Список використаних джерел

[1]. Т. Фесун. Елеваторна промисловість: традиції та інновації. Вітчизняний та світовий досвід. Київ: Національний університет харчових технолгій, 2021. [Електронний ресурс]. https://dspace.nuft.edu.ua/jspui/bitstream/123456789/34263/3/Elevator_industry_traditions_and_innovations._Domestic_and_world_experience.pdf

[2] Y.Bagljuk, and A. Shalyto. “SWITCH-technology. Algorithmic and programming methods in solution the logic control problems of shipping equipment”. Intern. conference on informatics and control. ICI & C'97. Proceedings. Vol.1, 1997.

[3]. Stateflow and Stateflow CoderUser’s Guide. USA:The MathWorks Inc, 2009.

[4] D. Shames, and D. Rosner. “Stateflow: A simulation environment for embedded systems”. IEEE Control Systems Magazine, 19(4), 49-57, 1999.

[5] M. Burke. Stateflow Best Practices. USA:The MathWorks Inc, 2012.

[6] https://uk.mathworks.com/products/simulink.html

[7] https://uk.mathworks.com/products/stateflow.html

References

[1]. T. Fesun. Elevator industry: traditions and innovations. Domestic and international experience. Kyiv: National University of Food Technologies, 2021. [Electronic resource]. https://dspace.nuft.edu.ua/jspui/bitstream/123456789/34263/3/Elevator_industry_traditions_and_innovations._Domestic_and_world_experience.pdf

[2] Y. Bagljuk, and A. Shalyto. "SWITCH-technology. Algorithmic and programming methods in solving the logic control problems of shipping equipment”. Intern. conference on informatics and control. ICI & C97. Proceedings. Vol. 1, 1997.

[3]. Stateflow and Stateflow CoderUsers Guide. USA: The MathWorks Inc, 2009.

[4] D. Shames, and D. Rosner. "Stateflow: A simulation environment for embedded systems". IEEE Control Systems Magazine, 19(4), 49-57, 1999.

[5] M. Burke. Stateflow Best Practices. USA: The MathWorks Inc, 2012.

[6] https://uk.mathworks.com/products/simulink.html

[7] https://uk.mathworks.com/products/stateflow.html

Отримана в редакції 26.04.2024. Прийнята до друку 20.05.2024. Received 26 Aprile 2024. Approved 14 May 2024. AvailableinInternet 23 July 2024

УДК 004.934.2

DEVELOPMENT OF VOICE ASSISTANT BASED ON NEURAL SYSTEMS

РОЗРОБКА ГОЛОСОВОГО ПОМІЧНИКА НА ОСНОВІ НЕЙРОСИСТЕМ

Svitlana Tyshchenko[1], Oleksandr Zhebko[2], Yevhen Zakharchenko[3], Yan Balyuk[4]
Світлана Тищенко[1], Олександр Жебко[2], Євген Захарченко[3], Ян Балюк[4]

[1,2,3,4] Mykolayiv National Agrarian University, Mykolayiv, Ukraine
ORCID: [1]https://orcid.org/0000-0001-7881-8740, [2]https://orcid.org/0009-0009-1604-5952
E-mail: [1]tyschenko@mnau.edu.ua, [2]zhebko@mnau.edu.ua, [3]jekazinferno@gmail.com, [4]balyuk.yangare@gmail.com

DOI: https://doi.org/10.15673/atbp.v16i2.2854

Abstract. *The development of a voice assistant based on neural systems is an actual line of research in the field of artificial intelligence. Existing voice assistants were analyzed and their advantages and disadvantages were identified in this paper. The following concepts were considered: natural language processing, and neural networks. Different methods of natural language processing significantly increase the quality of work of many programs, because they can*

interact with people at a natural level for humans without any problems. Modern voice assistants can help millions of people around the world at the same time, regardless of the time of the request. They are covering more and more spheres every year, from ordering food and taxis and making plans for the day to acting as translators.

The types of dialogue systems, the voice assistants` scope of use and their types, recurrent neural networks and data processing models Seq2seq and Transformer, as well as the work of the theoretical component of voice assistants based on neural networks, were considered. Node.js libraries were examined in detail, and an own voice assistant was created and outputs of this system were given. In addition, the possibilities of integrating the voice assistant with other systems and platforms, which expands its functionality and improves interaction with the user, were examined. General theoretical information about voice assistants and their components and principles of work were analysed. An environment for writing a voice assistant was chosen and a practical implementation of writing code for a voice assistant and an example of ready-made code were shown.

Анотація. *Розробка голосового асистента на основі нейронних систем є актуальним напрямком досліджень у сфері штучного інтелекту. У даній роботі проведено аналіз існуючих голосових асистентів, визначено їхні переваги та недоліки. Було розглянуто такі поняття: обробка природної мови, нейронні мережі. Різні методи обробки природної мови значно підвищують якість роботи безлічі програм, оскільки можуть взаємодіяти з людьми на природному для людини рівні без будь-яких проблем. Сучасні голосові асистенти одночасно здатні допомагати мільйонам людей по всьому світу, незалежно від часу запиту. Вони займають дедалі більше сфер з кожним роком, від замовлення їжі та таксі, складання планів на день, до функцій перекладачів.*

У результаті дослідження було розглянуто види діалогових систем, сфери використання голосових помічників та їх види, рекурентні нейронні мережі та моделі обробки даних Seq2seq й Transformer, а також роботу теоретичної складової голосових асистентів на основі нейронних мереж, детально розглянуто бібліотеки Node.js, а також створено власний голосовий асистент і наведено приклади роботи цієї системи. Додатково в роботі досліджено можливості інтеграції голосового асистента з іншими системами та платформами, що розширює його функціональність та покращує взаємодію з користувачем. Проаналізовано загальні теоретичні відомості щодо голосових асистентів та їх складових і принципів роботи. Було вибрано середовище для написання голосового асистента та відображено практичну реалізацію написання коду для голосового асистента та приклад готової роботи коду.

Keywords: voice assistant, machine learning, API, neural networks, Node.js.

Ключові слова: голосовий асистент, машинне навчання, API, нейронні мережі, Node.js.

I. Introduction

The voice assistant is a special system created to communicate with user in a natural way, in the form of a dialogue. Mostly, such models are created in the form of chatbots for communication, but a more complex structure allows the user not only to communicate, but also to interact with various services through the use of virtual assistants. That kind of systems require a high level of algorithm optimization and high-tech equipment to reduce training-time and improve the final quality of the neural network communication model.

II. Analytical review of the literature

ELIZA, created in 1965 and presented by Joseph Weizenbaum, is considered one of the first dialogue systems. ELIZA worked by extracting meaningful words from the user's sentence and substituting them into its patterns, thereby generating an answer. Although ELIZA was created for communication on the topic of psychotherapy, it is capable of communicating on general topics as well, but the quality of answers in this case is significantly reduced [1].

The first dialog system that passed a kind of Alan Turing test was PARRY, developed in 1972. This test itself is a normal dialogue between a group of people and some kind of computing device and is considered passed only if a certain part of the test takers, as a result of communication, declares that they communicated with a person, and not a computing machine. Psychiatrists could not distinguish the text received from PARRY from the dialogues of people suffering from schizophrenia. This was made possible with the help of both standard templates for answers that were implemented in ELIZA, and special software solutions that allowed PARRY to imitate some patterns of schizophrenia, which greatly increased the probability of passing the Alan Turing test in the end [2].

With the increase in computing power, dialog systems also developed. Thanks to this, it became possible to integrate neural network models related to natural language processing. That largely pushed many different companies to develop their dialogue systems that directly interacted not only with the user, but also with the system in which they were directly created. Such dialogue systems are usually called voice assistants. The peculiarity of voice assistants was that they are able not only to maintain a dialogue on various topics, to adapt to the manner of communication, but also to interact with system applications using voice requests from the user, which greatly improves the experience of using this or that system.

Apple was one of the first to introduce a full-fledged voice assistant. Created by them in 2011, Siri has become an integral part of the company's software. Able to perceive voice speech, answer questions and, most importantly, adapt to the manner of communication of each user, studying his preferences over a long period, thereby significantly improving the experience of using the company's products. Following Apple, many commercial companies started their development in this field, for example, in 2012, Google announced the Google Now voice assistant, after some time, in 2014, Microsoft Cortana and Amazon Echo became available, and in 2017, Yandex introduced Alice.

For large companies, voice assistants have become indispensable because they can provide basic support for millions of users while working completely autonomously. Virtual assistants perform more and more tasks than were previously performed by humans.

III. Object, subject and methods of research

The object of the study is a voice assistant based on neural networks.

The subject of the study is working tools with a voice assistant.

The aim of the work is to get acquainted with the theoretical aspects of a dialogue system for communication based on neural networks and to create own voice assistant using Node.js libraries and modules.

Task:

- to explore the theoretical aspects of voice assistants;
- to find methods that allow accurate processing of information;
- to choose software for implementing a voice assistant;
- to consider the libraries necessary for implementation;
- to analyze the results of the obtained system.

IV. Results

Let's consider the main types of dialog systems:

- task-oriented
- general purpose
- open domain
- closed domain.
- mixed

Let's consider each type in more detail.

1) task-oriented or Task-oriented systems - this is often the name given to the simplest voice assistants, because their main task is to help the user solve a certain task. Such systems can order pizza, transfer money, turn on the lights, or control subscriptions to various services. Thanks to neural network modules, voice assistants can both process the received information and remember special scenarios.

2) general purpose - such systems are designed for ordinary communication without special tasks. Most often they appear in the form of chatbots, unable to maintain dialogues on certain topics, as they are unable to understand the context. For the most part, the dialogue is built from unrelated questions from a person and answers from the dialogue system.

3) open domain or open-domain system - systems with an open domain are created to maintain dialogue on any free topic. Although they do not know how to perform certain actions as task-oriented systems, the skill of communication, understanding of dialogue, and context in general is a priority for such systems.

4) closed domain - such systems are very similar to the open domain, but their main difference is that they are not able to conduct dialogue in the area of the topics for which they were created.

5) mixed or Mixed system - such systems include most modern voice assistants. Their feature is that they can communicate on various topics, perform various tasks, make people's lives significantly easier, and constantly develop due to unique modular structures. Thus creating practically a project with open development, if desired, every person with the appropriate knowledge can add their skills to the conditional Alice from Yandex, such as the game of cities, and a translator [3].

Scope of use of voice assistants

Let's consider the main types of voice assistants:

1) voice assistants on devices: such as Siri on Apple devices, Google Assistant on Android devices, Alexa on Amazon Echo devices. They can perform various tasks such as searching for information, setting reminders, and controlling home automation devices.

2) voice assistants in cars: such as Apple CarPlay, Android Auto and Amazon Alexa Auto. They allow drivers to control their devices without taking their eyes off the road and perform a variety of tasks such as navigation and media control.

3) voice assistants in business: such as Microsoft Cortana and IBM Watson. They are used to manage business processes, analyze data and automate tasks.

4) voice assistants in healthcare: such as Nuance Dragon Medical and Suki AI. They are used to simplify the documentation of medical records and improve the quality of health care.

5) voice assistants in home automation: such as Google Home, Yandex Alice and Amazon Echo. They allow you to control home automation devices such as lighting, thermostats and smart door locks.

6) voice assistants in restaurants: such as Eatsa and Zume Pizza. They are used to automate food ordering and delivery processes.

7) voice assistants for communication: such as Google Duplex, Microsoft Xiaoice and Google Lambda, ChatGPT. These assistants have a more advanced natural language processing model to increase the quality of communication.

Each of these types of voice assistants has its characteristics and applications, but all of them are designed to simplify our lives and improve the quality of service.

Word embedding

A significant impetus to the development of dialogue systems for communication was the ability to represent words in vector form.

Word vectors are a numerical representation of words that maintain a semantic connection between them. So, for the word cat in a vector representation, one of the closest ones will be a dog. Such a connection is determined by the frequency with which these words occur together. Algorithms of vector representation work in such a way that only the distance between vectors is meaningful, and not the features of the vector itself. Thus, if the distance between the vectors is minimal, it means that these words are often mentioned in a general context, or one without the other is simply not used.

The process of presenting words in vector form: a) is the input layer, b) embedding layer, c) the output layer.

Let's analyze each of the stages in detail. An input layer that accepts one word in one-hot format. This format consists of the fact that the word is encoded by a binary vector with one unit representing the position of the word in the dictionary. The length of the vector dictionary is equal to the total number of non-repeating words.

The output layer is a vector whose length is equal to the size of the dictionary. This is the only layer that has a softmax activation feature. Each neuron in this layer shows the probability that a word belongs to a certain context.

After creating the dictionary, you need to select the input word and its context. The context refers to the closest words formed depending on the size of the context window.

The number of context words depends on the number of sentences and the size of the window. The word2vec algorithm searches for all sentences with the input word and the context around it. Vector representations of words are a very powerful tool that improves modern machine learning models. Significantly increasing their quality of learning dialogue systems for communication based on neural network technologies, due to the word2vec module [4].

Recurrent neural networks

Nowadays, there are already successful ways to generate text. It all started with the emergence of recurrent neural networks (RNN). This is a type of neural networks, where connections between elements form a directed sequence. Thanks to this, it becomes possible to work out a series of events in which not only the content is important, but also the order. Unlike known multilayer perceptrons, recurrent networks can use their internal memory to process sequences of arbitrary length. Such networks are applicable in tasks where something integral is broken into parts, because, unlike forward propagation neural networks, in recurrent neural networks, in addition to the new data that the neuron receives, information about the previous state of the network is transmitted. However, such networks have a problem, they remember recent data well, but are not able to store them for a long time.

Due to the shortcoming of RNN, its LSTM (Long short-term memory) modification was released after a short amount of time. In this network, cells with neurons have been changed: they have added a filter that is responsible for whether this information should be stored or not. As the learning progresses, the state of the cells will change: information will be added or removed from the state of the cell by filters that control the flow of information at the inputs and outputs of the module based on some conditions. They consist of a sigmoidal neural network layer and a pointwise multiplication operation.

Model Seq2seq

The seq2seq (sequence to sequence) model was created to solve many problems with different data. It consists of two recurrent networks, which are called Encoder and Decoder parts. The Encoder is responsible for processing the input data and consists of a stack of LSTM cells, which are responsible for encapsulating all input data, and also performing the function of the initial state of the Decoder stack. The decoder, in turn, also consists of a stack of LSTM cells, each of which takes previous hidden states and generates its own, so the output will contain all previous states. The initial state is determined using the softmax function to create a probability vector that will help us determine the final result.

This architecture helps to map sequences to each other, but it is not able to remember long sentences due to the large number of recurrent modules.

Seq2seq has revolutionized the translation process using deep learning. During the translation, not only the current word, but also its neighborhood is taken into account. Currently, this model is used for many different applications, such as: creating subtitles for images, dialogue systems, and text summarization [5].

Transformer data processing model

Transformer is a data processing model developed in 2017 by researchers at Google Brain that enables the processing of data sequences such as text, audio, and video. This model was developed to solve machine translation tasks, but it can also be used for other natural language processing tasks, such as text generation, text classification, question-and-answer systems, etc. Transformer uses an attention engine that allows the model to focus on the most important words and context, enabling it to produce more accurate predictions and responses. This model consists of two main components: encoder and decoder. The encoder takes a sequence of data as input and converts it into a hidden representation, which is then passed to the decoder. The decoder uses the hidden representation to generate output, such as a translation or an answer to a question.

Transformer has several advantages over other natural language processing models. It can process texts in context, enabling it to make more accurate predictions and responses. It can also use an attention mechanism, which allows it to focus on the most important words and context, which improves the quality of text processing, and also significantly increases the quality and speed of learning, outperforming other data processing models such as Seq2Seq and various convolutional neural networks.

This model was used to create different models of natural language processing, such as GPT (Generative Pre-trained Transformer), BERT (Bidirectional Encoder Representations from Transformers), RoBERTa (Robustly Optimized BERT Pretraining Approach) and others. These models have high accuracy and efficiency in natural language processing tasks and are widely used in various applications and software products [6].

Choosing a development environment for creating a voice assistant

There are many programming languages that can be used to write a voice assistant to communicate and receive various data: Python, C++, and others. Each language has its characteristics, pros and cons, but Python is considered the most popular in the community of developing neural network models, and here's why.

Python is a high-level programming language that has proven itself well in completely different fields, from web development, and data analysis, to programming various events and more.

Thanks to the popularity and active community in Python, many different libraries are closely related to neural networks and machine learning in general. The language is platform-independent, so it can be adapted to almost any operating system. Another advantage of Python is related to its openness - it is built based on Open Source technologies, so developers can access any language stack.

Speaking about the problems of Python, it is worth noting the difficulty of tracking errors in the code, which in some cases significantly increases the time of creating a project and its further improvement. However, it is worth noting that this problem is mostly solved by modern development environments (IDEs).

One of the oldest popular programming languages is C++. Having the capabilities of both a low-level and a high-level programming language at the same time, in the context of machine learning, C++ provides a higher level of control and efficiency than other programming languages. The flexibility of the language is suitable for resource-intensive applications. Given that C++ is a statically typed language, it can perform tasks at relatively high speed.

As for its disadvantages, the main one is that creating new applications based on C++ requires writing a large amount of complex code, which takes a lot of time and can cause great difficulties in maintenance. The C++ language is certainly difficult to master, and when creating new projects in it, novice programmers very often make casual mistakes.

Most programming languages do not have the necessary libraries for full-fledged implementation of the dialog system, they are difficult to master or require high computing power. Thus, the most popular and convenient programming language for machine learning is Python. Other environments also have their advantages, but currently, the code written by her is in greater demand on the market.

However, despite the clear pros and cons of these development environments, the main problem will be the lack of resources to create a really good and ready-to-use communication model and further integrate it into a voice assistant.

In these situations, API technology comes to the rescue, which allows you to use existing neural networks in projects, without spending time and resources on training your own processing models.

The main advantages of API:

1) reduction of development time: Using the API of existing neural networks allows you to significantly reduce the time needed to develop a voice assistant. Instead of spending months creating and training a model, you can use a ready-made API and focus on developing the interface and other functions of the assistant.

2) improved accuracy: Neural networks used in off-the-shelf APIs are usually trained on huge amounts of data and go through many optimization steps, enabling them to achieve high accuracy. Self-built models may not have enough data to train on and may have lower accuracy.

3) better support and updates: Big companies like Google, Amazon, and Microsoft have a large development team and can provide better support and frequent updates to their APIs. This allows you to get new features and improvements without having to update the models yourself.

4) scalability: Using the API of existing neural networks allows you to quickly scale the voice assistant as needed. You can easily increase the number of requests without worrying that the model will not be able to cope with the load.

5) economic efficiency: The creation and training of models require significant costs for computing resources and the payment of specialists. The use of ready-made APIs allows you to reduce these costs and focus on the development of business logic and user interface.

In general, using existing neural network APIs enables faster, more accurate, and more cost-effective development of voice assistants.

Since the API completely removes the need to write neural networks, the focus will be on speed and appearance. Since the voice assistant will be written in the form of a bot for Telegram, it is capable of receiving voice requests and composing appropriate responses, remembering the history of messages, and also understanding the context of the offer.

Node.js will be used to work with the API and further integrate the model into the chatbot.

The main advantages of this programming language:

1. high performance: Node.js uses an asynchronous input/output (I/O) model, which enables it to handle a large number of requests without blocking threads. This makes Node.js ideal for developing high-load applications.

2. ease of development: Node.js uses the JavaScript language that many developers already know. This makes developing applications on Node.js much easier and faster.

3. scalability: Node.js allows you to easily scale applications horizontally (adding new servers) and vertically (adding resources to existing servers).

4. Huge community: Node.js has a huge community of developers and many libraries and modules that can be used to speed up development.

5. Open Source: Node.js is an open source project that allows developers to easily make their changes and improvements to the platform.

6. WebSocket support: Node.js supports the WebSocket protocol, which enables browsers and servers to exchange data in real time.

7. Broad application: Node.js is used to develop various types of applications, including web applications, APIs, microservices, mobile applications, and more.

Thus, this programming language provides high-quality work with the support of many people connected to this voice assistant at the same time, as well as faster work with the API. That greatly simplifies and speeds up the overall operation of the system [7].

Description of libraries

To develop a voice assistant, let's stop at Node.js. The following libraries and various additional packages will be needed to implement most functions: Telegraf, axios, OpenAI, fluent-ffmpeg.

Let's consider each of the libraries in more detail.

Telegraf is a Node.js library designed for the development of chatbots on the Telegram platform. It provides a simple and convenient way to create and manage a bot using the Telegram Bot API.

The Telegraf library has several functions. She can:

1) process incoming messages and requests from users;
2) create buttons and menus for interaction with the user;
3) send messages, images, audio and video files to users;
4) update messages using already sent methods.

Telegraf uses an event model, so when creating a bot, you can define how to handle specific messages using events such as: text, photo, audio, video, location, contact, voice and many others. In addition, Telegraf has built-in support for sessions, which allow you to store user data, such as username or bot settings.

Telegraf provides a powerful and flexible tool for creating chatbots on Telegram, and its simplicity and ease of use allow you to quickly create bots without the need for special programming knowledge.

Axios is a JavaScript library that allows you to make HTTP requests using promises. It allows you to send and receive data from a web server, supports request and response interceptors, allows you to quickly configure requests, and much more.

A few examples of what you can do with axios:

1) execute GET and POST requests to download and send data to the server;
2) use URL parameters to pass data in requests;
3) set request headers, such as an authentication header or a JSON header;
4) execute requests with a timeout and handle errors;
5) use interceptors to change request or response parameters;
6) process responses using promises or a callback.

Axios also supports multiple platforms, including browsers and Node.js, and can be used to make requests to any web server, not just APIs.

Fluent-ffmpeg is a JavaScript library based on ffmpeg that is used to process and convert multimedia files (audio and video). The library allows you to perform almost any operation on multimedia files, such as slicing, converting, changing audio and video effects, applying filters, cropping, text overlay, and more.

With the help of Fluent - ffmpeg, you can easily create players for creating videos with specified parameters and convert video and audio to various formats. In addition, it has plugins that allow you to work with video and audio files using JavaScript calls and functions.

Examples of tasks that can be solved using Fluent-ffmpeg include:

1) converting files from one video format to another;
2) changing the resolution, bitrate, frames per second (FPS), sound formats and other audio/video parameters;
3) trimming of video files, creation of GIF-animation;
4) processing of video and audio files by applying special effects;
5) removing frames from video files.

Fluent - ffmpeg lets you use almost all ffmpeg options and features during development. The general approach to interacting with this library consists of two steps: defining the input file and results specified as an object, and defining the sequence of commands to be executed. Part of the sequence of commands can be implemented as JavaScript functions defined in the application.

OpenAI also has JavaScript and Node.js libraries that provide an interface to use OpenAI services directly from your JavaScript application. The OpenAI library for Node.js supports many of the services available in the Python version[8].

With the OpenAI library for Node.js you can:

1) work with the OpenAI API to perform natural language processing tasks;
2) use deep learning methods with OpenAI and create your own machine learning models based on this technology;
3) perform tasks related to text generation, answering questions, determining the context of the text, and many others;
4) use pre-trained models to perform language processing tasks;
5) send requests to the OpenAI API using JavaScript or Node.js through API methods.

The OpenAI library for Node.js has an intuitive interface and allows you to work with the OpenAI API from JavaScript applications. This makes it a powerful tool for building applications that use natural language processing and machine learning.

Thus, Node.js libraries are optimal for working with APIs to create a stable operation of a voice assistant, despite possible high user loads, the asynchronous model of the language allows you to perform many actions while consuming a small amount of resources.

Designing a general project structure

The main goal of this stage is to prepare the virtual environment to work with necessary libraries. In Node.js, similar to Python, external components are installed through commands in the terminal.

The command: *"npm init -w"* creates the workspace. It allows to group packages and use them together. It can be used to manage modules consisting of several packages, as well as to develop and test several scripts at once (Figure 1).

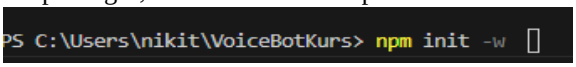

Fig. 1 – Creating a virtual environment

Voice messages representation in the text form

The ogg.js file is fully responsible for this stage. Voice messages by Telegram users are sent in the ogg format, which doesn`t comply with the standards set by the company that provides the API for the use of neural networks, so there is a demand for a processing model that will record information from voice messages in the text form.

For voice assistant designing 2 libraries were connected: Axios and ffmpeg. So, this task was divided into 3 parts:

1) receiving the ogg file by downloading from chat (Figure 2).
2) converting this file into mp3 format (Figure 3).
3) transformation voice message from mp3 to text format (Figure 4).

Let's take a closer look. The ogg file is downloaded using a direct link that is hidden in the telegram system.

```
async create(url, filename) {
    try {
        const oggPath = resolve(__dirname, '../voices', `${filena
        const response = await axios({
            method: 'get',
            url,
            responseType: 'stream',
        })
        return new Promise(resolve => {
            const stream = createWriteStream(oggPath)
            response.data.pipe(stream)
            stream.on('finish',() => resolve(oggPath))
        })


    } catch (e) {
        console.log('Error whlie creating ogg', e.message)
    }

}
}
```

Fig. 2 – Receiving the ogg file

This model is asynchronous to achieve parallelism, allowing modules to run at the same time. The input function receives a URL and a file name, which coincides with the user-ID, who wrote this request. And then, this information transfers to oggPath. The toMp3 function is used for the conversion ogg file into a mp3 file.

```
toMp3(input, output) {
    try {
      const outputPath = resolve(dirname(input), `${output}.mp3`)
      return new Promise((resolve, reject) => {
        ffmpeg(input)
          .inputOption('-t 30')
          .output(outputPath)
          .on('end', () => {
            removeFile(input)
            resolve(outputPath)
          })
          .on('error', err => reject(err.message))
          .run()
      })
    } catch (e) {
        console.log('Error whlie creating mp3', e.message)
    }
}
```

Fig. 3 – Creating an mp3 file based on .ogg

In this case, toMp3 function takes ogg file as input and a file in mp3 format is created on its basis with the help of the ffmpeg library, and the old ogg file is no needed and will deleted, freeing up the space on the disk.

The OpenAI library is necessary for converting voice data into text format. It has several data processors for this task, but the most affordable of them is the "whisper-1" model. Its advantage is high speed, as well as high-quality processing of natural language.

```
async transcription(filepath) {
    try {
        const response = await this.openai.createTranscriptio
            createReadStream(filepath),
            'whisper-1'
        )
        return response.data.text
    } catch (e){
        console.log('Error whlie transcription', e.message)
    }
}
```

Fig. 4 – Conversion of voice data into text format

Implementation of main components

This part is dedicated to the Telegram bot and the neural network, whose API will be used as a basis for creating a voice assistant. The voice assistant perceives both written text and live speech. Let's analyze in more detail how we will achieve such result.

```
bot.on(message('text'),async (ctx) => {
    ctx.session ??= INITIAL_SESSION
    try {
        await ctx.reply(code('Сообщение принял. Жду ответ от серв

        ctx.session.messages.push({role: openai.roles.USER, conte

        const response = await openai.chat(ctx.session.messages)

        ctx.session.messages.push({
            role: openai.roles.ASSISTANT,
            content: response.content
        })

        await ctx.reply(response.content)
    } catch (e){
        console.log('Error while voice message', e.message)
    }
})
```

Fig. 5 – Analysis of the written text

In this case, the second codeline serves to remember the context, next the constructor for finding errors, and the system message, then there is a direct request using the API and obtaining the corresponding result, with subsequent output one as a response (Figure 5).

When the request has a voice format, the mechanisms responsible for processing voice data and their further transformation start to work. The structure of interaction with the bot begins to change (Figure 6).

```
bot.on(message('voice'),async (ctx) => {
    ctx.session ??= INITIAL_SESSION
    try {
        await ctx.reply(code('Сообщение принял. Жду ответ от серв
        const link = await ctx.telegram.getFileLink(ctx.message.v
        const userId = String(ctx.message.from.id)
        console.log(link.href)
        const oggPath = await ogg.create(link.href, userId)
        const mp3Path = await ogg.toMp3(oggPath, userId)

        const text = await openai.transcription(mp3Path)
        await ctx.reply(code(`Ваш запрос: ${text}`))

        ctx.session.messages.push({role: openai.roles.USER, conte

        const response = await openai.chat(ctx.session.messages)

        ctx.session.messages.push({
            role: openai.roles.ASSISTANT,
            content: response.content
        })

        await ctx.reply(response.content)
    } catch (e){
        console.log('Error while voice message', e.message)
    }
})
```

Fig. 6 – Analysis of voice messages

To the already standard string for memorizing the context, there are constants with a link to download voice messages, and the user-ID who sent this message. Then there are 3 methods of processing voice messages and further output a response. The final part is the model for connection of neural network API (Figure 7).

```
class OpenAI {
    roles = {
        ASSISTANT: 'assistant',
        USER: 'user',
        SYSTEM: 'system',
    }

    constructor(apiKey) {
        const configuration = new Configuration({
            apiKey,
        });
        this.openai = new OpenAIApi(configuration);
    }

    async chat(messages) {
        try {
            const response = await this.openai.createChatCompleti
                model: 'gpt-3.5-turbo',
                messages,
            })
            return response.data.choices[0].message
        } catch(e){
            console.log('Error whlie ChatBot', e.message)
        }
    }
```

Fig. 7 – Integration the neural network to the bot

The large OpenAI class is divided into 3 subclasses, each responsible for important things. Roles is responsible for roles: system, user or assistant. Constructor is responsible for the API_KEY, and it allows to connect remotely trained neural networks, passing data for requests and returning responses. Chat is responsible for the neural network model and for creating a personal space for each user, so that users do not disturb each other. The result of communication with the voice assistant (Figure 8).

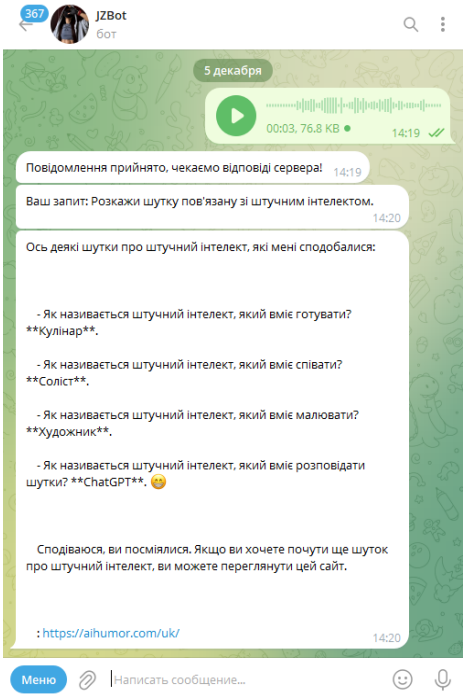

Fig. 8 – The written answer to a voice question

V. Conclusions

The main goal achieved as a result of the performed work was to develop a voice assistant for communication based on neural networks. To fulfill the target goal, the following tasks were solved:

- theoretical aspects of dialogue systems were considered;
- a voice information processing mechanism was created;
- selected software for the voice assistant;
- libraries for voice conversion into text were researched;
- the results of the developed system were analyzed.

Based on the research performed further development of the voice assistant will be planned, new functions and capabilities will be developed, and already implemented algorithms will be optimised.

Список використаних джерел
1. Діалогова система ELIZA. URL: https://en.wikipedia.org/wiki/ELIZA (дата звернення 10.02.2024).
2. Сучасні голосові асистенти. URL: https://bit.ua/2021/07/dlya-sproshhennya-roboty-dobirka-najzruchnishyh-golosovyh-pomichnykiv/ (дата звернення 12.02.2024).
3. Технологія Word2vec. URL: https://en.wikipedia.org/wiki/Word2vec (дата звернення 10.02.2024).
4. Навчання моделі seq2seq. URL: https://coderlessons.com/tutorials/mashinnoe-obuchenie/uchebnik-nltk/10-model-seq2seq (дата звернення 11.02.2024).
5. Бібліотеки Python. URL: https://docs.python.org/3/library/index.html (дата звернення 16.02.2024).
6. Драйвера Nvidia. URL: https://developer.nvidia.com/cuda-downloads (дата звернення 18.02.2024).
7. API ключі. URL: https://blog.bitbanker.org/ru/chto-takoe-api-klyuch-i-kak-ispolzovat-ego-bezopasno/chto-takoe-klyuch-api (дата звернення 09.02.2024).
8. Node.js. URL: https://nodejs.dev/en/learn/ (дата звернення 03.02.2024).
9. Документація ChatGPT. URL: https://chatgpt.com.ua/docs (дата звернення 07.02.2024).
10. Підключення API. URL: https://platform.openai.com/docs/guides/chat (дата звернення 07.02.2024).

References
[1] ELIZA dialog system [Online]. Available: https://en.wikipedia.org/wiki/ELIZA. [Accessed Feb. 10, 2024].
[2] Modern voice assistants [Online]. Available: https://bit.ua/2021/07/dlya-sproshhennya-roboty-dobirka-najzruchnishyh-golosovyh-pomichnykiv. [Accessed Feb. 12, 2024].
[3] Word2vec technology [Online]. Available: https://en.wikipedia.org/wiki/Word2vec. [Accessed Feb. 10, 2024].
[4] Teaching the seq2seq model [Online]. Available: https://coderlessons.com/tutorials/mashinnoe-obuchenie/uchebnik-nltk/10-model-seq2seq. [Accessed Feb. 11, 2024].
[5] Python libraries [Online]. Available: https://docs.python.org/3/library/index.html. [Accessed Feb. 16, 2024].
[6] Nvidia drivers [Online]. Available: https://developer.nvidia.com/cuda-downloads. [Accessed Feb. 18, 2024].
[7] API keys [Online]. Available: https://blog.bitbanker.org/ru/chto-takoe-api-klyuch-i-kak-ispolzovat-ego-bezopasno/chto-takoe-klyuch-api. [Accessed Feb. 9, 2024].
[8] Node.js [Online]. Available: https://nodejs.dev/en/learn. [Accessed Feb. 3, 2024].
[9] ChatGPT documentation [Online]. Available: https://chatgpt.com.ua/docs. [Accessed Feb. 7, 2024].
[10] API connection [Online]. Available: https://platform.OpenAI.com/docs/guides/chat. [Accessed Feb. 7, 2024].

Отримана в редакції 08.05.2024. Прийнята до друку 30.05.2024. Received 08 May 2024. Approved 30 May 2024. AvailableinInternet 30 July 2024

УДК 621.18-182.2

ДО ПИТАННЯ АВТОМАТИЗАЦІЇ КОТЛОАГРЕГАТУ ДКВр-10-13 В КОТЕЛЬНЯХ ХАРЧОВИХ ПІДПРИЄМСТВ

ON THE ISSUE OF AUTOMATION OF DKVr-10-13 BOILER UNITS IN BOILER ROOMS OF FOOD ENTERPRISES

Черняк О.І.[1], Світий І.М.[2]
Chernyak O.I.[1], Svity I.M.[2]

[1,2] Одеський національний технологічний університет, Одеса, Україна
1,2 Odessa National University of Technology, Odesa, Ukraine
ORCID ID: [1] https://orcid.org/0009-0001-5329-575X, [2] https://orcid.org/0000-0001-8524-5565
E-mail: [1]alex_c@te.net.ua, [2]switiy@yahoo.com

DOI: https://doi.org/10.15673/atbp.v16i2.2839

Анотація. *Технологічні процеси на підприємствах харчової промисловості пов'язані з великим споживанням*