Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

АРХІТЕКТУРА ПИТАЛЬНО-ВІДПОВІДАЛЬНОЇ СИСТЕМИ З ЕЛЕМЕНТАМИ САМОНАВЧАННЯ

Текстова частина

магістерської роботи

за спеціальністю „Комп'ютерні науки" 122

Керівник магістерської роботи
д.т.н., доц. А. М. Глибовець

_____
(підпис)

"____" _____ 2021 р.

Виконав студент
М. В. Андрощук
"____" _____ 2021 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри інформатики, к.ф.-м.н.

_____С. С. Гороховський

(підпис)

„____”_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на магістерську роботу

студенту 2 р.н. магістерської програми Комп’ютерні науки
Андрощуку Максиму Віталійовичу

Розробити архітектуру питально-відповідальної системи з елементами самонавчання

Зміст текстової частини до магістерської роботи:

Зміст
Анотація
Вступ
1 Питально-відповідальні системи
2 Огляд сервісів для створення питально-відповідальних систем
3 Нейронні мережі в NLP
4 Створення архітектури питально-відповідальної системи
Висновки
Список літератури
Додатки

Дата видачі „___” _____ 2020 р.

Керівник

Глибовець А.М., доктор технічних наук, доцент (підпис)

Завдання отримав

Андрощук М.В. (підпис)

Тема: Архітектура питально-відповідальної системи з елементами самонавчання

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу	04.11.2020	
2.	Огляд технічної літератури за темою роботи	27.11.2020	
3.	Виконання аналізу сучасних рішень	29.11.2020	
3.	Розробка архітектури питально-відповідальної підсистеми	27.12.2020	
4.	Реалізація програмного застосунку на базі розробленої архітектури	24.02.2021	
7.	Написання пояснювальної записки	21.04.2021	
8.	Створення слайдів для доповіді та написання доповіді	27.04.2021	
9.	Аналіз отриманих результатів з керівником, написання доповіді та попередній захист магістерської роботи	13.05.2021	
10.	Корегування роботи за результатами попереднього захисту	20.05.2020	
11.	Остаточне оформлення пояснювальної записки та слайдів	24.05.2020	
12.	Захист магістерської роботи	14.06.2020	

Студент _____

Керівник _____

“____”_____

ЗМІСТ

Анотація

В рамках даної роботи проведено огляд питально-відповідальних систем та їх архітектур, розроблено архітектуру питально-відповідальної підсистеми з елементами самонавчання з використанням Telegram, як системи для питань і відповідей нейронних мереж та Google DialogFlow.

Ключові слова: питально-відповідальні системи, нейронні мережі, Google DiaglowFlow

ВСТУП

Актуальність. Дуже важливим у сучасному світі можливість отримати відповіді на питання. Зростання використання дистанційних форм взаємодії між людьми, організаціями та компаніями. Це вимагає використання сучасних інформаційних технологій та мережі Інтернет. В цифровому світі знання і інформація зростають в об’ємах кожного дня, це ускладнює взаємодію з ними. Користувачі прагнуть до максимально простої взаємодії без втрати часу на пошук інформації. Для цього потрібне використання природньої мови та звичних каналів комунікації. Користувачі очікують індивідуальні, персоналізовані відповіді, що відповідають їх потребам. Це вимагає створення питально-відповідальних систем, що враховують попередні повідомлення отримані системою від людини, можливості обслуговувати велику кількість користувачів та можливість внесення змін відповідно до потреб.

Мета дослідження. Розробити архітектуру питально-відповідальної системи з елементами самонавчання

Завдання дослідження. Проаналізувати питально-відповідальні системи та підходи до їх створення. Вивчити та дослідити програмні рішення, які можуть використовуватися для побудови питально-відповідальної системи. Проаналізувати можливості і особливості компонентів для використання в рамках системи. Обрати інструменти для реалізації обраної архітектури.

Об’єкт дослідження. Питально-відповідальні системи, системи обміну повідомленнями, обробка та розуміння природної мови.

Предмет дослідження. Компоненти питально-відповідальної системи, інтеграції з системами, архітектура питально-відповідальної системи з елементами самонавчання.

Джерела дослідження. Електронні версії друкованої літератури, документація програм, електронні ресурси, вихідні коди програм.

Наукова новизна одержаних результатів дослідження полягає в створенні архітектури питально-відповідальної системи з елементами самонавчання, впровадження, застосування механізмів самонавчання для підвищення розуміння і ефективності обробки повідомлень користувача.

Практичне значення одержаних результатів. За рахунок поєднання різних практик при розробці архітектури питально-відповідальної системи, досягнуто краще розуміння повідомлень. Покращення досягнуто використанням окремих компонентів, що можливо замінити та конфігурувати, архітектура системи передбачає можливість розширення через взаємодії і.

РОЗДІЛ 1. Питально-відповідальні системи

1.1 Концепт питально-відповідальної системи

Одним з визначень питально-відповідальної системи можна вважати таке, що це система з наданням відповідей в автоматичному режимі на питання, задані людьми природною мовою, використовуючи попередньо структуровану базу даних або колекцію документів природною мовою.[1]

Питально-відповідальні системи є логічним розвитком інформаційного пошуку, адже кінцевою метою є надання користувачу інформації. Роль систем такого типу зростає разом із зацікавленістю. [1] Кількість інформації та даних, які доступні в мережі Інтернет, зростає щодня і для роботи з ними необхідні інструменти пошуку для виділення інформації, що очікує користувач.

Такі системи часто використовуються як альтернатива збіркам питань, які користувачі часто задають, це дозволяє показати користувачу лише ту інформацію, що він хоче отримати і не навантажувати зайвими даними. Альтернативним способом застосуванням є задіяння в клієнтській підтримці. Користувач спілкується з системою, а не живою людиною, що дозволяє зменшити навантаження на персонал та давати швидші відповіді користувачам.

До перших питально-відповідальних систем відносять BASEBALL та LUNAR. Завданням BASEBALL було давати відповіді на питання пов’язані з бейсбольною лігою США за рік, а LUNAR в свою чергу відповідав на питання про геологічний аналіз каміння, привезеного місією Apollo з місяця.[2]

1.2 Класифікації питально-відповідальних систем

Існують різні питально-відповідальні системи покликані розв’язувати різні задачі, але всіх їх можна виділяти у категорії за певними ознаками.

За пов’язаністю із сферою застосування питально-відповідальні системи поділяються на системи з обмеженим та відкритим доменом. Визначення відбувається за тим чи питання користувачів стосуються обмеженого домену чи відкритого домену. В обмеженому домені набір можливих питань обмежений, а у відкритих навпаки. Іншою відмінністю є відповіді, які

надаються. Так система з обмеженим доменом спирається на онтологію та термінологію домену, у відкритому домені ж це загальна онтологія та світові знання. До систем з обмеженим доменом відносяться системи Start, Webcoop. Системами з відкритим доменом Webclopedia, Answerbus, Mulder. [1]

Користувачі можуть задавати різні питання це буде вимагати від питально-відповідальної системи різних форматів відповідей. Різні стратегії застосовуються для кожного типу питань, щоб надати коректну відповідь. Так системи будуть поділе на такі: фактичні(що, хто, який, скільки), спискові(сутності за особливістю), підтверджуючі(чи), гіпотетичні(що б сталося), причинні(як, чому). [1]

Питально-відповідальна система може надавати користувачу задавати різні запитання, що надає можливість віднести одну систему одночасно до декількох категорій. До фактичних питально-відповідальних систем відносяться Webclopedia, Naluri, Start, Answerbus, Webcoop, Mulder, до спискових належать Naluri, Start, Webcoop. Підтверджуючими системами є Webclopedia, Webcoop, а причинним Webclopedia, Naluri, Answerbus, Webcoop, Mulder.[1]

Питально-відповідальним системам потрібно аналізувати та обробляти отримані питання, щоб з’ясувати потребу і показати релевантні відповіді. Документи, з якими працює питально-відповідальна система, обробляються і аналізуються таким самим чином.

До основних підходів відносяться такі: статистичний підхід, підхід на відповідність правилам та гібридний, який поєднує попередні. [1]

Питально-відповідальним системам для того, щоб сформувати відповідь необхідні дані. За видом походження цих даних можна поділити системи на такі типи: структуровані джерела даних, напівструктуровані джерела даних та неструктуровані джерела даних.[1]

В структурованих джерелах дані структуровані у смислові набори. Схожі сутності об’єднання у відношення(relations). Сутності в однакових відношеннях мають однакові атрибути. Опис всіх сутностей називається

схемою. Дані узгоджені відповідно до визначеного формату. Запит користувача перетворюється у запит до сховища, що дає точну відповідність питання і відповіді.[1]

У напівструктурованих джерелах самі дані і визначають схему. Неструктуровані джерела даних допускають будь-які типи даних. Відсутні семантичні зв'язки та строгі правила організації даних. Для роботи з таким форматом використовується NLP або методи інформаційного пошуку.[1]

До напівструктурованих належать системи Naluri, Start, Webcoop, а до неструктурованих Webclopedia, Answerbus, Mulder.[1]

Питально-відповідальні системи можна розділити за властивостями джерел даних: розмір, мова, гетерогенність, стиль, тип даних.[1]

Розмір джерела даних впливає на час пошуку та вибір правильних відповідей. Великий розмір збільшує шанси і точність надання правильної відповіді але, ризиками є збільшення NLP обробки та можливі проблеми при індексації.[1]

Документи різними мовами вимагають різних правил для обробки, при цьому універсальних правил для всіх мов не існує. Така особливість може дати перевагу у випадку комбінації даних, що містяться в різних документах різними мовами, але не всі мови мають чіткі граматичні правила, що ускладнить їх аналіз.[1]

Гетерогенність визначає об'єм різних сховищ даних і форматів з якими працює питально-відповідальна система. Така неоднорідність дозволяє отримувати кращі відповіді при розподілі інформації по різним джерелам. Але виникає потреба у перетворенні природної мови у запит, який джерело даних може опрацювати.[1]

Стиль даних та запитів може бути як формальним так і неформальним. Обробка неформальних даних стикається з труднощами через можливість відсутність синтаксису або формалізму. Це може призводити до неправильного аналізу запиту і документів у сховищі.[1]

Ще однією властивість є тип даних з якими працює питально-відповідальна система. Більшість систем працюють із текстовими документами, бо отримання відповідей є складною задачею на сьогодні.[3] Питально-відповідальні системи поділяються за формою відповіді, яку вони надають користувачу. Існує 2 основні способи: витяг і генерація.[1]

Витяг відповіді передбачає, що користувачу надається документ або його частина з джерела даних. Документи можуть бути поділені на речення, а система покаже найбільш релевантне з них. Найкраще підходить для фактичних і підтверджуючих питань. Іншим способом є розбиття документів на параграфи і користувач отримує найрелевантніший з них. Найкраще підходить для причинних і гіпотетичних питань. Ще одним варіантом витягу є відповідь користувачу у форматі мультимедіа.[1]

Генерація відповіді є альтернативним способом надання відповіді. Для підтверджуючих питань це буде “так” або “ні” за допомогою перевірки і підтвердження, що система проведе. Оцінки або рейтинги передбачають, що система надасть оцінку об’єкту або характеристиці об’єкта. Іншим способом є діалогові відповіді, коли система надає відповіді у форматі діалогу з користувачем.[1]

РОЗДІЛ 2 Огляд сервісів для створення питально-відповідальних систем

2.1 Dialogflow

Dialogflow - платформа для взаємодії користувачів із додатками, сайтами, пристроями, ботами та іншими застосуваннями. Декларує підтримку обробки природної мови, можливість обробки текстових і аудіо запитів та текстові або згенеровані аудіо відповіді.[4]

Перша версія була випущена у вересні 2014 року під назвою api.ai. У вересні 2016 року компанію із сервісом придбала компанія Google та у вересні 2017 року переіменувала на Dialogflow.[5]

Робота з Dialogflow базується на основних концептах: агент(agent), інтент(intent), сутність(entity), контекст(context). Вони є основою для побудови взаємодії з користувачами.

Агент у Dialogflow є віртуальним агентом, що обробляє бесіду з користувачами системи. Він є модулем, що розуміє природну мову та структурує дані отримані в рамках бесіди для подальшої обробки.[4]

Інтент класифікує намір користувача для ведення бесіди. Кожен агент може мати багато інтентів, які можуть бути використані при взаємодії з користувачем. Кожна репліка користувача співставляється з наявними інтентами та обирається найкраща.[4]

Інтент містить тренувальні фрази, що визначають які репліки користувача викликають даний інтент. Кожен інтент може містити багато тренувальних фраз. При цьому система декларує розширення вказаного списку схожими фразами за рахунок машинного навчання. Загальною рекомендацією є використання не менше 10 тренувальних фраз.[4]

Коли репліка користувача співпадає з якимось інтентом, Dialogflow витягує параметри з репліки. Кожен такий параметр має тип сутності, який визначає як дані будуть витягатися. Для підтримки такого витягання на тренувальних фразах потрібно розмітити параметри і вказати їх тип сутності.

Отримані параметри можна використати для побудови відповіді та передачі власному сервісу за допомогою webhook.[4]

Кожен інтент вимагає відповіді. Так Dialogflow підтримує статичні і динамічні відповіді. Статичні відповіді вказуються в інтенті, їх може бути декілька для різноманітності відповідей. Для динамічних відповідей є можливість додати параметри отримані з репліки користувача або використання власного сервісу із створення відповіді там. Базовою відповіддю є текст, але інші типи(картинка, звук тощо) також можливі в залежності від платформи, де працюватиме агент. При цьому відповідь може містити уточнююче питання, якщо не всі дані отримані в рамках одного інтенту.[4]

Сутність це параметр, який може бути у репліці користувача. Кожна сутність має тип. Dialogflow має початкові типи такі як: час, дата, довжини та інші. При цьому є можливість визначти власний тип, якщо готові не підходять. Кожному типу сутності відповідають конкретні сутності, які потрібно вказати. При цьому декілька різних слів можуть відповідати одній сутності.[4]

Контексти в Dialogflow близькі до контекстів у природній мові. Вони потрібні для побудови сценаріїв діалогу. Так кожен інтент може приймати і повертати контексти. При активації інтенту повертається вихідний контекст. І при наступному виборі інтенту будуть обиратися ті, в яких вхідний контекст співпадає попереднім вихідним.[4]

Для початку роботи з Dialogflow потрібно мати Google-акаунт та перйти до консолі сервісу(https://dialogflow.cloud.google.com/). Після цього потрібно обрати створення агента. Після цього відкриється вікно створення, де потрібно вказати ім’я агента, мову за замовчуванням та часовий пояс за яким запити будуть вказувати час. Після цього агент буде створений і відкриється вікно з налаштуваннями.

Далі потрібно обрати на яких платформах буде працювати агент. При обиранні з’являється діалогове вікно з інструкціями. У випадку Telegram потрібно буде ввести токен для бота, який буде використовувати агент

Dialogflow. Інтеграцію можна зупинити в будь-який момент змінивши значення перемикача.

На початку підтримуються 2 види інтентів: привітання та невдача(Fallback). Інтент невдачі використовується, коли решта інтентів не підійшли. Цих інтентів досить для перевірки конфігурації та інтеграції.

2.2 IBM Watson Assistant

IBM Watson Assistant - платформа для побудови питально-відповідальних систем. Створена компанією IBM в рамках проекту комп'ютерного штучного інтелекту IBM Watson .[6]

Основні концепти, якими оперує IBM Watson Assistant: інтенти та сутності

Інтент є колекцією реплік користувачів з однаковим значенням. Якщо у репліці користувача розпізнає інтент, то він піде по сценарію діалога. [6]

Сутність представляє інформацію із репліки користувача, яка стосується його наміру.[6]

Для початку роботи потрібно мати IBM Cloud аккаунт. Далі вибрати створення Watson Assistant. Після цього потрібно обрати регіон, де буде знаходитися сервіс, вказати тарифний план та обрати назву. Після цього натиснути на кнопку створення. Після цього відкриється вікно налаштувань сервіса.

Після цього можна перейти до налаштувань за кнопкою запуска Watson Assistant. Відкриється список поточних асистентів з можливістю додати нового. При виборі асистента відкриються його налаштування.

Підтримується інтеграція з власним сайтом, через окрему веб-сторінку, Facebook Messenger, Slack, Intercom. Також є можливість створити власну інтеграцію за допомогою API.[6]

На початку є лише діалог привітання та невдачі. Можливо додати вузол до дерева діалогу, як дочірній вузол іншого вузла.

Потрібно вказати ім'я діалогу, чи повинно користувачу пропонуватися перехід на цей діалог, умова запуску діалогу, відповідь користувачеві та дія

після надання відповіді. Після надання відповіді може звичайне очікування наступної репліки або перехід на інший діалог. Перехід може бути з очікуванням репліки, відповіддю при відповідності попередньої репліки та безумовною відповіддю.

2.3 Сервіси Microsoft Azure

Microsoft Azure - платформа хмарних обчислень, створена компанією Microsoft. Пропонує SaaS, PaaS, IaaS рішення. Для створення питально-відповідальної системи має декілька сервісів: Personalizer, для персоналізованих відповідей користувачу, QnA Maker, база знань для відповідей, Language Understanding, для виділення намірів у репліках користувача, Text Analytics, для виділення настрою, ключових фраз та іменованих сутностей, Azure Bot для взаємодії з користувачем та викликів інших сервісів.[7]

Language Understanding (LUIS) - хмарний сервіс, що надає API для обробки реплік користувача. Використовує машинне навчання для визначення значення та отримання релевантної детальної інформації з фраз користувача. Будь-який розмовний додаток, що працює з природною мовою, може використовувати LUIS, наприклад чат-боти. Сервіс підтримує англійську, китайську, французьку, італійську, німецьку, іспанську, японську мови та деякі інші, але української серед них немає.[8]

Додаток-клієнт відправляє текст, введений користувачем на ендпоінт LUIS. Після чого сервіс співставляє текст з можливими інтентами і вибирає найбільш імовірний. Далі намагається виділити сутності наявні в повідомленні. Після чого передає ці дані додатку-клієнту для їх подальшої обробки. [8]

QnA Maker - хмарний NLP сервіс, який створює шар природної мови над даними. Може використовуватися для найбільш доречної відповіді користувачу з бази знань при запиті природною мовою. Користувачами сервісу є розмовні додатки, які взаємодіють з користувачами природною мовою.[8]

QnA Maker рекомендують використовувати у випадку наявності статичної інформації у базі знань для відповідей або при однаковій відповіді різним користувачам на однакові питання. Прикладами такої інформації є FAQ, документація, брошури, тощо. наявність статичної інформації, наприклад pdf документи, веб ресурси з відповідями, тощо.[8]

Додаток-клієнт відправляє питання на ендпоінт QnA Maker. Після цього сервіс, використовуючи базу знань, надає відповідь і можливі уточнення для видачі найкращої відповіді. Додаток-клієнт отримує відповідь надану сервісом з оцінкою відповідністю і може перенаправити її користувачу або запропонувати уточнення.[8]

Для початку роботи потрібно мати аккаунт Microsoft, кошти на рахунку Azure для розміщення сервісів та увійти у Microsoft Azure.

Основою для роботи є Azure Bot Service Bot. Цей сервіс безпосередньо взаємодіє з користувачами, отримує та відправляє повідомлення, наприклад чат-боти.[8]

В Azure Portal потрібно обрати вікно Bot Services. Після цього відкриється список наявних ботів з можливістю створити нового.

Далі потрібно обрати створення нового бота, після чого сервіс пропонує обрати тип бота: Web App Bot або Bot Channels Registration. Перший варіант передбачає, що бот буде створений та розгорнутий всередині Azure як Azure App Service Web App. Другий же передбачає наявність бота розгорнутого поза Azure та його під’єднання до Bot Service.

Після обрання Web App Bot та переходу на створення, буде відкрито вікно з налаштуваннями. Потрібно вказати назву, параметри ресурсів Azure, регіон розгортання сервісу, та шаблон бота. На вибір дається Echo Bot, який повертає написане повідомлення, та Basic Bot, який містить LUIS та Bot Analytics.

Після цього можна перевірити поведінку бота у інтерфейсі Azure інструментом Web Chat, що доступний в консолі сторінки бота. Оскільки базова версія бота не має достатніх можливостей для створення власної

системи, то потрібно завантажити код бота та розпакувати в окрему директорію. Можливе використання шаблонів для ботів, які присутні в Інтернеті.

Перед початком роботи потрібно створити LUIS сервіс. При створенні потрібно вказати назву, мову, групу авторів, та ключ.

Після створення потрібно додати інтенти, які повинні розпізнавати сервіс. Є стандартні інтенти для погоди, автоматизації будинку, замовлення квитків та інших. Якщо ці інтенти не підходять, потрібно створити нові і додати фрази, які йому відповідають, і сутності, що можуть бути передані. Створимо два інтенти FavoriteUniversity та DeanComputerScience.

Далі необхідно запустити тренування моделі та опублікувати сервіс. Інтерфейс LUIS надає можливість відразу перевірити модель. Користувач надсилає повідомлення, а у відповідь йому повертається інтент з оцінкою ймовірності.

Наступним сервісом є QnA Maker. Потрібно створити нову базу знань, де вказати параметри розміщення сервісу в Azure, мову бази знань, її назву та документи для її наповнення.

Після наповнення базу знань можна перевірити і доповнити іншими парами питання-відповідь. Для кожної такої пари можна додати підказку для подальших запитань.

Після наповнення потрібно опублікувати сервіс для повноцінного використання.

Робота з ботом передбачає наявність директорії вихідних кодів сервісу. У директорії містяться файли, які відповідають за налаштування бота та виклики інших сервісів. Файл .env містить ідентифікатори, ключі, та адреси про сам сервіс бота та сторонні сервіси, що буде використовувати бот, в даному випадку LUIS та QnA Maker.

Поєднання декількох сервісів вимагає наявність диспетчеризації між цими файлами. Для цього є Dispatch tool, який потрібно додати через менеджер пакетів Node.js - npm.

Перевірити роботу бота можна розгорнувши його на власній машині або опублікувавши на Azure. Для локальної перевірки потрібно запустити бота командою npm start і використати Bot Framework Emulator - інструмент від Microsoft для локальної перевірки поведінки ботів. За замовчуванням бот стає доступний за адресою http://localhost:3978/api/messages. При відкритті бота необхідно вказати дані сервіса, що використовує бот: MicrosoftAppId та MicrosoftAppPassword. В іншому разі запити до сторонніх сервісів будуть невдалими через потребу в авторизації.

Для розгортання бота в Azure необхідно встановити Azure CLI та увійти у свій обліковий запис через команду az login. Далі архівувати директорію, де знаходяться вихідні коди бота та виконати команду az webapp deployment source config-zip, вказавши в параметрах групу ресурсів, назву сервісу та шлях до zip-архіву. Через декілька хвилин бот буде розгорнуто в Azure з можливістю взаємодії через Інтернет.

РОЗДІЛ 3 Нейронні мережі в NLP

3.1Архітектури нейронних мереж в NLP

Багатошаровий персептрон (Multilayer perceptron MLP) має три або більше шарів. Він використовує нелінійну функцію активації яка дозволяє класифікувати дані, які не можна лінійно розділяти. Кожен вузол шару підключається до кожного вузла наступного шару, роблячи мережу повністю підключеною. До таких програм відносяться розпізнавання мови та машинний переклад.[10]

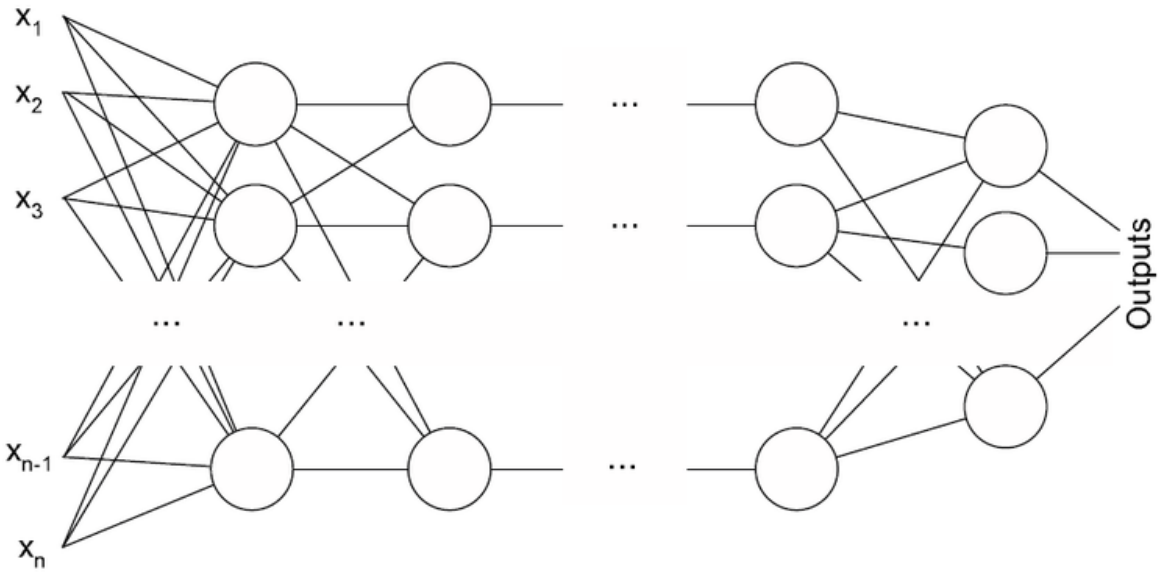

Рисунок 3.1[10]

Згорткова нейронна мережа (Convolutional neural network) містить один або кілька згорткових шарів, об'єднаних або повністю з'єднаних, і використовує різновид багатошарових персептронів, розглянутих вище. Згорткові шари використовують операцію згортки на вході, передаючи результат наступному шару. Ця операція дозволяє мережі бути глибшою з набагато меншою кількістю параметрів. Згорткові нейронні мережі показують добрі результати в застосунках пов'язаних із зображеннями та мовами. Вони можуть досягати видатні результати без знання слів, фраз, речень та будь-яких інших синтаксичних або семантичних структур щодо людської мови. Іншими застосуваннями є семантичний розбір, виявлення перифразів, розпізнавання мовлення [10]

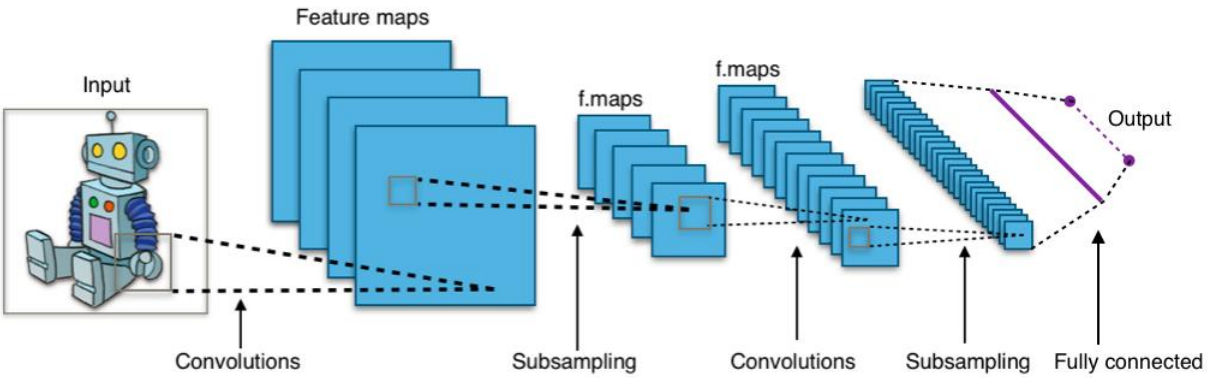

Рисунок 3.2[10]

Рекурсивна нейронна мережа (Recursive neural network) - це тип глибокої нейронної мережі, сформованої шляхом застосування одного і того ж набору ваг рекурсивно над структурою, щоб зробити структуроване передбачення щодо вхідних структур змінного розміру або скалярне передбачення на ній, обходячи задану структуру у топологічному порядку. У найпростішій архітектурі нелінійність, така як тангенс, і матриця ваги, що ділиться по всій мережі, використовуються для об'єднання вузлів у батьків. [10]

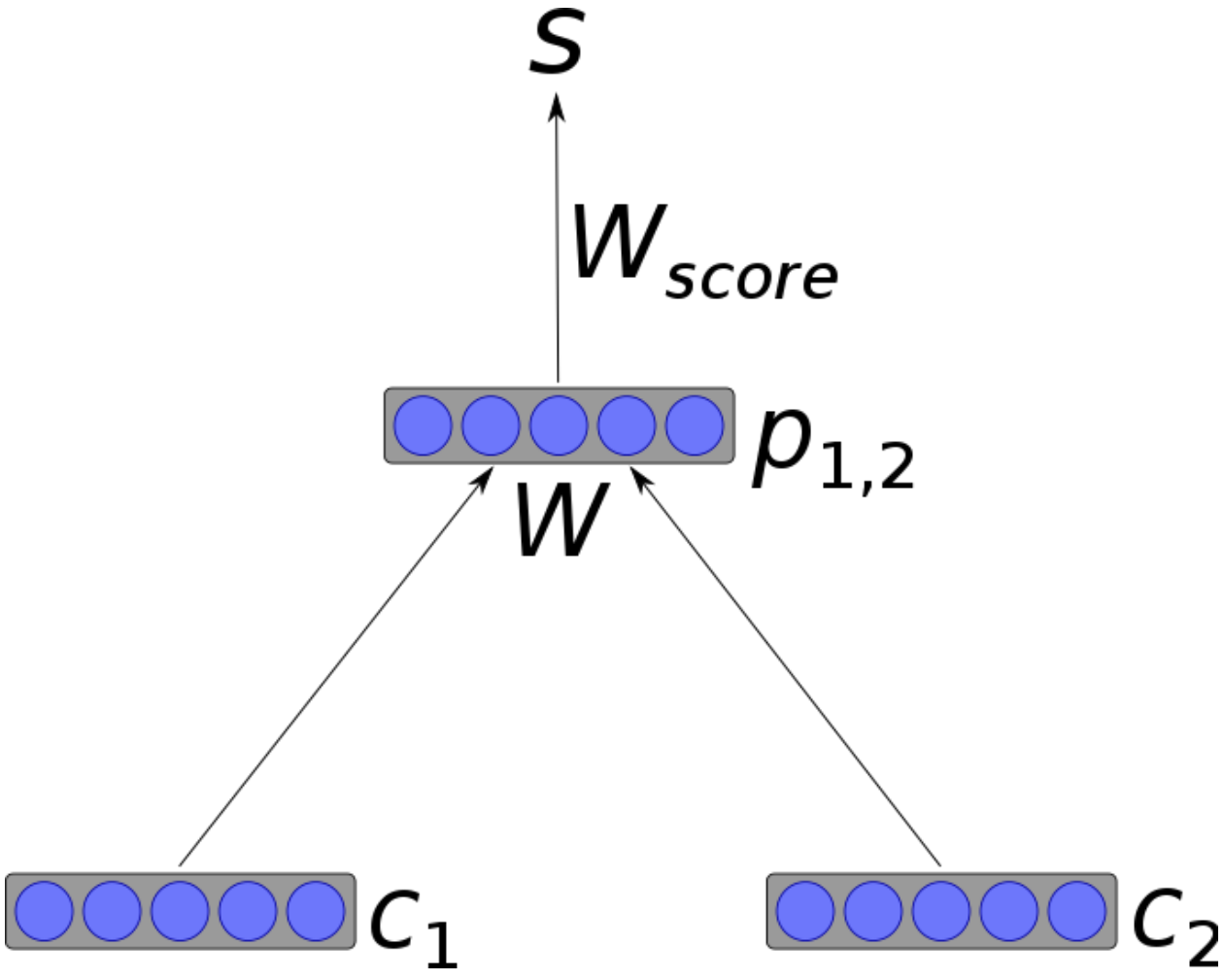

Рисунок 3.3[10]

Рекуррентна нейронна мережа (Recurrent neural network) є варіантом рекурсивної штучної нейронної мережі, в якій зв’язки між нейронами складають направлений цикл. Це означає, що вихід залежить не тільки від поточних входів, але і від стану нейрону попереднього кроку. Ця пам’ять дозволяє вирішувати такі проблеми, як розпізнавання рукописного вводу чи розпізнавання мовлення. [10]

Довга короткочасна пам’ять (Long short-term memory) - це специфічна архітектура рекуррентних нейронних мереж, яка була розроблена для більш точного моделювання тимчасових послідовностей та їхніх віддалених залежностей від звичайних рекуррентних мереж. Вона не використовує функцію активації у своїх рекуррентних компонентах, збережені значення не змінюються, а градієнт не має тенденції зникати під час навчання. Рішенням з використанням цієї архітектури досягають високого рівня при широкомасштабному акустичному моделюванні та у сфері позначень тегів.[10]

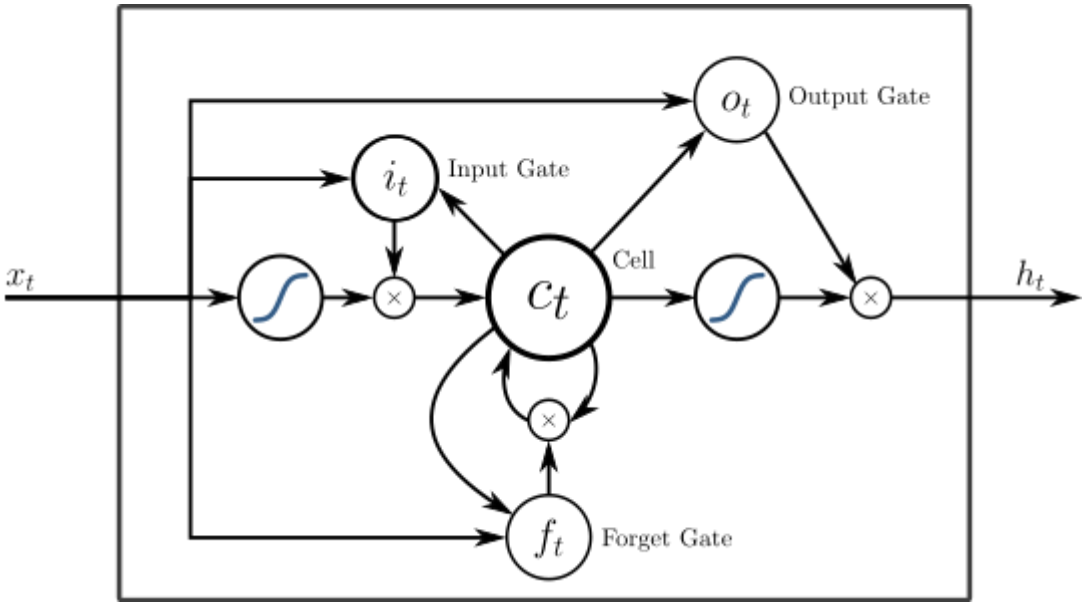

Рисунок 3.4[10]

Зазвичай моделі Sequence-to-sequence складаються з двох повторюваних нейронних мереж: кодувальника, який обробляє вхід, і декодера, який генерує результат. Кодувальник і декодер можуть використовувати однакові або різні набори параметрів. Моделі Sequence-to-sequence в основному

використовуються в питально-відповідальних системах, чатах і машинному перекладі. [10]

Окрім глибоких нейронних мереж, поверхневі моделі також є популярними та корисними інструментами. Наприклад, word2vec - це група дрібних двошарових моделей, які використовуються для створення вкладання слів. [10]

3.2 Самонавчання в нейронних мережах

Моделювання мови є завданням вивчення розподілу ймовірностей за послідовностями слів і, як правило, зводиться до побудови моделі, здатної передбачити наступне слово, речення чи абзац у даному тексті. Стандартний підхід полягає у навчанні мовної моделі шляхом надання їй великої кількості зразків, наприклад текст якоюсь мовою, що дозволяє моделі дізнатися ймовірність того, що різні слова можуть з’являтися разом у даному реченні.[11]

Нещодавно мовні моделі на основі нейронних мереж показали кращі показники, ніж класичні статистичні мовні моделі, такі як N-грами, приховані марківські моделі та системи, засновані на правилах. Моделі на основі нейронних мереж досягають таких більш високих показників завдяки: навчанню на дедалі більших розмірах корпусів лише з лінійним збільшенням кількості параметрів, параметризуванні вкладання слів та використання їх як вхідних даних для моделей, збір контекстної інформації як на рівні слова, так і на рівні речення[11]

Мовні моделі, засновані на нейронних мережах, є основою останніх розробок в обробці природних мов, прикладом яких є BERT, скорочення від Bidirectional Encoder Reprezentations від Transformers. [11]

Як випливає з назви, в архітектурі BERT використовуються перетворювачі, орієнтовані на увагу, які дозволяють збільшити можливості розпаралелювання, що потенційно може призвести до скорочення часу навчання для тієї ж кількості параметрів. Завдяки проривам, досягнутим за допомогою перетворювачам, що базуються на увазі, автори змогли навчити

модель BERT на великому текстовому корпусі, що поєднує Вікіпедію (2500 млн слів) та BookCorpus (800 млн слів), досягаючи великих результатів у різних завданнях при обробці природної мови. [11]

BERT, як і інші опубліковані роботи, такі як ELMo та ULMFit, навчався на основі контекстних подань у текстовому корпусі, а не безконтекстним способом, як це робилося у вкладанні слів. Контекстне подання враховує значення і порядок слів, що дозволяє моделям дізнатися більше інформації під час навчання. Однак алгоритм BERT відрізняється від інших вищезазначених алгоритмів використанням двонаправленого контексту, що дозволяє словам «бачити себе» як зліва, так і праворуч.[11]

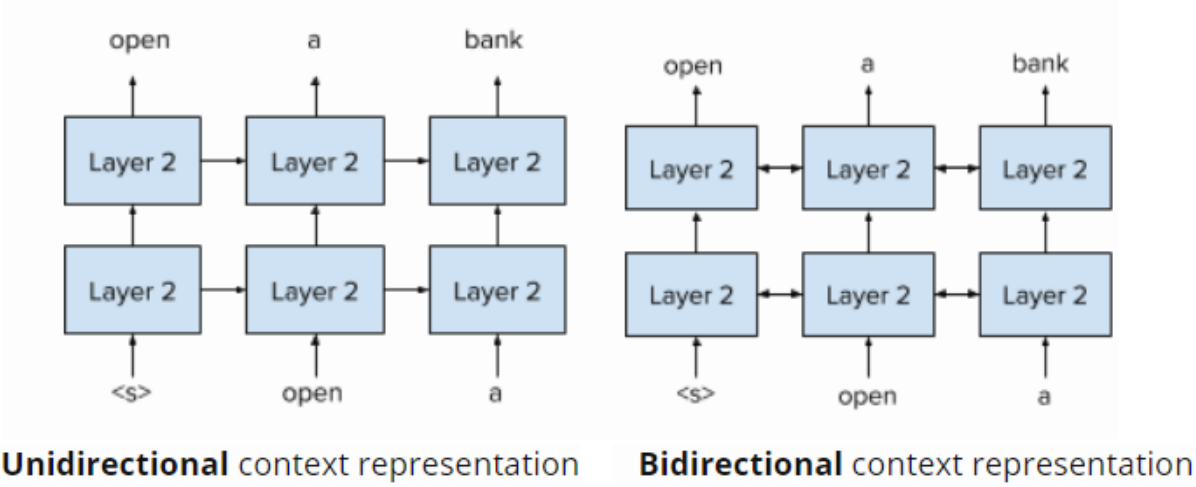

Рисунок 3.5[11]

BERT представив дві різні цілі, які використовуються під час попередньої підготовки: прихована мовна модель, яка випадковим чином приховує 15% слів від вхідного сигналу та навчає модель прогнозувати замасковане слово та передбачення наступного речення, яке бере пару речень, щоб визначити, чи відповідає останнє речення є фактичним реченням, яке продовжує попереднє речення або випадковим реченням. Поєднання цих навчальних цілей дозволяє чітко розуміти слова, а також дає змогу моделі дізнатися більше контексту відстані слово/фраза, який охоплює речення. Ці особливості роблять BERT підходящим вибором для таких завдань, як відповідь на запитання або порівняння речень. [11]

Попередньо навчена модель BERT може бути додатково відрегульована під конкретні завдання, такі як загальне розуміння мови, класифікація тексту, аналіз настроїв, запитання та відповіді тощо. Точне налаштування може бути здійснено шляхом заміни відповідних входів і виходів для даного завдання та, можливо, дозволяючи оптимізувати всі параметри моделі наскрізно. [11]

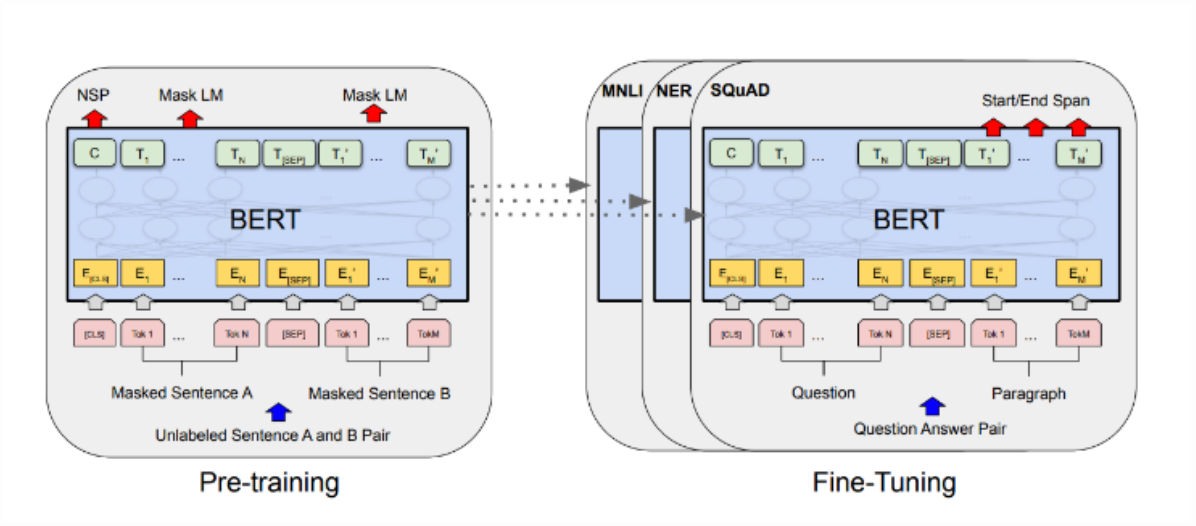

Рисунок 3.6[11]

3.3 Набори даних для нейронних мереж в NLP

Для навчання будь-якої нейронної мережі необхідні набори даних. Використовуючи дані із набору модель може навчатися та мінятися в залежності від виду набору.

Одним з таких наборів CoQA. CoQA - це масштабний набір даних для побудови розмовних систем відповідей на запитання. Метою завдання CoQA є вимірювання здатності машин розуміти уривок тексту та відповідати на низку взаємопов'язаних питань, які з'являються в розмові.[12]

CoQA містить 127 000+ запитань із відповідями, зібраними з 8000+ бесід. Кожна розмова збирається шляхом поєднання двох краудворків для спілкування про уривок у формі запитань та відповідей. Унікальні особливості CoQA включають 1) запитання розмовні; 2) відповіді можуть бути у вільній формі; 3) до кожної відповіді також додаються докази, виділені в уривку; та 4)

уривки зібрані із семи різноманітних сфер. CoQA має багато складних явищ, яких немає в існуючих наборах даних про розуміння читання, наприклад, стрижневі посилання та прагматичні міркування.[12]

SQuAD є іншим набором даних. Стенфордський набір запитань на відповіді на запитання (SQuAD) - це набір для розуміння читання, що складається з питань, заданих краудворками в наборі статей Вікіпедії, де на кожне запитання відповідає сегмент тексту або проміжок з відповідного уривка для читання або питання може бути без відповіді.[13]

SQuAD2.0 поєднує 100 000 запитань у SQuAD1.1 з понад 50 000 безвідповідальних запитань, написаних змагальними краудворками, щоб виглядати схожими на відповіді. Для успішного використання SQuAD2.0 системи повинні не тільки відповідати на запитання, коли це можливо, але й визначати, коли параграф не підтримує жодної відповіді, та утримуватися від відповіді.[13]

Загалом для навчання можна використовувати різні набори даних в тому числі створювати власні і навчати моделі на них. Головною проблемою буде розмітити обсяг достатній, щоб моделі навчалися.

РОЗДІЛ 4 Створення архітектури питально-відповідальної системи

4.1 Вибір сервісу для питально-відповідальної системи

Для створення було обрано Dialogflow з огляду на можливості інтеграції з окремими сервісами та наявності плану з безкоштовними повідомленнями.

В рамках безкоштовного плану надається 180 текстових запитів на хвилину, 60 запитів на зміну агента на хвилину, 100 запитів на запити в рамках сесії користувачів. Для прототипу цих обмежень достатньо.[9]

4.2 Інтеграції Dialogflow

У разі потреби динамічних відповідей Dialogflow дозволяє додати інтеграції з стороннім сервісом.Кожен інтент має налаштування, що вказує потребу в зовнішньому сервісі. Якщо воно не увімкнене, то буде використана статична відповідь. Тому для динамічних відповідей потрібно, щоб це поле було увімкнене. [4]

При визначення інтента з динамічною відповіддю Dialogflow надсилає запит на webhook сервісу з інформацією про інтент. Далі сторонній сервіс обробляє ці дані та повідомляє Dialogflow інформацію необхідну для відповіді через webhook. [4] На рисунку 3.1 зображено схему взаємодії користувача з Dialogflow та сторонньою системою через webhook.

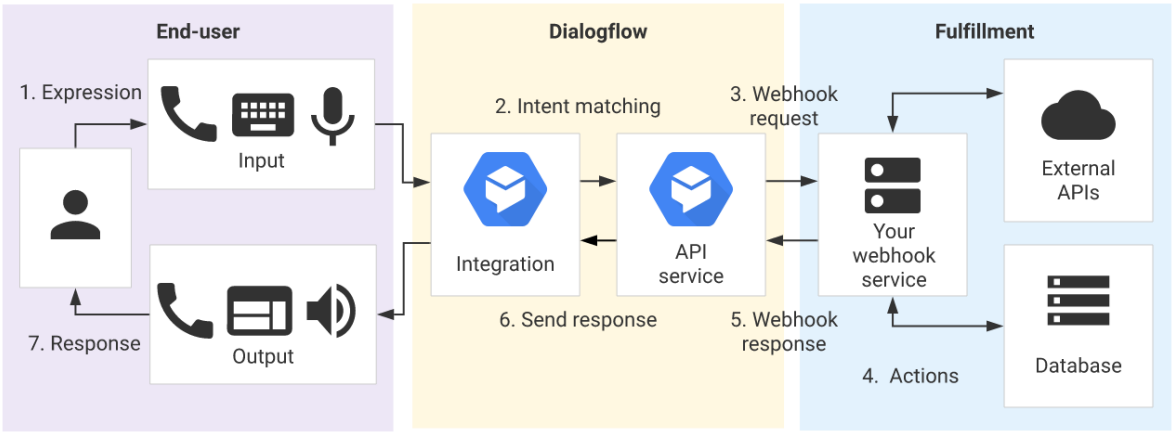

Рисунок 4.1[4]

Webhook стороннього сервісу повинен приймати та повертати повідомлення у форматі JSON відповідно до специфікації Dialogflow. Має

приймати HTTPS запити, бути публічно доступним, приймати POST запити WebhookRequest та відповідати WebhookResponse.[4]

4.3 Створення власного сервіса

Архітектура системи передбачає наявність Spring boot сервісу інтегрованого з Dialogflow. Для спрощення розробки таких сервісів була створена бібліотека Actions on Google для Java. Там присутні класи, які імплементують частину логіки Dialogflow: DialogflowApp, ActionContext, DefaultApp та інтерфейси для можливості власної реалізації.

Архітектура застосунку містиме контролер для обробки webhook запитів. Далі дані із запиту передаються власному класу, який розширює реалізовує API Dialogflow. Всередині класу містяться методи для обробки інтентів. Анотація @ForIntent мітить параметром назву інтента, який буде перехоплюватися і оброблятися цим методом. Кожен метод повинен повернути у відповідь об’єкт класу ActionResponse з даними, які отримає користувач.

Для роботи системи потрібное джерело даних, документи, тексти. При використанні інтенту з параметрами система використовує ці дані для надання відповіді.

4.4 Робота з нейронною мережею

Для роботи з нейронною мережею можна використати 2 різні підходи: мікросервісний та монолітний. Це передбачатиме створення окремого сервісу або додавання моделі до існуючого сервісу. Зазвичай моделі створюються за допомогою бібліотек PyTorch та Tensorflow. За замовчуванням для них рекомендують використовувати мову програмування Python. Але існує можливість використання і в Java за допомогою бібліотеки Deep Java Library.

4.5 Розгортання застосунку

Для розгортання застосунку було обрано сервіс Heroku. Heroku - хмарний сервіс типу PaaS, що дозволяє розгортати застосування. Підтримуються різні мови програмування такі як: Node.js, Java, Ruby та інші фреймворки.

Розгорнутий застосунок отримає доменне ім'я від Heroku, але при потребі можливо під’єднати і власне. Розгорнутий сервіс необхідно додати до Dialogflow, що дозволить йому передавати повідомлення саме на нього.

Висновки по роботі та рекомендації для подальших досліджень

В першому розділі роботи було розглянуто поняття питально-відповідальних систем, їх класифікації, проведено огляд різних видів питально-відповідальних систем.

Другий розділ містить огляд сервісів для створення питально-відповідальних системи, дослідження рішень, що можуть бути використані для побудови архітектури питально-відповідальної системи, в тому числі можливості, що вони надають, та варіанти інтеграції з месенджерами.

В третьому розділі розглянуто використання нейронних мереж у сфері обробки природньої мови, що можуть бути використані при створенні архітектури питально-відповідальної системи, яка матиме можливість навчатися на даних з діалогу та враховувати їх при наданні відповіді.

В четвертому розділі розглянуто архітектуру питально-відповідальної системи з елементами самонавчання. Запропоновано Dialogflow як точку інтеграції та створення окремого власного сервісу інтегрованого з Dialogflow та нейронною моделлю.

Отримана в результаті розробки архітектура може бути в змінена чи розширена новими частинами при потребі. Можна використовувати різні нейронні мережі, використання лише власного сервісу із заміною функцій Dialogflow. Можлива реалізація підключення різних джерел даних, використання даних з бесід для вдосконалення нейронних мереж.

Розроблений архітектура є ефективною, гнучкою та масштабованою. На її базі можливе створення схожих архітектура через незалежність компонентів та їх взаємодії через протоколи.

Цифровий світ є дуже динамічний, тому існуючі продукти і програми розвиваються, з’являються нові концепції та підходи. Вдосконалення та розвиток розробленої архітектури може бути частиною майбутньої роботи. Розширення можливе за рахунок розширення кількості каналів комунікації та способів взаємодії. Перспективним є додавання обробки голосу і використання перетворень голосу на рівні питально-відповідальної системи. Та

вдосконалення сценаріїв за рахунок підвищення точності та повноти відповідей.

Список літератури

1. Marco Antonio, Calijorne Soares, Fernando Silva Parreiras. “A literature review on question answering techniques, paradigms and systems”, 2018 [Електронний ресурс] URL: https://www.sciencedirect.com/science/article/pii/S131915781830082X?via%3Dihub
2. Question answering [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Question_answering
3. N. Indurkhya, F.J. Damereau (Eds.). “Handbook of Natural Language Processing” (second ed.), Chapman & Hall/CRC, Boca Raton (2010)
4. Dialogflow documentation [Електронний ресурс] URL: https://cloud.google.com/dialogflow/docs/
5. Dialogflow [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Dialogflow
6. IBM Cloud Docs [Електронний ресурс] URL: https://cloud.ibm.com/docs/assistant
7. Microsoft Azure [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Microsoft_Azure
8. Azure Cognitive Services documentation [Електронний ресурс] URL: https://docs.microsoft.com/en-us/azure/cognitive-services/
9. Quotas and limits [Електронний ресурс] URL: https://cloud.google.com/dialogflow/quotas
10. 7 types of Artificial Neural Networks for Natural Language Processing [Електронний ресурс] URL: https://medium.com/@datamonsters/artificial-neural-networks-for-natural-language-processing-part-1-64ca9ebfa3b2
11. Unsupervised NLP: How I Learned to Love the Data[Електронний ресурс] URL: https://medium.com/@ODSC/unsupervised-nlp-how-i-learned-to-love-the-data-1dde7dc4a3c1

12. A Conversational Question Answering Challenge[Електронний ресурс] URL: https://stanfordnlp.github.io/coqa/
13. The Stanford Question Answering Dataset [Електронний ресурс] URL: https://rajpurkar.github.io/SQuAD-explorer/