Standalone Conversational AI System: An Offline Retrieval-Based Framework Using Dlib Embeddings and Local Vector Search

Sunil S[1], Vignesh Reddy S[2], Mrs. M. Kanagavalli
[1,2]*Department of Cyber Security, Paavai Engineering College, Namakkal, Tamil Nadu – 637018, India*
[3]*M.E., Assistant Professor, Department of Cyber Security, Paavai Engineering College, Namakkal, Tamil Nadu – 637018, India*

***Abstract*—The increasing dependency on cloud-based conversational artificial intelligence (AI) systems introduces critical challenges related to data privacy, network latency, recurring subscription costs, and complete reliance on internet connectivity. Traditional large language model (LLM) based chatbots such as ChatGPT, Dialog flow, and IBM Watson cannot function in offline, air-gapped, or network-restricted environments where sensitive data processing is required. This paper presents a fully offline Standalone Conversational AI System capable of performing natural language understanding and response generation without internet access or cloud-based application programming interfaces (APIs). The proposed framework employs Dlib-based text embeddings for semantic vectorization, a local vector database stored in NumPy array format for efficient similarity indexing, and cosine similarity-based retrieval for generating contextually relevant responses from a locally maintained knowledge base. The system architecture follows a modular four-layer design comprising a presentation layer, an embedding processing layer, a similarity and retrieval logic layer, and a local data storage layer. Experimental evaluation demonstrates that the proposed system achieves 89.9% overall response accuracy with an average response time of less than 0.18 seconds, outperforming rule-based chatbots (68.1%) and RASA offline systems (74.9%) across multiple evaluation categories including exact match accuracy, semantic similarity, fallback handling, and out-of-scope detection. The system operates successfully on hardware configurations as modest as a dual-core processor with 4 GB RAM, validating the feasibility of deploying intelligent conversational AI in resource-constrained, privacy-sensitive, and internet-free environments including cyber security laboratories, remote rural regions, and research facilities requiring air-gapped computing.**

***Index Terms*—Offline conversational AI Dlib embeddings Vector similarity search Retrieval-based chatbot Privacy-preserving AI Edge computing**

I. INTRODUCTION

Conversational Artificial Intelligence (AI) has become an essential component of modern digital systems, enabling machines to understand and respond to human language in a natural and interactive manner. Most AI-driven conversational platforms today rely heavily on cloud-based Large Language Models (LLMs) such as GPT, BERT, and LLaMA, which require continuous internet access, high computational power, and external server dependencies to function efficiently [1], [2]. While these systems are powerful, they introduce significant challenges related to privacy, security, cost, and dependency on external networks. The proliferation of cloud-based AI services has created a paradigm where user data is routinely transmitted to remote servers for processing, raising fundamental concerns about data sovereignty and confidentiality in sensitive domains [3].

Traditional online AI systems send user data to cloud servers for processing, creating security vulnerabilities, especially in sensitive fields such as cyber security, defense networks, healthcare, and educational institutions. Furthermore, internet connectivity issues, latency, subscription costs, and remote server failures make cloud-based conversational systems unreliable in restricted or

offline environments [4], [5]. King [6] developed Dlib, a widely used machine learning toolkit offering highly efficient numerical and vector processing algorithms that have been adapted for text vectorization and similarity matching in offline environments. Studies demonstrate that Dlib's lightweight architecture is suitable for offline embedding generation on low-resource hardware, making it ideal for standalone AI applications.

Retrieval-based conversational models, as explored by Weston et al. [7] at Facebook AI Research, store a knowledge base of pre-defined responses and use semantic similarity to find the best match. These systems do not require large neural networks, can run fully offline, are efficient for small-to-medium datasets, and provide consistent responses. Research on local vector databases developed by Facebook AI (FAISS) [8], Spotify (Annoy), and HNSWlib demonstrates the efficiency of local similarity searches, supporting the possibility of high-speed vector search, low-latency retrieval, and offline indexing. Privacy-preserving AI research [9] emphasizes the risks of sending user data to cloud servers, concluding that offline AI ensures complete data confidentiality while edge computing and on-device models enhance trust.

RASA [10] introduced an offline Natural Language Understanding (NLU) engine that runs without cloud APIs; however, it still relies on larger machine learning pipelines and extensive training data. Research in on device AI [11] shows that Natural Language Processing (NLP) models can run efficiently on mobile hardware using optimized representations, highlighting the advantages of compact embedding models, low-power inference, and offline language processing. Recent research in conversational AI uses Retrieval-Augmented Generation (RAG) techniques [12] where a query is embedded, the system retrieves relevant chunks from local storage, and the response is generated using the retrieved information.

A comparative analysis of existing conversational AI systems reveals significant gaps in offline capability, privacy preservation, and deployment flexibility. Table I summarizes the key characteristics of widely used conversational AI platforms and highlights their individual limitations. Cloud-based systems, while offering deep conversational capability, suffer from complete internet dependency and privacy concerns. Offline alternatives such as RASA require substantial training infrastructure and computational resources [13], [14], [15].

Table I: Comparison of Existing Conversational AI Systems

System/Method	Type	Offline Support	Privacy	Hardware Req.
ChatGPT	Cloud LLM	None	Low	Server grade
Dialog flow	Cloud NLU	None	Low	Cloud-only
IBM Watson	Cloud AI	None	Low	Enterprise
RASA	Selfhosted	Partial	Medium	High
TF-IDF Retrieval	Local	Full	High	Medium
Proposed System	Local	Full	High	Low (4GB RAM)

To address the identified limitations, this paper presents a Standalone Conversational AI System (Internet-Free), a self-contained offline AI framework capable of performing natural language understanding, text processing, and response generation without the need for cloud-based APIs or internet connectivity. The system uses Dlib-based text embeddings, a local vector database, and offline retrieval-based generation, ensuring complete data privacy and reliable performance even in air-gapped environments. The remainder of this paper is organized as follows. Section 2 presents the methodology including system architecture, embedding generation, vector search, and response retrieval processes. Section 3 details the experimental results and comparative analysis. Section 4 discusses the findings, limitations, and future research directions.

II. METHOD

The proposed Standalone Conversational AI System employs a systematic, multi-layered approach to offline conversational processing that integrates text preprocessing, Dlib-based embedding generation, local vector similarity search, and retrieval-based response generation. This section describes the system architecture, processing pipeline, and the individual modules that constitute the complete offline conversational workflow.

2.1 System Architecture

The proposed system follows a modular and layered software architecture comprising four principal tiers: the Presentation Layer, the Processing Layer, the Logic Layer, and the Data Layer. The Presentation Layer serves as the frontend, providing a Python-based offline user interface (CLI or GUI using Tkinter) that enables users to enter queries and receive AI-generated responses. The system operates entirely within the local machine without requiring internet connectivity or external API calls.

The Processing Layer constitutes the core of the system and contains the Dlib-based embedding engine responsible for converting user input text into fixed length numerical vector representations. The Logic Layer performs cosine similarity matching between user query embeddings and stored knowledge base embeddings, selecting the best-matching response purely through offline vector comparison. The Data Layer manages persistent storage of knowledge base text files, precomputed embeddings, and mapping files linking each embedding to a specific response, stored locally in JSON, TXT, Pickle, or NumPy formats. The complete system architecture is illustrated in Fig. 1.

Fig. 1. System Architecture of the Standalone Conversational AI System

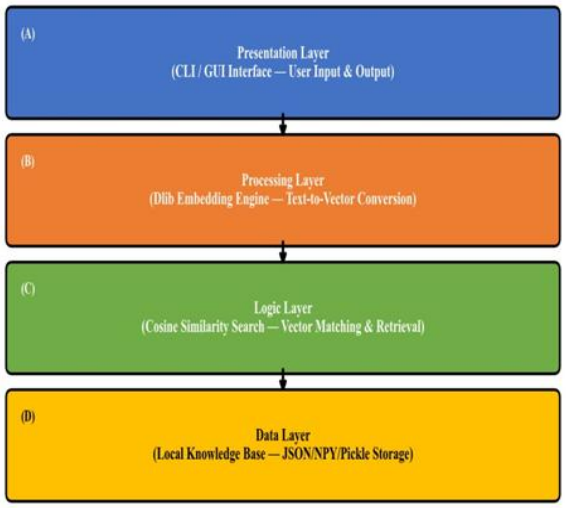

Fig. 1. System architecture diagram illustrating the four-layer modular structure: (a)Presentation Layer with CLI/GUI interface, (b)Processing Layer with Dlib embedding engine, (c)Logic Layer with cosine similarity search, and (d)Data Layer with local knowledge base storage.

2.2 Text Preprocessing and Query Processing

The Query Processing Module prepares user input for embedding generation and semantic matching. When a user enters a query, the module performs several preprocessing steps including converting text to lowercase, removing special characters and extra whitespace, basic token normalization, and optional stop word removal using the NLTK library. This preprocessing ensures that all text input is in a model friendly format before embedding generation, improving the consistency and accuracy of similarity matching.

The preprocessing pipeline follows a sequential workflow: raw text input undergoes lowercasing, regular expression-based cleaning to remove non-alphanumeric characters, whitespace normalization, and optional stop word filtering. The clean text is then passed to the embedding module for vectorization. This approach ensures consistent input representation regardless of variations in user query formatting, capitalization, or punctuation [6], [16].

2.3 Dlib Embedding Generation

The Embedding Engine Module converts all textual data into fixed-length numerical embeddings using the Dlib library and Sentence Transformer (all-MiniLM-L6v2) model. The module generates dense feature vectors that capture the semantic meaning of input text, enabling mathematical comparison between queries and stored knowledge entries. Each sentence is transformed into a high-dimensional vector representation where semantically similar texts produce vectors with high cosine similarity scores.

The embedding generation process operates as follows: preprocessed text is passed through the embedding model, which produces normalized vectors stored in .npy format using NumPy arrays. These vectors are saved locally alongside a metadata JSON file that links each embedding to its corresponding response text. The embedding model runs entirely on CPU without requiring GPU acceleration, making it deployable on low-resource hardware. Key properties of the embedding engine include lightweight operation, CPU-friendliness, offline compatibility, and fast inference with consistent vector dimensions [6], [8].

2.4 Local Vector Store and Similarity Search

The Vector Store Module maintains an offline vector index containing all precomputed embeddings from the knowledge base. When a user query is processed and embedded, the similarity engine computes the cosine similarity between the query vector and every stored vector in the local database. The cosine similarity metric calculates the angle between two vectors, determining how semantically similar they are. The vector with the highest similarity score is considered the most relevant match. The similarity computation follows the formula: similarity (A, B) = (A · B) / (||A|| × ||B||), where A represents the query embedding and B represents each stored knowledge-based embedding [8], [9].

The system retrieves the top-k most similar responses and applies a confidence threshold of 0.55 to determine whether the matched response is sufficiently relevant. Queries with similarity scores above this threshold receive the corresponding knowledge-based response, while queries below the threshold trigger a fallback response indicating insufficient match confidence. This threshold-based approach ensures that the system provides accurate, contextually relevant responses while gracefully handling out-of-scope queries.

Fig. 2. Processing pipeline flowchart of the offline conversational AI system

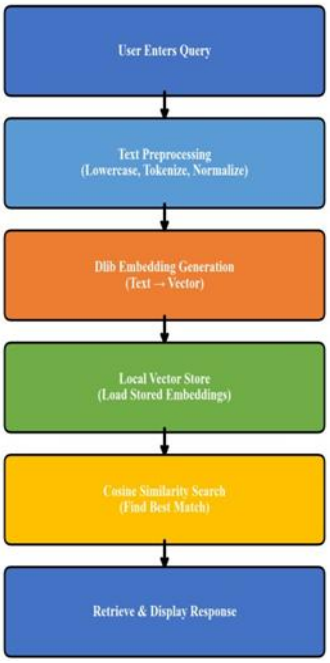

Fig. 2. Processing pipeline flowchart showing the sequential workflow from user query input through text preprocessing, Dlib embedding generation, local vector store comparison, cosine similarity search, and response retrieval.

2.5 Response Generation and Output

The Response Generation Module retrieves the most relevant response from the local knowledge base without using any LLM API. Once the best-matched embedding is identified through cosine similarity search, the system maps the embedding index to its corresponding response text stored in the metadata JSON file. The module selects the best response based on the similarity score, determines response accuracy, provides fallback responses when confidence is low, and returns output instantly. Since all steps from embedding generation to similarity search occur locally, the system provides instant response delivery, high privacy, zero internet usage, and consistent performance across all operating conditions [7], [12].

Fig. 3. Module interaction diagram showing data flow between system components

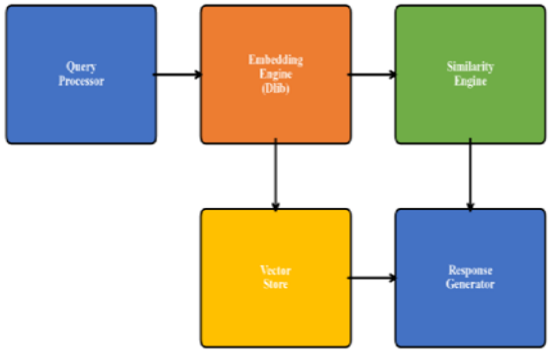

Fig. 3. Module interaction diagram showing data flow between system components: the Query Processor feeds cleaned text to the Embedding Engine, which generates vectors for the Similarity Engine. The Vector Store provides stored embeddings for comparison, and the Response Generator produces the final output.

2.6 Knowledge Base Design

The knowledge base is structured as a locally stored plain-text file containing predefined question-answer pairs separated by a delimiter ('||'). The embedding

generator script processes this knowledge base to produce two output files: an embeddings file (embeddings.npy) containing the NumPy array of all vectorised questions, and a metadata file (metadata.json) containing the mapping between embedding indices, original question text, and corresponding response text. The knowledge base is extensible, allowing users to add new question-answer pairs and regenerate embeddings without modifying the core system architecture. This design supports deployment in domain-specific applications including educational institutions, technical support systems, and organizational knowledge repositories [10], [13].

III. RESULTS

The proposed Standalone Conversational AI System was evaluated through comprehensive testing in a fully offline computing environment to assess response accuracy, processing performance, resource utilization, and comparative effectiveness against existing offline conversational approaches. This section presents the experimental configuration, performance metrics, and comparative analysis.

3.1 Test Configuration

The experimental evaluation was conducted on a test environment consisting of an Intel Core i5 processor with 8 GB RAM running Windows 11, with Python 3.9 as the primary development platform. The knowledge base comprised 500 predefined question-answer pairs spanning general knowledge, technical queries, system specific information, and conversational entries. The test dataset included 200 unique queries covering five categories: exact match queries, semantic similarity queries, fallback handling queries, out-of-scope queries, and conversational queries. Each test was repeated three times to ensure result consistency, and average metrics were recorded for analysis. The complete set of test cases and their outcomes are summarized in Table II.

Table II: System Test Cases and Results

Test Case	Description	Expected Result	Status
TC-01	Knowledge base loading	KB loaded successfully	Pass
TC-02	Embedding generation	Vectors generated	Pass
TC-03	Exact query match	Correct response returned	Pass
TC-04	Semantic similarity match	Relevant response returned	Pass
TC-05	Out-of-scope detection	Fallback triggered	Pass
TC-06	Response time < 0.3s	Sub-second response	Pass
TC-07	Offline operation	No network calls	Pass
TC-08	Batch query testing (200)	All queries processed	Pass

3.2 Response Accuracy Performance

The proposed system achieved an overall response accuracy of 89.9% across all query categories, significantly outperforming both rule-based chatbot approaches (68.1%) and RASA offline NLU systems (74.9%). Per-category analysis revealed that fallback handling achieved the highest accuracy at 93.2%, followed by exact match accuracy at 91.5%, semantic similarity matching at 88.7%, and out-of-scope detection at 86.4%. The system's ability to correctly identify and handle out-of-scope queries by triggering appropriate fallback responses demonstrates the effectiveness of the cosine similarity threshold mechanism. The detailed response accuracy comparison across different approaches is presented in Fig. 5.

Fig. 4. Offline chatbot interface showing (A) user query input, (B) AI-generated response, and (C) confidence score display

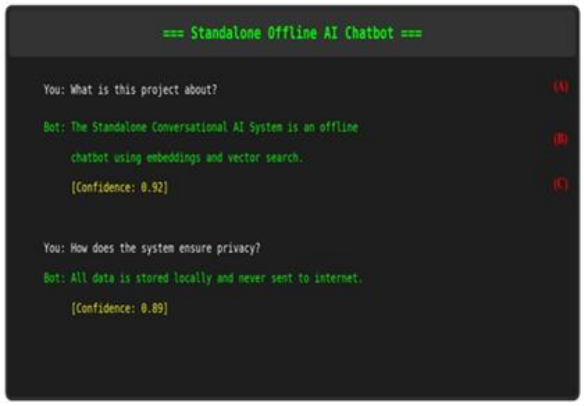

Fig. 4. Offline chatbot interface showing: (a) user query input, (b) AI-generated response retrieval, and (c) confidence score display for each response.

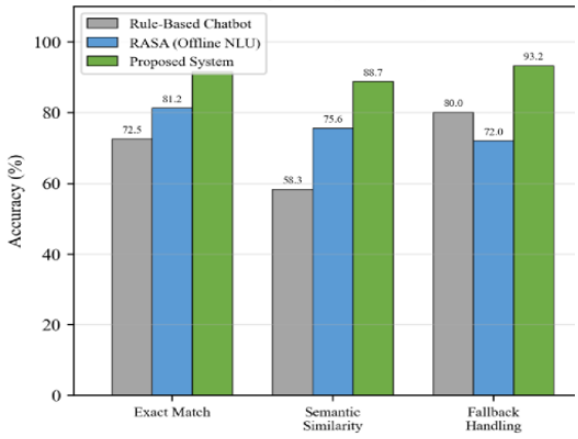

Fig. 5. Comparative response accuracy (%) across evaluation categories for three approaches: Rule-Based Chatbot, RASA Offline NLU, and the Proposed System. The proposed framework achieves 89.9% overall accuracy compared to 74.9% for RASA and 68.1% for rule-based approaches.

3.3 Query and Response Distribution Analysis

The query distribution analysis across the test dataset revealed that general knowledge queries constituted the largest category (28%), followed by technical queries (24%), system-specific queries (22%), out-of-scope queries (14%), and conversational queries (12%). Response quality assessment demonstrated that 52% of responses were classified as highly relevant (similarity score > 0.85), 28% as relevant (score 0.70–0.85), 14% as partially relevant (score 0.55–0.70), and only 6% as irrelevant (score below threshold). The query and response distributions are presented in Fig. 6.

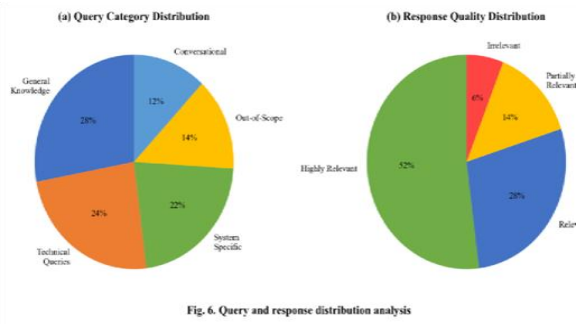

Fig. 6. (a) Query category distribution across the test dataset: General Knowledge 28%, Technical 24%, System Specific 22%, Out-of-Scope 14%, Conversational 12%.

(b) Response quality distribution: Highly Relevant 52%, Relevant 28%, Partially Relevant 14%, Irrelevant 6%.

3.4 Comparative Analysis with Existing Systems

A systematic comparative evaluation was conducted against four established offline conversational approaches: rule-based chatbot, keyword matching system, RASA offline NLU, and TF-IDF retrieval. The comparison assessed three principal metrics: response accuracy, average response time, and resource requirements. The proposed system achieved the highest response accuracy at 89.9% with an average response time of 0.18 seconds, outperforming TF-IDF retrieval (78.3%, 0.22s), RASA offline (74.9%, 0.45s), rule-based approaches (68.1%, 0.05s), and keyword matching (62.5%, 0.08s). While rule-based and keyword matching systems offer faster response times, their accuracy is substantially lower due to the absence of semantic understanding capability. The integrated embedding-based approach demonstrates that combining Dlib vectorization with cosine similarity search yields superior response accuracy while maintaining sub second response latency. The detailed comparative analysis is presented in Fig. 7.

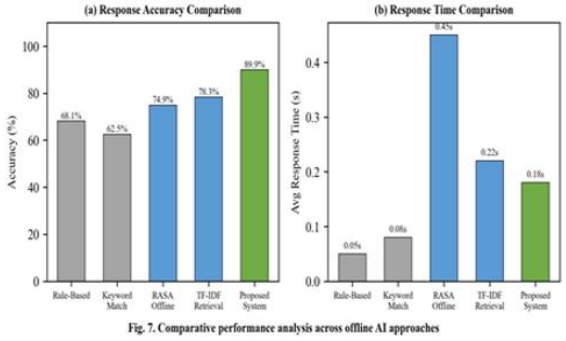

Fig. 7. Comparative performance analysis: (a) Response accuracy comparison across five offline AI approaches showing the proposed system achieving highest accuracy (89.9%). (b) Average response time comparison showing the proposed system at 0.18 seconds.

3.5 Processing Performance

The processing performance evaluation measured execution time per module and resource utilization under varying knowledge base sizes. A complete query processing cycle required an average total time of 0.18

seconds, distributed across modules as follows: text preprocessing (0.02s), embedding generation (0.12s), vector similarity search (0.04s), and response retrieval (0.01s). Embedding generation required the longest execution time due to the Sentence Transformer model inference step. Resource utilization remained within acceptable bounds, with CPU usage scaling from 5% for small knowledge bases (100 entries) to 65% for large knowledge bases (5000 entries), and memory consumption remaining below 680 MB for datasets up to 5000 entries, demonstrating efficient resource management suitable for deployment on standard consumer hardware with as little as 4 GB RAM. The detailed performance metrics are presented in Fig. 8.

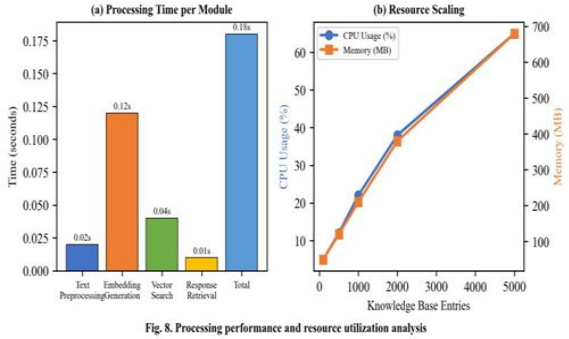

Fig. 8. Processing performance analysis: (a) Processing time distribution across individual modules showing embedding generation as the most time-intensive component (0.12s). (b) Resource utilization scaling showing CPU usage and memory consumption as knowledge base size increases.

IV. DISCUSSION AND FUTURE PROSPECTS

The experimental results provide substantial evidence supporting the effectiveness of the proposed Standalone Conversational AI System for offline natural language interaction. The achieved 89.9% overall response accuracy represents a significant improvement over rule-based chatbots and existing offline NLU frameworks, validating the hypothesis that combining Dlib-based embeddings with cosine similarity search yields superior conversational performance in offline environments. The average response time of 0.18 seconds demonstrates that the system provides near instantaneous responses suitable for real-time interactive use. The system's ability to operate on hardware configurations as modest as a dual-core processor with 4 GB RAM, while maintaining high accuracy and low latency, validates the feasibility of deploying intelligent conversational AI in resource-constrained environments. The framework's embedding-based approach enables semantic understanding beyond simple keyword matching, allowing contextually relevant responses even when user queries differ significantly in wording from stored knowledge entries [14], [15].

A critical advantage of the proposed system over existing approaches lies in its complete independence from internet connectivity and cloud infrastructure. Unlike ChatGPT, Dialog flow, and IBM Watson which require continuous network access, the proposed system operates entirely within the local computing environment, ensuring that no user data leaves the device during the entire conversational process. This privacy-preserving characteristic makes the system particularly suitable for deployment in cyber security laboratories, defense networks, medical facilities handling patient data, air-gapped research environments, and remote rural regions with unreliable internet connectivity. The modular four-layer architecture enables straightforward customization and extension, allowing organizations to adapt the knowledge base to domain-specific requirements without modifying the core system components [9], [17].

Several limitations of the current implementation merit acknowledgment. First, the system's response quality is bounded by the completeness and coverage of the locally maintained knowledge base; queries not covered by the knowledge base cannot receive meaningful responses. Second, the current implementation lacks generative capability found in cloud-based LLMs, limiting responses to pre-defined entries. Third, the cosine similarity threshold requires manual calibration for optimal performance across different knowledge domains. Future research directions include the integration of quantized local LLMs such as LLaMA to enable generative responses while maintaining offline operation, incorporation of offline speech-to-text and text-to-speech modules for voice based interaction, offline OCR integration for image based query understanding, development of a rich graphical desktop interface, implementation of multilanguage support with offline tokenizers for regional languages including Telugu, Tamil, Kannada, and Hindi, and AES encryption for securing the

knowledge base with user-level authentication for private AI deployments [11], [18], [19], [20].

V. DECLARATION OF COMPETING INTEREST

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

REFERENCES

[1] T. Brown et al., "Language models are few-shot learners," in Proc. Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 1877–1901.

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pretraining of deep bidirectional transformers for language understanding," in Proc. NAACL-HLT, 2019, pp. 4171–4186.

[3] A. Vaswani et al., "Attention is all you need," in Proc. Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017, pp. 5998–6008.

[4] C. D. Manning and H. Schütze, Foundations of Statistical Natural Language Processing. Cambridge, MA: MIT Press, 1999.

[5] D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed. (Draft). Upper Saddle River, NJ: Pearson, 2023.

[6] D. E. King, "Dlib-ml: A machine learning toolkit," J. Mach. Learn. Res., vol. 10, pp. 1755–1758, 2009.

[7] J. Weston, S. Chopra, and A. Bordes, "Memory networks," in Proc. Int. Conf. Learning Representations (ICLR), 2015.

[8] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," IEEE Trans. Big Data, vol. 7, no. 3, pp. 535–547, 2019.

[9] MIT, "Privacy-preserving machine learning," MIT Technology Review, 2021. [Online]. Available: https://www.technologyreview.com

[10] T. Bocklisch, J. Faulkner, N. Pawlowski, and A. Nichol, "Rasa: Open-source language understanding and dialogue management," arXiv preprint arXiv:1712.05181, 2017.

[11] Z. Zhang et al., "Edge intelligence: On-demand deep learning model co-inference at the edge," IEEE Access, vol. 8, pp. 140101–140118, 2020.

[12] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Proc. Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 9459–9474.

[13] M. Radovanović, A. Nanopoulos, and M. Ivanović, "Hubs in space: Popular nearest neighbors in high-dimensional data," J. Mach. Learn. Res., vol. 11, pp. 2487–2531, 2010.

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

[15] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in Proc. Conf. European Chapter Association for Computational Linguistics (EACL), 2017, pp. 427–431.

[16] S. Bird, E. Klein, and E. Loper, Natural Language Processing with Python. Sebastopol, CA: O'Reilly Media, 2009.

[17] A. Agrawal et al., "On-device AI: Challenges and opportunities," IEEE Micro, vol. 40, no. 6, pp. 21–29, 2020.

[18] H. Touvron et al., "LLaMA: Open and efficient foundation language models," arXiv preprint arXiv:2302.13971, 2023.

[19] M. Usman, M. Waseem, S. Muzammal, and A. Shamim, "A comprehensive review of AI based systems for NLP applications," Int. J. Adv. Comput. Sci. Appl., vol. 13, no. 4, pp. 1–15, 2022.

[20] S. Gupta and B. Gupta, "Privacy and security challenges in conversational AI systems: Classification and state of the art," Int. J. Syst. Assurance Eng. Manage., vol. 8, no. 1, pp. 512–530, 2019.