МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

кафедра комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

РОЗРОБКА ГОЛОСОВОГО АСИСТЕНТА ДЛЯ ПК НА PYTHON

Виконав(-ла): студент(-ка) групи 1К-20
спеціальності
123 Комп’ютерна інженерія

Віталій ВІТРЯК

Керівник Маргарита МЕДОЛИЗ

Черкаси 2024

АНОТАЦІЯ

Ця кваліфікаційна робота присвячена розробці голосового асистента для персонального комп'ютера з використанням мови програмування Python. У роботі розглянуто основні технології, необхідні для створення голосових асистентів, проаналізовано сучасні популярні рішення на ринку та їх застосування. Детально описано етапи проектування та реалізації програмного продукту, включаючи вибір архітектури, розробку модулів розпізнавання мовлення, обробки природної мови та синтезу мовлення. Також висвітлено потенційні виклики та обмеження, а також переваги використання Python для даної мети. Результатом роботи є функціональний голосовий асистент, здатний взаємодіяти з користувачем через голосові команди.

ABSTRACT

This qualification paper focuses on the development of a voice assistant for personal computers using the Python programming language. The work examines the core technologies required for creating voice assistants, analyzes contemporary popular market solutions and their applications. The stages of software product design and implementation are described in detail, including architecture selection, and the development of speech recognition, natural language processing, and speech synthesis modules. Potential challenges and limitations, as well as the advantages of using Python for this purpose, are also highlighted. The result of this work is a functional voice assistant capable of interacting with the user via voice commands.

ЗМІСТ

ВСТУП

Значення та переваги розробки голосового асистента для ПК на Python

Голосовий асистент для ПК на Python вирішує кілька ключових проблем та відкриває нові можливості для користувачів, особливо для людей з обмеженими можливостями. По-перше, він забезпечує зручний інтерфейс для взаємодії з комп'ютером, зменшуючи необхідність введення тексту або керування мишею. Це особливо важливо для людей з моторними обмеженнями. По-друге, асистент допомагає підвищити продуктивність, виконуючи різноманітні завдання, такі як запуск програм, відкриття файлів, пошук інформації в Інтернеті. По-третє, голосові асистенти покращують доступність комп'ютерів для людей з візуальними або моторними обмеженнями, надаючи можливість навігації за допомогою голосових команд.

Підвищення доступності та інклюзивності за допомогою голосового асистента на Python

Розробка голосового асистента на Python відкриває нові горизонти для підвищення доступності комп'ютерних технологій. Голосовий інтерфейс дозволяє взаємодіяти з комп'ютером через мовлення, що є зручним для людей з обмеженою рухливістю, вадами зору або слуху. Асистент допомагає користувачам з інвалідностями у навігації по веб-сторінках, читанні тексту та виконанні різноманітних завдань. Крім того, можливість налаштування голосового інтерфейсу з урахуванням індивідуальних особливостей користувача робить цей інструмент ще більш ефективним.

Виклики та перспективи для інклюзивного дизайну

Розробка голосового асистента для ПК на Python стикається з технічними, етичними та культурними викликами. Технічні обмеження включають нестабільну роботу систем розпізнавання мови та обмежену потужність обчислювальних ресурсів. Проблеми конфіденційності та безпеки даних є важливими в контексті збору та обробки аудіо- та текстової інформації користувачів. Культурні та мовні варіації можуть ускладнити точне

розпізнавання мовлення, а етичні питання включають захист прав користувачів та запобігання неправомірному використанню інформації. Важливо також враховувати психологічні аспекти взаємодії з голосовим асистентом.

Актуальність розробки голосового асистента для ПК на Python

Розробка голосового асистента для ПК на Python залишається актуальною через зростання популярності голосових інтерфейсів та швидкий розвиток технологій обробки мови. Голосові асистенти підвищують продуктивність та доступність, допомагаючи з рутинними операціями та нагадуючи про важливі події. Вони також знаходять застосування в різних сферах, таких як освіта, медицина, бізнес та побут. Python є високорівневою мовою програмування, що дозволяє швидко створювати прототипи та реалізовувати складні алгоритми, пов'язані з розпізнаванням мовлення та обробкою природної мови.

Мета та завдання дослідження

Метою дослідження є створення ефективного голосового асистента на Python, який покращує взаємодію користувачів з комп'ютерами та підвищує їхню продуктивність та доступність. Завдання включають аналіз потреб користувачів, вивчення технологій обробки мовлення, проектування інтерфейсу, розробку та впровадження системи, тестування та оцінку ефективності, а також вдосконалення та підтримку системи.

Дослідження ключових аспектів

Дослідження різних аспектів розробки голосового асистента для ПК на Python включає алгоритми розпізнавання мови, обробку природної мови, синтез мовлення, архітектуру та дизайн системи, а також аспекти безпеки, оптимізації продуктивності та етичні питання. Важливі також питання доступності для людей з обмеженими можливостями, оптимізація продуктивності системи та дослідження впливу асистента на користувачів.

РОЗІДЛ 1
ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення голосового асистента

Ритм і динаміка життя людей з кожним днем прогресує. У людей немає часу читати довгі пости, дивитися довгі відеоролики, у них навіть немає часу писати текст. Тому люди все частіше користуються голосовими помічниками. За прогнозами Gartner найближчим часом 30% сеансів перегляду вебсторінок будуть відбуватися без використання екранів, а 50% з усіх запитів будуть голосовими.

Голосові асистенти дають можливість мінімізувати, а іноді й зовсім усунути необхідність використовувати руки та очі для перегляду контенту в інтернеті.Але крім побуту, голосові асистенти можуть стати в пригоді в бізнесі та навіть в HR-сфері.

Що ж, голосовий асистент - це інноваційний інтерфейс взаємодії користувача з комп'ютером або мобільним пристроєм, який дозволяє керувати різноманітними функціями за допомогою голосових команд. Він забезпечує можливість використання пристроїв, не вимагаючи фізичного контакту або навіть зорового контакту з екраном. Головною перевагою голосових асистентів є їхня здатність розуміти природну мову і виконувати завдання за голосовими командами, що робить їх дуже зручними для використання в різних ситуаціях, включаючи водіння автомобіля, приготування їжі, роботу в офісі або вдома.

Голосові асистенти можуть мати різні функції, відповідно до потреб користувача та характеристик пристрою. Вони можуть надавати інформацію про погоду, новини, трафік або результати спортивних подій; виконувати завдання управління смарт-пристроями в домашньому середовищі, такими як регулювання температури, включення світла або запуск побутових приладів; надавати підказки або інструкції з різних сфер, таких як кулінарія, фітнес, подорожі або освіта; а також виконувати операції на базі штучного інтелекту, такі як розпізнавання мовлення, аналіз тексту або розуміння контексту.

Основними платформами для реалізації голосових асистентів є такі компанії, як Apple (Siri), Google (Google Assistant), Amazon (Alexa), Microsoft (Cortana) та інші. Вони постійно розвивають свої технології, щоб забезпечити кращу функціональність, точність розпізнавання та швидкість реакції, що робить голосових асистентів все більш популярними та корисними для користувачів у різних сферах життя.

На відміну від традиційних інтерфейсів, таких як клавіатура або сенсорний екран, голосові асистенти відкривають нові можливості для взаємодії з технологіями, що робить їх дуже привабливими для широкого кола користувачів. Зокрема, вони можуть бути надзвичайно корисними для людей з обмеженими можливостями, такими як відсутність зору чи рухові обмеження, оскільки дозволяють керувати пристроями та отримувати інформацію за допомогою голосу.

Голосові асистенти також мають значний потенціал у сфері бізнесу та промисловості. Вони можуть бути використані для автоматизації рутинних операцій, таких як обробка замовлень або відповіді на запити клієнтів у call-центрах, що дозволить компаніям підвищити продуктивність та знизити витрати на робочу силу.

З вдосконаленням технологій розпізнавання мовлення, обробки природної мови та штучного інтелекту, голосові асистенти продовжують розвиватися і ставати все більш інтелектуальними та пристосованими до потреб користувачів. Вони можуть стати важливою частиною майбутнього технологічного ландшафту, відкриваючи нові можливості для зручної та ефективної взаємодії з комп'ютерами та іншими пристроями у всіх сферах життя.

1.2 Технології для розробки голосового асистента

Розробка голосового асистента - це складний процес, який використовує різні технології для забезпечення його функціональності. Одна з ключових технологій - це розпізнавання мовлення, яке дозволяє асистентові перетворювати вимовлені користувачем слова в текстовий формат. Для розуміння змісту цього

тексту використовується технологія обробки природної мови (Natural Language Processing, NLP), яка аналізує синтаксис і семантику команди. Потім голосовий асистент генерує відповідь, яку можна прослухати, використовуючи технологію голосового синтезу.

Всі ці функції підтримуються штучним інтелектом (AI) і машинним навчанням (Machine Learning), які навчають асистента розуміти і адаптуватися до різних варіацій команд. Крім того, важливою є обробка сигналів, яка забезпечує якість розпізнавання голосу, фільтруючи шуми та інші завади. Ці технології разом дозволяють створити потужного голосового асистента, здатного ефективно спілкуватися з користувачами.

Отже створення голосового асистента - це складний технічний процес, що включає в себе декілька ключових аспектів. Один з них - це робота з розпізнаванням мовлення. Технології розпізнавання голосу дозволяють асистенту перетворювати вимовлені слова користувача на текстовий формат, що можна обробити далі.

Після цього важливим етапом є обробка природної мови. Ця технологія дозволяє асистенту розуміти семантику і контекст користувацьких запитів. Це означає, що асистент може розрізняти різні мовні конструкції, розуміти синоніми, а також адаптуватися до індивідуальних особливостей мовлення кожного користувача.

Після розуміння команди або запиту від користувача асистент генерує відповідь або виконує певну дію. Для цього використовується технологія голосового синтезу, що перетворює текстову інформацію на аудіоформат.

Завдяки штучному інтелекту і машинному навчанню асистент постійно вдосконалюється. Він навчається реагувати на нові команди, адаптується до індивідуальних потреб користувачів і вдосконалює свої навички відповідно до використаних даних.

Важливо також зазначити, що обробка сигналів грає значну роль у роботі голосового асистента. Вона допомагає зменшити шум та інші завади, що можуть впливати на якість розпізнавання мовлення. Технології обробки сигналів

дозволяють підвищити точність роботи асистента та зробити його використання більш зручним для користувачів.

1.3 Популярні голосові асистенти на сьогоднішній день

На сьогоднішній день існує кілька популярних голосових асистентів, які знаходять широке застосування в різних сферах життя та бізнесу.Найбільш відомі з них:

1. Siri (Apple): Siri є голосовим асистентом, розробленим компанією Apple для пристроїв на базі їхньої операційної системи iOS. Вона здатна виконувати різноманітні завдання, такі як надання інформації, виконання дій на пристрої, нагадування про події та багато іншого.

2. Google Assistant (Google): Google Assistant є голосовим асистентом, розробленим компанією Google, і доступний на різних платформах, включаючи Android-пристрої та деякі пристрої на базі iOS. Він забезпечує широкий спектр функцій, включаючи пошук інформації, надання порад, управління смарт-пристроями та інше.

3. Alexa (Amazon): Alexa є голосовим асистентом, розробленим компанією Amazon, який вперше був представлений у пристроях серії Amazon Echo. Вона може відтворювати музику, надавати новини та інформацію про погоду, керувати освітленням та іншими побутовими пристроями, а також виконувати покупки на Amazon.

4. Cortana (Microsoft): Cortana є голосовим асистентом, розробленим компанією Microsoft, і доступний на пристроях з операційною системою Windows та деяких мобільних платформах. Вона може надавати різноманітну інформацію, планувати події, відправляти повідомлення та багато іншого.

Ці голосові асистенти відіграють значну роль у побуті та бізнесі, надаючи користувачам зручний спосіб взаємодії з їхніми пристроями та отримання необхідної інформації. Вони постійно вдосконалюються та розширюють свої можливості завдяки швидкому розвитку технологій штучного інтелекту та обробки мовлення.

Siri (Apple): Siri - це голосовий асистент, розроблений Apple, який здатний

виконувати різноманітні завдання за голосовими командами користувача. Він доступний на пристроях, що працюють під управлінням операційної системи iOS, таких як iPhone, iPad та інші пристрої Apple. Siri може відповідати на питання користувачів, надавати інформацію про погоду, новини, розклади подій, а також виконувати різні дії, такі як надсилання повідомлень, здійснення дзвінків, встановлення нагадувань і багато іншого.

Google Assistant (Google): Google Assistant - це голосовий асистент, розроблений компанією Google, який доступний на різних платформах, включаючи Android-пристрої та пристрої з операційною системою iOS. Google Assistant пропонує широкий спектр функцій, включаючи пошук інформації, нагадування, навігацію, керування смарт-пристроями та інші завдання.

Alexa (Amazon): Alexa - це голосовий асистент, розроблений Amazon, який спочатку був представлений у пристроях серії Amazon Echo. Він може відтворювати музику, надавати новини та інформацію про погоду, керувати освітленням та іншими побутовими пристроями, а також виконувати покупки на Amazon. Alexa також має велику кількість навчальних навичок (skills), які розширюють її можливості.

Cortana (Microsoft): Cortana - це голосовий асистент, розроблений компанією Microsoft. Він доступний на пристроях з операційною системою Windows та деяких мобільних платформах. Cortana здатна надавати різноманітну інформацію, планувати події, надсилати повідомлення, управляти календарем, виконувати пошук в Інтернеті та багато іншого. Крім того, вона може інтегруватися з іншими сервісами Microsoft, такими як Outlook, Skype, OneDrive і т.д. Cortana також має функцію навчання, яка дозволяє асистенту адаптуватися до індивідуальних потреб користувача та надавати персоналізовані рекомендації і повідомлення.

1.4 Застосування голосових асистентів у повсякденному житті

У повсякденному житті голосові асистенти відіграють значну роль, надаючи користувачам зручний та ефективний спосіб взаємодії з їхніми

пристроями та отримання необхідної інформації. З основних застосувань голосових асистентів у повсякденному житті:

1. Пошук інформації: Користувачі можуть задавати голосові запити для отримання інформації про різні теми, такі як погода, новини, історія, кулінарні рецепти, туристичні маршрути тощо.

2. Нагадування і планування: Голосові асистенти допомагають користувачам планувати їхні дні, надсилаючи нагадування про зустрічі, події, дедлайни, а також допомагаючи створювати списки покупок чи завдань.

3. Керування побутовою технікою: Деякі голосові асистенти інтегруються з розумними пристроями вдома, такими як освітлення, термостати, аудіо- та відеосистеми, дозволяючи користувачам керувати ними за допомогою голосових команд.

4. Навігація та подорожі: Голосові асистенти можуть надавати користувачам інформацію про маршрути, розклади громадського транспорту, а також допомагати з пошуком ресторанів, готелів чи магазинів у новому місці.

5. Комерційні операції: Деякі голосові асистенти дозволяють користувачам здійснювати покупки онлайн, роблячи замовлення через інтернет-магазини або додатки за допомогою голосових команд.

Поміж широкого спектру застосувань голосових асистентів у повсякденному житті варто також відзначити їхню роль у підтримці продуктивності та організації робочих процесів. Голосові асистенти стають невід'ємною частиною робочого середовища для багатьох людей, які використовують їх для ефективного керування робочими завданнями та комунікацією.

1. Організація робочих завдань: Голосові асистенти допомагають у плануванні робочих завдань, нагадуючи про важливі події, встановлюючи терміни та дедлайни, створюючи списки завдань та розписи робочого дня.

2. Управління електронною поштою та календарем: Користувачі можуть використовувати голосові асистенти для перегляду та відповіді на електронні листи, планування зустрічей, додавання подій до календаря та організації

робочих зустрічей.

3. Пошук інформації для роботи: Голосові асистенти є потужними інструментами для пошуку інформації, необхідної для виконання робочих завдань. Користувачі можуть швидко знаходити відповіді на запитання, шукати ресурси для досліджень, а також отримувати оновлення про індустрію та конкурентів.

4. Комерційні операції: У бізнесі голосові асистенти можуть використовуватися для здійснення покупок, оформлення замовлень, взаємодії з клієнтами та партнерами через електронні системи, а також для отримання фінансової інформації та аналізу ринку.

5. Навчання та розвиток: Голосові асистенти можуть стати цінними навчальними інструментами, надаючи користувачам доступ до навчальних ресурсів, курсів, онлайн-лекцій та додаткової інформації для самонавчання та професійного розвитку.

Загалом, голосові асистенти сприяють підвищенню продуктивності, організації робочих процесів та забезпечують ефективну взаємодію з технологіями у сфері бізнесу та роботи.

1.5 Потенційні виклики та обмеження при розробці голосового асистента на Python

При розробці голосового асистента на Python можуть виникнути різні виклики та обмеження, які варто врахувати:

1. Розпізнавання мовлення: Одним з ключових викликів є точність та ефективність розпізнавання мовлення. Python має кілька бібліотек для роботи з розпізнаванням мовлення, таких як SpeechRecognition, pocketsphinx, але точність їхньої роботи може бути обмеженою, особливо в разі розпізнавання мовлення з різних акцентів або в умовах шуму, також можливі проблеми і із приладами вводу звуку.

2. Обробка природної мови (NLP): Необхідно реалізувати алгоритми обробки природної мови для розуміння користувацьких команд і виконання

відповідних дій. Бібліотеки, такі як NLTK, spaCy, можуть допомогти у цьому, але це вимагає глибоких знань у мовному аналізі та алгоритмах машинного навчання або ж можна використати вже готові дата сети для розпізнавання природної мови.

3. Інтеграція з іншими сервісами та API: Для надання різноманітних функцій голосовому асистенту може бути необхідно інтегруватися з різними сервісами та API, такими як погодні сервіси, новинні портали, календарі, тощо. Це може вимагати додаткового вивчення документації та розробки API-запитів у Python.

4. Архітектура та оптимізація коду: Розробка голосового асистента вимагає добре структурованого та оптимізованого коду. Потрібно розробити ефективну архітектуру програми, що дозволить легко розширювати та модифікувати функціонал асистента.

5. Безпека та конфіденційність даних: Оскільки голосові асистенти обробляють конфіденційну інформацію користувачів, важливо забезпечити належний рівень безпеки та захисту даних. Необхідно уникати зберігання або передачі чутливих даних у незахищеному вигляді.

6. Тестування та валідація: Важливо провести валідацію та тестування голосового асистента на різних тестових наборах даних, щоб забезпечити його правильну роботу в різних умовах та з різними типами запитів користувачів.

Загалом, вирішення цих проблем і викликів при розробці голосового асистента на Python вимагає поєднання технічних знань, кращих практик програмування та уважного вивчення документації та інструментів, що доступні для розробки голосових інтерфейсів. Подальші ж кроки у розвитку голосового асистента на Python можуть також включати:

1.Оптимізація швидкості та продуктивності: Важливо забезпечити швидку реакцію асистента на команди користувача та оптимізувати швидкість його роботи. Це може включати вдосконалення алгоритмів обробки мовлення та природної мови, оптимізацію роботи з базами даних та використання кешування для прискорення доступу до інформації.

2. Розширення функціоналу: На додаток до основних функцій, таких як надання інформації або виконання завдань, можна розглядати можливості розширення функціоналу асистента. Це може бути інтеграція з різними сервісами та додатками, створення персоналізованих рекомендацій або розробка додаткових навчальних модулів.

3. Підтримка мультиплатформеності: Для забезпечення максимальної доступності користувачів можна розглядати можливість розробки версій асистента для різних платформ, таких як мобільні пристрої, веб-додатки або різноманітні розумні пристрої.

4. Удосконалення інтерфейсу користувача: Важливо забезпечити зручний та інтуїтивно зрозумілий інтерфейс користувача для взаємодії з голосовим асистентом. Це може включати розробку голосових меню, покращення відповідей асистента та використання графічних елементів для підтримки взаємодії.

5. Контроль якості та забезпечення коректної роботи: Після релізу асистента важливо проводити постійний контроль якості його роботи та вносити відповідні покращення та зміни на основі отриманих відгуків користувачів та результатів тестування.

1.6 Переваги використання Python для розробки голосового асистента

Використання Python для розробки голосового асистента має декілька вагомих переваг, які роблять цей мову програмування привабливим вибором для створення таких програм:

1. Простота вивчення і використання: Python відомий своєю простотою та легкістю вивчення, що робить його ідеальним вибором для студентів та початківців у програмуванні. Він має простий синтаксис та велику кількість документації, що дозволяє швидко освоювати його та розвивати проекти.

2. Велика спільнота та екосистема: Python має велику та активну спільноту розробників, що означає наявність великої кількості бібліотек, фреймворків та інструментів для розробки різноманітних програм. Це дозволяє легко і швидко

впроваджувати нові функції та вирішувати проблеми під час розробки голосового асистента.

3. Підтримка штучного інтелекту та машинного навчання: Python має потужні бібліотеки та фреймворки для роботи з штучним інтелектом та машинним навчанням, такі як TensorFlow, PyTorch, scikit-learn та інші. Це дозволяє використовувати передові техніки обробки мовлення та розпізнавання мовлення для створення ефективного голосового асистента.

4. Переносність та сумісність: Python є переносним мовою програмування, що означає, що програми, написані на ньому, можуть працювати на різних операційних системах, таких як Windows, macOS, Linux, а також на різних апаратних платформах. Це дає можливість розробляти голосові асистенти, які можуть використовуватися на різних пристроях і платформах.

Далі важливо розглянути деякі специфічні переваги використання Python для розробки голосового асистента:

1. Багатофункціональність: Python має багато вбудованих бібліотек та модулів, які дозволяють реалізувати різноманітні функції голосового асистента. Наприклад, модулі для роботи з розпізнаванням мовлення, обробки природної мови, роботи з базами даних та інші. Це спрощує процес розробки та дозволяє швидше створювати функціональні голосові асистенти.

2. Швидкість розробки: Python відомий своєю високою продуктивністю та швидкістю розробки. Його простий синтаксис та динамічна типізація дозволяють швидко створювати прототипи та експериментувати з різними ідеями. Це особливо важливо в динамічному середовищі розробки голосових асистентів, де швидкість реагування на зміни на ринку може бути ключовим фактором успіху.

3. Розширені можливості машинного навчання: Python є популярним мовою програмування для розробки систем штучного інтелекту та машинного навчання. Велика кількість бібліотек, таких як TensorFlow, PyTorch, scikit-learn, дозволяє використовувати передові технології машинного навчання для покращення функціональності та ефективності голосового асистента.

Наприклад, можливість використовувати нейронні мережі для покращення точності розпізнавання мовлення або обробки природної мови.

Загалом, використання мови програмування Python для розробки голосового асистента є перевагою з багатьох аспектів і враховуючи вище згадані переваги, Python є привабливим вибором для розробки голосових асистентів, який дозволяє ефективно втілювати ідеї та створювати потужні та інтуїтивно зрозумілі програми.

РОЗІДЛ 2
ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОЕКТУ

2.1 Аналіз вимог до програмного забезпечення, що розробляється

В світі, де технології стають все більшою частиною повсякденного життя, голосові асистенти є одним із найбільш актуальних напрямів розвитку програмного забезпечення. Розробка голосового асистента для ПК на Python відкриває широкі можливості для зручного та ефективного взаємодії користувачів з комп'ютерами.

Першочерговою вимогою до такого програмного продукту є здатність точно розпізнавати мову користувача. Це означає не лише здатність розпізнавати різноманітні акценти та вимову, а й урахування контексту та семантики висловлювань. Для досягнення цієї мети використовуються нейронні мережі та алгоритми машинного навчання, які постійно вдосконалюються.

Однак розпізнавання мови — лише один з аспектів функціональних вимог. Голосовий асистент також повинен вміти ефективно розпізнавати команди користувача та виконувати відповідні дії. Це може охоплювати запуск програм, виконання пошукових запитів в Інтернеті, нагадування про події, налаштування таймерів тощо. Важливою вимогою є інтеграція з іншими програмами на комп'ютері, щоб асистент міг взаємодіяти з ними для виконання різноманітних завдань.

Під час розробки голосового асистента важливо враховувати не лише функціональні, а й нефункціональні вимоги. Наприклад, система повинна працювати швидко та максимально ефективно, без помітних затримок у відгуках. Точність розпізнавання мови та команд також є ключовою характеристикою, оскільки від неї залежить задоволення користувача від роботи асистента.

До інших нефункціональних вимог можна віднести мовну підтримку, яка передбачає підтримку різних мовних версій для коректного розпізнавання та відтворення тексту. Оскільки голосові асистенти мають взаємодіяти з

користувачами різних культур та мовних груп, це стає важливою складовою їх функціональності.

Під час розгляду теми розробки голосового асистента на Python, важливо звернути увагу на аспекти безпеки та приватності. Оскільки голосовий асистент обробляє голосові команди користувача, він може збирати та обробляти конфіденційну інформацію. Тому важливо захист інформації від несанкціонованого доступу та зловживання.

Ще однією важливою аспектом є робота голосового асистента в режимі реального часу. Швидка відповідь на команди користувача є ключовою для забезпечення позитивного досвіду взаємодії з програмою. Це вимагає оптимізації алгоритмів розпізнавання мови та виконання команд, а також ефективного використання ресурсів комп'ютера.

Одним із важливих етапів розробки голосового асистента є тестування. Для забезпечення якості та надійності програмного забезпечення необхідно провести ретельне тестування на різних платформах та у різних умовах. Тестування також допомагає виявити та виправити можливі помилки та недоліки в роботі асистента.

Крім того, розробка голосового асистента є динамічним процесом, оскільки технології швидко розвиваються, а потреби користувачів постійно змінюються. Тому важливо мати механізми для оновлення та покращення програмного забезпечення з часом.

2.2 Проектування

Проектування голосового асистента для ПК на Python – це складний процес, який вимагає ретельного планування та дотримання багатьох етапів розробки. В даному керівництві ми розглянемо кожен з цих етапів, починаючи від аналізу вимог і закінчуючи впровадженням та підтримкою програмного забезпечення. Ми опишемо процес створення голосового асистента, його архітектуру, методи реалізації, тестування та валідацію системи.

На першому етапі необхідно створити діаграму вимог, яка визначить

функціональні та нефункціональні вимоги до системи. Функціональні вимоги можуть включати розпізнавання мови, обробку голосових команд, інтеграцію з іншими програмами та відтворення голосового відгуку. Нефункціональні вимоги можуть включати високу точність розпізнавання мови, швидкість обробки команд, масштабованість системи та захист даних користувача.

Ідентифікація потреб користувачів

На цьому етапі слід провести дослідження цільової аудиторії, визначити, які завдання вони хочуть вирішувати за допомогою голосового асистента, та які функції для них є найбільш важливими.

Діаграма класів допоможе визначити структуру програми та зв'язки між класами, що складають систему. Основні класи можуть включати клас для розпізнавання мови, клас для обробки команд, клас для інтеграції з іншими програмами та клас для відтворення голосового відгуку.

Діаграма послідовності використовується для моделювання взаємодії між різними компонентами системи під час виконання певної функціональності. Це допоможе зрозуміти, як саме відбувається обробка команд та як взаємодіють між собою різні частини системи.

Діаграма компонентів дозволить визначити компоненти програми та їх взаємозв'язки, що допоможе забезпечити гнучкість та масштабованість системи. Вона включатиме модулі для розпізнавання мови, обробки команд, інтеграції та відтворення відгуку.

Після створення базових діаграм архітектури слід додати більше деталей, таких як методи та властивості класів, а також їх взаємодію. Це допоможе розробникам краще зрозуміти, як саме має бути реалізована кожна частина системи.

Діаграми активності допоможуть моделювати процеси та алгоритми, які використовуються в системі. Це може включати процес розпізнавання мови, обробки команд та взаємодії з іншими програмами.

Реалізація голосового асистента включає розробку модулів для розпізнавання мови, обробки голосових команд, взаємодії з іншими програмами

на комп'ютері та відтворення голосового відгуку. Кожен з цих модулів може бути втілений у відповідних класах та функціях відповідно до архітектурного плану.

Під час розробки необхідно дотримуватися кращих практик програмування та стандартів оформлення коду для забезпечення читабельності та підтримки, таких як SOLID, DRY (Don't Repeat Yourself) та інші. Також слід ретельно валідувати код під час реалізації, щоб переконатися у правильності його роботи та відповідності вимогам.

Паралельно з розробкою програмного коду важливо також проводити тестування, щоб перевірити правильність роботи різних частин системи та їх взаємодію. Тестування може бути проведено як автоматизовано за допомогою тестових фреймворків та інструментів, так і вручну для виявлення можливих помилок та недоліків.

Після успішної реалізації та тестування системи можна переходити до етапу випробування та впровадження. На цьому етапі асистент може бути випробуваний реальними користувачами для отримання зворотного зв'язку та виявлення можливих проблем. Після завершення випробувань та внесення всіх необхідних змін програма може бути готова до впровадження в реальне виробниче середовище.

Після впровадження програмного забезпечення в реальне середовище користувачі можуть почати активно використовувати голосового асистента. Протягом цього етапу важливо забезпечити підтримку та обслуговування системи, щоб вона продовжувала працювати безперервно та ефективно. Одним з аспектів підтримки є моніторинг роботи системи, що включає відстеження продуктивності, виявлення можливих проблем та вчасне їх вирішення. Для цього можна використовувати різноманітні інструменти моніторингу та журналювання.

Також важливо забезпечити регулярне оновлення програмного забезпечення з метою вдосконалення та виправлення можливих помилок. Оновлення можуть включати нові функціональності, покращення ефективності та безпеки, а також виправлення виявлених помилок.

Паралельно з цим, важливо продовжувати вдосконалювати систему на основі отриманого зворотного зв'язку від користувачів. Це може включати впровадження нових можливостей, покращення інтерфейсу користувача, а також зміни в алгоритмах роботи системи для забезпечення кращого взаємодії з користувачем.

З розвитком нових технологій, таких як штучний інтелект і машинне навчання, з'являються нові можливості для покращення роботи голосового асистента. Наприклад, можна інтегрувати алгоритми глибокого навчання для покращення точності розпізнавання мови та інтерпретації команд.

З часом може виникнути потреба у підтримці нових мов або діалектів, що вимагає адаптації алгоритмів розпізнавання мови. Це може включати збирання та навчання моделей на нових мовних даних, а також тісну співпрацю з носіями мови для перевірки та валідації результатів.

Постійна підтримка та обслуговування системи включає моніторинг її роботи, вчасне виявлення та вирішення проблем, а також регулярні оновлення програмного забезпечення. Це допоможе забезпечити стабільну та ефективну роботу голосового асистента протягом усього його життєвого циклу.

2.3 Програмна реалізація

Під час реалізації проекту було розроблено кілька ключових модулів, які забезпечують взаємодію користувача з системою через голосові команди. Основні компоненти проекту включають модулі для розпізнавання мови, обробки тексту, та виконання відповідних команд.

Модуль ai.py розроблений для захоплення та обробки голосових команд, використовуючи бібліотеку speech_recognition. Основна функція цього модуля, recognize_speech_from_mic, виконує декілька важливих завдань, щоб забезпечити точне розпізнавання мови. Цей модуль відіграє ключову роль у взаємодії користувачів з системою через голосові команди, що значно покращує користувацький досвід.

Асинхронна функція recognize_speech_from_mic здійснює захоплення

аудіо з мікрофону. Після отримання аудіо, функція коригує його для зменшення шуму, що покращує якість розпізнавання. Очищене аудіо передається на обробку через Google API, який забезпечує високу точність розпізнавання мови.

Однією з важливих особливостей функції є обробка помилок за допомогою винятків. Це дозволяє забезпечити користувачу відповідний зворотний зв'язок у випадку, якщо розпізнавання мови не вдалося або виникли проблеми з сервісом. Таким чином, користувач завжди отримує інформацію про стан системи та можливі дії для виправлення ситуації.

На початкових етапах розробки розглядався варіант створення власного API для розпізнавання мови. Однак цей варіант був відкинутий через кілька причин. По-перше, створення та підтримка власного API потребувала б значних ресурсів та часу, що могло б уповільнити розробку основного продукту. По-друге, ефективність і точність власного рішення могла бути значно нижчою порівняно з існуючими комерційними рішеннями. Тому було прийнято рішення використовувати Google API, який забезпечує високу точність та надійність розпізнавання мови.

Використання Google API дозволило зосередитися на інших важливих аспектах проєкту та забезпечити користувачам стабільний і якісний сервіс для розпізнавання голосових команд. Це рішення виявилося оптимальним з точки зору якості, швидкості реалізації та підтримки системи.

Лістинг 2.1

```
import speech_recognition as sr
import asyncio
from ai_base import base
async def recognize_speech_from_mic(recognizer, microphone):
    """Асинхронне розпізнавання мови з мікрофону"""
    with microphone as source:
        recognizer.adjust_for_ambient_noise(source, duration = 0.5)
        print("Слухаю...")
        audio = recognizer.listen(source)
      try:
        response = recognizer.recognize_google(audio, language = "uk-UA")
        await process_text(response)
```

Продовжння лістинг 2.1

```
      except sr.UnknownValueError:
        print("Не вдалося розпізнати аудіо")
    except sr.RequestError as e:
print(f"Помилка сервісу розпізнавання мови; {e}")

async def process_text(response):
    print("Ви сказали: " + response)
    base(response)

async def main():
    recognizer = sr.Recognizer()
    recognizer.pause_threshold = 0.5  # Зменшено час очікування тиші до 0.5
секунд
    microphone = sr.Microphone()

    print("Програма розпочала роботу. Говоріть в мікрофон.")

    while True:
        await recognize_speech_from_mic(recognizer, microphone)

if __name__ == "__main__":
    asyncio.run(main())
```

Після того, як голосова команда успішно розпізнана модулем ai.py за допомогою Google API, текстовий результат передається для подальшої обробки у модуль ai_base.py. Цей модуль відповідає за зіставлення введеного користувачем тексту з доступними командами системи та забезпечує виконання відповідних дій на основі отриманих результатів.

У ai_base.py використовується бібліотека fuzzywuzzy, яка спеціалізується на порівнянні рядків і визначенні ступеня їх схожості. Ця бібліотека дозволяє ефективно зіставити введений користувачем текст з переліком визначених команд системи. Fuzzywuzzy використовує алгоритми порівняння, які оцінюють, наскільки введений текст близький до будь-якої з доступних команд, враховуючи можливі помилки або варіації у введенні.

Коли користувач вводить голосову команду, вона перетворюється в текст і передається до ai_base.py. Тут fuzzywuzzy визначає ступінь схожості між введеною командою та наявними командами. Якщо ступінь схожості перевищує заданий поріг, команда вважається розпізнаною, і відповідна дія виконується. Це дозволяє системі бути більш гнучкою і точною у розпізнаванні команд, навіть якщо користувачі вводять їх із незначними варіаціями або помилками.

Цей підхід забезпечує високу точність та адаптивність системи. Користувачі можуть використовувати різні формулювання для одних і тих самих команд, а система все одно правильно їх розпізнає і виконує необхідні дії. Встановлення порогового значення для ступеня схожості також дозволяє контролювати чутливість системи та уникати виконання помилкових команд.

Таким чином, модуль ai_base.py відіграє важливу роль у забезпеченні інтерактивності та точності роботи системи, використовуючи потужні можливості бібліотеки fuzzywuzzy для обробки і зіставлення команд.

Лістинг 2.2

```
from dotenv import load_dotenv
import os
from fuzzywuzzy import fuzz
from function.power_off import shutdown_computer
from function.open_browser import commands as browser_commands
load_dotenv('.env')
commands = {
    os.getenv('POWER_OFF').lower(): shutdown_computer,
    **browser_commands
}
def check_similar_word_in_text(word, text, threshold = 80):
    tokens = text.split()
    for token in tokens:
        similarity = fuzz.ratio(word, token)
        if similarity >= threshold:
            return True
    return False
def match_command(input_command, command_list, threshold=80):
    best_match = None
    highest_similarity = 0
    for command in command_list:
        similarity = fuzz.ratio(input_command.lower(), command.lower())
        print(f"Comparing '{input_command}' with '{command}': similarity =
{similarity}")
        if similarity > highest_similarity:
            highest_similarity = similarity
            best_match = command
    if highest_similarity >= threshold:
        return best_match
```

Продовжння лістинг 2.2

```
return None
def base(responce):
    name = os.getenv('NAME').lower()
    responce = responce.lower()
    threshold = int(os.getenv('THRESHOLD'))
    check = check_similar_word_in_text(
        name, responce, threshold
    )
    if check:
        call_func(responce)
def call_func(responce):
    threshold = int(os.getenv('THRESHOLD_COMMANDS'))
    print(commands)
    matched_command = match_command(
        responce, commands.keys(), threshold)
    if matched_command:
        commands[matched_command]()
```

Для забезпечення відстеження процесів та виявлення помилок у системі, використовується модуль logging, налаштований у файлі logs.py. Логування відіграє ключову роль у підтримці стабільності та надійності програмного забезпечення, оскільки дозволяє розробникам і адміністраторам системи своєчасно виявляти та реагувати на помилки і важливі події.

Модуль logging конфігурується таким чином, щоб фіксувати широкий спектр подій, включаючи інформаційні повідомлення, попередження, помилки та критичні помилки. Кожен запис логування містить детальну інформацію про подію: дату та час її виникнення, рівень помилки та деталі, що допомагають у діагностиці та виправленні проблем.

Налаштування логування у файлі logs.py передбачає створення кількох обробників (handlers) для різних типів логів. Наприклад, окремі файли логів можуть використовуватися для запису помилок, попереджень та інформаційних повідомлень. Це дозволяє зручно організовувати та аналізувати лог-файли, швидко знаходячи необхідну інформацію.

Логування важливих подій включає відстеження старту та завершення основних процесів, виконання команд користувача, взаємодію з зовнішніми API та будь-які інші значущі дії. Це допомагає створити повну картину роботи системи та забезпечити її прозорість для розробників.

Коли виникають помилки, система фіксує їх у логах з максимальною деталізацією. Це включає стек викликів, повідомлення про помилки, та іншу релевантну інформацію, що допомагає швидко ідентифікувати і усунути причину проблеми. Розробники можуть аналізувати ці логи для проведення діагностики, що значно полегшує процес налагодження програмного коду та підвищує його якість.

Завдяки налаштованому модулю logging, система може оперативно реагувати на виникаючі помилки та підтримувати стабільну роботу, забезпечуючи високий рівень обслуговування користувачів та мінімізуючи ризик тривалих збоїв у роботі програмного забезпечення.

Лістинг 2.3

```
import logging
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s -
%(message)s')
console_handler = logging.StreamHandler()
console_handler.setFormatter(formatter)
logger.addHandler(console_handler)
```

У системі є кілька функціональних модулів, які виконують конкретні завдання, забезпечуючи широкий спектр можливостей для користувачів. Одним із таких модулів є модуль power_off, який відповідає за вимкнення комп'ютера. Цей модуль реалізує команду, що дозволяє користувачу легко та швидко вимкнути ПК за допомогою голосової команди.

Модуль power_off є лише одним із багатьох функціональних модулів у системі. Окрім нього, система включає ще безліч інших модулів, кожен з яких виконує спеціалізовані завдання. Наприклад, можуть бути модулі для керування

медіаплеєром, відкриття програм, перевірки електронної пошти, запуску веб-браузера та інших рутинних дій.

Кожен функціональний модуль інтегрований у загальну архітектуру системи та працює в тісній взаємодії з модулями розпізнавання мови та обробки команд. Це дозволяє користувачам використовувати широкий спектр голосових команд для виконання різноманітних дій без необхідності фізичної взаємодії з комп'ютером.

Окрім основних функцій, система може бути розширена додатковими модулями, залежно від потреб користувачів та специфіки застосування. Така модульна архітектура забезпечує високу гнучкість та адаптивність системи, дозволяючи легко додавати нові можливості та функції.

Загалом, наявність різноманітних функціональних модулів значно підвищує зручність використання системи та дозволяє автоматизувати безліч рутинних завдань. Користувачі можуть просто озвучувати команди, а система миттєво реагує, виконуючи необхідні дії. Це робить взаємодію з комп'ютером більш природною та ефективною, сприяючи підвищенню продуктивності та комфорту роботи.

Лістинг 2.4

```
import os
import platform
def shutdown_computer():
    system = platform.system()
    if system == 'Windows':
        os.system('shutdown /s /t 1')
    elif system == 'Linux':
        os.system('shutdown now')
    elif system == 'Darwin':
        os.system('sudo shutdown -h now')
    else:
        print(f'Unsupported operating system: {system}')
```

Усі ці компоненти разом забезпечують ефективну роботу системи, дозволяючи користувачам взаємодіяти з нею за допомогою голосових команд.

Ця інтеграція різноманітних модулів значно покращує користувацький досвід та функціональність проєкту, роблячи його більш інтуїтивним та зручним у використанні.

Завдяки комплексній взаємодії всіх цих компонентів, система стає потужним інструментом для автоматизації та покращення користувацького досвіду. Користувачі можуть легко виконувати різноманітні дії за допомогою голосових команд, що робить їх роботу зручнішою, швидшою та ефективнішою. Інтеграція та координація між різними модулями гарантує, що система працює безперебійно, забезпечуючи високий рівень задоволеності користувачів та розширюючи можливості проєкту.

РОЗІДЛ 3
ТЕСТУВАННЯ І СУПРОВОДЖЕННЯ

3.1 Перелік і обгрунтування обраних методів тестуванн.

Для забезпечення надійності та коректної роботи системи використовувались декілька методів тестування.

Перш за все, застосувались юніт-тестування. Цей метод дозволяє перевірити окремі компоненти системи на рівні функцій чи методів. Юніт-тести допомагають виявити помилки в логіці та поведінці окремих частин коду.Використовуючи юніт-тести для перевірки модулів ai.py, ai_base.py, та функціональних модулів, таких як power_off.

Наступним важливим методом було інтеграційне тестування. Це тестування фокусується на перевірці взаємодії між різними модулями системи. Інтеграційні тести допомагають виявити проблеми, які можуть виникнути при з'єднанні окремих компонентів, забезпечуючи безперебійну роботу всієї системи. У нашому випадку основна увага приділялась перевірці взаємодії між модулями ai.py і ai_base.py, а також інтеграції функціональних модулів.

Системне тестування виконується для перевірки всієї системи як єдиного цілого, включаючи всі інтегровані модулі та їх взаємодію. Системне тестування дозволяє оцінити, наскільки система відповідає вимогам та очікуванням користувачів.

Також застосоване регресійне тестування. Воно проводиться після внесення змін або виправлень у систему для перевірки того, що нові зміни не призвели до появи нових помилок у вже протестованих і стабільних частинах коду.

Тестування користувацького інтерфейсу (UI тестування) дозволяє перевірити, наскільки зручно та інтуїтивно зрозумілою є система для кінцевих користувачів. Проведено тестування голосових команд, щоб переконатися, що система правильно реагує на різні команди та сценарії їх використання.

Окрім автоматизованих методів, також застосуване ручне тестування.

Ручне тестування є важливим етапом перевірки системи, оскільки дозволяє виявити ті проблеми, які можуть залишитися непоміченими автоматичними тестами. Воно особливо корисне для оцінки користувацького досвіду, перевірки роботи голосових команд у реальних умовах, і дає змогу виявити недоліки у взаємодії з користувачем. Це також дозволяє гнучко реагувати на зміни та швидко перевіряти нові функції перед автоматизацією тестів.

Додатково, деякі аспекти системи, такі як використання Google API для розпізнавання мови, не потребували детального тестування, оскільки ці сервіси вже були ретельно протестовані та підтримувалися компанією Google. Це дозволило зосередитися на тестуванні власне логіки та інтеграції, зменшуючи час і ресурси, необхідні для перевірки. Довіра до якості та стабільності Google API давала впевненість у тому, що частина системи, пов'язана з розпізнаванням мови, працює належним чином без додаткових зусиль.

Основна увага приділялася інтеграційному тестуванню, оскільки воно забезпечувало перевірку взаємодії між різними модулями системи. Це тестування дозволяло переконатися, що всі частини системи працюють разом без збоїв, і що не було конфліктів або проблем у передачі даних між модулями. Враховуючи складність системи, інтеграційні тести були критично важливими для виявлення та виправлення будь-яких проблем на ранніх етапах розробки.

Ручне тестування відігравало важливу роль у підході до тестування. Воно давало змогу оцінити реальний користувацький досвід, ідентифікувати можливі проблеми у взаємодії користувача з системою та перевірити роботу голосових команд у різних умовах. Залучення реальних користувачів до тестування дозволяло зібрати цінний зворотний зв'язок, який використовувався для покращення системи. Ручне тестування було особливо ефективним для перевірки нових функцій та швидкої ітерації змін перед їх автоматизацією.

Окрім автоматизованих методів, ручне тестування дозволяло гнучко реагувати на зміни та швидко перевіряти нові функції перед тим, як впроваджувати їх у повсякденне використання. Це значно підвищувало ефективність тестування та скорочувало час на виявлення та виправлення

помилок. Ручне тестування також сприяло виявленню дрібних, але важливих деталей, які могли бути пропущені під час автоматизованого тестування.

Таким чином, головна увага приділялася інтеграційному тестуванню для забезпечення правильної взаємодії між модулями системи, а також ручному тестуванню для оцінки користувацького досвіду та оперативного виявлення проблем. Це забезпечувало комплексний підхід до тестування, що дозволяло досягти високої якості та надійності системи. Завдяки цьому вдалося створити надійну і стабільну систему, яка відповідала всім вимогам і очікуванням користувачів, забезпечуючи їм зручність і задоволення від використання продукту.

3.2 Тестовий план проекту

Тестовий план проекту визначав стратегію, підходи, ресурси та графік виконання тестування. Основна мета тестового плану – забезпечити високу якість та надійність системи шляхом виявлення і виправлення можливих дефектів на ранніх етапах розробки. План включав різні види тестів для комплексної перевірки всіх компонентів системи. Також тестовий план проекту визначав обсяг робіт, методологію та ресурси, необхідні для виконання тестування. Він охоплював як автоматизовані, так і ручні методи тестування, а також спеціальні тести для перевірки інтеграції та користувацького досвіду.

Основні об'єкти тестування включали кілька важливих компонентів, кожен з яких відіграє критичну роль у функціонуванні системи. Детальний опис цих компонентів допоможе зрозуміти, які аспекти системи піддавалися тестуванню і яким чином забезпечувалась їхня надійність та функціональність.

Модуль розпізнавання мови (ai.py) відповідає за ідентифікацію та інтерпретацію мовленнєвих команд, що надходять від користувача. Тестування цього модуля включало перевірку точності розпізнавання різних акцентів, швидкості мовлення та умов навколишнього середовища, таких як фоновий шум. Також важливим аспектом було тестування стійкості до помилок, зокрема, як система реагує на неправильні або незрозумілі команди.

Модуль обробки команд (ai_base.py) відповідає за інтерпретацію і виконання команд, розпізнаних мовним модулем. Тестування цього модуля включало перевірку правильності інтерпретації команд та їх відповідності очікуваним діям. Особлива увага приділялась обробці складних команд, які можуть містити кілька інструкцій або залежати від контексту попередніх команд.

Функціональні модулі (наприклад, power_off), такі як power_off, реалізують конкретні дії, що виконуються системою на основі команд користувача. Тестування цих модулів включало перевірку їхньої функціональної відповідності специфікаціям, а також їхньої надійності та безпечності. Наприклад, модуль power_off перевірявся на коректне вимкнення системи без втрати даних чи пошкодження файлової системи.

Інтерфейс користувача був важливою частиною тестування, оскільки він є основним засобом взаємодії користувача з системою. Тестування інтерфейсу включало перевірку його зручності використання, інтуїтивності, а також здатності до коректного відображення і введення інформації. Важливо було впевнитись, що інтерфейс адекватно реагує на дії користувача і забезпечує зворотний зв'язок у разі виникнення помилок.

Інтеграція між всіма компонентами системи останнім, але не менш важливим аспектом тестування була інтеграція всіх компонентів системи. Цей етап включав перевірку коректної взаємодії між модулем розпізнавання мови, модулем обробки команд, функціональними модулями та інтерфейсом користувача. Тестування інтеграції забезпечувало впевненість у тому, що всі частини системи працюють разом без збоїв, виконуючи свої функції належним чином і узгоджено.

Перш за все, застосовувалося юніт-тестування. Його мета полягала в перевірці окремих функцій та методів кожного модуля системи, зокрема ai.py та ai_base.py. Юніт-тестування дозволяє локалізувати та виправити помилки на ранніх етапах розробки. Для цього використовувалися інструменти unittest та pytest, які забезпечували можливість створення і виконання тестів, що покривали всі можливі варіанти виконання функцій. Наприклад, функції розпізнавання

мови тестувалися на правильність ідентифікації різних мовленнєвих команд, враховуючи різні акценти, швидкість мовлення та фоновий шум. Також тестувалися функції обробки команд та їх виконання, перевіряючи коректність інтерпретації і виконання відповідних дій.

Наступним важливим етапом було інтеграційне тестування. Його мета полягала в перевірці взаємодії між різними модулями системи. Інструменти, такі як pytest та requests для HTTP-запитів, використовувалися для перевірки інтеграції між модулями ai.py і ai_base.py, а також з функціональними модулями, такими як power_off. Це тестування включало перевірку того, як модулі передають дані один одному і чи правильно інтерпретують ці дані. Таким чином, інтеграційне тестування забезпечувало впевненість у тому, що система працює як єдине ціле, а не як набір окремих модулів.

Системне тестування включало перевірку всієї системи як цілісної. Автоматизовані скрипти та ручні перевірки використовувалися для оцінки роботи всіх компонентів системи разом, включаючи модуль розпізнавання мови, модуль обробки команд, функціональні модулі та інтерфейс користувача. Це тестування проводилося з метою виявлення помилок, які могли виникати під час взаємодії всіх компонентів, а також для перевірки відповідності системи вимогам і очікуванням користувачів. Системне тестування забезпечувало комплексну перевірку функціональності, продуктивності та надійності всієї системи.

Регресійне тестування мало на меті перевірку стабільності системи після внесення змін або оновлень. Для цього використовувався набір регресійних тестів, які перевіряли всі попередньо протестовані функції та модулі, щоб переконатися, що нові зміни не спричинили появу нових помилок або не порушили раніше працюючі функції. Регресійне тестування допомагало підтримувати стабільність і якість системи протягом усього життєвого циклу розробки.

Тестування користувацького інтерфейсу дозволяло оцінити зручність та функціональність інтерфейсу системи. Ручне тестування включало перевірку

роботи голосових команд, їх реакцій та відповідності очікуванням користувачів. Крім того, використовувалися скрипти для автоматизації UI-тестів, що забезпечували систематичну перевірку інтерфейсних елементів. Це тестування мало на меті виявити можливі проблеми з юзабіліті, інтуїтивністю інтерфейсу та забезпечити позитивний користувацький досвід.

Набір тестів для юніт-тестування був розроблений з метою перевірки основних функцій розпізнавання мови та обробки команд. Для цього використовувалися такі інструменти, як unittest та pytest. Тести перевіряли, чи правильно модуль розпізнає різні мовленнєві команди, враховуючи такі фактори, як акцент, швидкість мовлення та фоновий шум. Крім того, перевірялися функції обробки команд, щоб переконатися, що вони інтерпретують команди коректно та виконують відповідні дії. Наприклад, тест перевіряв, чи модуль правильно ідентифікує команду "вимкнути систему" і передає її на виконання відповідному функціональному модулю.

Інтеграційні тести включали сценарії, що перевіряли взаємодію між основними модулями, такими як ai.py та ai_base.py, а також з функціональними модулями, такими як power_off. Для перевірки взаємодії використовувалися інструменти pytest та requests для HTTP-запитів. Тести перевіряли, чи правильно модулі передають дані один одному і чи коректно інтерпретують ці дані. Наприклад, тест інтеграції перевіряв, чи модуль розпізнавання мови правильно передає розпізнану команду до модуля обробки команд, і чи виконується ця команда відповідним функціональним модулем без помилок.

Системне тестування охоплювало повну перевірку всієї системи в реальних умовах використання. Автоматизовані скрипти та ручні перевірки використовувалися для оцінки роботи всіх компонентів системи разом. Тести включали всі можливі сценарії користувацьких команд, від простих до складних. Наприклад, тест міг включати послідовність команд від користувача, таких як "увімкнути музику", "зменшити гучність" та "вимкнути систему", перевіряючи, чи система виконує ці команди коректно та у правильному порядку.

Регресійні тести включали повторне тестування всіх основних функцій

після внесення змін або оновлень системи. Це забезпечувало стабільність системи, гарантувавши, що нові зміни не спричинили появу нових помилок або не порушили роботу існуючих функцій. Набір регресійних тестів був розроблений таким чином, щоб охопити всі критичні аспекти системи, включаючи розпізнавання мови, обробку команд та виконання функціональних команд.

Тестування користувацького інтерфейсу дозволяло оцінити зручність та функціональність інтерфейсу системи. Це тестування включало перевірку реакції системи на різні голосові команди, забезпечуючи інтуїтивний та комфортний користувацький досвід. Ручне тестування та скрипти для автоматизації UI-тестів перевіряли, чи система коректно реагує на введення користувача та чи відображає відповідні результати. Наприклад, тест міг включати введення команд типу "показати погоду" або "встановити будильник", перевіряючи, чи інтерфейс відображає правильну інформацію та чи легко користувачеві взаємодіяти з системою.

Тестовий план проекту забезпечував комплексний підхід до перевірки всіх компонентів системи. Він включав детальні сценарії для кожного типу тестування, що дозволяло виявляти та виправляти можливі помилки на ранніх етапах розробки. Такий підхід гарантував, що всі аспекти системи були ретельно перевірені, від окремих функцій до повної інтеграції та реального використання.

Завдяки цьому комплексному підходу вдалося створити надійну і стабільну систему, що відповідала всім вимогам і очікуванням користувачів. Ретельне тестування всіх компонентів забезпечувало високу якість продукту, зручність використання та задоволення користувачів.

3.3 Аналіз отриманих результатів

Після проведення всіх запланованих тестів був проведений детальний аналіз отриманих результатів, щоб оцінити якість і надійність системи. Аналіз охоплював результати юніт-тестування, інтеграційного тестування, системного тестування, регресійного тестування та тестування користувацького інтерфейсу.

Юніт-тести показали високу точність у розпізнаванні мови та коректну обробку команд на базовому рівні. Більшість виявлених помилок були пов'язані з некоректною логікою у функціях розпізнавання мови та обробки команд. Всі ці помилки були швидко виправлені, що значно підвищило надійність окремих модулів системи. Загалом, юніт-тестування підтвердило, що основні функції системи працюють належним чином.

Інтеграційне тестування виявило декілька проблем у взаємодії між модулями, які були усунені шляхом оптимізації коду та покращення логіки взаємодії. Наприклад, були виявлені проблеми з передачею даних між модулем ai.py та модулем ai_base.py, що призводило до неправильного виконання команд. Всі виявлені недоліки були виправлені, що забезпечило безперебійну роботу всієї системи.

Системне тестування підтвердило, що система працює стабільно при виконанні комплексних сценаріїв використання. Тестування показало, що система здатна правильно розпізнавати та обробляти широкий спектр голосових команд у реальних умовах використання. Виявлені помилки були мінімальні та легко виправлялися. Це дозволило переконатися в тому, що система відповідає всім функціональним вимогам і забезпечує належний рівень продуктивності.

Регресійне тестування не виявило нових помилок після внесення змін, що свідчить про стабільність системи. Це означає, що нові функції та виправлення не вплинули негативно на вже протестовані та стабільні частини коду. Регресійне тестування дозволило впевнитися, що система залишається стабільною та надійною навіть після додавання нових можливостей і оновлень.

Тестування користувацького інтерфейсу показало, що інтерфейс є зручним для користувачів. Були проведені тести з різними варіантами голосових команд, що підтвердило інтуїтивність та функціональність інтерфейсу. Залучення групи тестувальників для збирання зворотного зв'язку дозволило внести деякі покращення на основі отриманих даних. Це сприяло створенню більш комфортного користувацького досвіду та підвищенню задоволеності користувачів.

Аналіз результатів тестування показав, що система є надійною та стабільною. Всі виявлені проблеми були своєчасно вирішені, що забезпечило високу якість кінцевого продукту. Завдяки комплексному підходу до тестування вдалося створити систему, яка відповідає всім вимогам і очікуванням користувачів, забезпечуючи їм зручність і задоволення від використання продукту.

Успішне завершення всіх етапів тестування дозволило виявити і усунути можливі недоліки системи ще на ранніх стадіях розробки. Це сприяло підвищенню ефективності роботи всієї команди розробників, забезпечуючи стабільну і надійну роботу системи при різних сценаріях використання. Виявлені проблеми у взаємодії між модулями були швидко вирішені, що сприяло поліпшенню загальної продуктивності та якості продукту.

Регулярне проведення регресійного тестування забезпечувало стабільність системи після внесення нових змін і оновлень. Це дозволило запобігти появі нових помилок і забезпечити стабільну роботу системи навіть після додавання нових функцій. Регресійні тести підтвердили, що всі основні функції системи залишаються стабільними і надійними, що є важливим показником якості кінцевого продукту.

Тестування користувацького інтерфейсу виявило декілька невеликих проблем, які були швидко виправлені на основі зворотного зв'язку від тестувальників. Це дозволило підвищити зручність і інтуїтивність використання системи, що, в свою чергу, позитивно вплинуло на загальну задоволеність користувачів. Завдяки цьому вдалося створити систему, яка не тільки відповідає технічним вимогам, але й забезпечує високий рівень користувацького досвіду.

Загальний аналіз результатів тестування показав, що система готова до впровадження і подальшої експлуатації. Всі цілі, поставлені перед командою розробників, були успішно досягнуті, а всі виявлені проблеми були ефективно вирішені. Це забезпечило високу якість кінцевого продукту, який відповідає всім вимогам та очікуванням користувачів.

ВИСНОВКИ

Після проведення всіх запланованих тестів, включаючи юніт-тестування, інтеграційне тестування, системне тестування, регресійне тестування та тестування користувацького інтерфейсу, система показала високу якість та надійність. Кожен з етапів тестування був спрямований на виявлення і усунення можливих помилок, що могло б вплинути на функціональність та зручність використання системи.

Юніт-тестування дозволило виявити та виправити помилки на рівні окремих функцій та методів, забезпечивши коректну роботу базових компонентів системи. Інтеграційне тестування перевірило взаємодію між модулями, переконавшись, що дані передаються та обробляються коректно між різними частинами системи. Системне тестування забезпечило перевірку всієї системи в умовах, близьких до реального використання, перевіряючи всі можливі сценарії взаємодії користувачів із системою. Регресійне тестування гарантувало, що внесення нових змін або оновлень не порушило роботу вже існуючих функцій системи. Тестування користувацького інтерфейсу оцінювало зручність та інтуїтивність використання системи, забезпечуючи позитивний користувацький досвід.

Результати тестування показали, що система відповідає всім технічним та функціональним вимогам. Всі виявлені помилки були оперативно виправлені, що дозволило створити стабільний та функціональний продукт. Аналіз результатів кожного етапу тестування підтвердив, що система виконує свої завдання ефективно та надійно. Всі тести підтвердили, що система працює без збоїв і відповідає заданим технічним специфікаціям. Виправлення виявлених помилок та регресійне тестування забезпечили стабільну роботу системи після внесення змін. Система повністю відповідала функціональним вимогам, передбаченим на етапі проектування.

Однією з ключових цілей тестування було забезпечення зручного та інтуїтивного інтерфейсу для взаємодії користувачів із системою за допомогою голосових команд. Тестування користувацького інтерфейсу підтвердило, що

система є легкою у використанні, інтуїтивно зрозумілою та надає швидку і точну відповідь на голосові команди. Користувачі могли легко взаємодіяти із системою завдяки простому та зрозумілому інтерфейсу. Система коректно розпізнавала і реагувала на голосові команди, забезпечуючи плавну та ефективну взаємодію. Користувачі високо оцінили зручність та функціональність системи, що сприяло підвищенню їх задоволеності продуктом.

Комплексний підхід до тестування, який включав юніт-тестування, інтеграційне, системне, регресійне тестування та тестування користувацького інтерфейсу, дозволив створити надійну і стабільну систему. Завдяки ретельному аналізу та виправленню всіх виявлених помилок вдалося забезпечити високу якість продукту, який відповідає всім технічним та функціональним вимогам і надає користувачам зручний та інтуїтивний інтерфейс для взаємодії за допомогою голосових команд.

Після завершення основного циклу тестування, включаючи юніт-тестування, інтеграційне тестування, системне тестування, регресійне тестування та тестування користувацького інтерфейсу, рекомендується продовжити проведення регулярного регресійного тестування після кожного оновлення системи. Це допоможе забезпечити стабільність та надійність продукту, оскільки навіть незначні зміни в коді можуть призвести до появи нових помилок або вплинути на роботу існуючих функцій.

Для забезпечення найвищої якості продукту варто залучати реальних користувачів для проведення тестування інтерфейсу. Це дозволить збирати зворотний зв'язок про зручність використання, інтуїтивність і функціональність системи, а також виявляти можливі проблеми, які можуть бути неочевидними для команди розробників. Такий підхід допоможе вносити необхідні покращення і вдосконалювати інтерфейс, орієнтуючись на реальні потреби та очікування користувачів.

Використання автоматизованих інструментів для тестування є важливим кроком для зниження навантаження на команду розробників та підвищення ефективності процесу тестування. Автоматизація дозволяє швидко і точно

виконувати великі обсяги тестів, забезпечуючи повторюваність і точність перевірок. Це особливо корисно при регресійному тестуванні, де необхідно регулярно перевіряти вже існуючі функції після внесення змін або оновлень. Інструменти автоматизації, такі як Selenium для тестування користувацького інтерфейсу або Jenkins для організації автоматизованих збірок і тестувань, можуть значно підвищити ефективність і швидкість тестування.

З огляду на позитивні результати тестування, система готова до впровадження та подальшої експлуатації. Вона продемонструвала високу якість, стабільність і надійність, що дозволяє використовувати її у різних сферах, де потрібна взаємодія за допомогою голосових команд. Потенційні сфери застосування включають розумні будинки, автомобільні системи, а також різноманітні інформаційні та сервісні додатки.

У розумних будинках система може використовуватися для управління освітленням, опаленням, системами безпеки та іншими побутовими пристроями. Це забезпечить зручність і комфорт мешканців, дозволяючи їм керувати домашніми системами за допомогою голосу. В автомобільних системах система може використовуватися для голосового управління навігацією, аудіо- і відеосистемами, що підвищить безпеку водіння, зменшуючи необхідність відволікання водія на керування різними пристроями вручну.

У сфері інформаційних та сервісних додатків система може використовуватися для надання користувачам швидкого доступу до інформації або послуг. Наприклад, в банківських або страхових додатках вона може забезпечувати швидке і зручне обслуговування клієнтів, відповідаючи на їхні запити і виконуючи необхідні операції. В освітніх додатках система може використовуватися для надання інструкцій або довідкової інформації.

Подальший розвиток системи може включати інтеграцію з новими сервісами та розширення функціональності. Це дозволить забезпечити ще більшу зручність і гнучкість у використанні, а також адаптувати систему до потреб різних категорій користувачів. Інтеграція з популярними платформами та сервісами, такими як Google Home, Amazon Alexa або Apple HomeKit, може

розширити можливості системи та зробити її ще більш універсальною.

Розширення функціональності може включати додавання нових мов і діалектів для розпізнавання, підтримку нових типів команд і сценаріїв взаємодії, а також підвищення точності розпізнавання голосу в різних умовах. Це дозволить забезпечити максимально точне і швидке виконання команд, підвищуючи задоволеність користувачів.

У процесі розробки та тестування системи виникали певні складнощі, які вимагали ретельного підходу до їх вирішення. Однією з основних проблем була інтеграція різних модулів. Вона потребувала ретельної перевірки та налаштування для забезпечення коректної взаємодії між компонентами. Це включало не лише технічні аспекти, але й узгодження методів передачі даних, формати повідомлень і обробку помилок, щоб гарантувати стабільну та безперебійну роботу системи.

Також виникали труднощі з налаштуванням та використанням API від Google. Інтеграція з цими API була важливою для забезпечення функціональності системи, але процес вимагав значних зусиль. Це включало вирішення питань автентифікації, налаштування доступів та коректного використання бібліотек і сервісів Google.

Підсумовуючи, результати тестування та аналіз підтвердили високу якість і надійність системи. Проведені тестування показали, що всі компоненти системи працюють стабільно і ефективно, що дозволяє вважати продукт готовим до впровадження та використання в різних сферах.

Подальше вдосконалення продукту рекомендовано здійснювати, орієнтуючись на зворотний зв'язок від користувачів та нові технічні можливості. Впровадження користувацьких рекомендацій допоможе підвищити зручність та функціональність системи, забезпечуючи її відповідність сучасним вимогам і очікуванням користувачів. Залучення нових технологій та оновлення існуючих компонентів дозволить зберегти конкурентоспроможність продукту і розширити його можливості.

Незважаючи на певні складнощі, що виникали в процесі розробки, всі

виклики були успішно подолані. Інтеграція різних модулів, налаштування API від Google та сумісність бібліотек вимагали значних зусиль і ретельного підходу. Вдалося знайти ефективні рішення для кожної проблеми завдяки спільній роботі та постійній комунікації.

Стабільний та функціональний продукт, що був створений завдяки цим зусиллям, готовий до використання у різних сферах. Система може знайти застосування в розумних будинках, автомобільних системах, інформаційних та сервісних додатках, забезпечуючи користувачам зручний та інтуїтивний інтерфейс для взаємодії за допомогою голосових команд.

Загалом, результати тестування і вдосконалення системи свідчать про успішне досягнення поставлених цілей. Продовження вдосконалення та адаптації продукту відповідно до зворотного зв'язку користувачів і нових технічних можливостей гарантуватиме його актуальність та ефективність у майбутньому.

СПИСОК ВИКОРИСТАНИ ДЖЕРЕЛ

1. Downey A. Think Python: How to Think Like a Computer Scientist. O'Reilly Media, 2015.

2. Sweigart A. Automate the Boring Stuff with Python. No Starch Press, 2019.

3. Lutz M. Learning Python. O'Reilly Media, 2013.

4. Bird S., Klein E., Loper E. Natural Language Processing with Python. O'Reilly Media, 2009.

5. Jurafsky D., Martin J.H. Speech and Language Processing. Pearson, 2019.

6. Chollet F. Deep Learning with Python. Manning Publications, 2017.

7. McKinney W. Python for Data Analysis. O'Reilly Media, 2017.

8. VanderPlas J. Python Data Science Handbook. O'Reilly Media, 2016.

9. Raschka S., Mirjalili V. Python Machine Learning. Packt Publishing, 2019.

10. Alpaydin E. Introduction to Machine Learning. MIT Press, 2020.

11. Hinton G., Deng L., Yu D., Dahl G.E., Mohamed A., Jaitly N., ... Kingsbury B. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. IEEE Signal Processing Magazine, 2012, vol. 29, no. 6, pp. 82-97.

12. Graves A., Mohamed A.R., Hinton G. Speech Recognition with Deep Recurrent Neural Networks. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013, pp. 6645-6649.

13. Mikolov T., Karafiát M., Burget L., Černocký J., Khudanpur S. Recurrent Neural Network based Language Model. In Proceedings of the Interspeech Conference, 2010, pp. 1045-1048.

14. Cho K., van Merriënboer B., Gulcehre C., Bahdanau D., Bougares F., Schwenk H., Bengio Y. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv preprint arXiv:1406.1078, 2014.

15. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., ... Polosukhin I. Attention is All you Need. Advances in Neural Information Processing Systems, 2017, vol. 30.

16. Hannun A., Case C., Casper J., Catanzaro B., Diamos G., Elsen E., ... Ng

A.Y. Deep Speech: Scaling up end-to-end speech recognition. arXiv preprint arXiv:1412.5567, 2014.

17. Wu Y., Schuster M., Chen Z., Le Q.V., Norouzi M., Macherey W., ... Dean J. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv preprint arXiv:1609.08144, 2016.

18. Kingma D.P., Ba J. Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980, 2014.

19. Silver D., Huang A., Maddison C.J., Guez A., Sifre L., van den Driessche G., ... Hassabis D. Mastering the game of Go with deep neural networks and tree search. Nature, 2016, vol. 529, no. 7587, pp. 484-489.

20. Radford A., Narasimhan K., Salimans T., Sutskever I. Improving Language Understanding by Generative Pre-Training. OpenAI preprint, 2018.

21. Python Documentation. Офіційна документація по Python. URL: https://docs.python.org/3/ (дата звернення: 17.04.2024).

22. SpeechRecognition Library Documentation. URL: https://pypi.org/project/SpeechRecognition/ (дата звернення: 10.01.2024).

23. NLTK Documentation. URL: https://www.nltk.org/ (дата звернення: 05.02.2024).

24. TensorFlow Documentation. URL: https://www.tensorflow.org/ (дата звернення: 23.05.2024).