ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки

КАФЕДРА ЕОМ

Інтегрована система штучного інтелекту в телеграм-бот для вивчення англійської мови

Кваліфікаційна робота

Другий (магістерський) рівень

АВТОР:
РЕУКА К. О.
СТУД. ГР. СПм-22-1

КЕРІВНИК:
ІЛЬЇНА І. В.
ДОЦ. КАФ. ЕОМ

Актуальність проблеми

Відсутність партнера для спілкування

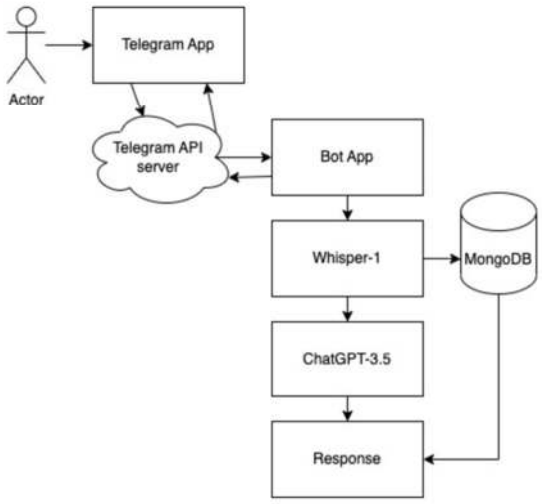
Неможливість самостійного покращення розмовного рівня

Страх та невпевненість

Постановка задачі

- Можливість діалогу за допомогою голосових повідомлень
- Можливість отримання синонімів для часто вживаних слів
- Можливість використання заданого рівня англійської мови
- Можливість перевірки граматики
- Можливість отримання інформації стосовно правильного використання слів

Архітектура



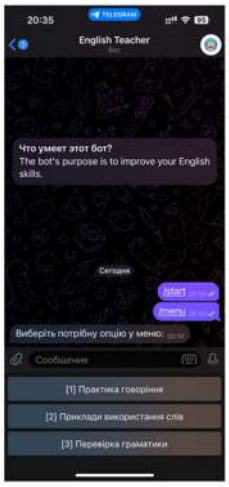
Порівняння моделей розпізнавання мовлення

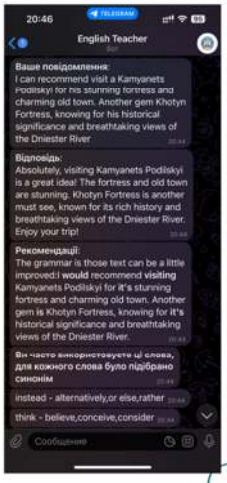
Набір даних	Whisper	wav2vec 2.0	Kaldi
Розмовний AI	51.3	57.1	68.1
Телефонні дзвінки	46.8	49.4	87.9
Зустрічі	63.7	64.8	81.4
Відео	9.8	22.8	42.1

Порівняння моделей генерації тексту

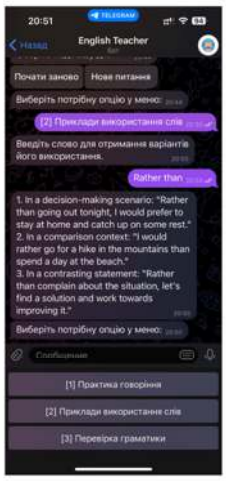
Назва	Використання	Набір даних	Дата випуску
GPT-2	Загальне	40 ГБ тексту та 8 мільйонів документів	14 лютого 2019
GPT-3	Загальне	570 ГБ звичайного тексту. В основному містить дані з наборів даних CommonCrawlm WebText, англійської вікіпедії	11 червня 2020

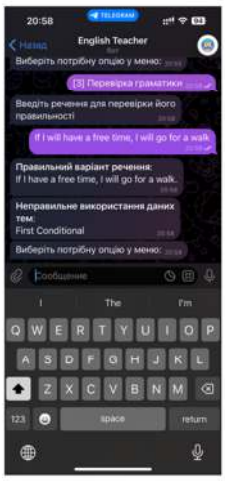
Режим практики говоріння





Використання слів та перевірка граматики





Висновки

- **Вирішено проблеми:**
  - **неможливість самостійного покращення розмовного рівня**
  - **відсутність партнера для спілкування**
  - **страх та невпевненість**

ДОДАТОК Б

Програмна реалізація

Б. 1 finalStage.js

```
      const processVoiceMessage = require("../../tools/processVoiceMessage")
      const removeFile = require("../../tools/files/removeFile")
      const { Extra } = require("telegraf")
      const getAnswerFromOpenAI = require("../../tools/getAnswerFromOpenAI")
      const { showTopUsedWords } = require("../../tools/showTopUsedWords")
      const { showRestartContinueMenu } =
require("../../tools/showRestartContinueMenu")

      async function finalStage(ctx) {
      const textFromVoice = await processVoiceMessage(ctx)
      await removeFile(ctx.session.filePath)
      ctx.session.messages.push(textFromVoice)
      const userMessage = textFromVoice.replaceAll(/[-.!()]/g, (match) =>
`\\${match}`)
      await ctx.reply(`*Ваше повідомлення*:\n${userMessage}`,
Extra.markdownV2())
      const { answerResponse, suggestionResponse } = await
getAnswerFromOpenAI(ctx.session.messages)
      const answer = answerResponse.content.replaceAll(/[-.!()]/g, (match) =>
`\\${match}`)
      const suggestion = suggestionResponse.content.replaceAll(/[-.!()]/g,
(match) => `\\${match}`)
      await ctx.reply(`*Відповідь*:\n${answer}`, Extra.markdownV2())
      await ctx.reply(`*Рекомендації*:\n${suggestion}`, Extra.markdownV2())

      await showTopUsedWords(ctx.message.from.id, ctx)
      await showRestartContinueMenu(ctx)
      }

      module.exports = {
      finalStage
}
```

Б. 2 checkGrammarScene.js

```
      const WizardScene = require("telegraf/scenes/wizard")
      const { processSentence } = require("./../tools/processSentence")
      const { showMainMenu } = require("../tools/showMainMenu")

      const checkGrammarScene = new WizardScene("checkGrammar",
      async (ctx) => {
            await ctx.reply("Введіть речення для перевірки його правильності")
            return ctx.wizard.next()
      },
      async (ctx) => {
            const response = await processSentence(ctx.message.text)
            await ctx.replyWithMarkdown(response)
            await showMainMenu(ctx)
            return ctx.scene.leave()
      })

module.exports = checkGrammarScene
```

Б. 3 checkDialogueScene.js

```
      const WizardScene = require("telegraf/scenes/wizard")
      const { connectToDataBase, getRandomQuestion } =
require("../tools/database/databaseInterractions")
      const { finalStage } = require("./stages/finalStage")

      const dialogueScene = new WizardScene(
      "continueDialogue",
      async (ctx) => {
            await connectToDataBase()
            const question = await getRandomQuestion(ctx.session.chosenTopic,
ctx.session.chosenEnglishLevel)
            if (ctx.session.messages) {
                  ctx.session.messages.push(question)
            } else {
                  ctx.session.messages = []
            }
            await ctx.reply(question)
            return ctx.wizard.next()
      },
      async (ctx) => {
            await finalStage(ctx)
            return ctx.scene.leave()
      }
      )

module.exports = dialogueScene
```

Б. 4 dialogueScene.js

```
      const WizardScene = require("telegraf/scenes/wizard")
      const { selectConversationTopic } =
require("../tools/selectConversationTopic")
      const { connectToDataBase, getRandomQuestion } =
require("../tools/database/databaseInterractions")
      const { Extra } = require("telegraf")
      const { selectEnglishLevel } = require("../tools/selectEnglishLevel")
      const { finalStage } = require("./stages/finalStage")



      const dialogueScene = new WizardScene(
      "dialogue",
      async (ctx) => {
            ctx.session.messages = []
            await ctx.reply("Режим *Діалог*", Extra.markdownV2())
            await selectEnglishLevel(ctx)
            return ctx.wizard.next()
      },
      async (ctx) => {
            ctx.session.chosenEnglishLevel = ctx.callbackQuery.data
            await ctx.reply(`Ви обрали: *${ctx.session.chosenEnglishLevel}*`,
Extra.markdownV2())
            await selectConversationTopic(ctx)
            return ctx.wizard.next()
      },
      async (ctx) => {
            ctx.session.chosenTopic = ctx.callbackQuery.data
            await ctx.reply(`Ви обрали: *${ctx.session.chosenTopic}*`,
Extra.markdownV2())
            await connectToDataBase()
```

```
            const question = await getRandomQuestion(ctx.session.chosenTopic,
ctx.session.chosenEnglishLevel)
            ctx.session.messages.push(question)
            await ctx.reply(question)
            return ctx.wizard.next()
      },
      async (ctx) => {
            await finalStage(ctx)
            return ctx.scene.leave()
      }
      )

module.exports = dialogueScene
```

Б. 5 wordExamplesScene.js

```
      const WizardScene = require("telegraf/scenes/wizard")
      const { processWord } = require("./../tools/processWord")
      const { showMainMenu } = require("../tools/showMainMenu")

      const wordsExampleScene = new WizardScene("wordExamples",
      async (ctx) => {
            await ctx.reply("Введіть слово для отримання варіантів його
використання.")
            return ctx.wizard.next()
      },
      async (ctx) => {
            const response = await processWord(ctx.message.text)
            await ctx.replyWithMarkdown(response)
            await showMainMenu(ctx)
            return ctx.scene.leave()
      })

module.exports = wordsExampleScene
```

Б. 6 databaseInterractions.js

```
      const mongoose = require("mongoose")

      const topicSchema = new mongoose.Schema({
      name: String,
      level: String,
      questions: [String],
      })

      const userWordsSchema = new mongoose.Schema({
      userId: String,
      words: {
            type: Map,
            of: Number,
      },
      })

      const Topic = mongoose.model("Topic", topicSchema)
      const userModel = mongoose.model("User", userWordsSchema)

      async function connectToDataBase() {
      await mongoose.connect("mongodb://127.0.0.1/test", { useNewUrlParser: true
, useUnifiedTopology: true} )
      }
```

```
async function getQuestionsByTopicAndLevel(topicName, englishLevel) {
const results = await Topic.find({ name: topicName, level: englishLevel })
return results[0].questions
}

async function getRandomQuestion(topicName = "Traveling", englishLevel =
"A1") {
const questions = await getQuestionsByTopicAndLevel(topicName,
englishLevel)
const randomIndex = Math.floor(Math.random() * questions.length)
return questions[randomIndex]
}

async function addWordsCountForUser(userId, wordsObject) {
const existingUser = await userModel.findOne({ userId })

if (existingUser) {
      Object.keys(wordsObject).forEach((key) => {
            if (existingUser.words.has(key)) {
                  existingUser.words.set(key, existingUser.words.get(key)
+ wordsObject[key])
            } else {
                  existingUser.words.set(key, wordsObject[key])
            }
      })
      existingUser.markModified("words")
      await existingUser.save()
} else {
      const newUser = new userModel({
            userId,
            words: new Map(Object.entries(wordsObject)),
      })
      await newUser.save()
}
}

async function getTopUsedWords(userId, threshold = 5) {
const results = await userModel.find({ userId })
const allUsedWords = Array.from(results[0].words.keys())
return allUsedWords.filter((word) => results[0].words.get(word) >=
threshold)
}

module.exports = {
connectToDataBase,
getRandomQuestion,
addWordsCountForUser,
getTopUsedWords,
}
```

Б. 7 downloadFile.js

```
const axios = require("axios")
const fs = require("fs")

const downloadFile = async (url, destination) => {
const response = await axios({
      method: "GET",
      url,
      responseType: "stream",
})

const writer = fs.createWriteStream(destination)
```

```
      response.data.pipe(writer)

      return new Promise((resolve, reject) => {
            writer.on("finish", resolve)
            writer.on("error", reject)
      })
      }

module.exports = downloadFile
```

Б. 8 removeFile.js

```
      const fs = require("fs")

      const removeFile = (path) => {
      return new Promise((resolve, reject) => {
            fs.unlink(path, (err) => {
                  if (err) {
                        reject(err)
                  } else {
                        resolve()
                  }
            })
      })
      }

module.exports = removeFile
```

Б. 9 getAnswerFromOpenAI.js

```
      const { chat } = require("../openai")

      async function getAnswerFromOpenAI(messages) {
      const answerRequirements =
              "Imagine that you are a real person. " +
              "Please, don't tell 'as an AI' and something about AI. " +
              "Comment the idea in the next message. Answer length is 2-3
sentences."
      const answerResponse = await chat(answerRequirements.concat(messages))

      const suggestionRequirements =
              "Check grammar, give possible suggestions for improvements in
terms of English language." +
              "How we can rephrase the same text but make it more clear." +
            "Explain my every mistake." +
              "Maximum answer length is 5 sentences." +
              "The following sentences:"
      const userAnswer = messages.slice(1)
      const suggestionResponse = await
chat(suggestionRequirements.concat(userAnswer))
      return { answerResponse, suggestionResponse}
      }

module.exports = getAnswerFromOpenAI
```

Б. 10 getUsedWords.js

```
      function getUsedWords(textMessage) {
```

```
      const splittedWords = textMessage
            .replaceAll(/[-,.!?\\///]|\/\//g, " ")
            .split(" ")
      const wordsWithValidLength = splittedWords.filter((word) => word.length >
3)
      const wordCountObject = wordsWithValidLength.reduce((wordsObject, word) =>
{
            wordsObject[word] = (wordsObject[word] || 0) + 1
            return wordsObject
      }, {})
      return wordCountObject
      }

module.exports = getUsedWords
```

Б. 11 processSentence.js

```
      const { chat } = require("../openai")

      async function processSentence(sentence) {
      const response = await chat(`Check grammar and find mistakes in a
sentence. You don't need to translate the sentence to ukrainian. Just explain my
every mistake using the ukrainian language, to make sure I understand everything
correctly: ${sentence}`)
      return response.content
      }

module.exports = { processSentence }
```

Б. 12 processVoiceMessage.js

```
      const downloadFile = require("./files/downloadFile")
      const { transcription } = require("../openai")
      const process = require("process")

      async function processVoiceMessage(ctx) {
      try {
            const voiceFileId = ctx.message.voice.file_id
            const voiceFile = await ctx.telegram.getFile(voiceFileId)
            ctx.session.filePath = voiceFile.file_path
            const voiceUrl =
`https://api.telegram.org/file/bot${process.env.TELEGRAM_TOKEN}/${ctx.session.fi
lePath}`
            await downloadFile(voiceUrl, ctx.session.filePath)
            return transcription(ctx.session.filePath)
      } catch (e) {
            await ctx.reply("Error while processing voice message", e)
      }
      }

module.exports = processVoiceMessage
```

Б. 13 processWord.js

```
      const { chat } = require("../openai")

      async function processWord(word) {
      const response = await chat(`Give a 3 examples of using this word in
different situations: ${word}`)
      return response.content
```

```
      }

module.exports = { processWord }
```

Б. 14 selectConversationTopic.js

```
      const { Markup } = require("telegraf")

      const topics = [
      { text: "Traveling", callback_data: "Traveling" },
      { text: "Hobbies", callback_data: "Hobbies" },
      { text: "Technology", callback_data: "Technology" },
      { text: "Entertainment", callback_data: "Entertainment" },
      { text: "Career", callback_data: "Career" },
      ]

      async function selectConversationTopic(ctx) {
      const keyboard = Markup.inlineKeyboard(
            topics.map((topic) => Markup.callbackButton(topic.text,
topic.callback_data)),
            { columns: 2 }
      )
      await ctx.reply("Оберіть тему запитань:", keyboard.extra())
      }

module.exports = { selectConversationTopic }
```

Б. 15 selectEnglishLevel.js

```
      const { Markup } = require("telegraf")

      const englishLevels = [
      { text: "A1", callback_data: "A1" },
      { text: "A2", callback_data: "A2" },
      { text: "B1", callback_data: "B1" },
      { text: "B2", callback_data: "B2" },
      { text: "C1", callback_data: "C1" },
      { text: "C2", callback_data: "C2" },
      ]

      async function selectEnglishLevel(ctx) {
      const keyboard = Markup.inlineKeyboard(
            englishLevels.map((topic) => Markup.callbackButton(topic.text,
topic.callback_data))
      )
      await ctx.reply("Оберіть ваш рівень володіння англійською:",
keyboard.extra())
      }

module.exports = { selectEnglishLevel }
```

Б. 16 showMainMenu.js

```
      const { Markup } = require("telegraf")

      async function showMainMenu(ctx) {
      await ctx.reply(
            "Виберіть потрібну опцію у меню:",
            Markup.keyboard([
                  ["[1] Практика говоріння"],
```

```
                  ["[2] Приклади використання слів"],
                  ["[3] Перевірка граматики"],
            ]).oneTime().resize().extra()
      )
      }

      module.exports = {
      showMainMenu
}
```

Б. 17 showRestartContinueMenu.js

```
      const {Markup} = require("telegraf")
      const { showMainMenu } = require("./showMainMenu")

      async function showRestartContinueMenu(ctx) {
      const keyboard = Markup.inlineKeyboard([
            Markup.callbackButton("Почати заново", "restart_conversation"),
            Markup.callbackButton("Нове питання", "continue_conversation")
      ])
      await ctx.reply("Оберіть подільшу дію:", keyboard.extra())
      await showMainMenu(ctx)

      }

module.exports = { showRestartContinueMenu }
```

Б. 18 showTopUsedWords.js

```
      const { getTopUsedWords } = require("./database/databaseInterractions")
      const { Extra } = require("telegraf")
      const { getSynonym } = require("./wordsAPI/getSynonyms")

      async function showTopUsedWords(currentUserId, ctx) {
      const topUsedWords = await getTopUsedWords(currentUserId)
      await ctx.reply("*Ви часто використовуєте ці слова, для кожного слова було
підібрано синонім*", Extra.markdownV2())
      for (const word of topUsedWords) {
            const synonym = await getSynonym(word)
            await ctx.reply(`${word} \\- ${synonym.slice(0, 3)}`,
Extra.markdownV2())
      }
      }

      module.exports = {
      showTopUsedWords
}
```

Б. 19 main.js

```
      const {
      Telegraf, Stage, session
      } = require("telegraf")
      const process = require("process")

      const dialogueScene = require("./scenes/dialogueScene")
      const continueDialogueScene = require("./scenes/continueDialogueScene")
      const checkGrammarScene = require("./scenes/checkGrammarScene")
      const wordExamplesScene = require("./scenes/wordExamplesScene")
```

```
	const { showMainMenu } = require("./tools/showMainMenu")

	require("dotenv/config")

	async function setup() {
	const bot = new Telegraf(process.env.TELEGRAM_TOKEN)
	const stage = new Stage()

	stage.register(dialogueScene)
	stage.register(continueDialogueScene)
	stage.register(checkGrammarScene)
	stage.register(wordExamplesScene)

	bot.use(session())
	bot.use(stage.middleware())

	bot.command("dialogue", (ctx) => ctx.scene.enter("dialogue"))
	bot.command("checkGrammar", (ctx) => ctx.scene.enter("checkGrammar"))
	bot.command("wordExamples", (ctx) => ctx.scene.enter("wordExamples"))

	bot.command("menu", ctx => showMainMenu(ctx))
	bot.action("restart_conversation", async (ctx) => {
		ctx.session = {}
		await ctx.scene.enter("dialogue")
	})

	bot.action("dialogue", async (ctx) => {
		ctx.session = {}
		await ctx.scene.enter("dialogue")
	})

	bot.hears("[1] Практика говоріння", async (ctx) => {
		ctx.scene.enter("dialogue")
	})

	bot.hears("[2] Приклади використання слів", async (ctx) => {
		ctx.scene.enter("wordExamples")
	})

	bot.hears("[3] Перевірка граматики", async (ctx) => {
		ctx.scene.enter("checkGrammar")
	})

	bot.action("wordExamples", async (ctx) => {
		await ctx.scene.enter("wordExamples")
	})

	bot.action("checkGrammar", async (ctx) => {
		await ctx.scene.enter("checkGrammar")
	})

	bot.action("continue_conversation", async (ctx) => {
		await ctx.scene.enter("continueDialogue")
	})

	await bot.launch()
	}

setup().then()
```

Б. 20 openai.js

```
	const { OpenAI } = require("openai")
```

```
	const { createReadStream } = require("fs")

	const openai = new OpenAI({
	apiKey: "process.env.OpenAI_KEY
	})

	async function chat(messages) {
	const chatCompletion = await openai.chat.completions.create({
		model: "gpt-3.5-turbo",
		messages: [{ role: "user", content: `${messages}` }],
	})

	return chatCompletion.choices[0].message
	}

	async function transcription(filepath) {
	const response = await openai.audio.transcriptions.create({
		model: "whisper-1",
		file: createReadStream(filepath),
	})

	return response.text
	}

module.exports = { chat, transcription }
```