Building a Chatbot to Adopt an Effective Learning Strategy for Graduate Courses in Computer Science

Alex Elentukh(✉), Shravan Motipally, Jacob Kustra, and Toby March

Metropolitan College of Boston University, 1010 Commonwealth Ave., Boston, MA 02215, US
{Elentukh,Shravanm,JKustra,TVMarch}@bu.edu

Abstract. Courses offered in a Computer Science program of a large university differ greatly in their structure. Such a variety is driven by the diversity of backgrounds of faculty members, as well as by the breadth and depth of topics covered. The pragmatic goal of our research is to help students to adopt an effective learning strategy while engaging in a graduate program of several dozen very different courses. As a first step, we established a database of frequently asked questions regarding class logistics. Building on this foundation we launched the chatbot to be readily available to all program registrants. This paper can serve as a step-by-step introduction to How-To-Build-A-Chatbot while dealing with challenges and successes encountered. A student is able to quickly browse through relevant questions and responses. Additionally, a student can actively participate in a conversation with the bot, select among five language models, adjust semantic matching, and switch on generative mode. We used Ragas framework to formally evaluate and measure the performance of the bot, while focusing on its ability to simulate the human-like interaction. The experimental nature of exchange with the bot served as a strong motivation for students to keep exploring and learning. The evolution from a list-bound bot design into a more fluid approach involving language technologies is not trivial. As a benefit, we were able to offer students the personalized and engaging learning experience.

Keywords: computer science education · software engineering · teaching and learning effectiveness · language models · chatbots · prompt engineering · retrieval augmented generation

1 Introduction

We can theorize about artificial intelligence destroying humanity [1, 2] or, alternatively, we can host a useful application responding to actual questions – easily accessible by a multitude of real people. Our clear direction is toward the second goal.

QBot application has been built by a team of students as a term project during two twelve-weeks courses titled “Software Quality Management” [3]. The app includes all attributes of a real software product as it has been actively used by students of several

T. Zlateva and G. Tuparov (Eds.): CSECS 2024, LNICST 609, pp. 276–296, 2025.
https://doi.org/10.1007/978-3-031-84312-9_19

consecutive running of this course. The relevance of QBot is due to the fact that its users are the same students who built it. QBot is a live application readily available at, https://qbot-slak.onrender.com/. Figure 1 shows the QBot home page. The name of the course is at the top of the left pane. If you ask a question, QBot will instantly respond.

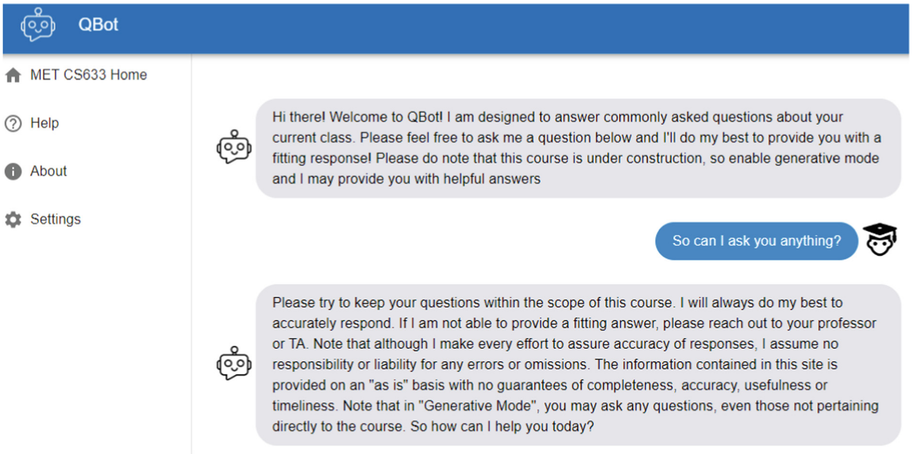

Fig. 1. Home Page of QBot.

Settings Menu is shown at Fig. 2, allows to select among various language models and to adjust models' parameters. Generative mode is defaulted to gpt3.5-turbo model.

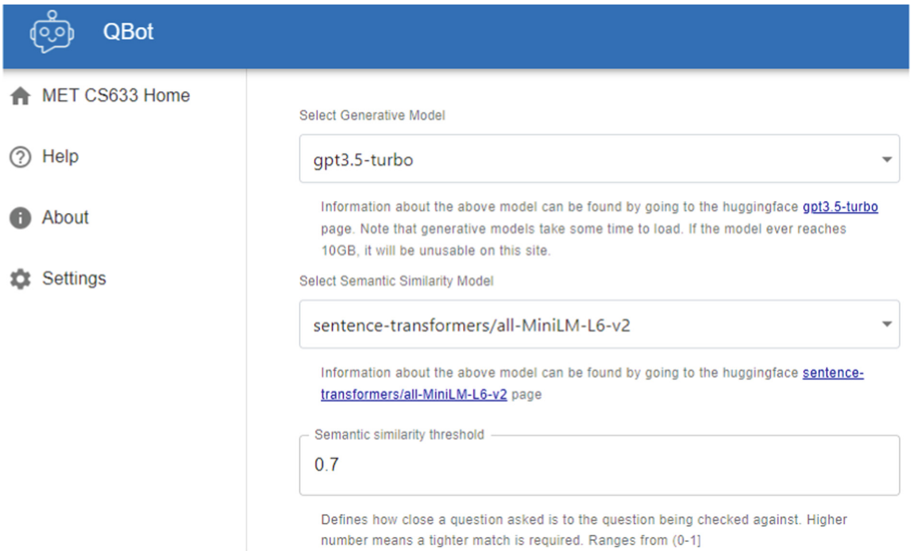

Fig. 2. Screen with Settings for model tailoring.

Semantic similarity threshold (as opposed to direct or lexicographical similarity) is manually tailored for each embedding model - to define the proximity toward the so-called "ground truth", which is the set of predefined responses. Students use QBot to navigate their personal strategy toward the most effective path of learning.

2 Examples of Questions and Responses Predefined in QBot

Literature review offers a broad range of chatbot applications, [4–6]. The focus of our QBot is on logistics and mechanics of taking a class. Consider that logistics of each class is naturally influenced by the class's body of knowledge. In other words, if we are to draw a Venn Diagram between Logistics and Content, both circles will have a significant overlap. QBot literally responds to the question, "What is the effective way to learn?" It gives the following plausible response, "Good question. First point is that the amount of 'learning' will be proportional to the amount 'effort' you invest. Such concept is opposite to a common thinking that 'someone else must teach me'. The second point is about importance to strategize and do things in a certain order. For example, if you assume that you can pick up all topics during the night before final exam, such strategy might prove flawed. Many students treat this class as a project with all its activities being well-scheduled."

Here are another two frequent questions along with their responses.

Question, "Are there any constraints in selecting a topic for a term project?". Response, "Your grade is the constrain. Projects that will be used by students of future classes - will receive the top grade. Projects that are little to do with software engineering and do not contribute to the scope of this class - will lose points. To develop this thought further, the project HoneyLemonMacroons has little to do with software engineering. On another hand, there is an actual bakery in Texas that is using the app developed here. The project Fashion Portal has a remarkable UI. Someone in California has a successful small business of selling clothes, while using this app. Any project that instigated a real business will receive additional points."

Question, "What is the common misconception about a term project?". Response, "Projects in this class are done by real students, who learn things as they go along, often learn through committing mistakes. Some parts of project reports are good and other are not as good. Although looking at reports for the first time a student might not be able to distinguish the good from the bad, which is an apparent difficulty of this learning method."

All questions posed by students are ranked by the frequency of how often they are being asked. The following question always bubbles up to the top, "Is this class difficult or easy?". The standard QBot response is as follows, "The class covers multiple topics. It is rewarding for a student to learn new material and be able to apply it. Six modules include over a thousand pages of course notes, quizzes, assignments, and term project."

The fact that this is the most frequently asked question offers an insight, into the state of mind of a student. In order to respond to such question, one had to delve deeper, "What actually are they asking?" Responding to questions literally would not be sufficient. Students feel insecure in front of a new material that they have to absorb. Students are asking for an encouragement to alleviate their natural anxiety, and such an encouragement must be based on real facts about the course.

Another common question is as follows, "One member of our team does not contribute. What do we do?" Apparently, some students feel that team dynamic is fundamentally unfair. Since all members share the same grade, although some students work so much harder than others. Professor has to explain that if a team member does not contribute then other members have to complete his/her tasks. There is no alternative way to do it. Even if we decrease the grade of a free-rider – this will not help a team to complete a project. The notion that, "University projects are designed to simulate a real-life scenario" – puts events in perspective. A team of a software project is a wrong place to look for fairness in distribution of tasks. You just have to do your personal best at whatever you do. It is hardly helpful to dwell on thinking that someone on your team did less than you are.

The following example highlights the differences between logistics of various courses, which makes it imperative for a student to learn the "culture" of each course.While dealing with quizzes most professors rely on LMS (Learning Management System) Blackboard, where questions and grades are administered automatically. It is a vivid "sign of times" that evolution of technology led many professors to abandon LMS and start using paper copies of each quiz to minimize the cheating. Paper quizzes are administered in the class for all students at once, which eliminates the possibility for several students to cooperate on this inherently-individual activity. Otherwise one student can do the initial part of a quiz, while another student does the later part, so then to exchange and combine the result of their "collaboration".

Delving into dynamics of interactions among students does open the complexity which needs to be addressed. It is a mistake to disregard this complexity. A student entering a class needs to quickly understand the rules of engagement and find the way to succeed. Our QBot offers a path for a student, which has proven to be quite effective.

3 Three Personas and Use Cases

Figure 3 shows the interrelation among three defined personas along with corresponding use cases. Addition of Account Admin increased greatly the system complexity, since each class needs a distinct space in MongoDB with questions and course material.

Definition of personas and use cases was the prudent initial step of our project, as it laid foundation for all future efforts.

4 Evolution of QBot Design – Implementation Perspective

While building QBot (and trying to use it at the same time) we implemented several significant enhancements. In effect, we had a "production environment" where real customers used the product and fed their suggestions back to the development team. The goal of these multiple improvements to QBot has been to keep refocusing on the task at hand, namely, on guiding students toward the most effective way to learn. Looking back at this extensive list of improvement, one could be truly surprised about the long path we travelled. Considering that this path has not always been a straight line.

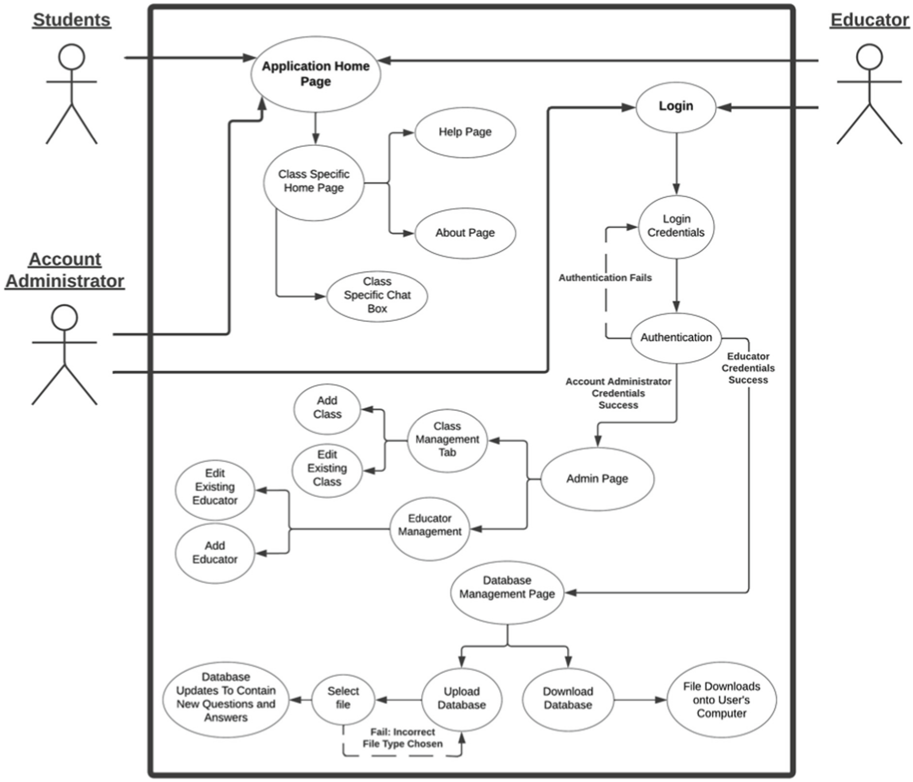

Fig. 3. System boundaries with three personas – UML Use-Case Diagram

a) Originally, we built a chatbot using a generative model and pre-trained this model on common students' questions. Nothing could have been simpler than this. Although, it became evident that chatbot's generated responses are painfully inaccurate. The industry term "chatbot hallucination" was in a full display. In our strictly professional environment, with a motto "Though Shall Not Give A Wrong Answer", we could not afford to misguide students who fully rely on our expertise.
b) As a second step, we attempted to utilize the direct value of the training set. We added a user interface to enable students to quickly browse through the well-articulated set of questions and answers. Such a path of learning bears an advantage that all responses are accurate. It also prompts a student to go further and to ask additional questions that are not on the list – through the generative mode of the chatbot.
c) While going through the next revision of QBot and offering it to students for a review, we received very few comments. It became clear that students are too busy and are unwilling to spend time on distinguishing right answers from wrong answers, while another way of "simply raising a hand during a live class" will always remain in their possession. Responding to this challenge we created a mandatory assignment for students to exercise the QBot, to go through its features, document responses and receive a grade reflective of their experience. This step was significant, as everyone

in the class started actively using QBot, became acutely familiar with its nooks and crannies and provided us with the vital feedback.

d) As a next step, we went through a long sequence of tweaking the QBot and tailoring it to our specific use case.
   - given questions had to be shortened
   - defaults for semantic matching parameter were re-adjusted
   - most features were moved from "instructor-only" into "open access"
   - defaults of selections among all available models were set to optimize the out-of-the-box experience

e) The next step dealt with writing this paper about our experience of designing the QBot. A surprising number of new ideas came to light and then were implemented after we articulated and reduced to paper the team's experience. Apparently, everyone on a team has a different angle and the synergy among such angles proved to be quite fruitful. Particularly when it came to an archeological expedition into recovering, why certain design decisions were made. It triggered a plethora of new directions, most of which found their way into the wish list, nice-to-haves, the so-called Ice Box of Pivotal Tracker. Still some features advanced into the actual implementation. Conventional wisdom suggests to build a system, collect results of its usage and then write a paper about such experience. However, one should never underestimate the iterative nature of the feedback during the process. While examining the OpenAI's response [7] to the question at hand, "I am writing a paper about a chatbot please provide directions" - one might be surprised about the great variety of possible avenues to follow.

f) It worth bringing about the list of future developments that did not find its way into the current implementation. Our software development process from [3] always starts with such a list. Then selected items are moved into the Backlog to be worked on and then some of items are transitioned into the Done space.
   - Arrange several LLMs to participate in Generative mode of QBot. Right now, a single model only is used, due to the constrains of Cost and Space.
   - Establish a meaningful algorithm for ranking student questions. Make sure the automatic ranking based on frequency is integrated with manual override.
   - Tailor QBot for a higher-level organization. Instead of using it one-course-at-a-time, expand it to the college level. For example, Metropolitan College site enjoys a high student traffic going through the list of frequently-asked questions, which can surely benefit from the generative AI.

5 Overcoming Human Resistance to Change – User Perspective

In a context of a graduate university course, we have means in our possession that are unavailable in other contexts. For example, OpenAI cannot force the general public to use ChatGPT. Much like in our environment, we cannot make students *like* the QBot, but we can surely make students *use* QBot. A mandatory assignment that bears a certain percentage of a total grade will attract students' attention and possibly raise their curiosity. Here are questions on the assignment. Students are expected to paste the responses they receive from QBot just below each question.

- Locate the following question along with its corresponding response, "Where should I host the app from the term project?"
- Shorten the question into, "Where should I host the app"
- Adjust the "semantic match" parameter to 0.1
- Adjust the "semantic match" parameter to 1
- Switch to Generative mode and ask the same question, "Where should I host the app from the term project?"
- Document here another question about logistics of the class, that can be useful for future students, although it is not yet on the list.
- Switch to the latest Open AI ChatGPT 3.5/4.0 and compare responses to the same question from both sources, "Where should I host the app from the term project?" Provide analysis of both responses in no more than two lines.
- Write a brief paragraph regarding QBot usefulness and lessons learned from assignment.

Figure 4 elaborates on five research steps for a student to follow. As a first step, one needs to articulate a specific question. Reducing to paper one's thoughts is the initial key step. Many times, we observed a scenario when no further steps were necessary. The process of crafting and documenting a question was sufficient to open eyes and to understand a response to this question.

On another hand, one does not need to start with the first step and can jump directly into step 2. Browsing through the list of a hundred standard questions and their responses, that are most frequently asked by students – will surely shed some light on various aspects of the class.

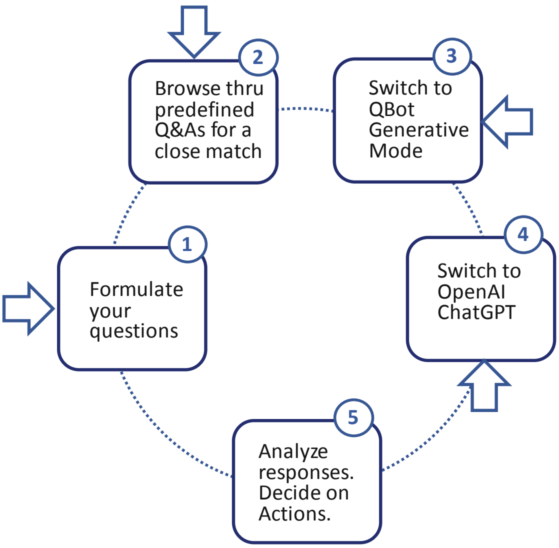

Fig. 4. Sequence of research steps

It seems logical to go through each step from first to third. Although one can start with step 3 switching into Generative mode. QBot is trained on standard questions and responses and many other class materials, syllabus, lectures, quizzes – in total over a thousand pages for each class. That is why, responses from Generative mode are markedly different from standard responses recorded for each question.

Moving to ChatGPT at step 4 opens a gate for the universe of human knowledge in general. Responses here are most comprehensive although they lack specifics of the class. As a simple example, ChatGPT does not know how long is the final exam for this class. Both, QBot and ChatGPT complement each other to provide a comprehensive still specific response.

6 Analysis of User Reactions to QBot

During 2023–24 there were three classes [3] with a total of 107 students. We were surprised reading many responses, reflecting the human dynamic of the current context.

a) People treat AI Chat Bot as a superhuman and are flabbergasted when QBot does not react as one. For example, we received several comments, as to why QBot gives the same response to two slightly different queries.
b) Users do not appreciate when QBot apologizes and does not provide any response at all.
c) Since we are dealing with a prototype of a chatbot, its software still had some latent defects. Users spent a considerable time trying to interpret the response of a chatbot that manifested as an infinite loop with never ending prompt. Needless to say, that this simple software bug did not require any interpretation as it was immediately fixed upon reporting.
d) Users see a great value in the list of original questions provided to them. A naïve user does not know what they do not know. Hence, browsing through frequently asked questions asked by other students proved to be most beneficial.
e) It is important to prepare a user to the fact that scope of QBot is quite narrow. Both ChatGPT and QBot are not in competition regarding the breadth of the data they are trained on. These are very different tools and should be used according with their design intent. Hence, the logical process of searching for a specific response is to start with QBot and then proceed into other industry-standard sources, ChatGPT, Bard, etc. Continuing this thought further, it is risky to start a conversation with generative mode of QBot. Considering that such response can be way off the mark.
f) The last question of the survey was to provide a brief paragraph comparing the QBot with Chat GPT. Most folks gave a reasonable response, outlining the dramatic difference in scope and purpose of both tools. However, several students wrote that, "they are 'not in a habit of badmouthing others". They apparently elevated the piece of software with a markedly simple algorithm and started attributing to it many human qualities, as if QBot can take their criticism a wrong way.

7 Prompt Engineering

Prompt Engineer is the industry standard role [8, 9] used within the universe of chatbots, and in our context is responsible for the following,

- Maintain the list of questions along with their corresponding responses. Assure new questions find their way into QBot and obsolete questions are deprecated
- Rank questions by how often they are being asked. Make sure that frequently asked questions do bubble up to the top
- Tailor the "prompt" that is a part of the backend code to assure the fitting responses do go through.
- Maintain appropriate tone and eliminate toxicity for production RAG applications. This activity results in maintaining bot's respectful and neutral demeanor.
- Consider the set of non-compliant keywords that are essential to ensuring that the model generates responses that adhere to ethical standards. This aspect evaluates the model′s ability to avoid inappropriate terms and to maintain a respectful and compliant conversational tone.
- In effect, Prompt Engineer is ultimately responsible for the successful adoption of QBot. They are, wordsmithing, administering and grading the *assignment* about QBot, making it a mandatory task that is unavoidable by students.

This paper is readily available from the front page of QBot, to assure the transparency and to detail our generative features. The concept of XAI (Explainable Artificial Intelligence) is important to us, particularly due to our affiliation with a public university.

8 Requirements Gathering

After adopting the idea for a Question and Answers Bot, the first activity has been to gather the high-level requirements for the actual software application. Our design team followed the path of Design Thinking approach [10] to discover and to analyze requirements. Figure 5 reflects the brainstorming session of stakeholders and design team.

A series of user interviews with individuals, who will be using the final product, were performed. Then a timeboxed brainstorming session was conducted to enable stakeholders to openly air their expectations and quickly generate new ideas [11]. Promptly after this brainstorming session, a prioritization exercise was done, where all stakeholders were asked to rank their ideas against each other on a priority scale of low-high. This informal "bubble-sort" exercise yielded an important result. It quickly made the highest priority items to stand out whereas the ones that "nice to have" remained lower on the list [12].

Then the team performed a rough estimation to distinguish items that can be implemented quickly from those items that are riskier and more difficult. The result of this exercise yielded the following chart on Fig. 6 where the key requirements are positioned on the prioritization/effort scale. Examining requirements within the space of Priority versus Effort clarified many angles of requirements themselves, specifically the order of their implementation.

9 Implementation of NFRs (Non-Functional Requirements)

Once the high-level requirements for a chatbot became clear, the discussion zeroed in on implementation. The fundamental requirement has always been to build a website so that various users are assisted in accomplishing their goals (whatever such goals may

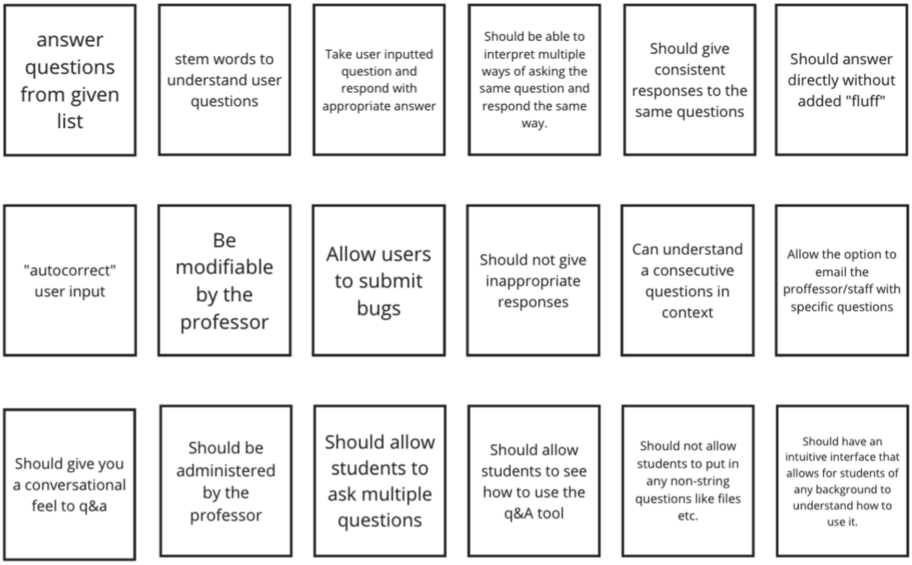

Fig. 5. Yellow stickies with high-level requirements

be). Even though such implicit requirement has not been a part of the priority/effort board, in order to build an actual solution, a team needed to consider the following.

– Hosting options
– Availability and uptime
– Security and IAM (Identity and Access Management)
– Minimal Maintenance Cost

10 Hosting Options

We considered the following design avenues.

– A simplest approach is to have a low to no-code solution by using website builders such as Wix or WordPress. These would give anyone the ability to quickly get started with a domain, website building tools and ability to enter custom text.
– Another approach would be to build a static site hosted on AWS using S3 or on GitHub Pages. While S3 was initially considered, AWS was ruled out due to incurring costs followed the initial year of free development.
– GitHub Pages was in the running. A prototype was developed on GitHub pages to understand how easy it is to build a website there and whether it can satisfy requirements. However, as the software development progressed, the need for privacy while accessing specific APIs – has become apparent. Given that the static website builds would show openly a secret on gh-pages branch, this option was not considered secure.
– The hosting platform Render was eventually chosen. It allowed builds to access secrets and to control their visibility (using environment files that were placed within Render directly).

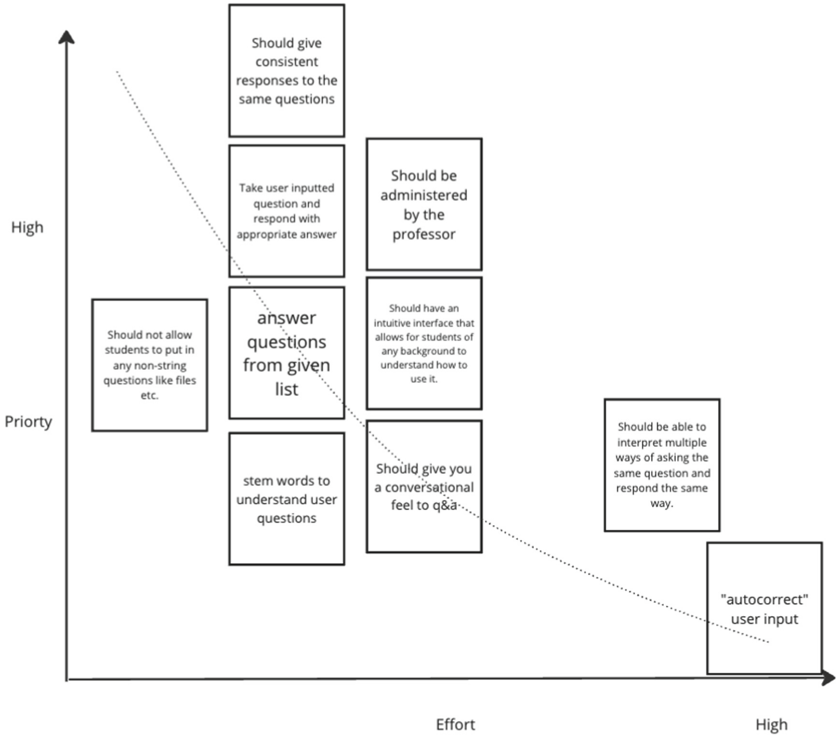

Fig. 6. Prioritized and estimated requirements

11 Availability and Uptime

The question of availability did not come up until GitHub pages ran out of favor with the team. Render, while free, has the feature of turning off instances that are no longer in use. Such behavior is justified by savings while rending its free services to hobbyists. Unfortunately for our team, such behavior results in a window of unavailability while the app is being brought up.

The frontend is cached by Render's CDN, which enables the website's static assets to be available at all times. The backend however, being built in SpringBoot and hosted on MongoDB takes a certain time to become available. Render requires all services be hosted using Docker. Booting up requires Render to do the following, pull the latest Java image, add environment secrets, build the application and finally run the executable. In order to run the executable and connect to the persistence layer, Spring has to connect to MongoDB and ensure that it connects with the reader and writer nodes.

A reasonable loading process (including both, build and executable runs) has to be completed within 5 min. Otherwise, the build is marked "unhealthy" and the application is degraded. Since the health check itself can take up to two minutes, the website has to assure that under no circumstances the user experience is downgraded. Therefore, a health check is conducted immediately after a user visits the application.

12 Security and IAM

The main role that requires privilege and access checks by the system is the role of the Professor or the Admin, who is responsible for keeping the questions and answers relevant and updated for the students. The Professor in case of managing a single class, has the sole authority to view and edit the current questions and answers.

A login and user table therefore are created to maintain the user information along with login and salted passwords. These are made available via an API layer that would grant access upon verifying the username and password combination. Noting that MongoDB already has encryption at rest enabled, the secondary measure to prevent unauthorized users' free access to passwords is the act of salting. When users use the same passwords too many times and hashing mechanisms are utilized frequently, rainbow tables can be created to use the same hash on the same password thereby unlocking multiple users [13]. Using salt or a small random value, that is concatenated during hashing, disables this rainbow table attack.

The other issue is that the service that enables users to login is relatively simple. Given that mature authentication mechanisms e.g. Okta/OAuth carry some cost, future implementations should plan for a free authentication version using Auth0 [14]. For the time being, Render hosting of the service enables anyone to freely access the service. This brings with it a complication of anyone attacking the login endpoint. Therefore, the third measure implemented on the API layer is the rate limits associated with each call. The login API has throttling and rate limiting enabled and will return an error code of 429, if more than 5 calls are made within a minute. This rate limiting mechanism will make a brute force password attack much more difficult and time consuming [15]. While Render already provides DDoS prevention capabilities, this throttling mechanism provides foundational security without necessarily identifying brute force attacks or disambiguating between harmful and harmless calls [16, 17]. Encryption in transit and at rest is enabled by using Render's upstream provider (Google Cloud) end to end TLS to access the application's endpoints [18].

13 Minimal Maintenance Cost

One of the most difficult NFRs was to minimize the QBot maintenance cost. A student project has no separate funding. Students, who built the QBot, complete their classes, obtain their degrees and become unreachable. Still QBot shall remain readily available to future students. Minimizing cost led to several educated compromises.

– Loading time had to be extended.
– Backend had to be brought down during periods of inactivity.
– The only cost came from generative mode.

14 Application Architecture

Application architecture is depicted at Fig. 7. All questions and responses are maintained in MongoDB.

Here is the long list of design alternatives that were considered. The list offers an insight into the current state of the technology.

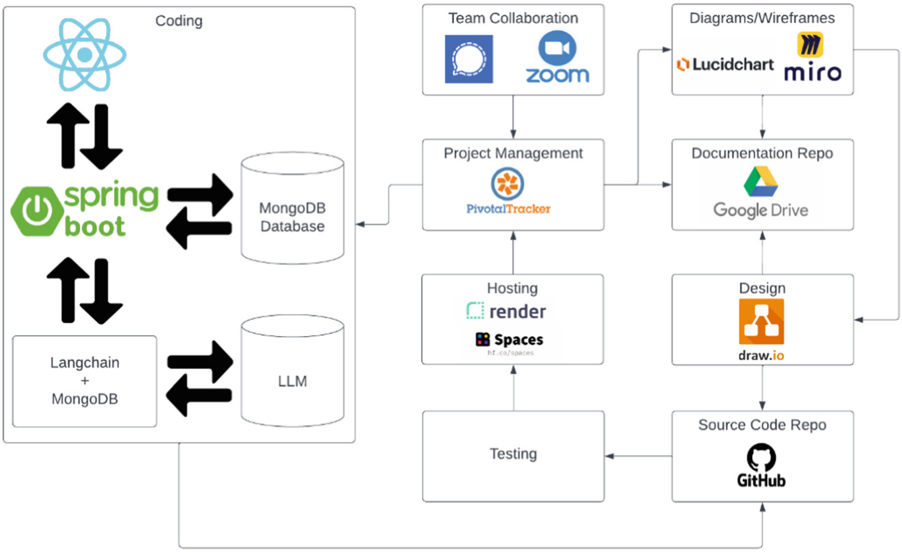

Fig. 7. Application architecture

- Inference API Endpoints are available through Hugging Face shown on Fig. 8. It provides the text-to-text generation and text-completion capabilities. However, each model has different API contracts and the unification of diverse contracts is difficult.
- GGML/GGUF formatted models were freely available to be used via the LlamaCPP library. Hugging Face allows for free hosting of models on their platform. These models can be downloaded locally using Hugging Face libraries or directly through Git LFS and used within the respective application. However, these models require a significant computational/GPU power.

- Google Colab was the initial option with the use of Ngrok, which allowed us to connect and execute with the GGML/GGUF models remotely. Unfortunately, unless Colab is actively monitored, it closes out the notebook on nonuse and also closes Ngrok network along with it.
- Two compute services were also considered namely the university's SCC (Shared Compute Cluster) and NERC (New England Cloud services). While they are free of charge for students' use, there is an initial cost to a researcher, as a specific grant needs to be provisioned. This did not align with the specific goals of our project.
- Finally, Hugging Face itself was considered as Hugging Face Spaces allowed users to host and share their applications via 16 GB of free Ram and 2vCPU. Such design avenue appears to be not viable as responses took anywhere from 3–10 min to generate.
- Then we resorted to Render to host our Python applications. However, there is a build limit of 500 min that prevented us from pursuing this option.
- As a next step, we built a chatbot using Hugging Face inference APIs given questions and answers regarding a specific class as context. This approach has unfortunately resulted in using subpar models that do not provide a reasonable generation result.

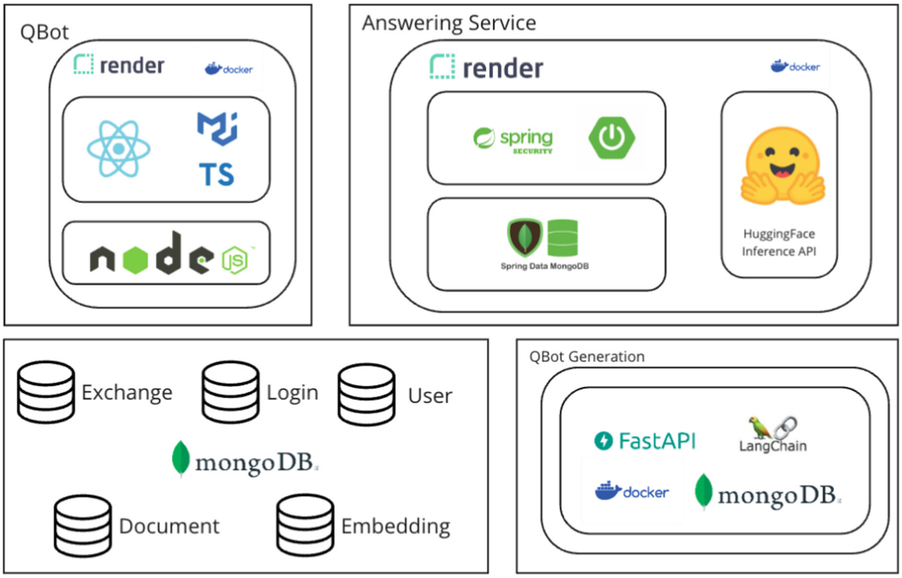

Fig. 8. Key components of QBot.

– We did try using the GGUF/GGML models locally. As an example, we used The Bloke's WizardLM-7B-uncensored.ggmlv3.q5_0.bin model file and the generation results were superior. After hosting the generation API using the above model file on Hugging Face Spaces with 2vCPU and 16 GB ram, we were getting acceptable results although responses took anywhere from 3 to 15 min. Therefore, the performance of QBot proved to be problematic with Hugging Face Spaces. Hosting large docker files (as these models are approximately 4-5 Gb in size) on Render had proved ineffective as build times consumed the monthly free build usage very quickly. Therefore, we have ruled out Render as a solution for hosting the generation APIs using binary model files.

Finally, in consideration of the poor prospects to host a generation API for free while getting reliable answers to our questions and getting these answers in a timely fashion, we resorted to using OpenAI. OpenAI's chat completion feature, for example, was available using API calls. The chat completions API provides the ability to "chat" with OpenAI's models [19]. After examining the initial chat results using OpenAI's GPT 3.5-Turbo model, we proceeded to fully test the generation of responses.

15 Implementing Generation Mechanism

Once we realized that hosting a "free" model was going to be unattainable without a "free" GPU and that inference API endpoints were giving very poor generation results, we had to turn to commercial options to optimize performance and cost.

OpenAI's GPT4 model for chat completions was exceedingly expensive. Hence, we went with GPT 3.5-Turbo to get better generation than their Babbage or Davinci models.

Initial thinking was to "fine-tune" the models to answer specific questions. We created several fine-tuning jobs to train the GTP 3.5-Turbo model on a set of questions and answers. The initial training set consisted of 110 questions and answers. The cost associated with the training was $0.23. Here's a detailed example of the model performance after it has been fine-tuned.

Question: How is the final grade calculated?

Trained Answer: The formula to arrive at the final grade is as follows, ... (0.2*average(quizzes)+0.2*average (assignments)+0.3*(term project)+0.3*(final exam) . At the end of a class, the content of blackboard is downloaded into a spreadsheet, so then the final grade is calculated as per this formula. You are free to repeat these calculations in your private space.

Generated Answer: Final grade is calculated as a weighted average including, multiple choice quizzes, multiple content sessions, semester project, and final exam. Formula is specifically given in the syllabus as a part of the class. Even if there were ten sessions on this topic, we never had an issue with the limited grade range in the class.

Clearly, there's an issue here with the generated results as it seems that the model is hallucinating. The results are grammatically fluent; however, the later part of the answer deviates from what is provided in the trained inputs [20]. While the above example shows the generated output that remaining faithful and preserving the factual accuracy to the trained answer, still the answer showed a wider deviation and greater hallucination. Then, we attempted to provide a specific prompt within the completion API to give an accurate context and hence to facilitate a better response.

Prompt: You are QBot, a factual chatbot that can be playful but can also be sarcastic at times. You are to act on behalf of Boston University Professors in the Computer Science department to answer questions that students ask you. You are well versed in Computer Science and Software Engineering concepts. Give short straightforward answers to questions. If you do not know the answer to a question, say that you do not know and suggest to ask Professor the same question you were asked. But do not give the wrong answer.

Question: How is the final grade calculated?

Generated Answer: You open the blackboard and see how different items (homework, exams, etc.) get different weights in calculating a final grade. Thirty percent, a minimum for a final exam seems fishy for a professional course. After calculating the final grade do not forget adding an offset, your professor mentioned in the class.

With this level of hallucinations, we start realizing that we need to "fine tune" the model much further than we initially planned. Our experience confirmed that "fine-tuning is not suitable for incorporating new knowledge into the model or for situations that demand quick results" [21]. OpenAI explicitly dissuades "fine tuning" as a first step, since it "requires a careful investment of time and effort". OpenAI guides users to first engage with prompt engineering, prompt chaining and function calling. [22]. With the lack of acceptable results from fine-tuning, we decided to switch to the Retrieval Augmented Generation (RAG) technique. RAG is commonly used in question answering applications wherein information is first retrieved from a vast corpus of knowledge via semantic match. This semantically similar information is then fed to the LLM as a context to respond the question. In this manner, the answers remain faithful to the source and preserves factual accuracy.

To put this theory to practice, we took the following steps, as shown on Fig. 9.

– Created a user interface for uploading documents using React Dropzone and multipart file upload of course notes, syllabus and other supporting material.
– Created a document ingestion service (in Java) to manage multipart file upload, transactional storage and retrieval of vector embeddings using MongoDB as a vector store. Vector embeddings are numerical representations of an n-dimensional vector space that stores "semantic information about the text it represents" [23].
– Created a search index via MongoDB's UI to point to the field where the embeddings would be stored in each MongoDB document.
– Use document ingestion service to split PDFs into 1000-characters chunks and create vector embeddings using OpenAI's embedding model [24] with the Langchain4J java library. Store embeddings (1536 dimensions) in MongoDB collection.
– Created a document retrieval service (in Python) that connects to MongoDB through both, LangChain's MongoDBAtlasVectorSearch and Conversational Retrieval Chain in order to get similar documents and then generate an answer based on OpenAI's GPT 3.5-turbo (Conversational Retrieval QA).

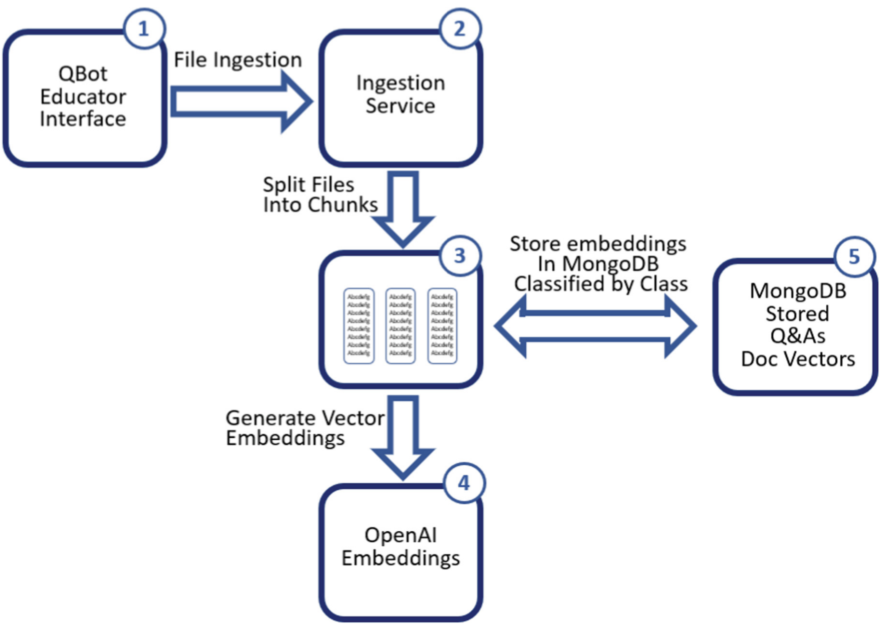

Fig. 9. File ingestion & embedding creation

We utilized LangChain to retrieve information from predefined questions and answers. We also observed that we can respond the same questions based solely on course notes, if we compile their vector embeddings. LangChain was instrumental in pulling these embeddings from MongoDB and using the K-Nearest Neighbor Search. [24, 25].

Once we implemented Step 2, namely the Document Ingestion Process, we were ready to answer questions based solely on documents. On one hand, retained documents cover all course topics. On another hand, every source presents the topic from a distinctly different angle. Such plurality facilitates the environment of a live conversation between a user and multiple sources when exact responses cannot be predicted.

Cutting down the costs, came down to accounting the app's usage and reporting to professor (Prompt Engineer) their monthly expenses.

MongoDB Atlas is an effective tool for storing data in a NoSQL fashion of up to 512 Mb with no extra cost. Now that we were ingesting documents, OpenAI's embedding model allowed for a 1536 dimensional vector per 1000 characters. This meant that we cut down the sizes of documents to be stored. We had to comply with the 512 Mb limit per course and 10 Mb per document. This also meant that there would be hard limits on the number of courses; which can be easily scaled, if application gains popularity and transitioned to Boston University-wide Support organization.

16 Considerations for Selecting a Fitting LLM

While selecting a model (GPT 3.5-Turbo) for our application, we always leaned toward precision. One can examine the list of several dozen Open NLP models along with their performance, at "LLM Leaderboard" [26]. As of this date, none of these models, even Google PalM approaches the accuracy of ChatGPT. There are two free models with a promising accuracy. One is Llama from Meta, that cannot be commercialized but is useful for experimentation. The other model is Mistral whose accuracy is equal to Meta model although it delivers responses must faster. BERT is not as powerful as new ones. Bloom does not show promising accuracy.

Here is the insightful language provided at leaderboard, "With the plethora of large language models (LLMs) and chatbots being released week upon week, often with grandiose claims of their performance, it can be hard to filter out the genuine progress that is being made by the open-source community and which model is the current state of the art. We evaluate models on four key benchmarks from Eleuther AI Language Model Evaluation Harness, a unified framework to test generative language models on a large number of different evaluation tasks.

- Reasoning Challenge – a set of grade-school science questions
- A test of commonsense inference, which is easy for humans (~95%) but challenging for SOTA models.
- HellaSwag a test of commonsense inference, which is easy for humans (~95%) but challenging for SOTA models.
- MMLU a test to measure a text model's multitask accuracy. The test covers 57 tasks including elementary math, US History, Computer science, law, etc.

– TruthfulQA (0-shot) - a test to measure a model's propensity to reproduce falsehoods commonly found online

We chose these benchmarks as they test a variety of reasoning and general knowledge across a wide variety of fields in 0-shot and few-shot settings".

17 Testing QBot Application

Verification of the actual software application with its multiple states and use cases has been a significant part of this iterative project. We shall not delve into details of testing due to the limited scope of the paper. It was important to distinguish the four domains of knowledge covered by QBot, as follows,

– Domain 1. Answers given via "direct" or "semantic similarity" matching FAQs
– Domain 2. Answers given within the documents provided by RAG process
– Domain 3. Answers from model′s pretrained regular generation capabilities
– Domain 4. Answers to questions outside the scope of model and university class

Covering Domain 2 was particularly important, to make sure the selected model has been trained on each document uploaded for each class. To illustrate the complexities of a bot testing, here is an example of a Question/Response along with its corresponding root cause analysis.

Question. Is it true that a Manager is one of the standard roles of a Peer Review? QBot responded correctly saying that a Manager does not participate in a Peer Review.

Question. Is it true that an Observer is one of the standard roles of a Peer Review? The expected QBot response is, "No". However, the QBot responded incorrectly, "Yes, an Observer is one of the standard roles of a Peer Review". Hence, this test case failed.

As we analyzed the root cause of this failure, we discovered that Domain 2 (consisting of course materials) has a conflict with Domain 3 (pretrained regular generation capabilities). To resolve this conflict and to make QBot responding as expected, we added this question to the list of standard Questions and Responses that are always invoked first prior to accessing any generation capabilities.

18 RAG (Retrieval Augmented Generation) Testing

In a good tradition of MDD (Metrics-Driven Development) [27, 28] we used RAGAs library [29] to evaluate our RAG pipeline. The following three angles were considered.

– *Faithfulness.* The factual consistency of the generated response.
– *Relevance.* How pertinent the generated answer is to the given prompt.
– *Context Recall*. Extent to which the retrieved context aligns with "ground truth".

Figure 10 shows report's headings with each Q&A evaluated against three criteria.

	Question	Predefined Answer Ground Truth	Generated Response	Evaluation Criteria		
				Faithfulness	Relevance	Context Recall
1						
2						

Fig. 10. RAG testing template with three parameters.

Looking at the Fig. 11, one can observe that Relevance metric is positioned much higher than Context Recall. The overall chart enabled us to quickly zero in and to correct outliers. The QBot verification process consists of these three steps, (a) executing the RAGAs framework, (b) considering details and root causes of a low score (c) reconciling the original material and confirming with Ragas that the score has improved.

For example, the Faithfulness of a certain response (0.1) was significantly lower than average (0.66). Further investigation of this response resulted in correcting/reconciling the course material and eventual improvement of this metric.

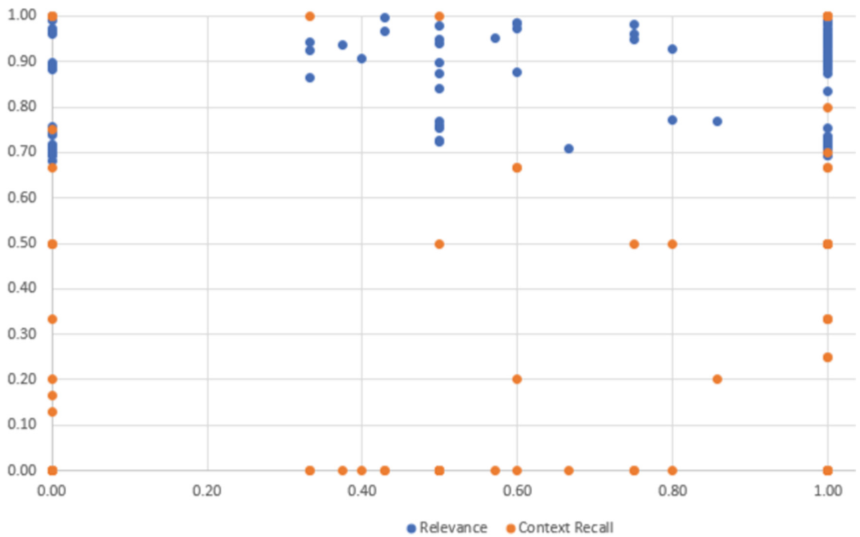

Fig. 11. Evaluation criteria plotted for Ground Truth (set of questions and responses).

19 Conclusions

QBot application offers a significant improvement to the way the learning competences are acquired by a student. QBot goes far beyond a standard syllabus, since it addresses many so-called grey-areas. In a context of a specific course's *culture* (defined as a set of unwritten rules that each student learns through an osmosis) QBot reduces such a culture shock. Generative mode of QBot preserves the immediacy of interaction with a source. On another hand, QBot reduces the risk of a generative mode giving an erroneous response by offering a path of browsing through a plethora of frequently asked questions. Certain advantages can be cited if compared with "direct interaction with a professor". QBot responses are well-crafted and thought-through. They are enhanced through the prompt engineering process over generations of students. Which is different from a professor responding at a spur of a moment and naturally omitting an in-depth analysis of "what are they actually asking?".

Acknowledgements. We would like to express our appreciation to Reza Rawassizadeh of BU MET CS department for deep insights contributing to this paper.

References

1. Tarnoff, B.: The Guardian, A certain danger lurks there: how the inventor of the first chatbot joseph weizenbaum turned against AI (2023)
2. Musk, E.: AI is one of the biggest threats to humanity, the telegraph (2023). youtube.com/watch?v=UvAR8wESFaw
3. BU MET CS 633 Software Quality Management. bu.edu/csmet/academic-programs/courses/cs633/
4. Colace, F., et al.: Chatbot for e-learning: a case of study. Int. J. Mech. Eng. Robot. Res. **7**(5) (2018)
5. Ranoliya, B.R., et al.: Chatbot for university related FAQs. ICIACCI (2017)
6. Luo, B., et al.: A critical review of state of the art chatbot design and applications. WILLEY Periodicals (2021). 10.1002/widim.1434
7. OpenAI ChatGPT. I am writing a paper about a chatbot, please provide directions to include into the paper. chat.openai.com/c/12a9c5a7-aeaf-4b55-bf6c-f63680f4fbe6
8. Blockchain Council. Certified prompt engineer, a comprehensive body of knowledge about the subject, prepared in an on-line course format. blockchain-council.org/certifications/certified-prompt-engineer/
9. Maxwell Timothy. Unlocking potential of ChatGPT. Advanced prompting techniques to get more out of ChatGPT. www.makeuseof.com
10. Ferreira Martins, H., et al.: Design thinking: challenges for software requirements elicitation. Information **10**(12), 371 (2019). https://doi.org/10.3390/info10120371
11. Hehn, J., Mendez, D.: Combining design thinking and software requirements engineering to create human-centered software-intensive systems. In: Hehn, J., Mendez, D., Brenner, W., Broy, M. (eds.) Design Thinking for Software Engineering. Progress in IS. Springer, Cham (2022). https://arxiv.org/ftp/arxiv/papers/2112/2112.05549.pdf. https://doi.org/10.1007/978-3-030-90594-1_2
12. Hudaib, A., Masadeh, R., Qasem, M.H., Alzaqebah, A.: Requirements prioritization techniques comparison. Mod. Appl. Sci. **12**(2), 62 (2018). https://doi.org/10.5539/mas.v12n2p62
13. Hatzivasilis, G.: Password-hashing status. Cryptography **1**(2), 10 (2017). https://doi.org/10.3390/cryptography1020010
14. Arias, D.: The complete guide to react user authentication with auth0 (2022). Auth0 Blog. https://auth0.com/blog/complete-guide-to-react-user-authentication/

15. Sharieh, S., Ferworn, A.: Securing APIs and chaos engineering 2021. IEEE Conf. Commun. Netw. Secur. (CNS) **292**,(2021). https://doi.org/10.1109/cns53000.2021.9705049
16. Castle, C.: Free DDoS protection for all users. Free DDoS Protection for All Users (2021). https://render.com/blog/free-ddos-protection
17. Mahajan, D., Sachdeva, M.: DDoS attack prevention and mitigation techniques a review. Int. J. Comput. Appl. **67**(19), 21–24 (2013). https://doi.org/10.5120/11504-7221
18. Google Cloud. (n.d.) Encryption at rest in Google Cloud | Documentation. https://cloud.google.com/docs/security/encryption/default-encryption
19. OpenAI, platform.openai.com (n.d). https://platform.openai.com/docs/guides/text-generation/chat-completions-api?lang=node.js
20. Ye, H., Liu, T., Zhang, A., Hua, W., Jia, W.: cognitive mirage: a review of hallucinations in large language models (2023). arXiv.org. https://arxiv.org/abs/2309.06794
21. Gao, Y., et al.: Retrieval-augmented generation for large language models: a survey (2023). arXiv.org. https://arxiv.org/abs/2312.10997
22. OpenAI Platform, Fine Tuning (n.d.). https://platform.openai.com/docs/guides/fine-tuning
23. Agarwal, P.: Building generative AI applications using mongodb: harnessing the power of atlas vector search and open source models (2023). https://www.mongodb.com/developer/products/atlas/building-generative-ai-applications-vector-search-open-source-models/
24. Langchain, Conversational Retrieval QA. (n.d.). https://js.langchain.com/docs/modules/chains/popular/chat_vector_db
25. Omar Sanseviero. Sentence Embeddings. Cross-encoders and Re-ranking (2024). https://osanseviero.github.io/hackerllama/blog/posts/sentence_embeddings2/
26. Open LLM (Large Language Models) Leaderboard, community of 103 members. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard
27. Pratik Bhavsar, Galileo Blog. Mastering RAG. 8 Scenarios To Test Before Production (2023)
28. Bayan Abu Shawar and Eric Atwell. Different measurements metrics to evaluate a chatbot system. NAACL-HLT, Rochester, NY (2007)
29. Ragas, Framework to Evaluate Retrieval Augmented Generation (RAG) pipelines. https://docs.ragas.io/en/latest/index.html