НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В.Коваль
(підпис) (ініціали, прізвище)
“___”_____2019 р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки
6.050101 “Комп’ютерні науки”

на тему: Розпізнавання ключових слів у потоці мовлення

Виконав: студент 4 курсу, групи ТР-52

Борозенець Михайло Романович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к.т.н. Стативка Юрій Іванович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050101 “Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту
Борозенецю Михайлу Романовичу
(прізвище, ім’я, по батькові)

1. Тема роботи “Розпізнавання ключових слів у потоці мовлення”
керівник роботи доцент, к.т.н. Стативка Юрій Іванович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”___”_____ 201__ р. № _____

2. Строк подання студентом роботи ___ _____ 201__ р.
3. Вихідні дані до роботи графічне відображення результатів розпізнавання ключових слів в аудопотоці.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати проблематику розпізнавання ключових слів у потоці мовлення, провести порівняльний аналіз сучасних програмних рішень розпізнавання ключових слів, створити та реалізувати власну систему розпізнавання ключових слів у потоці мовлення .
5. Перелік ілюстраційного матеріалу (з точним зазначенням обов’язкових креслень) 1. Постановка задачі та проблематика. 2. Огляд існуючих програмних рішень. 3. Огляд технологій та засобів програмування. 4. Опис програмної реалізації 5. Висновки .

6. Публікації: _____

Дата видачі завдання ”___”_____ 201__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі		
2.	Розробка архітектури та загальної структури системи		
3.	Розробка структур окремих підсистем		
4.	Підготовка матеріалів		
5.	Програмна реалізація системи		
6.	Захист програмного продукту		
7.	Оформлення пояснювальної записки		
8.	Передзахист		
9.	Захист		

Студент _____ Скитенко Р.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Стативка Ю.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Метою роботи була розробка та реалізація системи розпізнавання ключових слів у потоці мовлення, що дасть змогу користувачу аналізувати вхідний аудіосигнал в режимі реального часу та розпізнавати задані ключові слова. Набір ключових слів та словник може бути вказаний користувачем. Користувацький додаток представляє собою вікно, за допомогою якого користувач вводить вхідну інформацію та отримує результат.

Ключові слова: розпізнавання ключових слів, прихована марковська модель, звукові характеристики.

Записка містить 59 сторінкок, 19 рисунків та 20 посилань.

ABSTRACT

The purpose of the work was to develop and implement a keyword recognition system in the broadcast stream, which will allow the user to analyze the incoming audio signal in real time and recognize the specified keywords. A set of keywords and a dictionary can be specified by the user. The custom application is a window by which the user inputs the input information and receives the result.

Key words: key word recognition, hidden Markov model, sound characteristics.The note contains 59 pages, 19 figures, 2 and 20 references.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CMU Sphinx - відкритий пакет для роботи з системами мовлення

Node.js – платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript.

Google Assistant API – частина Google Cloud та продукт компанії Google спеціально вбудованих пристроїв

PocketSphinx – це спеціальна реалізація CMU Sphinx, з повністю відкритим кодом. Особливості системи полягають у простій інтеграції в будь-які інші системи на базі ARM процесора.

CLI – інтерфейс командної строки

ВСТУП

Нині все частіше використовується системи розпізнавання ключових слів в потоці мовлення. Дана технологія практично незамінна для створення додаткового каналу управління в різних автоматизованих системах. Проте, в даний час для української мови безкоштовних робочих рішень немає.

Хоча існує безліч алгоритмів для розпізнавання ключових слів, більшість з них засновані на прихованих марковських моделях або нейромережах. Приховані марковські моделі базуються на імовірнісному співвідношенні порядку фонем у слові. Вони добре працюють при невеликому розмірі словника, але при розмірі словника більше певної межі цей метод практично перестає працювати, Таким чином, розробка методу, здатного розпізнавати ключові слова в потоці мовлення з великим розміром словника є актуальним завданням.

Для певного набору прикладних задач можна використовувати CMUSphinx – відкритий пакет для роботи з системами мовлення. Для роботи з ним треба вибрати акустично модель. На жаль, в вільному доступі моделі для української мови немає, проте є інші, які мають схожі властивості.

Основна мета дипломної роботи полягала в дослідженні методів розпізнавання ключових слів в потоці мовлення і розробці програмного продукту для розпізнавання ключових слів українською мовою.

Для досягнення цієї мети в ході виконання дипломної роботи вирішувалися такі основні завдання:

• Дослідження існуючих методів розпізнавання ключових слів в потоці мовлення.

• Дослідження особливостей української мови, що впливають на розпізнавання ключових слів в потоці мовлення.

• Програмна реалізація пропонованих алгоритмів і демонстрація результатів поетапної обробки мовного сигналу.

Для демонстрації програмного продукту створений програмний продукт, що приймає вхідних потік звуків, та реагує на набір ключових слів, що може бути

вказаний користувачем. Програма має інтерфейс командної строки та може бути запущена разом с сервером у Докер контейнері.

Звіт з переддипломної практики містить 4 розділи.

У першому розділі описується постановка задачі та проблематика.

У другому розділі відбувається огляд та порівняння існуючих програмних рішень.

У третьому розділі вказуються основні засоби розробки даної системи розпізнавання ключових слів.

У четвертому розділі дано опис реалізованого програмного продукту, його архітектури та користувацького інтерфейсу .

1. ПОСТАНОВКА ЗАДАЧІ РОЗПІЗНАВАННЯ КЛЮЧОВИХ СЛІВ У ПОТОЦІ МОВЛЕННЯ ТА ПРОБЛЕМАТИКА

Використовуючи теоретичний матеріал та існуючі технічні засоби, дослідити та створити систему розпізнавання ключових слів у потоці мовлення, яка буде надавати можливість користувачу використовувати переваги систем дистанційного керування. Система має підтримувати українську мову та акустичні моделі інших мов, має бути простою у використанні та запускатися на мікроконтролерах та вбудованих інтернет речах.

В результаті розробки програмного продукту, користувачу буде надана можливість створити власну граматику та словник для обробки будь-яких власних ключових слів.

Для розробки інформаційної системи була використана мова програмування JavaScript. Розробка графічного інтерфейсу користувача відбувалась на основі ReactJS.

В ході роботи буде проаналізовано та досліджено принцип роботи систем CMU Sphinx, PocketSphinx, Porcupine, Google Speech та інших існуючих аналогів. Було протестовано та зроблено детальний аналіз існуючих систем розпізнавання ключових слів у потоці мовлення.

Метою розробки програми аналізу ключових слів у потоці мовлення є аналіз існуючих рішень та створення програмного продукту, що дасть змогу отримати вільне безкоштовне рішення для розпізнавання ключових слів з підтримкою української мови.

Розроблена система повинна:

1. Надавати користувачу можливість введення власного словника.
2. Надавати можливість вивести список розпізнаних слів.
3. Надавати можливість задати декілька ключових слів або словосполучень.

4. Проводити розпізнавання мови вказаного аудіо файлу та в реальному режимі часу, використовуючи адаптовану та не адаптовану моделі;

7. Надавати можливість додати власні ключові слова для будь-якого словника.

8. Надати можливість користувачу керувати налаштуваннями граматики що робить використання сервісу більш швидким та зручним.

1.1 Проблематика керування інформаційними системами

Сучасні інформаційні системи часто потребують альтернативного каналу керування. А у деяких прикладних задач класичний підхід до керування пристроями просто неможливий [1]. У таких випадках часто використовують системи голосового керування, що зазвичай використовують механізми розпізнавання ключових слів як триггер для початку розпізнавання [2]. Тоді замість звичних засобів керування користувач має змогу використовувати голосові команди, це може значно покращити досвід роботи з програмою.

Таким чином, можемо визначити, що від якості систем розпізнавання ключових слів залежить якість систем дистанційного керування голосом.

1.2. Процес розпізнавання ключових слів

Визначення ключових слів у потоці мовлення в контексті комп’ютерних наук означає, що якщо існуюча програмована система аналізує фонові звуки, то коли користувач вимовляє одне або декілька однозначно заданих слів, програма виконує певну дію.

Отже, можемо умовно задати системі два стани. Стан очікування та стан дії або робочій стан [2].

Класичний процес переходу системи визначення ключових слів із сплячого режиму в робочий режим проходить достатньо інтуїтивно. Спершу пристрій вмикається, завантажує ресурси і переходить у сплячий режим. Коли користувач

вимовляє ключові слова, вони виявляються аналізатором, і пристрій переходить у робочий режим та виконує дії або очікує на подальшу вимогу від користувача.

Тобто це створює базис для керування пристроєм повністю тільки за допомогою голосу, без необхідності рук. Також ця схема може заощадити багато енергії або комп'ютерних ресурсів в системах роботи з мовленням, оскільки пристрій не працює в режимі постійної роботи [3].

Зазвичай розпізнавання слів проходить наступним чином: звукова хвиля розбивається по ділянках тиші. Залежно від системи і використовуваних даних встановлюється певний мінімум гучності, наприклад, 10 Дб, і час підтримки цього рівня, наприклад, 100 мс, або 10 фреймів [4]. Інколи також може розглядатися зниження рівня гучності на певну кількість Дб. Після цього робляться спроби зіставлення можливих комбінацій слів з аудіо рядом, та вибирається найкраща комбінація.

Далі сигнал розбивається на фрейми (зазвичай, по 10 мс), і для кожного фрейма обчислюється вектор ознак [5]. Пошук найбільш ефективного способу обчислення цих чисел все ще є предметом вивчення, проте в загальних випадках обчислюється похідна від спектра. Далі, отримані вектори розглядаються в рамках певних моделей для підбору найбільш підходящої комбінації слів. Моделлю розпізнавання патернів в мові зазвичай є або прихована марковська модель [5] або нейронна мережа [7]. Таким чином, система отримує найбільш вірогідну комбінацію, що однозначно виділяє слово. У кожен момент часу підтримуються поточні кращі комбінації, які з плином часу розширюються за рахунок пошуку комбінацій для наступного фрейму.

1.3 Критерії роботи систем розпізнавання ключових слів

Для визначення якості системи розпізнавання ключових у потоці мовлення треба використовувати певні критерії. Визначимо основні критерії оцінки ключових слів.

1) Частота відкликання [1], що означає відношення кількості спрацювань програми віл загальної кількості сказаних заданих ключових слів. Більш високий рівень відкликання означає кращу продуктивність.

2) Частота помилкової тривоги [2], що означає ймовірність того, що ключове слово буде помічено, але воно не було промовлене. Більш низька кількість частота помилкової тривоги означає кращу продуктивність.

3) Фактор реального часу [2], що означає швидкість реакції пристрою. Визначення ключових слів вимагає швидкої реакції.

4) Енергоспоживання – актуально для мікроконтролерів, більшість з яких живляться за допомогою акумулятора або від батареї. Низьке енергоспоживання гарантує тривалу витримку батареї.

1.4 Структура мовлення

У сучасній практиці структуру мовлення розуміють як безперервний аудіопотік, у якому деякі, відносно стабільні стани, чергуються з динамічними, менш стабільними станами [9].

Це створює послідовність станів, у якій можна визначити більш-менш подібні класи звуків. Можна хибно припустити, що слова завжди створюються визначених класів, але це не так [4]. Справа у тому, що акустичні властивості сигналу, що відповідають класам, можуть сильно відрізнятися, в залежності від багатьох факторів (контексту сигналу, стилю мовлення тощо). Тому, у більшості випадків переходи між фонемами більш інформативні, ніж стабільні частини мовлення. Саме тому розробники часто виділяють дифони, тобто аудіо сигнал між двома фонемами.

Інколи фонеми розглядаються в певному контексті. Такі фонеми називають трифони та квінфони. Розгляд їх має місце бо, звучання конкретної фонеми може відрізнятися в залежності від оточуючих її фонем. На відміну від діфонів, трифонам відповідають ті ж відрізки звукової хвилі, що і фонемам [5] (рисунок 1.1)

З фонем часто формують підслова, тобто мовні склади. Іноді їх описують як стійкі до редукції, бо при прискоренні мови фонеми можуть змінюватися, а склади залишаються незмінними.

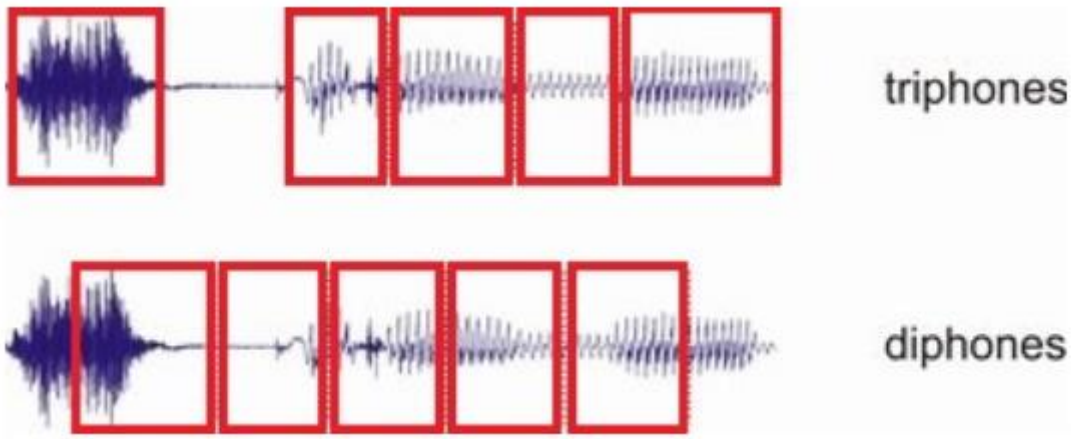

Рисунок 1.1 - відрізки аудіосигналу, що відповідають до трифонам та дифонам

Різні комбінації підслів формують слова. Підслова та нелінгвістичні звуки, такі як зітхання, кашель, формують набір підслів.

Підслова важливі для розпізнавання мовлення, оскільки вони значно обмежують комбінації фонем. Так, наприклад, якщо ми маємо для розпізнавання 40 фонем і слово, в середньому, має 7 фонем, то існує 40^7 варіантів. На щастя, люди дуже рідко використовують на практиці більше 20 тисяч слів. Це робить цей спосіб розпізнання більш точним [6].

1.5 Висновки до розділу

У даному розділі було проведено огляд проблематики та мотивації створення систем голосового керування у контексті розпізнавання ключових слів у потоці мовлення.

2. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

При розробці будь-якого програмного продукту важливим чинником є швидкість розробки та якість програмного продукту. Саме тому розробники часто використовують вже існуючі програмні пакети, модулі або API від експертів відповідної галузі. Розпізнавання ключових слів у потоці мовлення не стало виключенням.

Розглянемо деякі з найвідоміших програмних засобів та виконаємо порівняння за критеріями якості системи.

2.1 CMU Sphinx

CMU Sphinx - це комплекс програмних утиліт для розпізнавання мови, що базується на прихованих марковських моделях [3]. Sphinx абсолютно стійкий до тривалого мовного сигналу. За багато років його існування для нього було створено багато повних акустичних моделей та величезну кількість словників, що містять слова, поділені на фонеми для більш ніж двадцяти найпопулярніших мов світу. Нажаль, для української мови вільної акустичної моделі та словника немає. Sphinx написаний повністю на мові програмування Java. Компанія Sun Microsystems внесла великий внесок у розвиток Sphinx і допомогу в програмної експертизі проекту [3].

Центральна задача проекту представляє собою створення універсальної системи автоматичного розпізнавання мовлення, тобто точне перетворення голосового сигналу до текстового вигляду. Після обробки сигналу, Sphinx здатний передавати подію для використання у власних програмних системах.

Сам процес розпізнавання проходить наступнім чином: аудіо сигнал розбивається на фрейми (зазвичай, по 10 мс), і для кожного фрейму обчислюється вектор спеціальний ознак. В CMU Sphinx даний вектор складається з 39 параметрів [3], зміст яких залежить від методу вилучення ознак. Зазвичай, перші 13 ознак - кепстр сигналу, другі - перша похідна від кепстру, треті - друга похідна. Як вже був

сказано раніше, комбінація параметрів та найбільш ефективного способу обчислення цих чисел все ще є предметом вивчення, тому часто дорабляється розробниками Sphinx, проте в загальних випадках обчислюється похідна від спектра.

Далі, отримані вектори розглядаються в рамках певних моделей – мовної та акустичної, а також фонетичного словника. З яких формується підбору найбільш підходящих комбінації слів або слова. Моделлю для розпізнавання патернів в мові є приховані марковські моделі [11] (рисунок 2.1).

```
115  void
116  hmm_dump(hmm_t * hmm,
117           FILE * fp)
118  {
119      int32 i;
120
121      if (hmm_is_mpx(hmm)) {
122          fprintf(fp, "MPX   ");
123          for (i = 0; i < hmm_n_emit_state(hmm); i++)
124              fprintf(fp, " %11d", hmm_senid(hmm, i));
125          fprintf(fp, " ( ");
126          for (i = 0; i < hmm_n_emit_state(hmm); i++)
127              fprintf(fp, "%d ", hmm_ssid(hmm, i));
128          fprintf(fp, ")\n");
129      }
130      else {
131          fprintf(fp, "SSID  ");
132          for (i = 0; i < hmm_n_emit_state(hmm); i++)
133              fprintf(fp, " %11d", hmm_senid(hmm, i));
134          fprintf(fp, " (%d)\n", hmm_ssid(hmm, 0));
135      }
136
```

Рисунок 2.1 – частина функції обробки станів для CNU Sphinx для реалізації алгоритму прихованих марковських мереж

Отже, в даному випадку, обраною комбінацією є та, яка є найбільш вірогідною за статичної моделлю.

2.2 PocketSphinx

PocketSphinx – це спеціальна реалізація CMU Sphinx, з повністю відкритим кодом [3]. Особливості системи полягають у простій інтеграції в будь-які інші системи на базі ARM процесора. PocketSphinx постійно активно розвивається і робить власний внесок в розвиток систем голосового керування, розпізнавання ключових

слів у потоці мовлення тощо. Частина коду та API працює на Python та вимагає BaseSphinx як ключову залежність. Має пакет для розпізнавання ключових слів.

PocketSphinx можна використовувати як Python модуль для створення власних застосунків (рисунок 2.2). У такому випадку нам знадобиться задати у конфігураціях модуля шлях до акустичної моделі та словника. PocketSphinx має власні особливості, серед яких достатньо висока точність, високий рівень енергозбереження та простота використання, робить його цікавим інструментом для створення власних застосунків. У тому числі з використанням систем розпізнавання ключових слів у потоці мовлення.

```
#!/usr/bin/env python
from os import environ, path

from pocketsphinx.pocketsphinx import *
from sphinxbase.sphinxbase import *

MODELDIR = "pocketsphinx/model"
DATADIR = "pocketsphinx/test/data"

# Create a decoder with certain model
config = Decoder.default_config()
config.set_string('-hmm', path.join(MODELDIR, 'en-us/en-us'))
config.set_string('-lm', path.join(MODELDIR, 'en-us/en-us.lm.bin'))
config.set_string('-dict', path.join(MODELDIR, 'en-us/cmudict-en-us.dict'))
decoder = Decoder(config)

# Decode streaming data.
decoder = Decoder(config)
decoder.start_utt()
stream = open(path.join(DATADIR, 'goforward.raw'), 'rb')
while True:
  buf = stream.read(1024)
  if buf:
    decoder.process_raw(buf, False, False)
  else:
    break
decoder.end_utt()
print ('Best hypothesis segments: ', [seg.word for seg in decoder.seg()])
```

Рисунок 2.2 – приклад використання PocketSphinx для створення власних застосунків

PocketSphinx також має власні CLI програми, які можна використовувати як для автоматизації простих задач, так и для створення власних застосунків. Гнучкі конігурації дозволяють інтегрувати ці програми будь де [3].

Цікавим фактом є можливість компіляції під WebAssembly (рисунок 2.3). У такому випадку PocketSphinx можна запустити у браузері та створювати системи голосового керування на власному веб сайті.

```
<script type="text/javascript">
  var Module = {
      locateFile: function() {return "/path/to/pocketsphinx.wasm";},
      onRuntimeInitialized: function() {
      // Now I can start using it
      ...
      }
    };
</script>
<script src="path/to/pocketsphinx.js"></script>
```

Рисунок 2.3 – використання PocketSphinx у браузері за допомогою WebAssembly

Алгоритм роботи PocketSphinx аналогічний до повної версії, проте з деякими спрощенням для комфортної роботи на інтегрованих пристроях [3]. Розробники вказують, що програма успішно використовується на пристроях IPhone, Nokia Maemon і Windows Mobile.

2.3 Google Assistant API

Google Assistant API – частина Google Cloud та продукт компанії Google спеціально вбудованих пристроїв [17]. Він дає змогу розробникам створювати голосових помічників, конвертувати аудіо в текст, розпізнавати ключові слова, роботи системи голосового управління. API розпізнає 120 мов та підтримує велику кількість користувачів одночасно. Технологія може інтегрована в мобільні телефони, комп'ютери, та підтримують введення інформації за допомогою голосу, та будь які інші пристрої, навіть без підключення до інтернету. Використовуючи сервіси Google, можливо визначити, якою мовою говорить користувач, та використовувати отримані дані для додаткової обробки. Нажаль, продукт повністю закритий, та щоб з ним

працювати треба створити Google Cloud аккаунт (та робити інтеграції їх API до своїх власних сервісів).

Наразі немає відкритої повної інформації про те, як працюють алгоритми Google. Можемо лише однозначно сказати, що для передачі звуку відкривається канал передачі потоку звукового сигналу на сервера Google. Вони приймають вхідний потік та розбивають його на фрейми. Далі створюються спектрограма голосу, що далі розбивається і відправляється на 8 різних потужних серверів [3], що являють собою частину екосистеми Google Cloud (рисунок 2.4). Далі, проходить процес схожий на той, що працює у CNU Sphinx, проте замість марковських моделей алгоритм використовує складні нейронні мережі.

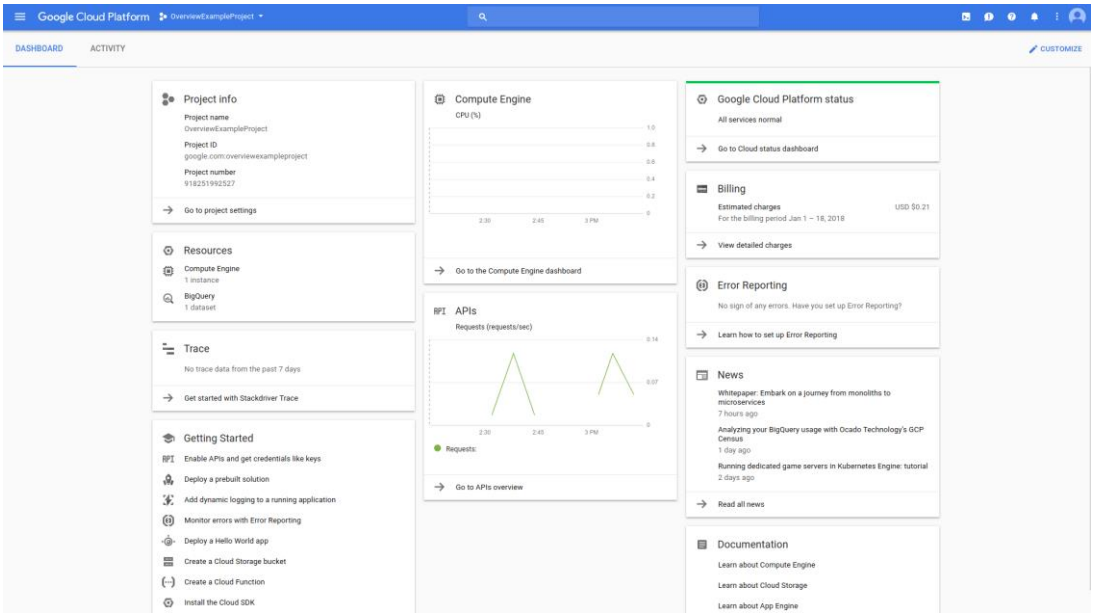

Рисунок 2.4 – Вигляд Google Cloud аккаунта

Після обробки, користувач API отримує данні про мову та може використовувати далі для створення власного програмного забезпечення. Так, наприклад, можна створити Node.js додаток (рисунок 2.5), що буде використовувати Google Assistance API для будь-яких систем голосового керування, розпізнавання ключових слів, тощо.

```
const GoogleAssistant = require('./googleassistant');
const deviceCredentials = require('./devicecredentials.json');

const CREDENTIALS = {
  client_id: deviceCredentials.client_id,
  client_secret: deviceCredentials.client_secret,
  refresh_token: deviceCredentials.refresh_token,
  type: "authorized_user"
};

const assistant = new GoogleAssistant(CREDENTIALS);
```

Рисунок 2.5 – приклад коду для інтеграції використання Google Assistans у власних додатках

2.4 Picovoice Purcupine

Picovoice Purcupine - відкрита система розпізнання ключових слів у потоці мовлення з подвійною ліцензією [16]. Вона дозволяє розробникам створювати власні додатки з використанням власного API (рисунок 2.6). На відміну від попередніх рішень, Purcupine має лише модуль розпізнавання ключових слів у потоці мовлення, та не має модуля для аналізу мовлення, в цілому.

```
func getNextAudioFrame() -> UnsafeMutablePointer<Int16> {
    //
}

while true {
    let pcm = getNextAudioFrame()
    var keyword_index: Int32 = -1

    let status = pv_porcupine_multiple_keywords_process(handle, pcm, &keyword_index)
    if status != PV_STATUS_SUCCESS {
        // error handling logic
    }
    if keyword_index >= 0 {
        // detection event logic/callback
    }
}
```

Рисунок 2.6 – приклад коду для інтеграції використання Picovoice Porcupine у власних додатках

Purcupine використовує глибокі нейронні мережі [16] та набір оптимізованих алгоритмів для розпізнавання ключових слів у потоці мовлення. Продукт компактний і обчислювально-ефективний, повністю придатний для мобільних пристроїв. Програма на базі Purcupine може працювати всього з 20 Кб оперативної пам’яті та мінімальним процесом.

Ядро системи використовує ANSI C, а тому пакет повністю кросплатформений. В даний час підтримує Raspberry Pi, Beagle Bone, Android, iOS, watchOS, Linux, Mac, Windows і веб-браузери з WebAssembly (рисунок 2.7). У випадку комерційної ліцензії підтримує такоє інші процесори ARM Cortex-A і ARM Cortex-M (M4 і M7) і DSP.

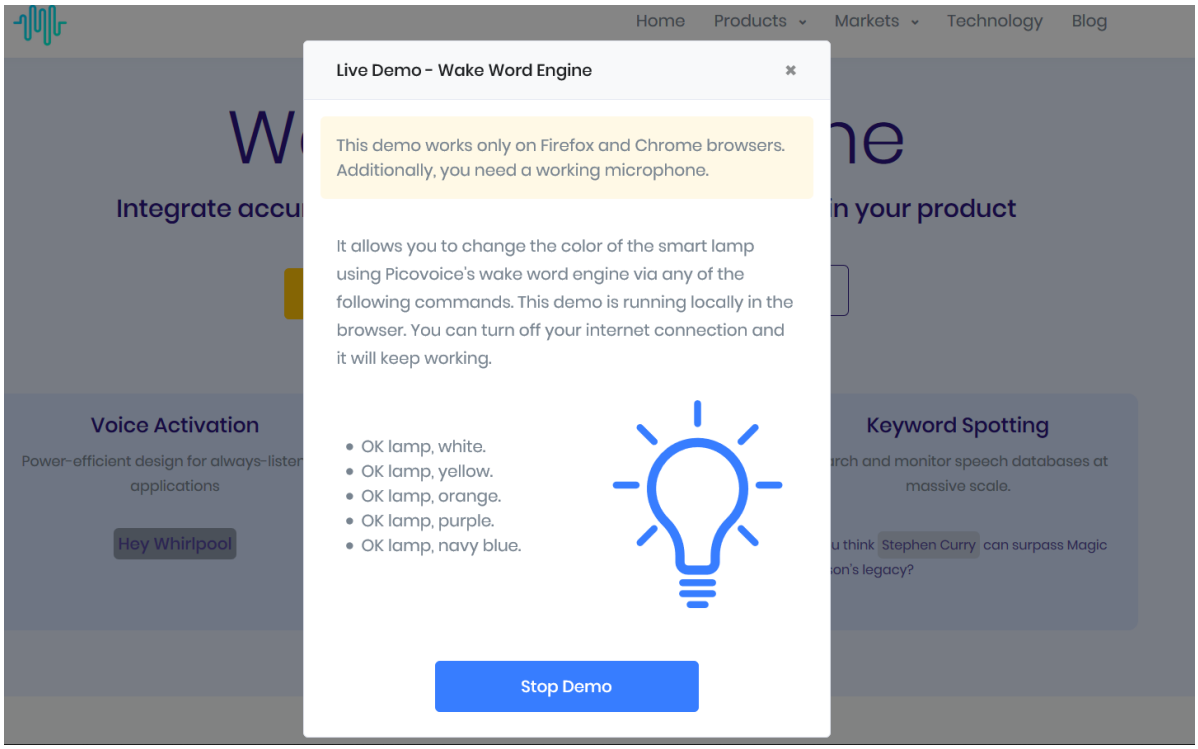

Рисунок 2.7 – приклад використання Porcupine у веб браузері для розпізнавання ключових слів

Porcupine може виявляти декілька голосових команд одночасно, не використовуючи більше CPU та оперативної пам'яті. Розробники мають змогу вибирати з набору попередньо визначених фраз на різних платформах і використовувати їх безкоштовно. Крім того, розробники можуть створювати власні

фрази пробудження (нажаль з певними обмеженнями і лише на Linux, Mac або Windows).

2.5 Порівняння існуючих рішень

Зрозуміло, що кожне існуюче рішення має власні особливості. А для того, щоб виконати більш-менш об’єктивне порівняння, зробимо власне дослідження на основі заданих критеріїв [19].

1. Частота відкликання системи [20]. Пам’ятаймо, що цей параметр означає відношення кількості спрацювань програми віл загальної кількості сказаних заданих ключових слів. Виконаємо порівняння основане на 50 тестових промовлень ключових слів у тихому приміщенні для кожної системи. Отримаємо дані (рисунок 3.8), що демонструють високих показних частоти викликання системи для кожної системи

Рисунок 3.8 – порівняння частоти відкликання системи

2. Частота помилкової тривоги. На противагу від частоти відкликання системи, цей параметр означає кількість помилкових спрацювань за одиницю часу. Отже, залишимо наші системи розпізнавання ключових слів на 6 годин та отримаємо результати (рисунок 3.9)

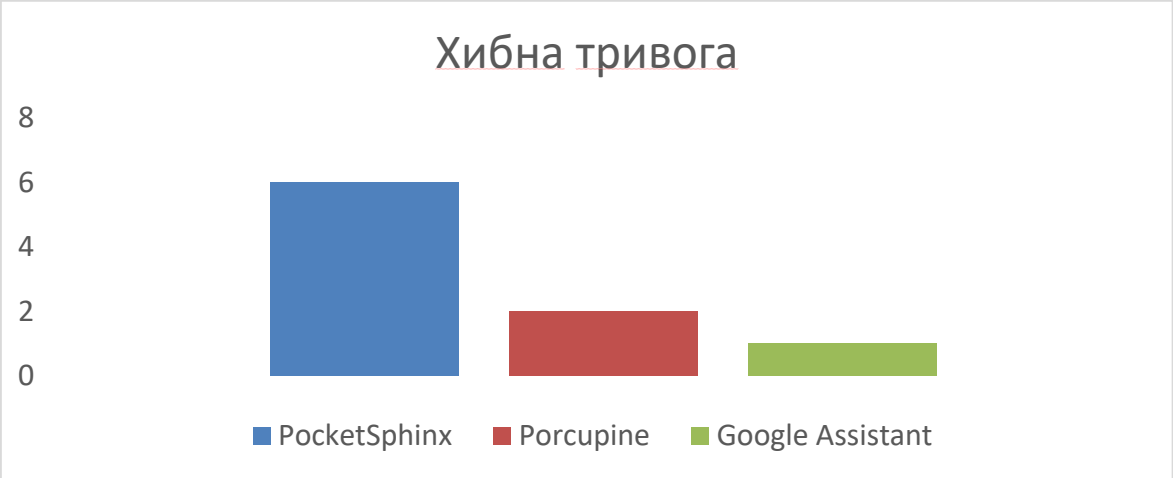

Рисунок 3.9 – порівняння хибної тривоги за 6 годин

3. Енергоефективність. Для спрощення замірів, будемо вважати, що енергоефективність в першу чергу залежить від навантаження на центральний процесор. Зробимо порівняння навантаження систем розпізнавання ключових слів (рисунок 3.10).

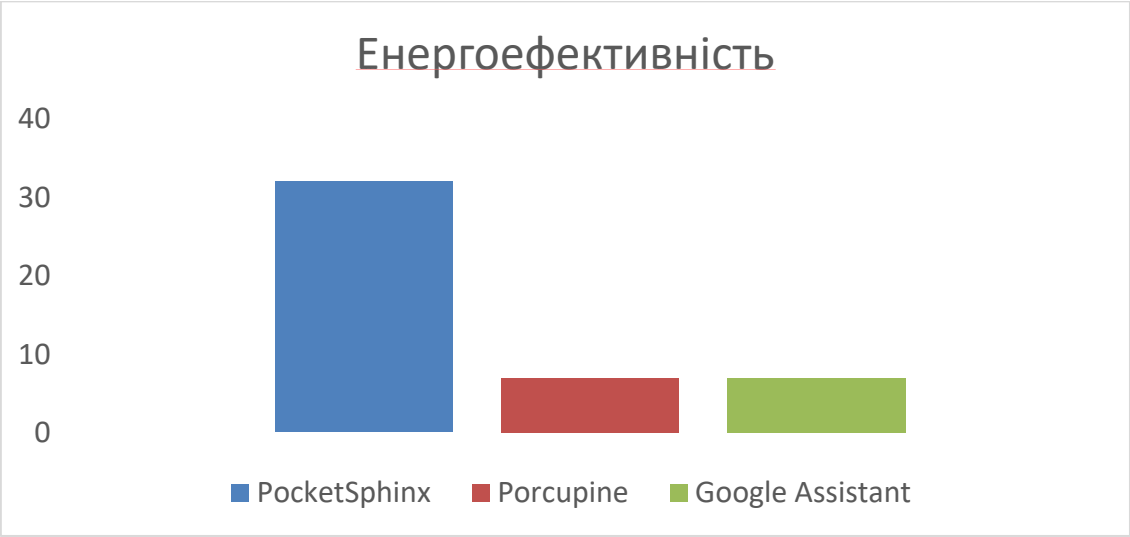

Рисунок 3.10 – порівняння хибної тривоги за 6 годин

У якості системи для порівняння будемо використовувати Raspberry Pi 3.

4. Зрозуміло, що окрім базових критеріїв якості системи розпізнавання ключових слів у потоці мовлення існують багато інших, менш пов’язаних з

предметною областю. Так, наприклад з трьох рішень лише PocketSphinx має повністю вільну ліцензію та відкритий до модифікацій, лише Google Assistant може працювати з більш ніж 120 мовами одночасно, а лише Porcupine має підтримку старих мобільних процесорів.

2.6 Висновки до розділу

У даному розділі було проведено проведено аналіз та порівняння інсуючих систем розпізнавання ключових слів у потоці мовлення.

Нажаль, наразі ідеального рішення наразі немає. Для даної роботи найбільш цікавим рішенням для мене є PocketSphinx, тому було вирішено створювати власний програмний продукт на його основі.

3. ОГЛЯД ТЕХНОЛОГІЙ ПРОГРАМУВАННЯ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ РОЗПІЗНАВАННЯ КЛЮЧОВИХ СЛІВ У ПОТОЦІ МОВЛЕННЯ

Незалежно від предмету розробки, важливим чинником, під час розробки програмного продукту, є вибір засобів програмної реалізації та технологій. Середовищем розробки інформаційної системи було обрано Visual Studio Code. Для створення графічного інтерфейсу системи використовувався інтерфейс програмування додатків React JavaScript.

Для обміну сигналами з клієнтом було використано Node.js та WebSocket протокол.

3.1 Visual Studio Code

Visual Studio Code – редактор коду для створення, редагування та підтримки сучасних застосунків і програм для хмарних систем [18]. Visual Studio Code має вільну ліцензію та працює з Windows, macOS і Linux.

Він постачається з вбудованою підтримкою Javascript, TypeScript, NodeJs (автозаповнення, перевірка синтаксису та рефакторинг). За визнеченням Github комюніті є одним з найкращих редакторів коду. Він має зручний інтерфейс (рисунок 4.1) та екосистему плагінів для підтримки інших мов (C, C ++, C #, Python та ін). Користувач має змогу додатково встановити розкладки клавіатур з текстових редакторів, таких як термільний atom або vim.

Інструмент повністю кросплатформений, тобто підтримує більшість популярних платформ та навіть запуск у веб браузері [18].

Продукт розробляє компанія Microsoft, як крос платформну альтернативу платному Vistual Studio Середовище розробки стало першим кросплатформений продуктом у лінійці Visual Studio.

За основу для Visual Studio Code використовуються напрацювання вільного проекту Atom, що розвивається компанією GitHub. Зокрема, Visual Studio Code є надбудовою над Atom Shell, що використовують браузерний рушій Chromium і Node.js. Примітно, що про використання напрацювань вільного проекту Atom на сайті Visual Studio Code і в прес-релізі і в офіційному блозі не згадується.[3]

Редактор містить вбудований дебагер, інструменти для роботи з Git і засоби рефакторингу, навігації по коду, автодоповнення типових конструкцій і контекстної підказки. Продукт підтримує розробку для платформ ASP.NET і Node.js, і позиціонується як легковаге рішення, що дозволяє обійтися без повного інтегрованого середовища розробки. Серед підтримуваних мов і технологій: JavaScript, C++, C#, TypeScript, jade, PHP, Python, XML, Batch, F#, DockerFile, Coffee Script, Java, HandleBars, R, Objective-C, PowerShell, Luna, Visual Basic, Markdown, JSON, HTML, CSS, LESS і SASS, Haxe.

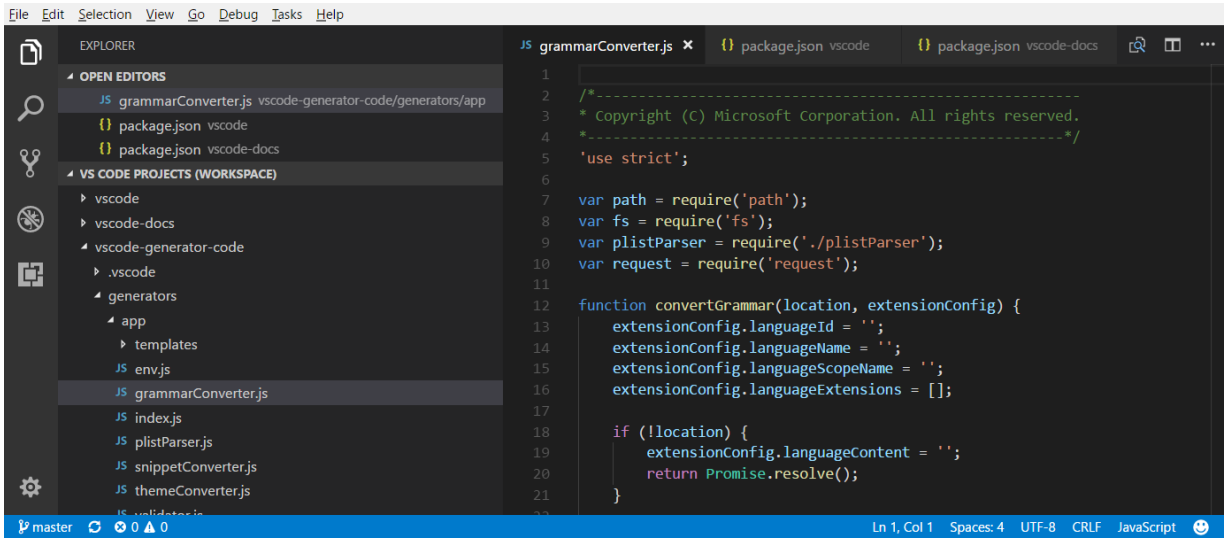

Рисунок 3.1 – інтерфейс Visual Studio Code

3.2 Бібліотека React

React — відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з

якими стикаються в розробці односторінкових застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників [14].

Відповідно до офіційної документації, React робить зручним створення багаторазово використовуваних компонентів інтерфейсу, що містять динамічна дані, які можуть змінюватись з часом. Має власних унікальний синтаксис (рисунок 3.2)

```
function HelloMessage({ name }) {
  return <div>Hello {name}</div>;
}

ReactDOM.render(
  <HelloMessage name="Taylor" />,
  document.getElementById('container')
);
```

Рисунок 3.2 – приклад коду з використанням React

Бібліотека надає можливість використовувати MVC паттерн та працювати абсолютно абстрактно від DOM. Замість цього, вона надає власну більш просту модель програмування та забезпечує кращу продуктивність, у більшості випадків. За допомогою такого рівня абстракції React можливо рендерити на сервері, використовуючи Node.js. Або навіть створювати мобільні додатки за допомогою React Native [15] (рисунок 3.3).

Бібліотека реалізує одностороннє зв’язування даних, що знижує необхідність працювати зі збереженням комплесних даних та робить код більш інтуїтивним, ніж у традиційних бібліотеках.

Як і будь-яка інша бібліотек React має власні особливості. Серед яких JSX, що представляє собою розширення синтаксису JavaScript. Його не обов’язково використовувати, проте при розробці за допомогою React це рекомендується розробниками бібліотеки та значно покращує процес розробки.

Компоненти React - це програмні одиниці веб застосунку. Розбиваючи весь проект на унікальні компоненти, можна значно полегшити розробку та підтримку складних довгострокових проектів. Їх можливо використовувати як шаблони

відображення даних, що значно покращує читабельність та допомагає підтримувати великі програми.

```
import React, {Component} from 'react';
import {Text, View} from 'react-native';

class HelloReactNative extends Component {
  render() {
    return (
      <View>
        <Text>
          If you like React, you'll also like React Native.
        </Text>
        <Text>
          Instead of 'div' and 'span', you'll use native components
          like 'View' and 'Text'.
        </Text>
      </View>
    );
  }
}
```

Рисунок 3.3 – приклад React Native програми

Ключовими недоліками технології є:

1. Бібліотека реалізує лише інструменти для роботи з відображенням даних. Тому для всього іншого все одно потрібно вибрати інші технології, щоб отримати повний набір інструментів для розробки.
2. Вбудовані шаблони JSX можуть інколи реалізовувати логіку програмного застосунку, що може значно погіршити читабельність та підтримку коду

Отже, React дозволяє розробникам створювати комплексні веб-застосунки, які використовують величезну кількість даних, що можуть змінюватись з часом, без жодного перезавантаження сторінки. На його основі можливо створити додатки, що будуть швидкі та масштабовані. React має інструменти лише для створення інтерфейсу у веб застосунках та може бути використаний у поєднанні з іншими JavaScript бібліотеками або фреймворками.

Тобто, React лише відповідає частині відображення у шаблоні модель-вид-контролер (MVC).

3.3 Платформа Node.js

Node.js — платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript [12]. Якщо раніше JavaScript застосовувався для обробки даних в браузері на сторонні користувача, то Node.js надав можливість виконувати JavaScript скрипти на сервері (рисунок 3.4) та відправляти користувачеві результат їх виконання. Платформа Node.js перетворила JavaScript на мову загального використання з великою спільнотою розробників.

```
// App initialization
const dotenv = require('dotenv');
dotenv.config();

require('@babel/register');

// Utils
const createApp = require('./src/createApp');
const logger = require('winston');

createApp().then(app => {
  const port = process.env.PORT || 6001;
  logger.info(`Start ${process.env.NODE_ENV} mode`);
  app.listen(port, () => logger.info(`Server running on port ${port}`));
});
```

Рисунок 3.4 – приклад ініціалізації веб серверу за допомогою Node.js

Для керування модулями використовується пакетний менеджер npm. Що надає змогу зручно встановлювати, видаляти та тестувати мільйони вільних пакетів.

Платформа Node.js призначена для виконання високопродуктивних мережевих додатків, написаних мовою програмування JavaScript. Платформа окрім роботи із серверними скриптами для веб-запитів, також використовується для створення клієнтських (рисунок 3.5) та серверних повноцінних програм.

```
const {app, BrowserWindow} = require('electron')

let mainWindow

function createWindow () {
  mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true
    }
  })

  mainWindow.loadFile('index.html')
```

Рисунок 3.5 – приклад сворення вікна клієнтського додатку на Node.js

В платформі використовується розроблений компанією Google V8 [12].

Для забезпечення обробки великої кількості паралельних запитів у Node.js використовується асинхронна модель запуску коду, заснована на обробці подій в неблокуючому режимі та визначенні обробників зворотніх викликів (callback).

Node.js базується на відкритому програмного забезпеченні. Так, наприклад, для мультиплексування з'єднань використовується бібліотека libuv, для створення пулу потоків (thread pool) задіяна бібліотека libeio, а для виконання DNS-запитів у неблокуючому режимі інтегрований c-ares. Всі системні виклики, що спричиняють блокування, виконуються всередині пулу потоків. А потім, як і обробники сигналів, передають результат своєї роботи назад через неіменовані канали (pipe).

3.5 Фреймворк Express.js

Express.js, або просто Express — програмний каркас розробки веб-застосунків для Node.js (рисунок 3.6), реалізований як вільне і відкрите програмне забезпечення під ліцензією MIT. Він спроектований для створення веб-застосунків і API[2]. Де-факто є стандартним каркасом для Node.js [12]. Автор фреймворка, TJ Holowaychuk, описує його як створений на основі написаного на мові Ruby каркаса Sinatra, маючи

на увазі, що він мінімалістичний, але має велику кількість плагінів, шо підключаються.

```
const express = require('express')
const app = express()

app.get('/', function (req, res) {
  res.send('Hello World')
})

app.listen(3000)
```

Рисунок 3.6 – приклад реалізації базового веб серверу за допомогою Express

Express є бекендом для програмного стека MEAN, разом з базою даних MongoDB і каркасом AngularJS для фронтенду.

3.6 Висновки до розділу

У даному розділі було проведено проведено детальний огляд інструментів розробки, що я використовував для створення програмного застосунку. Були показано приклади їх використання на практиці та порівняння з аналогами.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ТА ВИПРОБУВАННЯ ПРОГРАМИ

В розділі цьому описується процес створення та тестування додатку з використанням системи розпізнавання ключових слів для української мови. Нажаль, вільної безкоштовної акустичної моделі для української мови наразі немає, тому будемо використовувати російську.

4.1 Створення власного додатку на основі PocketSphinx

PocketSphinx можна використовувати як Python пакет або як CLI системну програму. Наведемо приклад використання Python пакету PocketSphinx пакету для розпізнавання ключових слів (рисунок 5.1).

У якості параметрів маємо: hmm – шлях до акустичної моделі, keypharase – ключове слово, dict – фонетичний словник.

```
pocketsphinx_dir = os.path.dirname(pocketsphinx.__file__)
model_dir = os.path.join(pocketsphinx_dir, 'model')
config = pocketsphinx.Decoder.default_config()
config.set_string('-hmm', os.path.join(model_dir, 'rus'))
config.set_string('-keyphrase', 'привіт')
config.set_string('-dict', os.path.join(model_dir, dict_name))
config.set_float('-kws_threshold', self.threshold)
p = pyaudio.PyAudio()
stream = p.open(format=pyaudio.paInt16, channels=1, rate=16000,
input=True, frames_per_buffer=20480)
stream.start_stream()
buf = stream.read(1024)
if buf:
    decoder.process_raw(buff)
else:
    break;
if decoder.hyp() is not None:
    print("Hotword Detected")
    decoder.end_utt()
    start_speech_recognition()
    decoder.start_utt()
```

Рисунок 4.1 – реалізація системи розпізнавання ключових слів на базі PocketSphinx за допомогою Python

Фонетичний словник зіставляє слово словника до відповідної послідовності фонем. Це може виглядати так: привіт PP R I V I TT

Так ми зробили скрипт, що здатний розпізнавати слово привіт. У випадку розпізнавання скрипт передає сигнал на обробник, де той може виконати будь-яку дію.

Так ми отримали готове рішення для розпізнавання ключових слів у потоці мовлення для української мови. Проте, цей підхід має декілька недоліків:

1. Ми можемо використовувати лише Python у якості мови програмування
2. Pip версія PocketSphinx не має останнії оновлень продукту

Отже, спробуємо зробити власне більш універсальне рішення. Для цього, спочатку треба скомпілювати SphinxBase та PocketSphinx під відповідну платформу. Так, отримаємо вихідний код з офіційного репозіторію PocketSphinx за допомогою git та відповідної команди клонування (рисунок 4.2)

```
git clone https://github.com/cmusphinx/pocketsphinx.git
```

Рисунок 4.2 – отримання коду PocketSphinx за домомогю git

Далі запустимо процес з установки з офіційних bash скриптів (рисунок 4.3). Для установки нам знадобиться встановлений C компілятор та make.

```
$ ./configure
$ make clean all
$ make check
$ sudo make install
```

Рисунок 4.3 – команди компіляції PocketSphinx

Тепер можна використовувати засоби CLI PocketSphinx такі як pocketsphinx_continius.

Спробуємо у CLI режимі запустити запустити команду для розпізнавання ключових слів у потоці мовлення. Для цього створими словник та додамо туди слово “відкрити”, що має фонетичну форму vv i d k r yy t yy. Далі, вже можемо виконати команду для початку розпізнавання (рисунок 4.4). Де hmm - шлях до акустичної моделі, dict - шлях до словника, a keypase - задане ключове слово.

```
pocketsphinx_continuous -hmm zero_ru.cd_cont_4000 -inmic
yes -keyphrase відкрити -dict wordlist.voc
```

Рисунок 4.4 — команда запуску pocketsphinx_continius для розпізнавання ключових слів у потоці мовлення.

Тепер, після запуска команди початку аналізу можемо бачити логи програми, та відповідні розпізнані слова (рисунок 4.5).

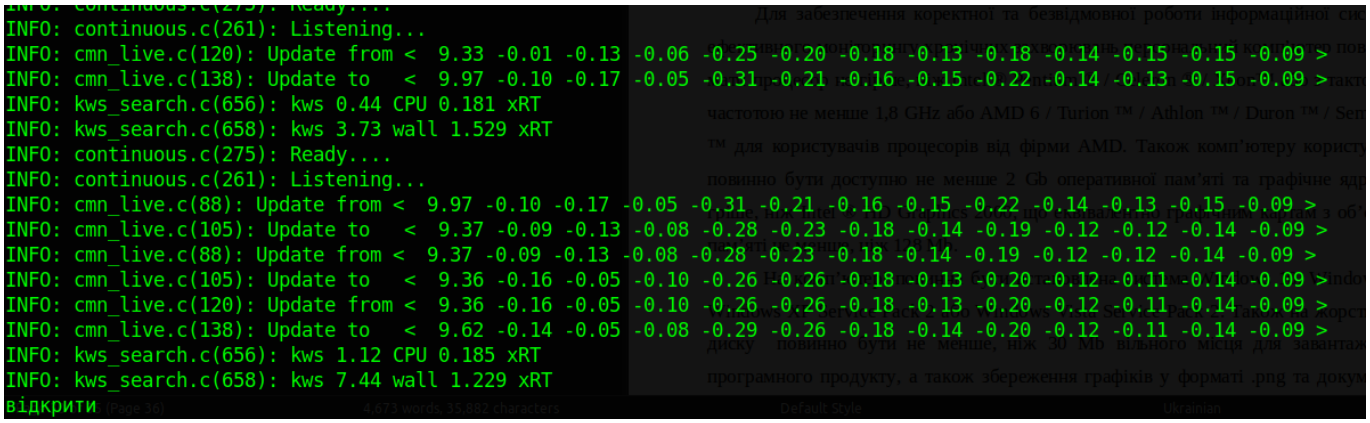

Рисунок 4.5 — Логи pokcetsphinx_continius

Цікаво, що системні логи надходять до терміналу з unix stderr потоку, а ключові слова надходять до stdout. Це надає нам змогу легко відкокремлювати логи від ключових слів в момент їх розпізнавання, а отже використовувати момент розпізнання у власних додатках.

Так для прикладу створемо Node.js сервер, що буде запусати pocketsphinx_continius у системному терміналі та обробляти його системний вивід (рисунок 4.6).

Тепер, коли ми можемо обробляти дані с PocketSphinx на веб сервері, було би зручно вивести дані на у веб браузер. Для цього створимо новий проект за допомогою React генератора, та будемо обмінюватись даними за допомогою веб сокетів, що створюють стабільне двостороннє підключення клієнта та серверу. У випадку розпізнавання відправляти подію sphinxData на клієнт (рисунок 4.6) та виводити розпізнане ключове слово клієнту.

Цікаво, що клієнт можна запустили на будь-якому іншому пристрої та відслідковувати розпізнанні слова.

```
module.exports.run = async (req, res) => {
obj = new SphinxService(
"pocketsphinx_continuous -hmm zero_ru.cd_cont_4000
-jsgf grammar.jsgf -dict wordlist.voc -inmic yes"
);
obj.run({
    onData: async data => {
    console.log(data);
    await emitEvent("sphinxData", data);
},
    onError: async error => {
    console.log(error);
    await emitEvent("sphinxError", error);
},
onComplete: async () => {
    await emitEvent("sphinxComplete", { success: true });
}
});
return res.json({ success: true });
};
```

Рисунок 4.6 – обробка подій на сервері

Таким чином, ми створили майже універсальне рішення на базі PocketSphinx, що має достатньо високу продуктивність та може використовуватись на практиці.

4.2 Робота з програмним продуктом

Інтерфейс додатку побудований веб засобами (рисунок 4.7):

Перед початком роботи користувач має задати граматику та набір фонем. Після цього запустити сервер, що здатний запускати систему розпізнавання ключових слів.

Правила граматики моють відповідати до JSGF - незалежному від платформи текстовому представленню граматик, для використання в розпізнаванні мовлення.

Граматики буде використовуватись розпізнавальним модулем для визначення комбінацій ключових слів.

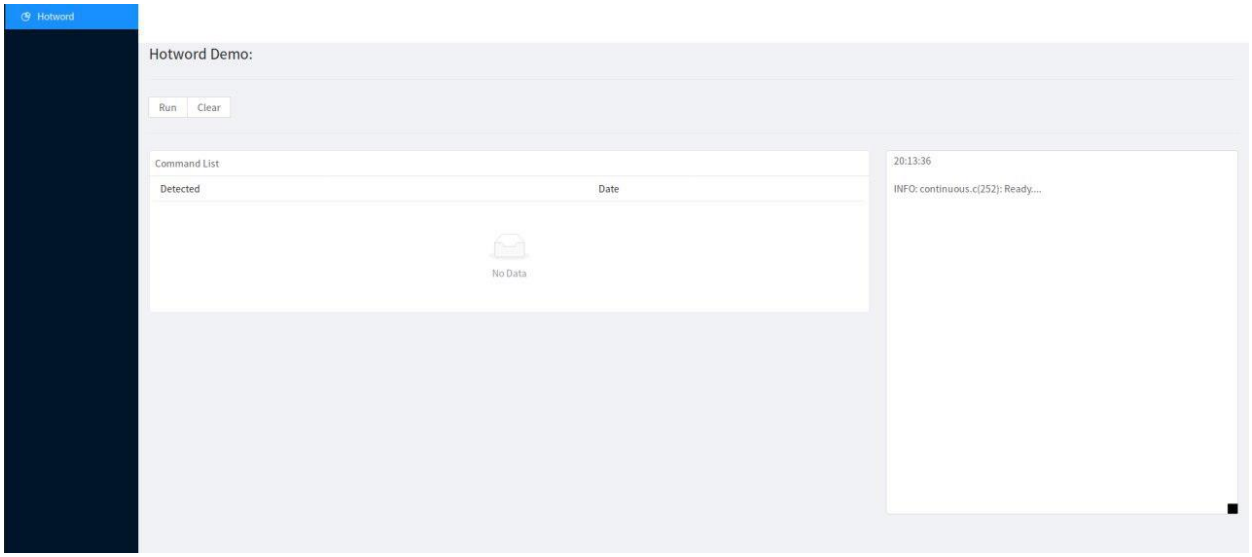

Рисунок 4.7 – Головне вікно додатку

Окрім того, програма очікую словник та набір фонем. Словник можна використовувати існуючий.

Після того, як слово було визначене, воно буде відображено у графічному інтерфейсі (рисунок 4.8).

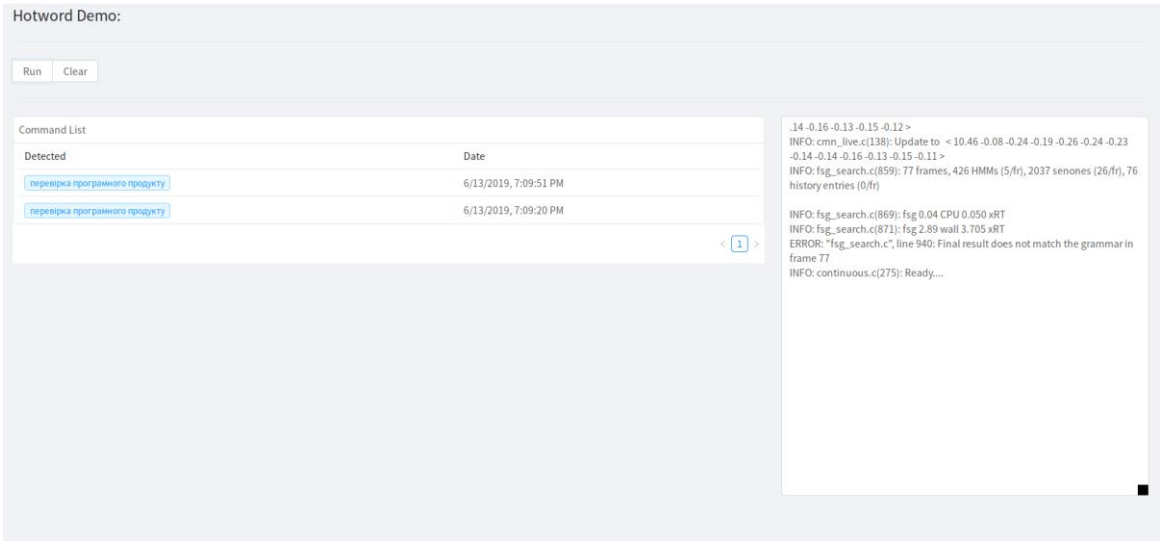

Рисунок 4.8 – ввідображення ключових слів у інтерфейсі

Тепер, користувач може задати команди для реагування на задані ключові слова у клієнті.

Зрозуміло, що серверну частину легко запустити окремо тільки у CLI режимі (рисунок 4.9), тоді не даному пристрої буде графічного інтерфейсу, проте

навантаження на центральний процесор значно зменшиться. Цей режим можна використовувати у мікроконтролерах, а керувати налаштуваннями через інтернет по веб сокет протоколу.

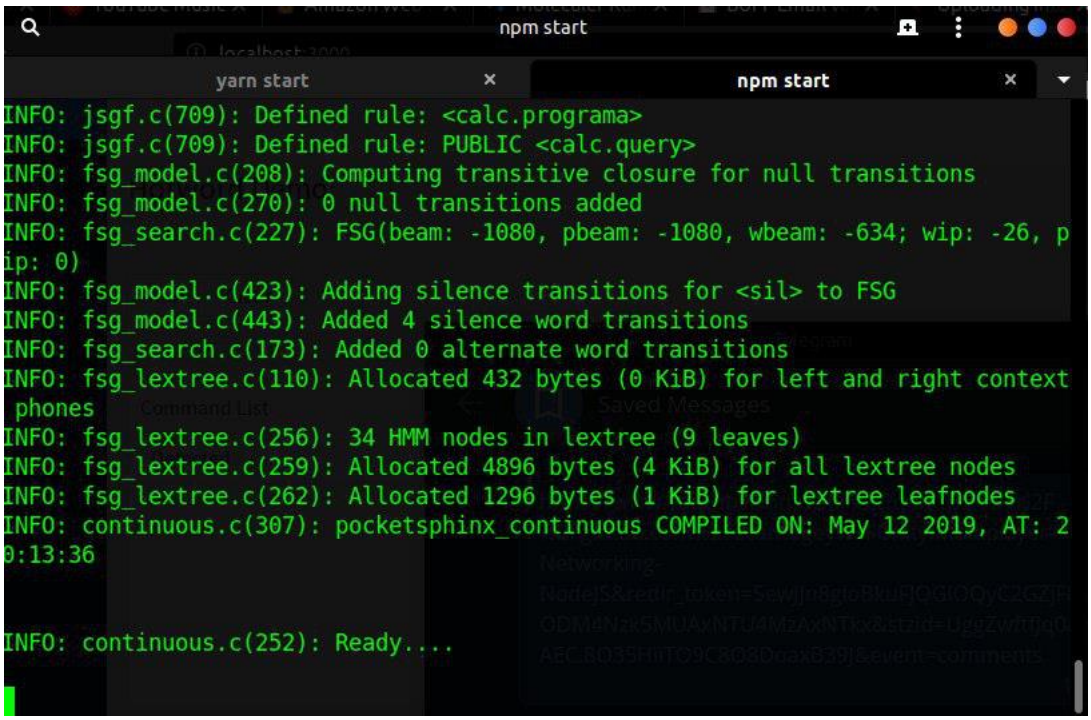

Рисунок 4.9 – CLI режим

5.3 Висновки до розділу

У даному розділі було розглянуто два способи створення додатку для розпізнавання ключових слів на базі PocketSphinx, розглянути плюси та мінуси кожного підходу.

Був описаний процес покрокового створення власного додатку для використання та оглянуто користувацький інтерфейс.

ВИСНОВКИ

В ході виконання даної роботи було розроблено програму, що має змогу розпізнавати ключові слова у потоці мовлення

Додаток було написано на мові програмування JavaScript з використанням React та Node.js.

Програма може бути використана для зручного використання систем розпізнавання ключових слів у потоці мовлення.

Був створений програмний продукт, що приймає граматику, словник, та реагує на набір ключових слів, що може бути вказаний користувачем. Окремо програма має cli інтерфейс та може бути запущена разом с сервером у Докер контейнері та графічний веб інтерфейс.

В ході роботу було проаналізовано ефективність роботи системи CMU Sphinx, та була покращена ефективність розпізнавання за рахунок адаптації акустичної моделі та створення більш компактного словника.

Для роботи з даним програмним забезпеченням необхідний лише будь-який мікроконтролер або комп’ютер.

Користувач має змогу власноруч адаптувати існуючу акустичну модель під свій голос та умови навколишнього середовища.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алимурадов, А. К. Обзор и классификация методов обработки речевых сигналов в системах распознавания речи.[Електронний ресурс] - 2015, №2. – С. 27-35. — Режим доступу: http://cyberleninka.ru/article/n/obzor-i-klassifikatsiya-metodov-obrabotkirechevyh-signalov-v-sistemah-raspoznavaniya-rechi

2. Мещеряков, Р. В. Структура систем синтеза и распознавания речи. [Електронний ресурс]– 2009. – С. 121-126. Режим доступу: http://cyberleninka.ru/article/n/struktura-sistem-sinteza-i-raspoznavaniya-rechi

3. CMU Sphinx tutorial. [Електронний ресурс] Режим доступу: https://cmusphinx.github.io/wiki/tutorialconcepts/

4. Sohn, Kim, Sung. A Statistical Model-Based Voice Activity Detection. 1999 – 56с

5. M. Y. Appiah, M. Sasikath, R. Makrickaite, M. Gusaite, Robust Voice Activity Detection and Noise Reduction Mechanism — Institute of Electronics Systems, Aalborg University, 2015. — 81 с.

6. Ravi Ramachandran; Richard Mammone (6 December 2012). Modern Methods ofSpeech Processing. Springer Science & Business Media. 102 p.

4. M. H. Savoji, «A robust algorithm for accurate end pointing of speech,» Speech Communication, pp. 45–60, 1989.

7. Ramirez, J.; J. M. Gorriz; J. C. Segura Voice Activity Detection. Fundamentals and Speech Recognition System Robustness. In M. Grimm and K. Kroschel (ed.). Robust Speech Recognition and Understanding, 2007. — 22 с.

8. Alan V. Oppenheim, Ronald W. Schafer. From Frequency to Quefrency: A History ofthe Cepstrum. 2004

9. Shreya Narang, Ms. Divya Gupta. Speech Feature Extraction Techniques: A Review. 2015

10. Jing Dong, Dongsheng Zhou, Qiang Zhang. Robust Feature Extraction Based onTeager-Entropy and Half Power Spectrum Estimation for Speech Recognition. 2015

11. HTK Speech Recognition Toolkit. http://htk.eng.cam.ac.uk/

12. Node.js official page. [Електронний ресурс] Режим доступу: https://nodejs.org/en/

13. React.js tutorials. [Електронний ресурс] Режим доступу: https://ru.reactjs.org/

14. React Native official page. [Електронний ресурс] Режим доступу: https://facebook.github.io/react-native/

15. React Native official page. [Електронний ресурс] Режим доступу: https://facebook.github.io/react-native/

16. Picovoice web page. [Електронний ресурс] Режим доступу: https://picovoice.ai/

17. Google Assistance docs. [Електронний ресурс] Режим доступу: https://developers.google.com/assistant/sdk/overview

18. Visual Studio Code web page. [Електронний ресурс] Режим доступу: https://code.visualstudio.com/

19. Balakrishnan Narayanaswamy. Improved Text-Independent Speaker Recognitionusing Gaussian Mixture Probabilities. 2005

20. C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri. OpenFst: a general andefficient weighted finite-state transducer library. 2007

ДОДАТОК А

Система розпізнавання ключових слів у потоці мовлення

Специфікація

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5276_19Б

Аркушів 2

Київ 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5276_19Б	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5276_19Б	Index.html	Файл веб інтерфейсу програми
УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5276_19Б	index.js	Веб сервер програми

ДОДАТОК Б

Система розпізнавання ключових слів у потоці мовлення

Текст програми

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5276_19Б

Аркушів 7

Київ 2019

Клас роботи з PocketSphinx.

```
const { spawn } = require("child_process");
const kill = require("tree-kill");

class SphinxService {
  constructor(command, options) {
    this.command = command;
    this.options = options;
    this.process = null;
  }

  stop() {
    console.log(this.process);
    if (this.process) kill(this.process.pid, "SIGKILL");
  }

  run({ onData, onError, onClose, onComplete }) {
    // Clear response
    this.response = [];

    // Spawn new process
    this.process = spawn(this.command, {
      shell: true
    });

    console.log("Init process...");

    this.process.stdout.on("data", async data => {
      const dataStr = data.toString("utf-8");
      onData && (await onData(dataStr));
    });

    this.process.stderr.on("data", async err => {
      const errorStr = err.toString("utf-8");
      onError && (await onError(errorStr));
    });

    this.process.on("error", async err => {
      const errorStr = err.toString("utf-8");
      onError && (await onError(errorStr));
    });

    this.process.on("close", async code => {
```

```
      onClose && (await onClose(this.response, code));
      onComplete && (await onComplete(this.response, code));
    });
  }
}

module.exports = SphinxService;
```

Контролер Sphinx.

```
const SphinxService = require("../../services/Sphinx");
const { emitEvent } = require("../../socket");

let obj = null;

/**
 * Run pocketsphinx
 */
module.exports.run = async (req, res) => {
  obj = new SphinxService(
    "pocketsphinx_continuous -hmm zero_ru.cd_cont_4000 -jsgf
grammar.jsgf -dict wordlist.voc -inmic yes"
  );
  obj.run({
    onData: async data => {
      console.log(data);
      await emitEvent("sphinxData", data);
    },
    onError: async error => {
      console.log(error);
      await emitEvent("sphinxError", error);
    },
    onComplete: async () => {
```

```
      await emitEvent("sphinxComplete", { success: true });
    }
  });

  return res.json({ success: true });
};

/**
 * Stop pocketsphinx
 */
module.exports.stop = async (req, res) => {
  obj.stop();

  return res.json({ success: true });
};
```

ДОДАТОК В

Система виявлення голосової активності в звуковому сигналі

Опис програмного коду

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5276_19Б

Аркушів 8

Київ 2019

АНОТАЦІЯ

Даний додаток містить опис системи розпізнавання ключових слів у потоці мовлення. Створений програмний продукт може визначати ключові слова для української мови та виконує такі завдання:

– виявлення ключових слів у потоці мовлення для української мови;

– візуалізація роботи методів;

– можливість роботи з власними словниками та ;

Даний застосунок було написано на платформі JavaScript з використанням програмних пакетів PocketSphinx.

ЗМІСТ

1. ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис системи розпізнавання ключових слів у потоці мовлення. У додатку Б міститься програмний код головних частин розробленої системи.

Для роботи з розробленим додатком необхідно мати встановлений виконуваний файл застосунку, Node.js, мікрофон та/або записаний аудіофайл у форматі .wav.

При розробці програмного продукту використовувалась платформа Node.js, мова програмування JavaScript та середовище розробки Visual Studio Code.

2. ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблені компоненти виконують завдання розпізнавання ключових слів у вхідному аудіопотоці та візуалізації результатів. Це відбувається за рахунок об’єднання комплексних методів аналізу аудіо потоку.

Розроблену програму можна використовувати для розпізнавання ключових слів у мікроконтролерах або для створення систем голосового керування.

Функціональні обмеження на використання додатку полягають у ресурсах, що потрібні для виконання процесу розпізнавання у реальному часі.

3. ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Для забезпечення повноцінної роботи та досягнення високої точності та зручності роботи системи розпізнавання ключових слів було обрано платформу Node.js, використання якої дозволяє максимально просто реалізувати необхідні алгоритми та легко здійснити інтеграцію з PocketSphinx, а також має досить швидку обробку вхідних запитів користувача.

Також для роботи з обробкою звуку використовується додаткова PocketSphinx.

Розроблений додаток працює в операційній системі Linux, як десктопна програма або у мікроконтролерах, як серверна частина.

4. ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для реалізації сервісу швидкого прототипування було розроблено програму розпізнвання ключових слів на основі системи PocketSphinx. Для роботи з українською мовою була використана акустична модель російської. Всі алгоритми будуються на аналізі вхідного сигналу та прихованих марковських моделях.

Загалом, всі обчислення дискретизації вхідного звукового сигналу, та подальшої його обробки алгоримами, з подальшою візуалізацією результатів.

5. ВИКЛИК І ЗАВАНТАЖЕНЯ

Для початку роботи необхідно запустити сервіс розпізнавання.

Після запуску користувач бачить вікно сервісу де і вибирає необхідний йому варіант роботи.

6. ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для розробленого додатку є фіксований мікрофоном аудіопотік, або аудіофайл у форматі .wav.

Вихідними даними є графічне відображення результатів аналізу вхідних даних.