iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 10 **Issue:** VII **Month of publication:** July 2022

DOI: https://doi.org/10.22214/ijraset.2022.45891

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue VII July 2022- Available at www.ijraset.com

Chatbot Web Application using DIET

Arvind G[1], Anchith Acharya U[2], Abhinandan G D[3], Adhish N[4], Dr. B. N. Shankar Gowda[5]
[1, 2, 3, 4]*Student,* [5]*Associate Professor, Bangalore Institute of Technology, Dept. of Computer Science and Engineering, Bangalore-560004, Karnataka, India*

Abstract: ***A chatbot is a computer program that is capable of communicating with the user in natural language in order to understand and parse the technical query. Such a program is frequently employed by organizations in order to automate the task of customer care and services. We have focused on designing a chatbot for our college, the Bangalore Institute of Technology. The current chatbot that is present in the official website for BIT is only an interface connecting the user to a live agent and is therefore unavailable for use most of the time. This inspired us to design a much better alternative for the existing chatbot using the DIET algorithm, that is accessible all the time and provides accurate responses in the blink of an eye.***
Keywords: Chatbot, DIET Algorithm, Rasa framework, Intent classification, Entity extraction

I. INTRODUCTION

A chatbot (derived from "chat robot") is a software program that simulates human conversation, either via voice or text communication. Also known technically as conversational interfaces, chatbots present a new way for individuals to interact with computer systems.

Traditionally, to get a question answered by a software program involves using a search engine or filling out a form. A chatbot allows a user to simply ask questions in the same manner that they would address a human. The most well-known chatbots currently are seen by us every day on our smartphones: Alexa, Google Assistant and Siri.

Organizations use chatbots to engage with customers alongside the classic customer service channels of phone, email, and social media. Chatbots provide the organization the benefit of answering trivial and repetitive queries of the customer without the need of any human intervention whatsoever. It also aids the user in navigating the services provided by the organization.

BIT Chatbot is an intelligent, semi-automated chatbot service that not only aims to assist users in navigating the official website of the Bangalore Institute of Technology but is also able to hold fluent conversations with the user, in natural language. It serves as a highly intuitive method of gaining various information about the institute, such as the departments, courses, etc. Further, it also allows a student to access their personal information such as marks and attendance and manage placement activities by logging-in to the application.

II. RELATED WORKS

Since Markov chains have been successfully used for web request reformulation via absorbing stochastic process (Wang, Huang, and Wu 2015), and modelling query utility (Jansen, Booth, and Spink 2005), demonstrating that Markov processes can be used to explain the user web request behaviour (Xiaofei Zhu 2012). Here, we introduce a novel Markov chain approach to query reformulation that is both extremely scalable and interpretable thanks to simple definitions of transition probabilities. The Markov chain is employed in this study for the first time, to the best of author' knowledge, for query reformulating in speech virtual assistants.

One significant distinction between the reformulation of a web inquiry and Alexa's use case is the requirement that the user's utterance be smoothly replaced in order to reduce friction.

Every time we want to reformulate, asking users for their approval is a friction point that we attempt to minimise. Another distinction is how a user's engagement with a voice-based chatbots system defines success and failure. When dealing with Alexa, we use both implicit and explicit customer feedback to determine the absorption stages of processes and outcomes.

Markov-based algorithm daily consumes the analysed Alexa log data to learn from users' reformulations and refreshes the above online database in order to get new rewrite options and maintain the feasibility of existing rewrites. In following sections of this work, we go over the characteristics of the data and also how our model accomplishes this.

The entire ingestion to update process occurs offline, and the low-maintenance feature-toggling (also known as feature-flag) approach is used to update the rewrites in the database. Furthermore, we also have an asynchronous blacklisting method that assesses the redesigns from our Markov chain by separately measuring their friction ratio against that of the initial utterance, and should they score worse against that, filters them out of being uploaded to the database.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue VII July 2022- Available at www.ijraset.com

III. PROPOSED SYSTEM

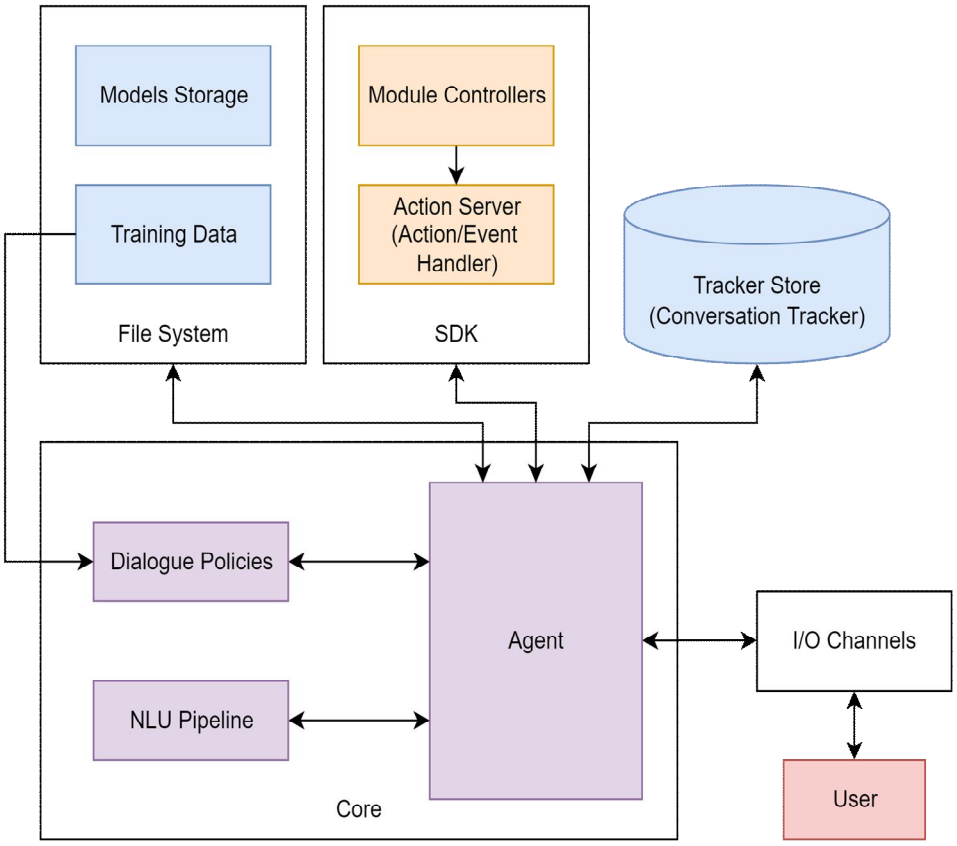


Figure 3.1: Architecture of the proposed system

Rasa is an open-source machine learning framework to automate text and voice-based conversations. Using Rasa, one can build contextual assistants capable of having layered conversations with lots of back-and-forth. DIET is a multi-task transformer architecture that handles both intent classification and entity recognition together. It provides the ability to plug and play various pre-trained embeddings like BERT, GloVe, ConveRT, and so on. Large-scale pre-trained language models aren't ideal for developers building conversational AI applications. DIET is different because it:

1) Is a modular architecture that fits into a typical software development workflow
2) Parallels large-scale pre-trained language models in accuracy and performance
3) Improves upon current state of the art and is 6X faster to train

The chatbot is used to assist in navigating the website and also in performing various user-specific operations. The chatbot issues queries to the data repository in order to obtain data that is needed to fulfill the request. This data can be processed by the other modules too. The chatbot may defer the control to the other modules as part of a response if the request demands it. The chatbot consists of a core NLU component that communicates with the Filesystem, SDK, and the Tracker Store. The core component consists of the following:

a) *Agent:* The central component which coordinates all other activities and operations.
b) *NLU Pipeline:* A set of algorithms and NLU operations used to handle intent classification, entity extraction, and response retrieval.
c) *Dialogue Policy:* A set of rules specifying how the Agent must respond to specific inputs.

The user communicates with the Agent through I/O channels. The trained chatbot models are stored in the File System, along with the data used to train the models. Dynamic responses can be generated by the Agent by referring to the Action/Event Handlers. The controllers of individual modules communicate with these Handlers to generate a suitable response to the request. These interactions are specified through the SDK, which is also capable of interacting with the conversation tracker. The Tracker Store keeps track of previous conversations which can later be used for analysis/further predictions.

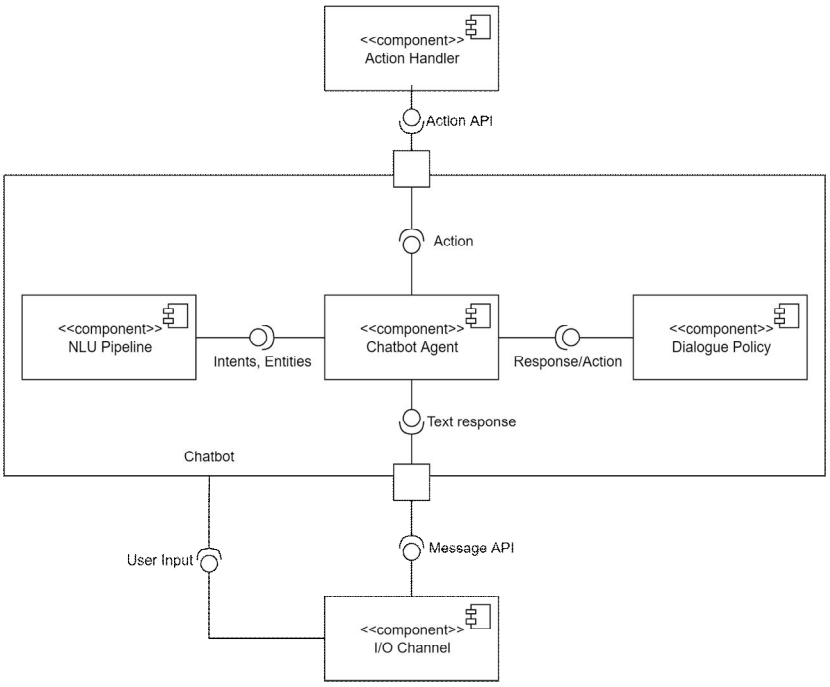


Figure 3.2: Component diagram of the proposed system

The components diagram of the chatbot is shown above. The chatbot agent exposes a message API through which it can send and receive messages that act as inputs and outputs of the chatbot. Whenever the user types the input query through whatever I/O channel is in use (which can be through a website client, native desktop client, mobile application client, etc.), the message is passed to all the components of the chatbot, which are: NLU Pipeline, Chatbot Agent and Dialogue Policy.

The Chatbot Agent is the main coordinator of all tasks of the chatbot. The NLU Pipeline performs the required NLU operations on the message received and detects a set of intents and entities. These intents and entities are then passed to the Agent.

The Chatbot Agent then consults the Dialogue Policy to determine what action should be taken next, given the extracted intents and entities. If the action is a simple response, then the response is fetched from the Dialogue Policy and returned to the user through the message API. If the action to be taken is otherwise a complex procedure that is specified by the developer, then the Agent forwards the intents and entities to the Action Handler.

The Action Handler exposes an API through which the developer can insert their own logic on how to generate a response for a particular query. This custom action is then executed, and the resulting response is returned back to the Agent, which forwards it to the user.

IV. ALGORITHM DESIGN

A. *Intent Classification*

1) *Input Embedding:* The first step is feeding out input into a word embedding layer. A word embedding layer can be thought of as a lookup table to grab a learned vector representation of each word.

2) *Positional Encoding:* The next step is to inject positional information into the embeddings.

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/dmodel})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/dmodel})$$

3) *Scaling Down the Soft Score value:*Then we take the attention weights and multiply it by the value vector to get an output vector. The higher SoftMax scores will keep the value of words the model learns is more important.

$$SoftMax(x_i) = \exp(x_i)/sum\ \exp(x_j)$$

4) *Linear Classifier Using Cosine Similarity:* Response = cosineSimilarity(xi,intents)

```
if response
Return intentIdentifier(response)
else
Return fallBackIntent()
```

B. Entity Extraction

1) *Converting to lowercase:* Here we use inbuilt lower() method to convert string/query into lower case for processing: query.lower()
2) *Tokenizing using NLTK'S tokenizer:* Tokenizers divide strings into lists of substrings. For example, tokenizers can be used to find the words and punctuation in a string:
   word_tokenize(query) Tokenization is done so that the NLU operations can be carried out individually on each token so that entities can be extracted from the query.
3) *Removing Punctuations:* After tokenization we have to filter punctuations using nltk regular expression tokenizer, where we specify the specific punctuations in addition that we want to filter
   Call nltk.RegexpTokenizer(pattern) with pattern as r"\w+" to filter a tokenizer that uses pattern to remove punctuations.
   nltk.RegexpTokenizer(query)
4) *Remove Stop Words:* First, we store a list of words that are considered as stop words. NLTK has a list of stop words inbuilt for each language
   stopwords = set(setOfStopWords)
   while (w <= query.split()) != null
   if check stop(w) remove(w)
   end StopWordRemoval
5) *Lemmatization:* Here we group together the different inflected forms of a word so they can be analysed as a single item. We use WordNet for lemmatization where words are lemmatized by WordNetLemmatizer.lemmatize(listOfWords)

V. EXPERIMENTAL RESULTS

Sequence learning and neural machine translation tasks have been built using sequence-to-sequence (seq2seq) architectures. As a result, we used a Long Short-Term Memory (LSTM) architecture as an alternate technique to produce rewrites by phrasing the job of query writing as an expansion of sequence learning. In other words, we initially used a rephrase detection ML model to mine three months' worth of rephrase data so that the original utterance was flawed, and the phrase was successful. The seq2seq model was then trained using these data so that, given a first utterance, it would generate the second utterance.

The data used to further train and fine-tune the NLU model of the chatbot to make it behave in the desired way for interaction with our target audience, was specified in a file using the YAML format. Initially, the intents to be recognized were listed, with several natural language examples provided for each intent. Entities in these examples were marked/annotated, and their corresponding roles, names and synonymic forms were specified too. Next, the responses for each of these intents were created, with multiple variations to be chosen randomly from. Finally, rules were created in order to associate each intent with its corresponding response. In case of custom actions, the logic was implemented using the Rasa SDK and then the actions were linked to the corresponding intent using a rule.

Table 4.1: Tabular results detailing the variation of training and testing time, and accuracy with change in the number of training examples for each intent.

Sl no.	No. of training sentences per intent	Training time	Intent recognition time	No. of test passes	No. of test fails	Accuracy
1	3	~2 mins	1.18 ms	8	12	40%
2	5	~3 mins	0.72 ms	10	10	50%
3	8	~5 mins	0.44 ms	16	4	80%
4	10	~6 mins	0.32 ms	17	3	85%
5	12	~9 mins	0.31 ms	15	5	75%

Overall, each intent contained an average of 9 examples. This count was further increased dramatically whenever an example made use of an entity with synonyms. In such cases, the intent would consist of around 15-20 examples. With this count of examples, the algorithm was able to successfully interpolate the query between the provided examples and understand the intent as well as extract the entity.

Figure 4.1: Graph showing the variance of the number of training sentences vs. training time of the model, showing linear relationship.

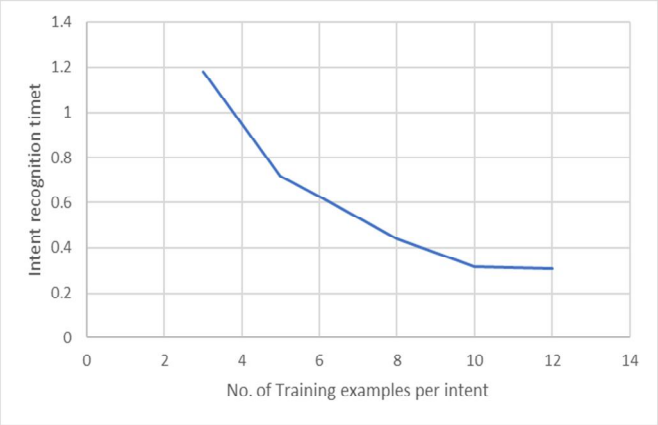


Figure 4.2: Graph showing the variance of the number of training sentences vs. time taken to recognize the query intent. The relationship is linear up to a certain point, after which, increase in training sentences results in minimal increase in recognition time.

As we can see from the above graphs, the number of training examples provided for each intent leads to a direct increase in the time taken to train the model. However, it also results in a quicker intent recognition time, as well as an increase in accuracy.

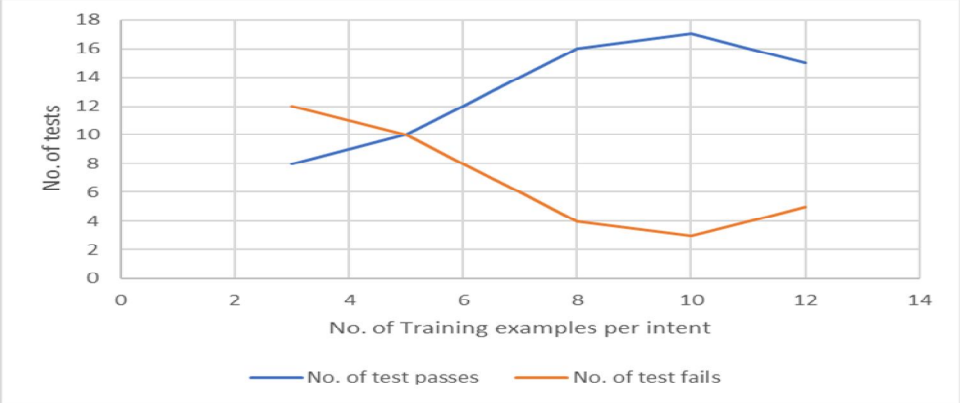


Figure 4.3: Graph plotting the number of training sentences against the test results (pass/fail). Increasing the no. of examples leads to increase pass cases until it reaches the saturation point at 10 examples.

Figure 4.4: Graph plotting the number of training sentences against the overall accuracy of the model

We considered 20 intents for testing the model. From the above graphs we observe how the accuracy of the model increases proportionally to the number of training examples per intent. However, after crossing 10 examples per intent, the decrease in intent recognition time is diminishing, and the accuracy also suffers a major drop due to overfitting. But at the same time, the training time will be increased considerably when more examples are provided. Therefore, the optimal number of training examples, which was used in our model, was an average of 9 training sentences for each intent.

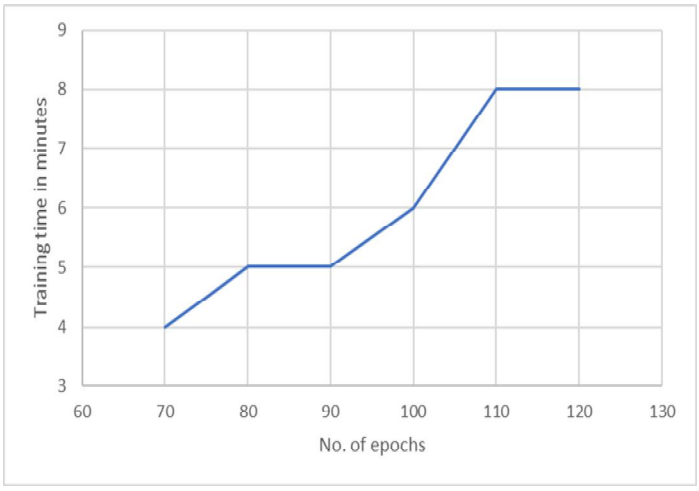


Figure 4.5: Graph plotting the number of epochs for the DIET Classifier vs. the training time of the model, in minutes.

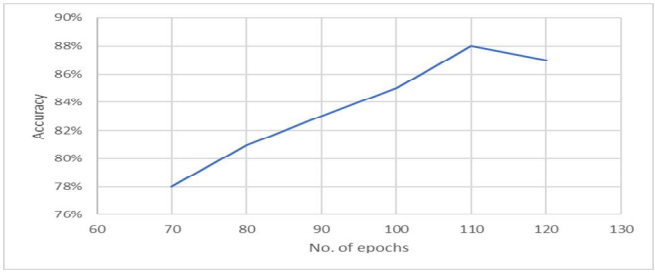


Figure 4.6: Graph plotting the number of epochs for the DIET Classifier vs. the accuracy of the model. The accuracy increases linearly with the epoch count, until it gets saturated at 110 epochs; then decreases

From graphs 4.5 and 4.6, we can see that the trend between the number of epochs used for training and the training time (considering 10 example sentences per intent) as well as the accuracy of the model, peaks at about 110 epochs. However, we chose the default value of 100 epochs for our model to cut down on the training time during development.

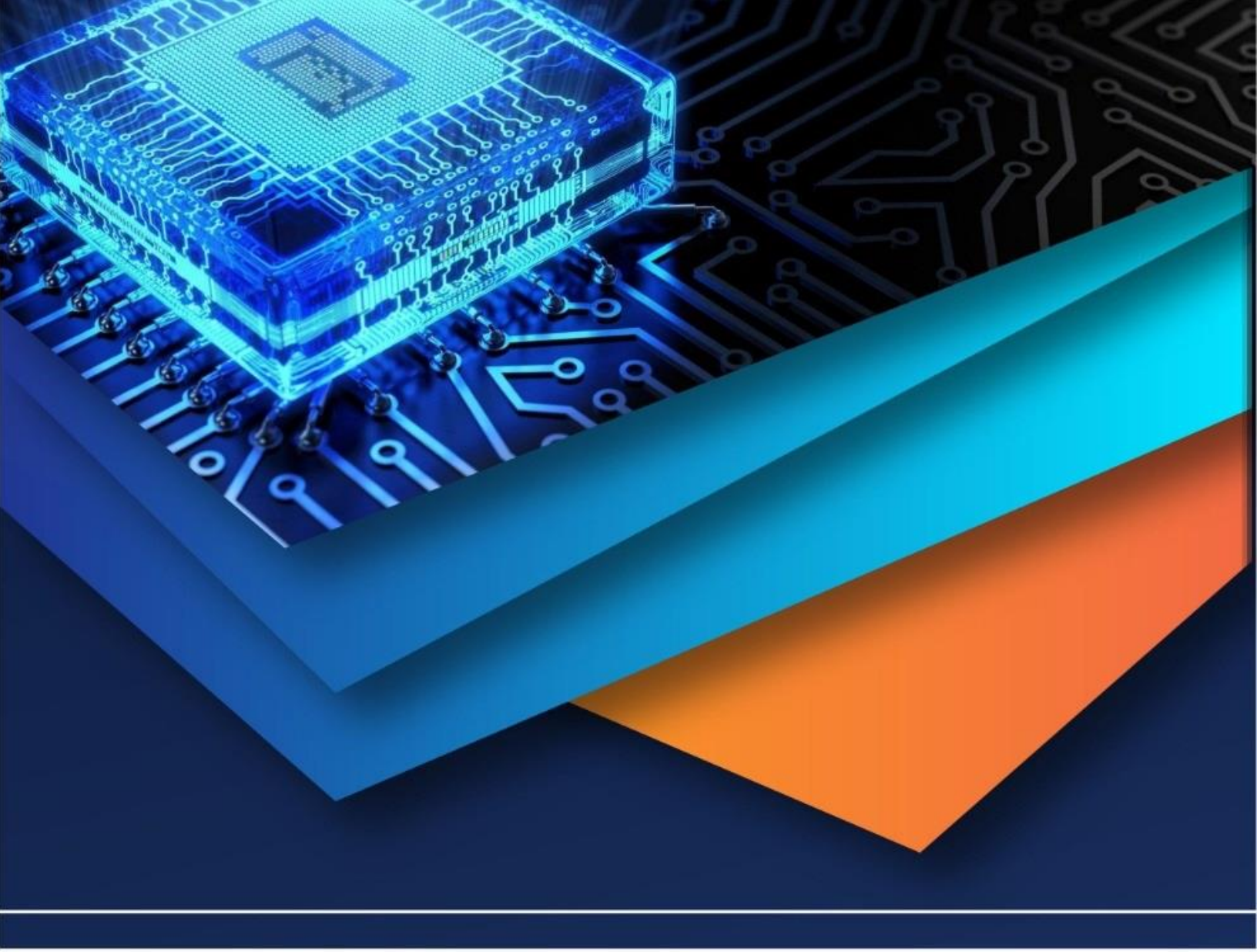
VI. CONCLUSIONS

We presented a novel architecture in this research that can be applied to retrieval-based chatbots. The new architecture considers contexts from the training dataset that could be potential matches for the contender response and analyzes them to the present context when predicting the degree to which a contender answer and a conversation scenario match. The model enhances the BERT Bi-Encoder framework while barely slowing down inference. The output trade-off between the aforementioned designs is outlined in this section.

It is crucial for these systems to include self-learning methods to solve the reoccurring problems consistently with little human interaction as conversational agents gain popularity and expand into other domains. In this study, we introduced a self-learning system that, by query reformulation, effectively targets and corrects both systemic and user problems during runtime. To surface effective phrase reformulations in chatbot agents, we specifically presented a highly scalable collaborative-filtering approach based on an absorptive Markov chain. In order to deliver the rewrites in a look-up manner online without negatively affecting users' perceived latency, our system aggregates large amounts of pass data in an offline manner. Our system has been evaluated and put into use.

REFERENCES

[1] M. M. Khan, "Development of An e-commerce Sales Chatbot," in 2020 IEEE 17th International Conference on Smart Communities: Improving Quality of Life Using ICT, IoT and AI (HONET), 2020, pp. 173-176
[2] V. Subramanian, N. Ramachandra and N. Dubash, "TutorBot: Contextual Learning Guide for Software Engineers," in 2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE), 2019, pp. 16-17
[3] S. Tadvi, S. Rangari and A. Rohe, "HR Based Interactive Chat bot (PowerBot)," in 2020 International Conference on Computer Science, Engineering and Applications (ICCSEA), 2020, pp. 1-6
[4] Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, Asli Celikyilmaz, “End-to-End Task-Completion Neural Dialogue Systems”, Feb 2018, arXiv:1703.01008
[5] Zi Yin, Keng-hao Chang, Ruofei Zhang, “DeepProbe: Information Directed Sequence Understanding and Chatbot Design via Recurrent Neural Networks”, 13 August 2017, arXiv:1707.05470
[6] Y. Nishihara, M. Ikuta, R. Yamanishi and J. Fukumoto, "A Generation Method of Back-Channel Response to Let a Chatting Bot Be a Member of Discussions in a Text-Based Chat," in 2017 6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI), 2017, pp. 324-329
[7] S. Meshram, N. Naik, M. VR, T. More and S. Kharche, "College Enquiry Chatbot using Rasa Framework," in 2021 Asian Conference on Innovation in Technology (ASIANCON), 2021, pp. 1-8
[8] Amir Ali, Muhammad Zain Amin, “Conversational AI Chatbot Based on Encoder-Decoder Architectures with Attention Mechanism”, Aug 2017, DOI:10.13140/RG.2.2.12710.27204
[9] Vladimir Vlasov, Johannes E. M. Mosig, and Alan Nichol, “Dialogue Transformers”, 1st May 2020, arXiv:1910.00486
[10] Tanja Bunk, Daksh Varshneya, Vladimir Vlasov, Alan Nichol, “DIET: Lightweight Language Understanding for Dialogue Systems”, 11th May 2020, arXiv:2004.09936



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY