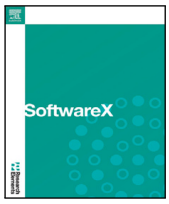
SoftwareX 29 (2025) 102029

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx



Original software publication

LLM based QA chatbot builder: A generative AI-based chatbot builder for question answering

Md. Shahidul Salim [a], Sk Imran Hossain [a,*], Tanim Jalal [a], Dhiman Kumer Bose [a], Mohammad Jahid Ibna Basher [b]

[a] Khulna University of Engineering & Technology, Bangladesh
[b] Chattogram University of Engineering and Technology, Bangladesh

ARTICLE INFO

Keywords:
LLM
Chatbot
Retrieval augmented generation

ABSTRACT

Large language model (LLM) based interactive chatbots have been gaining popularity as a tool to serve organizational information among people. Building such a tool goes through several development phases i.e. (a) Data collection and preprocessing, (b) LLM fine-tuning, testing, and inference, and (c) Chat interface development. To streamline this development process, in this paper, we present the LLM Question–Answer (QA) builder, a web application, which assembles all the steps and makes it easy for technical and non-technical users to develop the LLM QA chatbot. The system allows the instruction fine-tuning of following LLMs: Zepyhr, Mistral, Llama-3, Phi, Flan-T5, and user provided model for organization-specific information retrieval (IR), which can be further enhanced by Retrieval Augmented Generation (RAG) techniques. We have added an automatic web crawling based RAG data scrapper. Also, our system contains a human evaluation feature and RAG metrics for assessing model quality.

Code metadata

Current code version	V1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00446
Permanent link to Reproducible Capsule	None
Legal Code License	MITLicense.
Code versioning system used	Git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	python 3.10, cudatoolkit, torch, torchvision, torchaudio, jupyter, langchainhub, sentence-transformers, faiss-gpu, docx2txt, langchain, bitsandbytes, transformers, peft, accelerate, pynvml, trl, datasets, uvicorn, gradio, chroma, ragatouille
If available Link to developer documentation/manual	https://github.com/shahidul034/LLM-based-QA-chatbot-builder/blob/main/README.md
Support email for questions	ss@cse.kuet.ac.bd

1. Motivation and significance

In artificial intelligence and Natural Language Processing (NLP), developing intelligent chatbots has gained momentum. A chatbot is a QA system that can provide answers in natural language, either by referring to a set of documents or without any context (as in closed-book QA tasks).

Question Answering is a research area that combines research from different, but related, fields which are IR, Information Extraction and NLP [1–3]. QA presents specific challenges within the field of IR, such as retrieving precise, relevant information to answer complex questions rather than simply returning a list of documents. It has been used as one of the most natural types of human–computer communication. For

* Corresponding author.
E-mail address: imran@cse.kuet.ac.bd (Sk Imran Hossain).

https://doi.org/10.1016/j.softx.2024.102029
Received 28 August 2024; Received in revised form 19 December 2024; Accepted 22 December 2024
Available online 31 December 2024
2352-7110/© 2024 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

LLM [4], a high-quality dataset typically helps identify the context, such as the portion of the input prompt or surrounding information, that assists the model in solving a particular task. In RAG, for instance, context includes relevant documents retrieved from external sources. Most of the chatbots are developed using deep learning models [5], such as LLMs, and some of them are pattern matching or a rule-based approach [6]. It differs from the data-driven Question–Answering system model, which can be created using data or conversation history that has been conducted, allowing for sufficient development to train the model with the available data [7]. According to some deep learning studies, neural network models produce results that are good enough to be utilized in a QA system. One of them achieves a good performance by using the sequence-to-sequence approach [8–10]. Recently, some open-source LLMs like Zepyhr [11], Mistral [12], Llama-3 [13], Phi [14] and Flan-T5 [15] had been trained on huge amounts of data and can understand the context correctly. These models can be used to develop intelligent chatbot for QA. Furthermore, RAG, a method that combines generative models with external knowledge retrieval, can increase the model performance for factual QA. Fig. 1 illustrates a RAG framework for enhancing the performance of LLMs and outlines the specific process required to develop an LLM-based QA chatbot. While open-source models can be fine-tuned for specific tasks to improve performance, many of these models are also capable of handling various tasks without additional fine-tuning. Data collection, data curation, fine tuning, testing, inference and deployment are the needed steps to develop an intelligent question–answering system using LLM. However, there is no open-source free software or framework to help with the development of QA chatbot for technical or non-technical people. It is difficult to collect data individually for instruction fine-tuning LLMs and building RAG. Moreover, we need to preprocess and structure the data for fine tuning. After fine-tuning, it is important to check the model's performance. There are some evaluation metrics, but they are not good enough for evaluating LLM models [16]. So, human evaluation is the most appropriate way of testing LLM's performance.

There is an open-source application Verba [17], which offers an end-to-end, streamlined, and user-friendly interface for RAG. To use Verba API keys are needed for various components depending on the chosen technologies. There is no way to collect datasets from users, validate them, and deploy the model for the users. There is another application named autotrain-advanced [18], which supports only fine-tuning of the LLMs.

Our software surpasses LocalRQA [19] by offering support for 4-bit and 8-bit quantized models and automatic web crawling based RAG data scrapper, making it efficient for low-resource consumer PCs. Additionally, we included the latest models such as Llama-3, Phi-3, Mistral, and Zephyr, ensuring cutting-edge performance and capabilities.

We have developed software that facilitates the end-to-end process of developing a LLM QA application. We added features to collect data from users. Our software features an automatic RAG builder using web crawler, which is further used for inferencing. We added a human evaluation interface to collect user ratings about the models for specific questions provided by the user. After evaluation, we have added another feature to deploy the model for the user.

The open-source Python module LangChain [20] is used in our code to connect with the LLM. For RAG, we need a vector database. First, we need to convert the RAG data into a vector database, and then save the vector database in a local folder. To convert data into a vector database, we used the ensemble of"bge-large-en-v1.5" [21] and "ColBERT" [22] text embedding models. For storing and processing vector databases, we used Chroma [23], which is also open-source library. We used Chroma DB, an open-source vector store, for storing the vector database. Chroma DB is utilized for the storage and retrieval of vector embeddings.

In this work, we have proposed a framework and software to develop an intelligent chatbot using open-source models. Moreover, we apply this method to provide context using external sources; as a result, the user does not need to fine-tune the model each time when new data is added. This user-friendly, easy-to-use software allows for directly developing an LLM-based QA chatbot without programming requirements. With the aid of this software, researchers will be able to effortlessly assess the performance of both their new LLMs and their benchmark dataset.

2. Software description

A general description of this software is given in Section 2.1. Following, details on the implementation of each feature are provided in Section 2.2.

2.1. Software architecture

The software consists of six main functionalities: data collection, fine-tuning, testing data generation and RAG customization, human evaluation, inference, and deployment. Each functionality is described in Section 2.2. The LLM QA chatbot builder utilize different open-source LLMs, embedding models, and Python libraries as its basis, as there are different considerations to be taken into account, particularly in terms of the balance between speed and quality. We added features to collect data from users and automatically build RAG. We have added the recent 7/8B parameter models (Mistral, Zepyhr, and Llama-3) and some light-weight models (Phi-3 and Flan-T5) for low-configuration devices, and an embedding model fine-tuning option. The tool features a customizable LLM that has the potential to incorporate enhanced LLMs in the future. We have added a feature to collect user ratings and integrated RAG evaluation metrics to assess the performance of different models, data chunking methods, and embedding approaches. Lastly, we added an option for deploying the model. Fig. 2 shows the software architecture of LLM based QA builder.

2.2. Software functionalities

Data collection, fine-tuning, testing data generation and RAG customization, human evaluation, inference, and deployment are the six tabs that make up the software. Each tab is described below in detail.

2.2.1. Data collection

The first tab, the "Data collection" tab, is primarily used to collect training, testing and embedding data from the user. We have divided this tab in three sections. In the first section, the user can manually insert the training and testing data using two files:"finetune_data" and "testing_data". The "finetune_data" file is used for training the models, and the "testing_data" file is used for testing the models. For these files the extension can be either *xlsx* or *csv*. Both files are placed in the data folder. Training file has two columns: "question" and "answer", and testing file has three columns: "question", "ground_truth", and "context". The embedding file format depends on the selected loss function and is displayed once the loss function is chosen. There is generally no minimum data requirement for instruction fine-tuning LLMs, but the exact amount depends on several factors such as model size, downstream task etc. Fine-tuning, especially instruction fine-tuning, requires high-quality data in sufficient quantity to ensure the model learns effectively without overfitting. Table 1 shows the training data format for fine-tuning.

In the second section, the user interface is given below for collecting training, testing data, and embedding data. This interface can be used to collect data from other users. This tab is divided into three sub-tabs: "Training Data Generation", "Testing Data Generation", and "Embedding Data Generation". The "Testing Data Generation" sub-tab is further divided into two tabs: "Existing Questions" and "Custom Questions". In the "Existing Questions" tab, the user can answer the existing questions that are given by the user and then save the question and answer in an *xlsx* or *csv* file using the "Save the Answer" button.

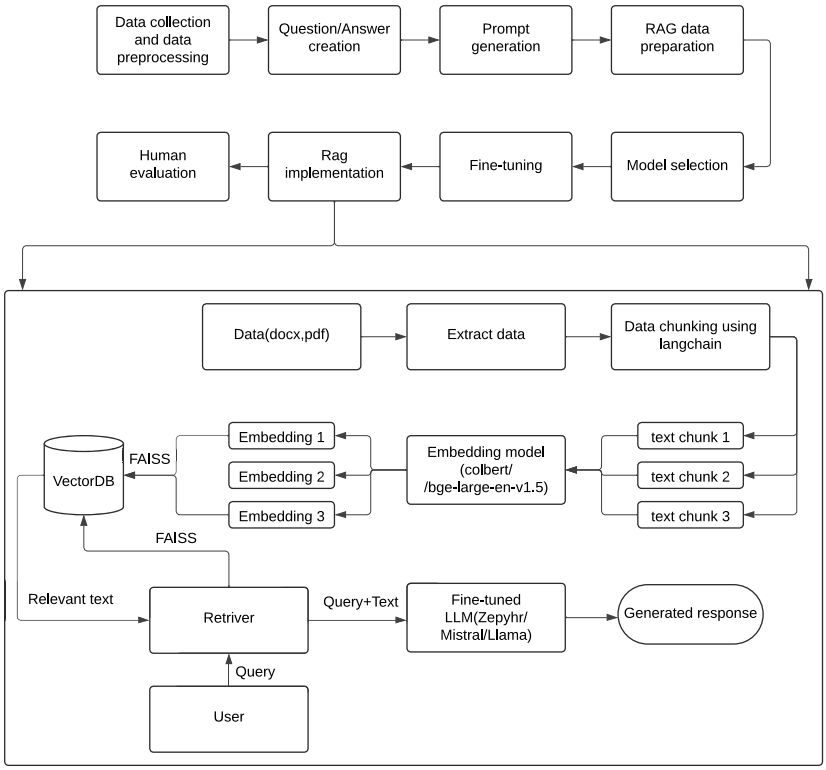


Fig. 1. Pipeline of developing LLM QA chatbot.

Table 1
Training data format.

Question	Answer
What does KUET stands for?	KUET stands for Khulna University of Engineering & Technology.
How many halls in KUET?	KUET has a total of 7 halls: 6 for male students and 1 for female students. The largest hall, Amar Ekushey Hall, accommodates 563 students.

The user can generate a new question using "Generate New Question" if the user does not want to answer current question". In the "Custom Questions" tab, the user can enter new questions and answers. Then save the answer using the "save the answer" button. All the questions and answers are saved in the "finetune_data" file in the "Training Data Generation" tab. In the "Testing Data Generation" tab, questions, ground truths and contexts are collected from the user for testing the model. "Embedding Data Generation" tab collects data to fine-tune embedding model.

In the third section, the user can add RAG data for inference, which is used as a context for LLM. Users can parse the data from the website using the parse data button. Here, the user will give the target website a link for parsing data. After clicking the parse button, it will recursively parse the data from the website and save it as a "rag_data.xlsx" or "rag_data.csv" file. This data can be used for further implementation of RAG. Fig. 3 shows the data collection tab.

2.2.2. Fine-tuning

In the fine-tuning tab, the user fine-tunes the given models using the already stored data in "data" folder. Users must provide a HuggingFace token to access the latest LLM models. There is a drop-down box to select the model. Initially, five models are given for fine-tuning: Mistral, Zepyhr, Llama-3, Phi, and Flan-T5, and a custom model is also given so that the user can give their own model. To train Mistral, Zephyr, and Llama models, you need 24 GB of VRAM and, for inference, 16 GB in 8-bit quantization [24]. We have added a quantization technique for saving GPU memory. Quantization technique to reduce computational and memory costs without compromising the performance of the model. For the Phi and Flan-t5 models, training requires 5 GB of VRAM, and inference needs 4 GB. To fine-tune a custom model, select the "custom model" option in the "Select the Model for Fine-tuning" dropdown, then configure it by editing the code section. After fine-tuning, the model will be saved in the "models" folder. Fig. 4(a) shows the fine-tuning user interface. First, we need to provide an excel file, which was previously created in the "Data Collection" tab. Fine-tuning is mainly used to learn the context of the subject. Users have the option to change the parameter and the code by clicking "Advance Code Editing". The embedding model can be fine-tuned in "Embedding model" sub-tab. Users also have the option to select embedding models and fine-tune them for specific datasets.

2.2.3. Testing data generation and RAG customization

This tab is primarily utilized for generating answers using the fine-tuned models, aiding users in evaluating the model's performance. Users can select the data chunking parameters and embedding model, and save this configuration as a favorite for deployment. To evaluate the performance of RAG, we incorporate several key metrics with human evaluation: answer correctness, answer similarity, answer relevancy, faithfulness, context recall, and context precision.

The user selects the desired fine-tuned model and clicks the "Generate Answer for Testing Dataset" button. The generated data allows the user to evaluate the model's performance in the human evaluation tab. Fig. 4(b) illustrates the user interface for generating human evaluation test data.

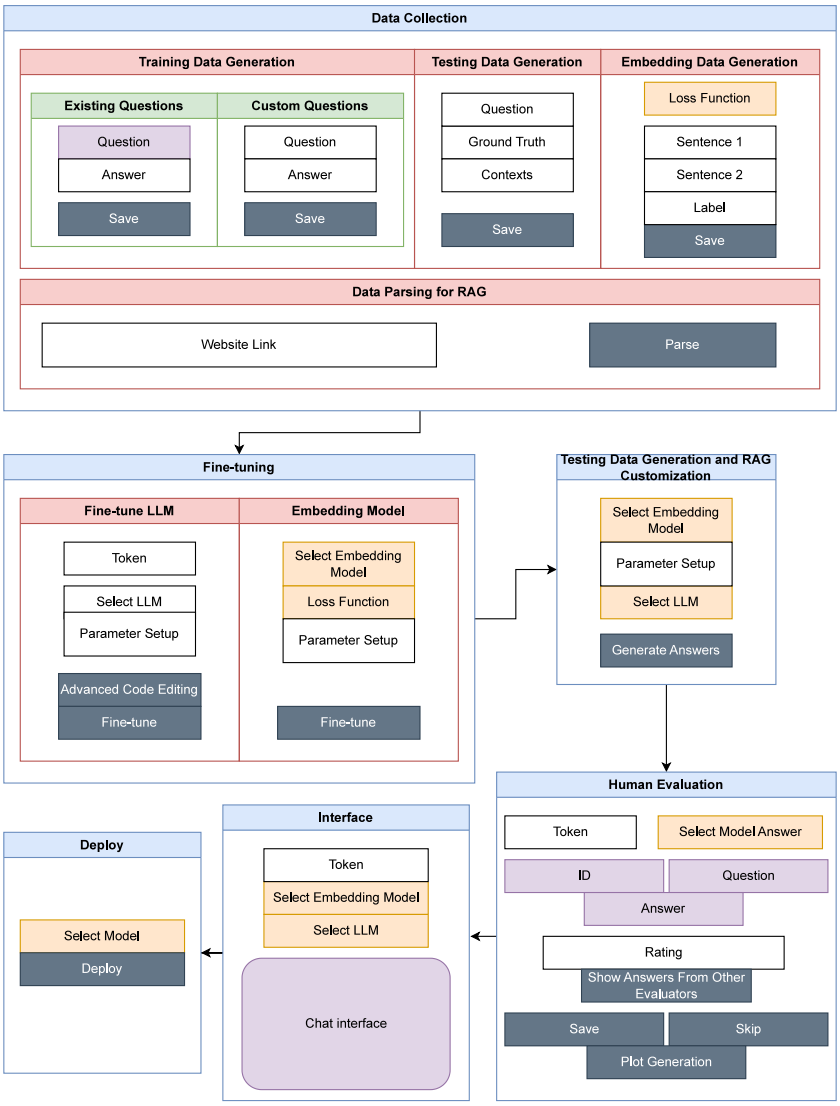


Fig. 2. Software architecture.

2.2.4. Human evaluation

The human evaluation tab is used to check the model's performance by involving human judgement. In recent works, GPT 4 [25,26] and ROUGE-L [27] has been used for evaluating model answers, but in this case, GPT 4 shows bias about answers and sometimes makes mistake without knowing context. There is poor correlation between ROUGE-L and GPT 4 [19]. To solve these issues, we have added human evaluation, where other users directly evaluate a model by rating model's answers. The rating can be an integer value within the range 1 to 5 where a higher value indicates a higher acceptance by the evaluator. First, we provide unique token to an evaluator if the evaluator is new otherwise, evaluators can resume their activity using the provided token. Then, the evaluator needs to select the desired fine-tuned model for human evaluation. In this tab, the "model name" and "number of questions" are displayed. The user rates each answer provided by the model for the questions. After clicking the "save" button, the questions, answers, and ratings are saved in an Excel file. Each evaluator's answers are saved in separate Excel files utilizing the evaluator tokens. Evaluators can pause their rating activity and resume later at their convenience. The "Show Answer from Other Evaluators" button shows the answer from the other evaluators if the answer is already provided by other evaluators. An evaluator can skip answering a question using the "Skip" button. The "Plot generation" button shows a summary of the average performance of the model based on available ratings. Fig. 5 shows the human evaluation user interface.

2.2.5. Inference

The Inference tab offers configurable options for users to customize the inference process, including selecting the embedding model, splitter type, chunk size, overlap, separator, and max tokens. Users can choose between fine-tuned or standard models and interact via a chat interface. The system leverages RAG to provide responses informed by relevant data sources. Fig. 6(a) shows the inference user interface.

2.2.6. Deployment

The user can deploy the fine-tuned model in the deployment tab. First, the user must provide the model's name. Models are saved in the "models" folder. After clicking the deploy button, all the code for deployment is saved in the "deploy" folder. The user needs to install required dependencies, which are given in the "requirement.txt" file of the "deploy" folder."pip install -r requirements.txt" command can be used to install the dependencies. After installation, the model will be deployed. Fig. 6(b) shows the user interface of the deployment tab.

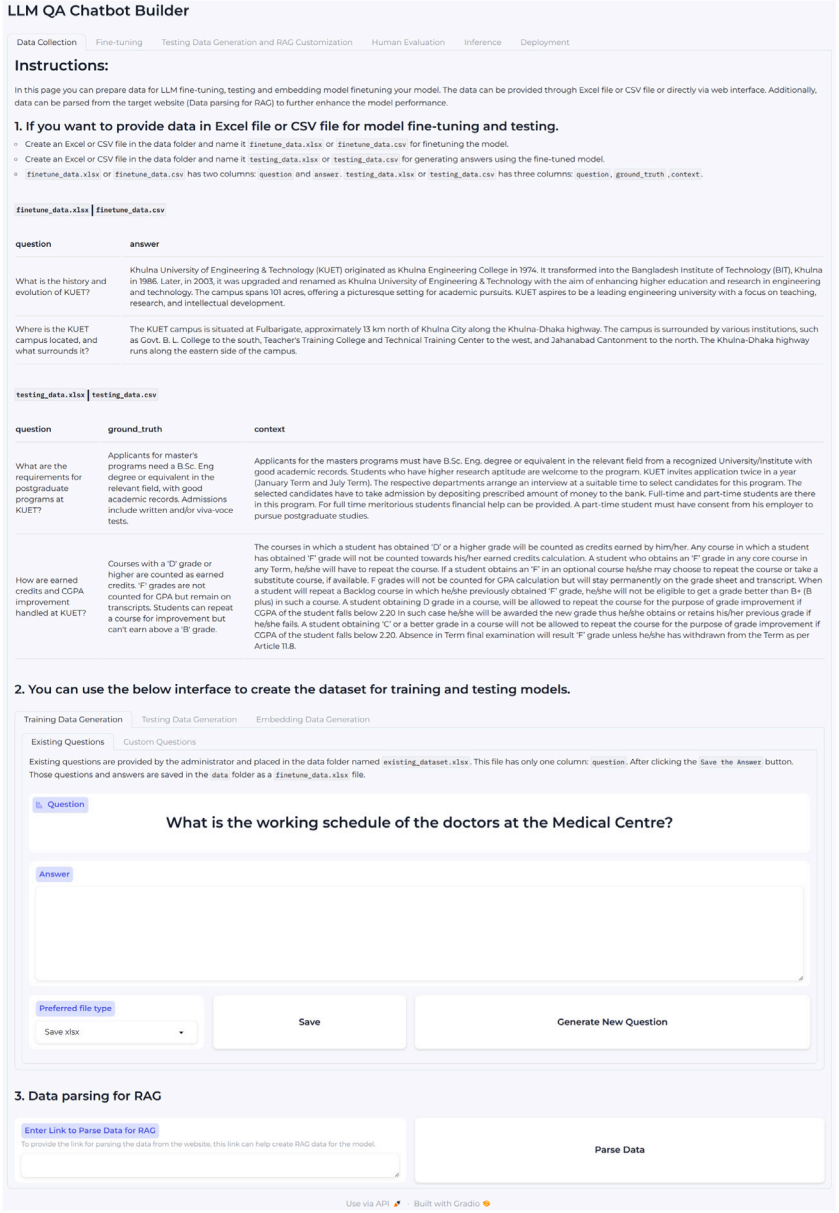


Fig. 3. Data collection.

3. Illustrative examples

A screencast titled *LLM QA Chatbot Builder.mp4*,[1] available in our code repository, provides a comprehensive demonstration of our software, showcasing the use cases of each component with practical examples.

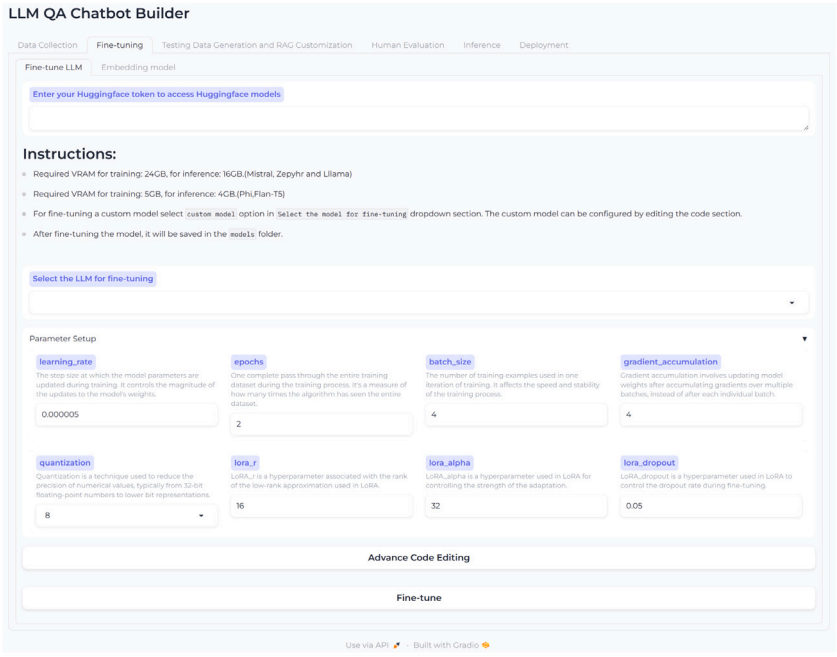
4. Impact

Small businesses and academic institutions increasingly require chatbot solutions capable of efficiently responding to user inquiries. However, developing a LLM-based QA chatbot poses significant challenges, particularly for non-technical users lacking expertise in LLMs and programming. Our software addresses this gap by enabling users to develop chatbots without the need for prior programming knowledge. Additionally, it offers a code editing option within the finetuning interface, allowing technical users to modify model hyperparameters and train custom models as needed.

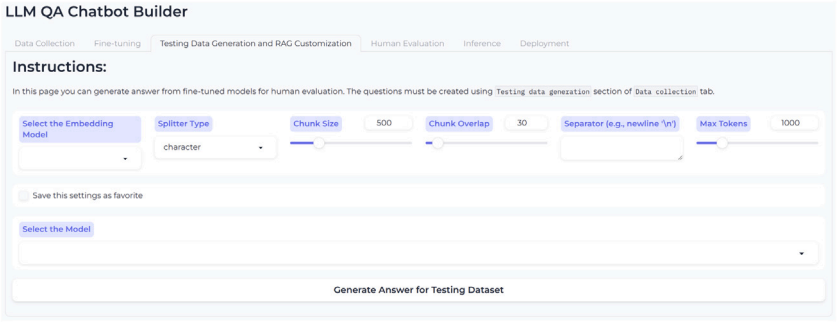
A critical barrier to the widespread adoption of LLM-based chatbots is the computational power required to train large models. Our software mitigates this issue by incorporating a quantization mechanism, which significantly reduces the computational resources required, making it feasible for users with consumer-level computers to develop robust chatbots.

The potential impacts of this tool are substantial. The most immediate benefit is the significant reduction in time required for data collection, which enhances both efficiency and consistency in model training. By standardizing the data collection process, the tool ensures that all data is uniformly formatted and structured, leading to higher-quality, more diverse datasets that improve the overall performance of the trained models. As data collection and analysis become increasingly complex, the need for reliable, adaptable, and efficient tools becomes more critical.

[1] https://github.com/shahidul034/LLM-based-QA-chatbot-builder/blob/main/LLMQAChatbotBuilder.mp4



(a) Finetuning



(b) Testing data generation from model

Fig. 4. Finetuning and testing data generation from model.

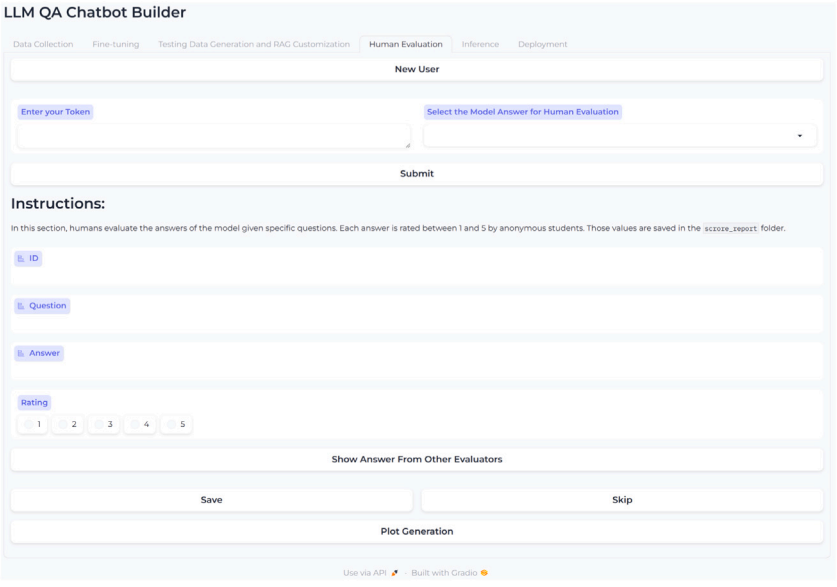
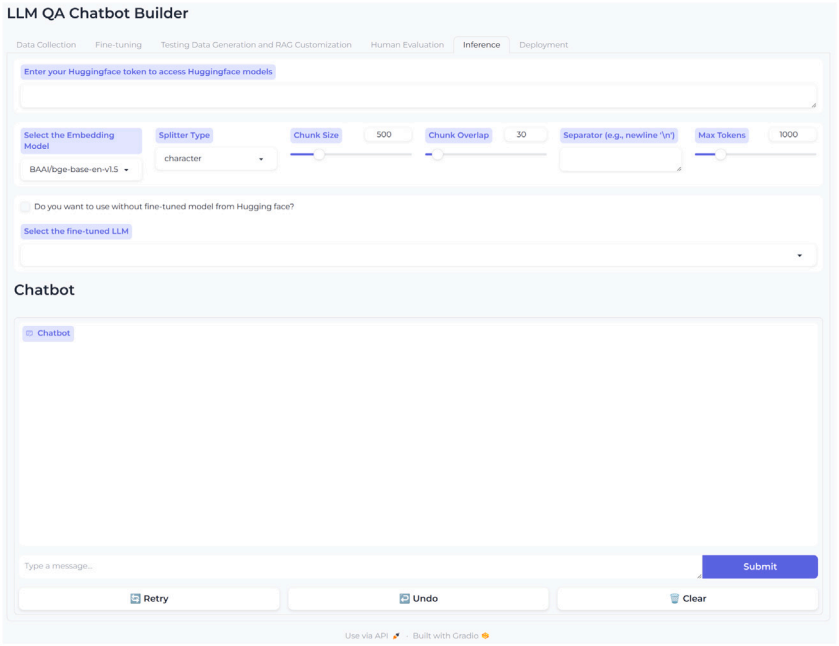


Fig. 5. Human evaluation.



(a) Inference

(b) Deployment

Fig. 6. Inference and Deployment.

Furthermore, this tool enhances research quality by providing researchers with additional time to focus on optimizing model architectures. Researchers can experiment with multiple LLM models and assess their performance, thereby developing more accurate models capable of addressing specific user queries. The inclusion of the latest open-source LLM models, known for superior instruction tuning, further boosts the tool's utility. Future updates will continue to integrate new open-source models, enabling users to seamlessly test and incorporate these advancements into their research or chatbot development projects.

5. Conclusions

The LLM-based QA chatbot builder software is designed to empower researchers and organizations with limited computational resources to develop their own LLMs using open-source libraries and models. This tool eliminates the need for paid API keys and allows users to fine-tune models without requiring technical expertise. For those with technical knowledge, the software also offers flexibility to modify code and integrate new models, enabling the creation of customized solutions beyond the pre-included options. Moreover, this software provides a comprehensive, end-to-end solution for QA model development, including testing, inference, and deployment capabilities often absent in existing software systems. As an ongoing project, the software continues to evolve, incorporating the latest high-performance models recognized on the Hugging Face leaderboard. This commitment to continuous improvement ensures that users have access to cutting-edge tools for building and refining their chatbots.

CRediT authorship contribution statement

Md. Shahidul Salim: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Sk Imran Hossain:** Writing – review & editing, Validation, Supervision, Software, Project administration. **Tanim Jalal:** Software, Methodology, Data curation. **Dhiman Kumer Bose:** Validation, Data curation. **Mohammad Jahid Ibna Basher:** Writing – review & editing.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the authors used ChatGPT in order to improve the readability and language of the manuscript. After using this service, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Dr. Sk Imran Hossain reports financial support was provided by University Grants Commission of Bangladesh. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is funded by the University Grants Commission of Bangladesh. The authors thank MST. Jesiara Khatun, MST. Sadia Islam, Omar Bin Sariffuzzaman, and the CSE 2K20 batch from Khulna University of Engineering & Technology for their assistance in evaluating the model's performance.

References

[1] Du J-l, Yu P-f. Towards natural language processing: A well-formed substring table approach to understanding garden path sentence. In: 2010 2nd international workshop on database technology and applications. 2010, p. 1–5. http://dx.doi.org/10.1109/DBTA.2010.5659102.

[2] Santana Suárez O, Carreras Riudavets FJ, Hernández Figueroa Z, González Cabrera AC. Integration of an XML electronic dictionary with linguistic tools for natural language processing. Inf Process Manage 2007;43(4):946–57. http://dx.doi.org/10.1016/j.ipm.2006.08.005, URL https://www.sciencedirect.com/science/article/pii/S0306457306001415.

[3] Métais E. Enhancing information systems management with natural language processing techniques. Data Knowl Eng 2002;41(2):247–72. http://dx.doi.org/10.1016/S0169-023X(02)00043-5, URL https://www.sciencedirect.com/science/article/pii/S0169023X02000435.

[4] Birhane A, Kasirzadeh A, Leslie D, Wachter S. Science in the age of large language models. Nat Rev Phys 2023;5:277–80. http://dx.doi.org/10.1038/s42254-023-00581-.

[5] Daungsupawong H, Wiwanitkit V. Letter: The use of large language models as medical chatbots in digestive diseases. Alimentary Pharmacol. Therapeut. 2024;60(7):971. http://dx.doi.org/10.1111/apt.18105, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/apt.18105, URL https://onlinelibrary.wiley.com/doi/abs/10.1111/apt.18105.

[6] Shang L, Lu Z, Li H. Neural responding machine for short-text conversation. In: Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: long papers). Beijing, China: Association for Computational Linguistics; 2015, p. 1577–86. http://dx.doi.org/10.3115/v1/P15-1152, URL https://aclanthology.org/P15-1152.

[7] Chandwani V, Kumar S, Singh PK. Long short-term memory based conversation modelling. In: 2020 3rd international conference on emerging technologies in computer engineering: machine learning and internet of things. ICETCE, 2020, p. 105–9. http://dx.doi.org/10.1109/ICETCE48199.2020.9091753.

[8] Yin J, Jiang X, Lu Z, Shang L, Li H, Li X. Neural generative question answering. In: Proceedings of the workshop on human-computer question answering. San Diego, California: Association for Computational Linguistics; 2016, p. 36–42. http://dx.doi.org/10.18653/v1/W16-0106, URL https://aclanthology.org/W16-0106.

[9] Su P-H, Mrkšić N, Casanueva I, Vulić I. Deep learning for conversational AI. In: Proceedings of the 2018 conference of the North American chapter of the association for computational linguistics: tutorial abstracts. New Orleans, Louisiana: Association for Computational Linguistics; 2018, p. 27–32. http://dx.doi.org/10.18653/v1/N18-6006, URL https://aclanthology.org/N18-6006.

[10] Vinyals O, Le Q. A neural conversational model. 2015, http://dx.doi.org/10.48550/arXiv.1506.05869, arXiv:1506.05869, URL https://arxiv.org/abs/1506.05869.

[11] Tunstall L, Beeching E, Lambert N, Rajani N, Rasul K, Belkada Y, et al. Zephyr: Direct distillation of LM alignment. 2023, http://dx.doi.org/10.18653/v1/N18-6006, arXiv:2310.16944.

[12] Jiang AQ, Sablayrolles A, Mensch A, Bamford C, Chaplot DS, de las Casas D, et al. Mistral 7B. 2023, http://dx.doi.org/10.48550/arXiv.2310.06825, arXiv:2310.06825, URL https://arxiv.org/abs/2310.06825.

[13] Grattafiori A, Dubey A, Jauhri A, Pandey A, Kadian A, Al-Dahle A, et al. The llama 3 herd of models. 2024, http://dx.doi.org/10.48550/arXiv.2407.21783, arXiv:2407.21783, URL https://arxiv.org/abs/2407.21783.

[14] Abdin M, Aneja J, Awadalla H, Awadallah A, Awan AA, Bach N, et al. Phi-3 technical report: A highly capable language model locally on your phone. 2024, http://dx.doi.org/10.48550/arXiv.2404.14219, arXiv:2404.14219, URL https://arxiv.org/abs/2404.14219.

[15] Chung HW, Hou L, Longpre S, Zoph B, Tay Y, Fedus W, et al. Scaling instruction-finetuned language models. 2022, http://dx.doi.org/10.48550/arXiv.2210.11416, arXiv:2210.11416, URL https://arxiv.org/abs/2210.11416.

[16] Stureborg R, Alikaniotis D, Suhara Y. Large language models are inconsistent and biased evaluators. 2024, http://dx.doi.org/10.48550/arXiv.2405.01724, arXiv:2405.01724, URL https://arxiv.org/abs/2405.01724.

[17] Edward. Verba. 2024, https://github.com/weaviate/Verba.

[18] Thakur A. Autotrain-advanced. 2024, https://github.com/huggingface/autotrain-advanced.

[19] Yu X, Lu Y, Yu Z. LocalRQA: From generating data to locally training, testing, and deploying retrieval-augmented QA systems. In: Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 3: system demonstrations). Bangkok, Thailand: Association for Computational Linguistics; 2024, p. 136–51. http://dx.doi.org/10.18653/v1/2024.acl-demos.14, URL https://aclanthology.org/2024.acl-demos.14.

[20] Langchain AI Developers, A framework for developing applications powered by large language models (LLMs), URL https://github.com/langchain-ai/langchain, Accessed: June 11, 2024, ongoing.

[21] Xiao S, Liu Z, Zhang P, Muennighoff N, Lian D, Nie J-Y. C-pack: Packed resources for general Chinese embeddings. In: Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval. SIGIR '24, New York, NY, USA: Association for Computing Machinery; 2024, p. 641–9. http://dx.doi.org/10.1145/3626772.3657878.

[22] Khattab O, Zaharia M. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In: Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval. SIGIR '20, New York, NY, USA: Association for Computing Machinery; 2020, p. 39–48. http://dx.doi.org/10.1145/3397271.3401075.

[23] Chroma Developers, Chroma: The Open-Source Embedding Database, URL https://github.com/chroma-core/chroma, Accessed: June 11, 2024, ongoing.

[24] Jacob B, Kligys S, Chen B, Zhu M, Tang M, Howard A, et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: 2018 IEEE/CVF conference on computer vision and pattern recognition. 2018, p. 2704–13. http://dx.doi.org/10.1109/CVPR.2018.00286.

[25] Zheng L, Chiang W-L, Sheng Y, Zhuang S, Wu Z, Zhuang Y, et al. Judging LLM-as-a-judge with MT-bench and chatbot arena. 2023, http://dx.doi.org/10.48550/arXiv.2306.05685, arXiv:2306.05685, URL https://arxiv.org/abs/2306.05685.

[26] Liu Y, Iter D, Xu Y, Wang S, Xu R, Zhu C. G-eval: NLG evaluation using gpt-4 with better human alignment. In: Proceedings of the 2023 conference on empirical methods in natural language processing. Singapore: Association for Computational Linguistics; 2023, p. 2511–22. http://dx.doi.org/10.18653/v1/2023.emnlp-main.153, URL https://aclanthology.org/2023.emnlp-main.153.

[27] Lin C-Y. ROUGE: A package for automatic evaluation of summaries. In: Text summarization branches out. Barcelona, Spain: Association for Computational Linguistics; 2004, p. 74–81, URL https://aclanthology.org/W04-1013.