https://doi.org/10.2352/EI.2025.37.3.MOBMU-319

Integration of protocol-driven chatbots with generative AI and a case study

Hasmik Yengibaryan, Yerevan State University, Yerevan, Armenia
David Akopian; Department of Electrical & Computer Engineering, The University of Texas at San Antonio, San Antonio, USA

Abstract

The integration of deterministic protocol-specified chatbots with generative AI bridges the gap between precise, protocol-driven logic and conversational flexibility. This paper introduces MachineQuizzing, a chatbot designed to enhance learning in machine learning through gamified quizzes and real-time explanations. Leveraging platforms like Dialogflow for structured logic and Gemini for generative capabilities, the chatbot demonstrates how the integration of these technologies can enhance conversational experience.

Introduction

Chatbots have become a pivotal part of our daily lives, enhancing human-machine interaction across various domains. They are tools that are reshaping business-customer interactions. Chatbots are increasingly important in various industries, from customer service to healthcare, education, and finance.

Chatbots, or conversational agents, are computer programs designed to simulate human conversation with users, usually through text or voice interactions [1].

Platforms such as Google Dialogflow [2], Amazon Lex [3], IBM Watson [4], ManyChat [5], and Chatfuel [6] have played a significant role in advancing this technology. These platforms mostly focus on protocol-driven chatbots to configure human-machine conversations. These platforms excel in configuring conversations based on predefined rules and possibly expert knowledge, making them highly effective for structured, protocol-driven applications.

However, LLM advancements resulted in more flexible, contextually aware, and human-like chatbots. Models such as GPT [7], Gemini [8], Llama [9], and Claude [10] have revolutionized the field. Unlike their protocol-driven counterparts, generative AI (genAI) systems leverage vast general knowledge and advanced natural language understanding to engage in more dynamic and intuitive conversations.

Despite the impressive capabilities of protocol-driven platforms and generative AI systems, a notable gap exists between these approaches. Protocol-driven platforms offer robust solutions for specific tasks through structured interactions, but they often lack the broader conversational flexibility and contextual richness of generative AI. On the other hand, generative AI does well with general knowledge and flexibility but fails to accomplish very specific or structured tasks without additional help [11].

Addressing this gap presents an opportunity to combine the best of both worlds by integrating the strengths of both approaches. By combining the precise, protocol-driven logic of protocol-specified chatbots with the vast knowledge base and conversational flexibility of generative AI, we can create more powerful and dynamic chatbot systems.

In particular, Google provides several conversational AI tools, including Dialogflow [12] for protocol-driven chatbot creation and Gemini for general chatting.

This paper explores the feasibility of integrating protocol-specified chatbots with generative AI. It demonstrates this concept by examining the integration of Dialogflow and Gemini to bridge the gap between protocol-driven and generative chatbots. A case study integration MachineQuizzing chatbot is developed for an enhanced learning experience.

Problem formulation

Integrating protocol-driven chatbots with genAI is to enrich the limited capabilities of protocol-driven chatbots' specialized functionalities with the broad knowledge base and human-friendly interface of generative AI.

In this work, we will explore this integration with a practical example. As an example, we chose a chatbot assistant that can help users deepen their knowledge of machine learning with gamified quizzes. Chatbot's protocol-driven part should be able to provide tests to users, which include multiple choice questions about various machine learning techniques and methods. It should also catch incorrect responses, grade their performance, save their score, and provide user statistics about their progress on overall and individual scores. The Generative part should correct and explain user mistakes and answer any questions about machine learning that users may have at any given moment. It should have an easy-to-use and user-friendly UI to help the gamification learning process.

Dialogflow and integrations

Dialogflow is a conversation design platform that integrates natural language understanding. It makes it easy to design and integrate a conversational user interface into a mobile app, web application, device, bot, interactive voice response system, and so on [13]. As shown in the schematic description in Figure 1, Dialogflow serves as a middleman for end-user and internal logic for the given chatbot.

Dialogflow does not provide the end-user agent, but integrates with many popular conversation platforms like Google Assistant, Slack, Telegram, and Facebook Messenger. These integrations are fully supported by Dialogflow and are configured with the Dialogflow Console. Each integration handles end-user interactions in a platform-specific way. [12]

For example, to set up integration, a Telegram bot can be created, e.g., using Telegram botfather. Then, the bot's Access Token should be given to Dialogflow Console. Dialogflow's fulfillment feature can be used to process the chatbot logic. There are two options for this: inline editor and Webhook service. The inline editor is intended for simple fulfillment testing and prototyping. Webhook service is used for more complicated fulfillments. It also allows connection to external databases and external API calls. User interactions with the integrated platform will be processed by Dialogflow and transferred to the backend to identify the corresponding intent. Then, the response logic for the intent will be activated. It can be a simple response, a

communication with a database, or an API call. This conversation cycle may repeat as many times as needed.

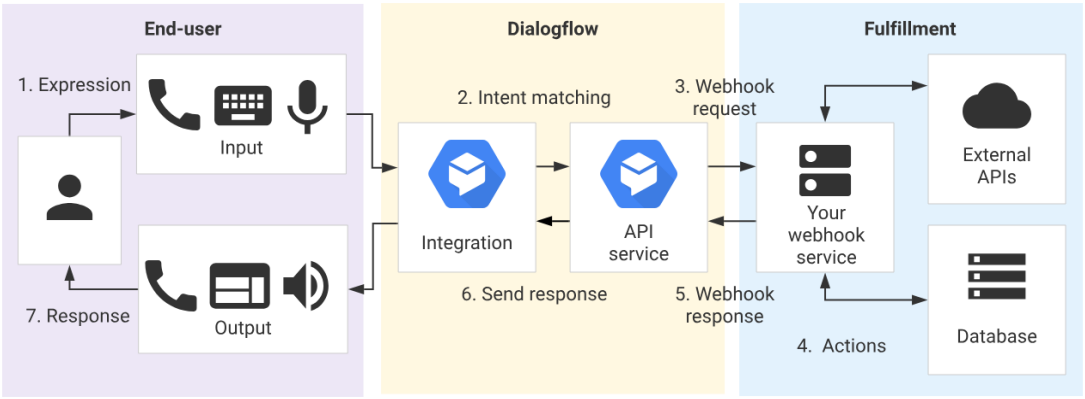


Figure 1. Chatbot workflow using Dialogflow

In particular, protocol-driven chatbots created with Dialogflow can be integrated with Generative AI using dedicated APIs. The Gemini API gives access to the whole Gemini model family, which are Google's latest models [14]. To gain access to models, an API key should be created in Google AI Studio. Then, in the backend, the model in use should be chosen and configured.

Method description

We created a chatbot called MachineQuizzing. Various tools and technologies such as Dialogflow, MySQL, Python,

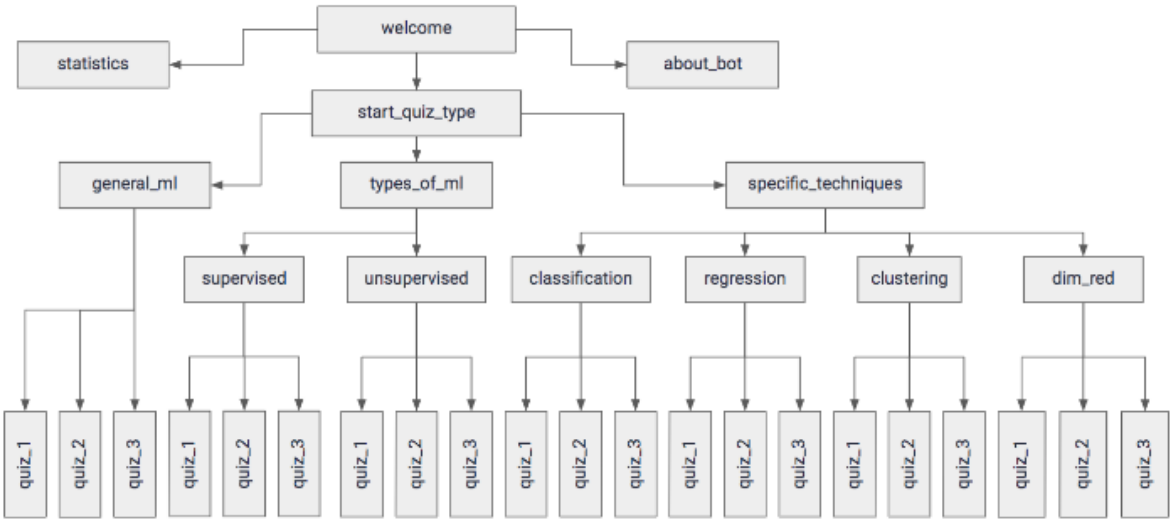


Figure 2. Contexts and Intents

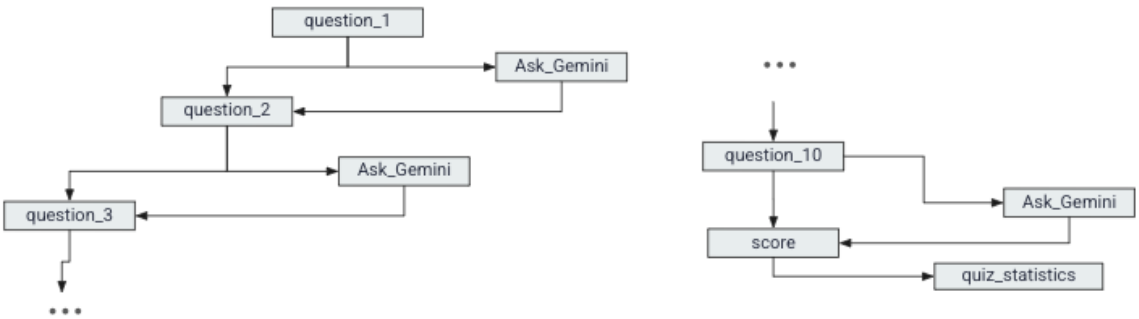


Figure 3. Quiz Structure

FastAPI, Gemini API, Telegram Messenger were used to build the chatbot. Each tool played a crucial role in various aspects of chatbot functionality, from natural language understanding to backend logic development and user interface.

- Building the protocol-driven bot component: Dialogflow serves as the main platform for building and designing the conversational chatbot agent [13].
- Intent classification: for Intent classification chatbot was taught using Dialogflow's natural language understanding capabilities. Training phrases are needed to distinguish the intents, which are sample phrases that can be typed by the end user. For each intent, 5-20 training phrases were created. From practical observations, it is observed that for very similar expressions, the system had difficulty classifying expressions that were sufficiently different from them. However, when taught with distinct expressions, intent classification occurred more precisely. Therefore, when building MachineQuizzing, we considered this and chose expressions for training that were less in number, but as comprehensive as possible.
- Slot filling, contexts: For these, we also used Dialogflow's features. Slot filling determines the user's preferred test type at the beginning, see Figure 2, and quiz answer options during the test, see Figure 3. Contexts guide users through the conversations. In Figure 2, three boxes signify the intents, and arrows are their connecting contexts.
- Backend: in order to process the external logic of the chatbot, it makes Webhook requests to the backend through the fulfillment feature of Dialogflow. Then the requests are received and processed using the FastAPI Web framework and the Python programming language, respectively. This backend infrastructure made it possible to process user data from Telegram, interact with external services (Gemini API) and databases (MySQL).

```
async def handle_request(request: Request):
    # Retrieve the JSON data from the request
    payload = await request.json()
    question_id = None

    # Extract necessary information from the payload

    # Get username and intent from payload
    username =
payload['originalDetectIntentRequest']['payload']['data']['from']['id']
    intent = payload['queryResult']['intent']['displayName']

    # Get parameters and query text
    parameters = payload['queryResult']['parameters']
    query_text = payload['queryResult']['queryText']

    # Get output context
    output_context = payload['queryResult'].get('outputContexts', None)
        # Call function for the corresponding intent
    ...
```

- User Interface: for UI Telegram instant messaging service was used. Chatbots integration with it enables users to interact with the chatbot directly within Telegram. The platform provides users with accessibility across different devices and environments, such as Telegram's mobile and desktop applications, as well as Telegram Web. Additionally, through integration, user data is collected in the backend—specifically, their usernames, which serve as the primary key in the database.
- Database system: MySQL database management system (RDBMS) was used to store and manage user's score data.

```
def insert_or_update_progress(username, quiz_name, score):
    # Connect to the database
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="password",
        database="quiz_progress"
    )
    cursor = conn.cursor()

    # Check if entry exists for the given username and quiz_name
    cursor.execute(f"SELECT * FROM {quiz_name} WHERE
username=%s", (username,))
    existing_entry = cursor.fetchone()

    if existing_entry:
        # Entry exists, update the values
        first_attempt = existing_entry[1]  # Get existing first_attempt score
        average = ((existing_entry[2] * existing_entry[4]) + score) /
(existing_entry[4] + 1)
        last_attempt = existing_entry[5]
        num_attempts = existing_entry[4] + 1
        current_attempt = score

        # Update the row
        cursor.execute(f"UPDATE {quiz_name} SET first_attempt=%s,
average=%s, last_attempt=%s, num_attempts=%s,"
                       f" current_attempt=%s WHERE username=%s",
                       (first_attempt, average, last_attempt, num_attempts,
current_attempt, username))
    else:
        # Entry does not exist, insert new row
        first_attempt = score
        average = score
        last_attempt = score
        num_attempts = 1
        current_attempt = score

        # Insert new row
        cursor.execute(
            f"INSERT INTO {quiz_name} (username, first_attempt, average,
last_attempt, num_attempts, current_attempt) "
            f"VALUES (%s, %s, %s, %s, %s, %s)",
            (username, first_attempt, average, last_attempt, num_attempts,
current_attempt))

    # Commit changes and close connection
    conn.commit()
    conn.close()
```

- Generative AI integration: for MachineQuizzing's generative part Google's Gemini Pro was used [8]. The chatbot has been integrated with the Gemini API, which can be activated in two cases: for mistake corrections and question answers. If the wrong answer to a quiz is given, the model is activated by quizzes intent and explanation why the answer is wrong is generated.

```
def question(user, parameters: dict, output_context: list, quiz: dict,
quiz_answer: dict, question_id, score):
  ...
    else:
        # Provide feedback if answer is incorrect
        fulfillment_text = api_answer(
            f"Question is:{quiz[str(int(question_id) - 1)]} and my answer is
{parameters['quiz-answer']} explain why it is wrong and what is the correct
answer") + \
                       " \n Ask me other questions you may have or type 'next' if you
want to see the next question."
        active_context = prefix + f"context_question{int(question_id) - 1}"
        return JSONResponse(content={
            "fulfillmentText": fulfillment_text,
            "outputContexts": [
                {
```

```
            "name": f"{active_context}",
            "lifespanCount": 2
        }]
    })
```

If a user asks a question Default Fallback intent is activated as no intent is matched and the answer is generated.

```
def fallback(context: list, query_text: str):
    # Generate response using API
    fulfillment_text = api_answer(query_text)
    print(fulfillment_text)
    return JSONResponse(content={
        "fulfillmentText": fulfillment_text,
        "outputContexts": [
            {
                "name": f"{context[0]['name']}",
                "lifespanCount": 2
            }
        ]
    })
```

Encountered problems and their solutions

During the building process of MachineQuizzing we encountered some problems mainly connected to webhook timeout limitation [12]. When no matching intent is found, for example when a user has asked a question about machine learning, the Default Fallback intent is activated and the query is sent to Gemini's API. Dialogflow has a maximum webhook timeout limit of 5 seconds, that is, if no response is received from the backend within 5 seconds, it returns the default response of the system, for example, "Sorry, can you say that again?", "I didn't get that. Can you repeat?" or "One more time?".

One of the problems arose when a response received from the Gemini API was being processed.Gemini's replies are written in Common Markdown, which is incompatible with Telegram's MarkdownV2. All translation attempts were causing a violation of the aforementioned 5-second time limit. So the format was converted from Markdown markup language to plain text.

Another problem was Gemini's response time, which also conflicted with Dialogflow's webhook timeout limit. In order to avoid this limitation and get more accurate answers, prompt tuning was done using the following texts:

"When asked questions, keep your answers as short as possible and don't give any extra information unless specifically asked."

"Picture yourself as my trusted ML advisor, I will ask you ML related questions, please answer them. I'll also give you quizzes and the answer options, my answer which will likely be wrong, don't give me the quiz answers just give me direction. I'm turning to you for assistance with my quizzes or any ML inquiries. Offer insights, hints, or answers to help me progress. Please help me learn."

This action reduced the time it takes Gemini to generate a response by 0.038 seconds on average. For example, this action took Gemini's response generation time from 0.005616 seconds to 0.043894 seconds, reducing it by 0.038278 seconds for "What is linear regression?" query, and for "What is a confusion matrix?" query response was reduced by 0.038303 seconds from 0.045353 to 0.00705.

While testing the chatbot, we also encountered a problem with hallucinations during the generation of the wrong answer explanations [15]. Initially the current test question and the user's wrong answer with the following prompt were given to Gemini: "Question is: {current question} and my answer is {user's wrong answer}." Sometimes wrong version answers were generated as correct ones. Here, too, we applied prompt tuning, we added "explain why it is wrong and what is the correct answer" to the above prompt. As a result, the number of hallucinations decreased.

Results

We created MachineQuizzing by combining a protocol-driven chatbot created via Dialogflow with generative AI represented by Gemini. Our chatbot successfully addresses all the elements outlined in the problem description.

We'll consider each point in detail below.

- Quizzes: MachineQuizzing provides several quizzes, ranging from general machine learning to specific techniques such as dimensionality reduction, classification, and regression. There are 21 tests of 7 types, each consisting of 10 multiple-choice questions with only one correct answer. If the answer is correct, the user earns points, and the next question is presented. Quizzes allow users to test their understanding of key machine learning concepts and principles.
- Evaluation system: during the quiz, the chatbot evaluates the user's answers. Each correct answer is valued 10 points, giving the opportunity to score a maximum of 100 points. The user does not receive points for skipped and wrong answered questions. After completing each quiz, the user receives the score.
- Statistics: in addition to grading tests, chatbot allows users to compare received scores with the results of other users, as well as with the results of previous attempts. This feedback mechanism allows users to track their progress and identify gaps in their knowledge.
- Explanation of wrong answers: if the user answers a question incorrectly, the chatbot explains why the answer is incorrect. This instant feedback helps users understand concepts more deeply through simple and concise explanations.
- Answering questions: the chatbot can also answer questions that arise during the test. Thanks to this feature, users have access to the relevant information when they need it and can reach the correct quiz answer through the questions.

Thus, thanks to the above features, the chatbot enables users to easily grasp the complex concepts of machine learning and strengthen their knowledge as can be seen in the screenshots below or by following the QR code:

Analysis and Conclusion

This paper shows a successful example of integration of protocol-specified chatbot with GenAI. With MachineQuizzing, you can see the advantages of parts of integration. Its protocol part provides the selection, presentation, and evaluation of tests and provides users with statistics on their progress. The GenAI part provides answers to questions and explanations of mistakes. Since the explanations are activated only in the case of an incorrect answer, the protocol-specified part also controls the truthfulness of

the generated answer. In conclusion, integration resulted in a simple and convenient assistant for familiarization with machine learning, its branches, and algorithms. A chabot operation illustration example can be observed through the link of Figure 4, and sample screenshots are shown in Figure 5.

Figure 4. QR code: chatbot operation

References

[1] E. Adamopoulou and L. Moussiades, "An overview of Chatbot technology," in IFIP advances in information and communication technology, 2020, pp. 373–383. doi: 10.1007/978-3-030-49186-4_31.

[2] "Conversational agents and dialogflow," Google Cloud. https://cloud.google.com/products/conversational-agents

[3] "Build and deploy conversational AI interfaces with Amazon Lex," Amazon Web Services, Inc. Available: https://aws.amazon.com/lex/

[4] "IBM Watson." https://www.ibm.com/watson

[5] "Chat Marketing Made Easy with Manychat," manychat.com. https://manychat.com/

[6] Chatfuel, "Chatfuel | AI agents for automated sales | Meta's partner," Chatfuel. https://chatfuel.com/

[7] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018. [Online]. Available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[8] "Gemini: a family of highly capable multimodal models," arXiv, Jan. 2023, doi: 10.48550/arxiv.2312.11805.

[9] Llama H. Touvron et al., "LLAMA: Open and Efficient Foundation Language Models," arXiv, Jan. 2023, doi: 10.48550/arxiv.2302.13971.

[10] Anthropic. The Claude 3 model family: Opus, Sonnet, Haiku, *Anthropic*, 2024. Available: https://anthropic.com/claude-3-model-card

[11] H. Chen, X. Liu, D. Yin, and J. Tang, "A survey on dialogue systems," ACM SIGKDD Explorations Newsletter, vol. 19, no. 2, pp. 25–35, Nov. 2017, doi: 10.1145/3166054.3166058.

[12] Google Cloud, "Dialogflow ES documentation." Available: https://cloud.google.com/ dialogflow/es/docs.

[13] N. Sabharwal and A. Agrawal, "Introduction to Google Dialogflow," in Apress eBooks, 2020, pp. 13–54. doi: 10.1007/978-1-4842-5741-8_2.

[14] Google AI for Developers, "Gemini API Overview." Available: https://ai.google.dev/gemini-api/docs/api-overview

[15] Z. Xu, S. Jain, and M. Kankanhalli, "Hallucination is Inevitable: An Innate Limitation of Large Language Models," arXiv , Jan. 2024, doi: 10.48550/arxiv.2401.11817

Author Biography

Hasmik Yengibaryan received her BS in Statistics from Yerevan State University, Armenia. She is currently a graduate student at the faculty of Mathematics and Mechanics in YSU. Her research interests include Machine Learning and AI technologies.

David Akopian received his Ph.D. degree from Tampere University, Finland. He is currently a Professor with The University of Texas at San Antonio (UTSA). Before joining UTSA, he was a senior scientist at Nokia. His current research interests include mobile computing and positioning, He is a Fellow of US National Academy of Inventors.

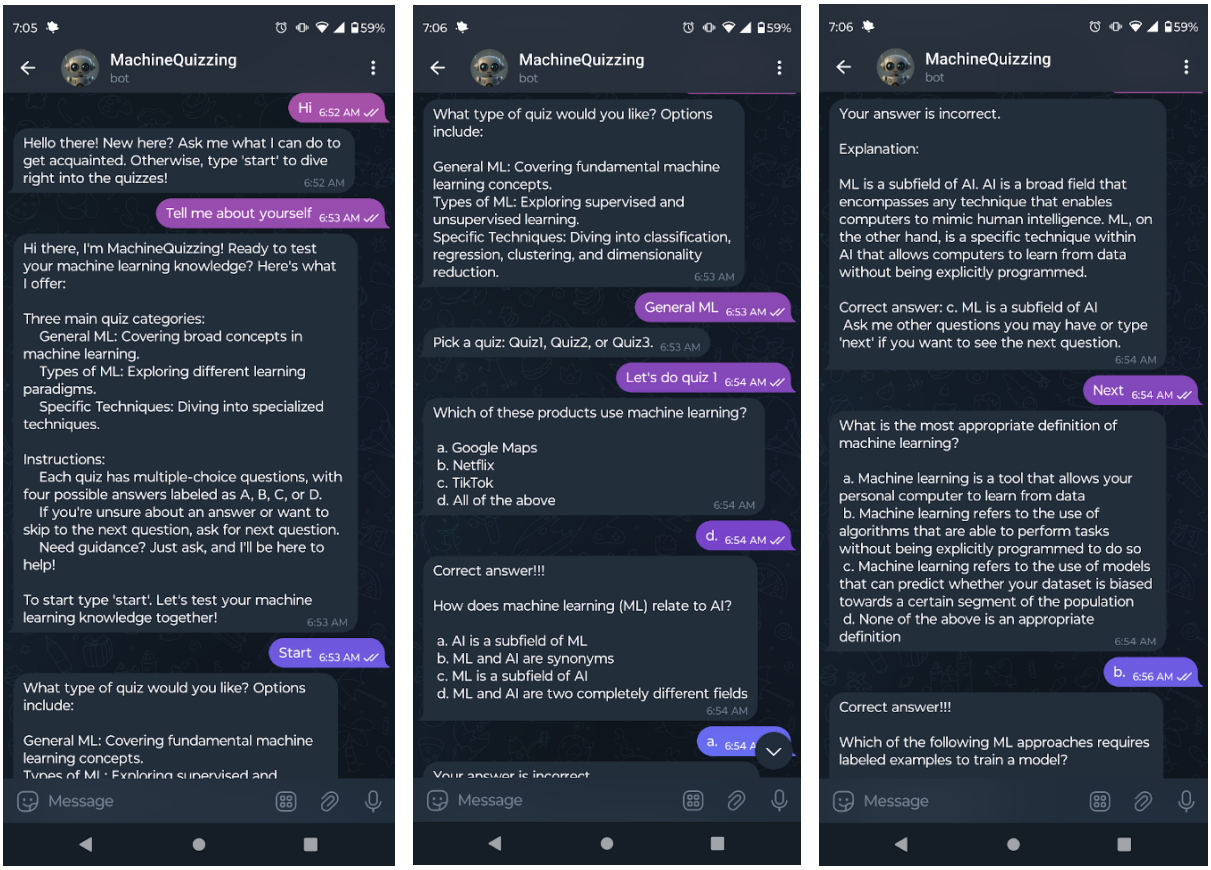
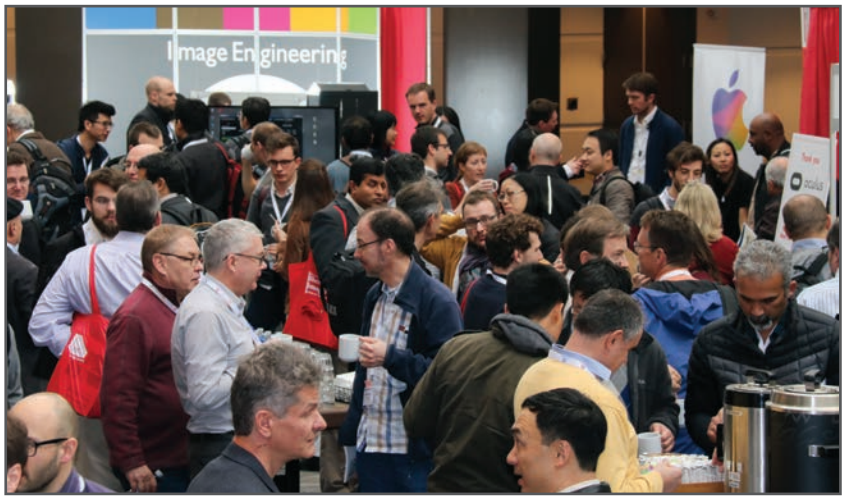


Figure 5. Sample screen snapshots of the chatbot operation

JOIN US AT THE NEXT EI!

electronic IMAGING

Imaging across applications . . . Where industry and academia meet!



• SHORT COURSES • EXHIBITS • DEMONSTRATION SESSION • PLENARY TALKS •
• INTERACTIVE PAPER SESSION • SPECIAL EVENTS • TECHNICAL SESSIONS •

www.electronicimaging.org

ADVANCING IMAGING SCIENCE 75 1947 IS&T 2022 imaging.org