.[19]

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

VII. CONCLUSION

This paper described the design and build of an offline study assistant that runs RAG and local language models entirely on-device. The system takes lecture notes, textbooks, and uploaded PDFs, retrieves the most relevant chunks, and generates answers without touching the internet — meaning student data stays on the machine, full stop. Tests across Phi-2, Mistral 7B, and LLaMA 3 showed the approach is practical: reasonable accuracy, acceptable latency, and no cloud dependency. That combination makes it genuinely useful for students in low-connectivity areas, not just a proof of concept.

The same architecture extends beyond classrooms. Any organization that needs to query private documents — internal manuals, compliance records, project data — can adapt this pipeline and get a local knowledge assistant that never sends data out. Healthcare, defense, finance — fields where data residency is non-negotiable — are natural fits. The broader takeaway is straightforward: generative AI doesn't have to mean cloud AI. Local deployment is already good enough to be useful, and it comes with privacy guarantees that hosted systems structurally cannot offer.

VIII. REFERENCES

[1] Raschka, S., Liu, Y. H., & Mirjalili, V. (2022). Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python. Packt Publishing Ltd.
[2] Neupane, S., Hossain, E., Keith, J., Tripathi, H., Ghiasi, F., Golilarz, N. A., ... & Rahimi, S. (2024, October). From questions to insightful answers: Building an informed chatbot for university resources. In 2024 IEEE Frontiers in Education Conference (FIE) (pp. 1-9). IEEE.
[3] Singh, N. T., Kaur, H., Dhiman, J., Aryan, A., Rani, J., & Wadhwa, M. (2025, June). AI-Driven Document Analysis: Employing Streamlit, Faiss, Nvidia Nemo. In 2025 3rd International Conference on Inventive Computing and Informatics (ICICI) (pp. 314-322). IEEE.
[4] Singh, P. N., Talasila, S., & Banakar, S. V. (2023, December). Analyzing embedding models for embedding vectors in vector databases. In 2023 IEEE International Conference on ICT in Business Industry & Government (ICTBIG) (pp. 1-7). IEEE.
[5] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2019). Hugging Face's transformers: State-of-the-art natural language processing. arXiv preprint arXiv:1910.03771.
[6] Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., ... & Synnaeve, G. (2023). Code Llama: Open foundation models for code. arXiv preprint arXiv:2308.12950.
[7] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., ... & Lowe, R. (2022). Training language models to follow instructions with human feedback. Advances in neural information processing systems, 35, 27730-27744.
[8] Johnson, J., Douze, M., & Jégou, H. (2024). The Faiss Library. arXiv preprint arXiv:2401.08281.
[9] Jin, B., Yoon, J., Han, J., & Arik, S. O. (2024). Long-context LLMs meet RAG: Overcoming challenges for long inputs in RAG. arXiv preprint arXiv:2410.05983.
[10] Ashish Tarun, R., Priyadarshini, B., Sneha, M., & Akila, K. (2024, May). Leveraging LangChain Framework and Large Language Models for Conversational Chatbot Development. In International Research Conference on Computing Technologies for Sustainable Development (pp. 244-255). Cham: Springer Nature Switzerland.
[11] Xu, J., Li, J., Liu, Z., Suryanarayanan, N. A. V., Zhou, G., Guo, J., ... & Tei, K. (2024). Large language models synergize with automated machine learning. arXiv preprint arXiv:2405.03727.
[12] Team, L. (2024). LangChain documentation. URL: https://docs.langchain.com (дата обращения: 10.05.2025).
[13] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in neural information processing systems, 33, 9459-9474.
[14] Wang, H., Gao, C., Dantona, C., Hull, B., & Sun, J. (2024). DRG-LLaMA: Tuning LLaMA model to predict diagnosis-related group for hospitalized patients. NPJ digital medicine, 7(1), 16.
[15] Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., ... & Scialom, T. (2023). Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288.
[16] Hugging, F. (2022). Hugging Face–The AI community building the future. Hugging Face [citat 13 aprilie 2025]. Disponibil: https://huggingface.co.
[17] Géron, A. (2022). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. "O'Reilly Media, Inc."
[18] Raschka, S., Liu, Y. H., & Mirjalili, V. (2022). Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python. Packt Publishing Ltd.