ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Кафедра ЕОМ

«Метод обробки даних електроенцефалограми з використанням засобів машинного навчання»

Кваліфікаційна робота
Другий (магістерський) рівень

Виконав:
ст. гр. СПм-20-1
Чернов Д.В.

Керівник:
доц. каф. ЕОМ
Іващенко Г.С.

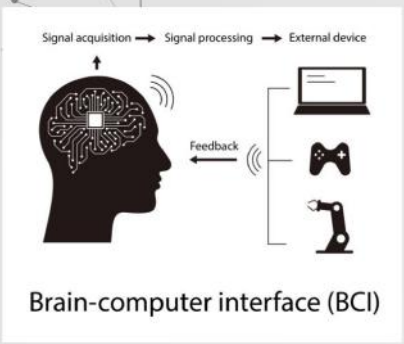
Актуальність роботи

Технологія електроенцефалограми вийшла на споживчий ринок завдяки компаніям-виробникам ЕЕГ-обладнання, що сприяло її використанню у різних сферах.

Засоби машинного навчання набули широкого використання у ЕЕГ-пристроях для виявлення та передбачення можливих артефактів при вимірюванні.

Присутність артефактів може негативно вплинути на різноманітні дослідження або бути корисною при застосуванні в різних ЕЕГ-пристроях. Одним з широко відомих типів артефактів є моргання очей, яке може з'являтись при будь-якому вимірюванні ЕЕГ.

Існуючі рішення



Brain-computer interface (BCI)

Нейрогарнітура NeuroSky MindWave Mobile

Засоби машинного навчання набули широкого використання у ЕЕГ-пристроях, які виконують роль інтерфейсу між мозком людини та комп'ютером. BCI-пристрої можуть виявити складні мозкові хвилі у реальному часі для контролю або управління різними пристроями, що може допомогти особам з легкими та важкими руховими порушеннями, включаючи тих, хто не може спілкуватися з іншими.

При використанні BCI-пристроїв людина уявляє як виконує діяльність або зосереджуються на об'єкті, не покладаючись на рухи м'язів, завдяки перетворенню специфічної активності мозку на функції управління та команди. Прикладами можуть виступати керування візками або переміщення курсору на екрані.

3

Постановка задачі

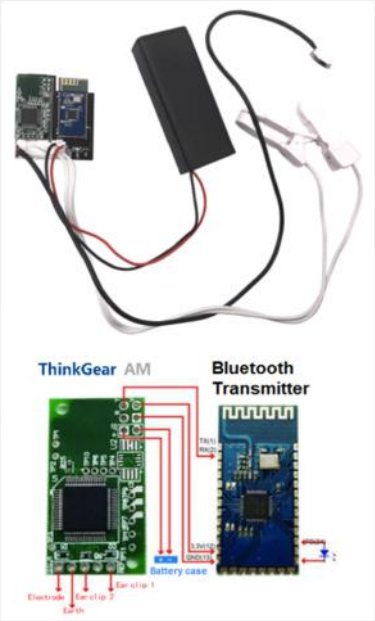
Метою кваліфікаційної роботи є реалізація методу обробки даних електроенцефалограми з використанням засобів машинного навчання.

- ❖ підготовити апаратну частину для отримання необроблених ЕЕГ-даних;
- ❖ забезпечити збір необроблених ЕЕГ-даних від апаратної частини;
- ❖ виявлення різних типів мозкових хвиль з необробленого сигналу (альфа, бета, гамма, дельта і тета хвилі);
- ❖ попередня обробка даних для використання засобами машинного навчання;
- ❖ прогнозування артефактів (закритого або відкритого стану очей людини) на основі оброблених даних.

4

Апаратна частина для отримання вхідних даних

- ThinkGear ASIC-модуль від компанії NeuroSky для отримання ЕЕГ-даних
- Bluetooth-передавач для з'єднання з програмною частиною
- Акумуляторна батарея для забезпечення автономності
- Один електрод, який розміщують на лобі



5

Отримання необроблених даних від апаратної частини

1. Виконання читання байтів з потоку, поки не зустрінеться байт SYNC, який дорівнює 0xAA.

2. Читання наступного байту та перевірка, що він також є байтом SYNC. Якщо це не байт SYNC, необхідно повернутись до кроку 1. В іншому випадку перейти до кроку 3.

3. Читання наступного байту із потоку як PLENGTH. Якщо PLENGTH дорівнює 0xAA (десятичне 170), то повторення кроку 3. Якщо значення PLENGTH перевищує 170, повернення до кроку 1. В іншому випадку перейти до кроку 4.

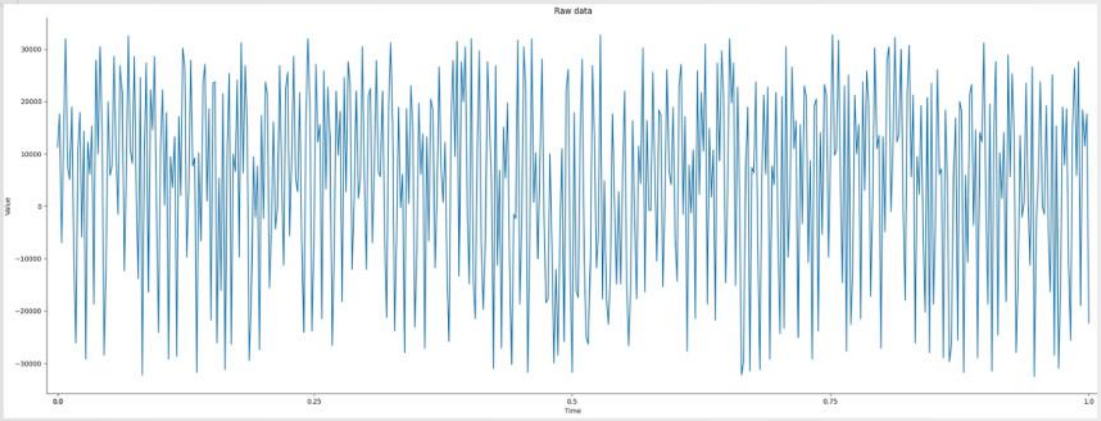
4. Читання PLENGTH байту з PAYLOAD потоку байтів, зберігаючи їх у сховищі. Підсумування кожного байту під час його читання шляхом збільшення акумулятора контрольної суми.

5. Взяття найнижчих 8 біт акумулятора контрольної суми та їх інвертування.

6. Читання наступного байту із потоку як байт CHKSUM. Якщо CHKSUM не відповідає обчисленій сумі, то ігнорувати дані, вивести попереджуюче повідомлення, перейти до кроку 1. В іншому випадку зберегти отримані дані. Якщо необхідна кількість даних зібрана, то повернути результат, у іншому випадку – перейти до кроку 1.

6

Приклад необроблених даних від апаратної частини



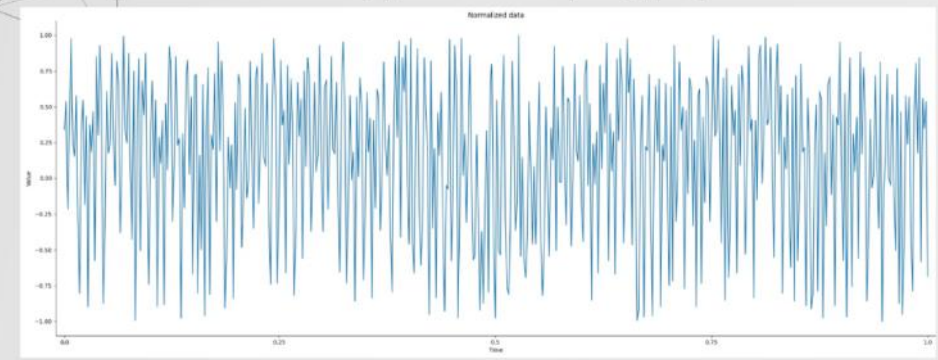
7

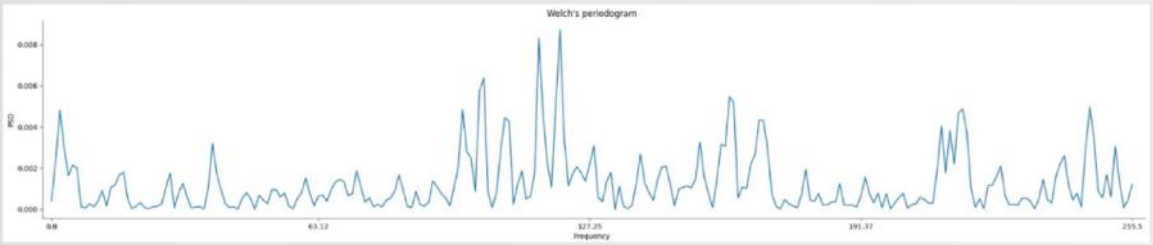
Визначення типів мозкових хвиль

Основним етапом аналізу ЕЕГ-даних є виявлення різних типів мозкових хвиль: альфа, бета, гамма, дельта і тета хвилі. Кожен з цих типів хвиль має власний діапазон частот, може бути виявлений з неопрацьованого ЕЕГ-сигналу та характеризує різний стан людського мозку. Після отримання необроблених даних від апаратної частини виконується їх нормалізація, отримання періодограми Уелча для підрахунку спектральної густини потужності, визначення сили діапазону хвиль мозку за допомогою правила Сімпсона для отримання площі під кривою графіка спектральної густини потужності.

8

Нормалізовані дані та періодограма Уелча

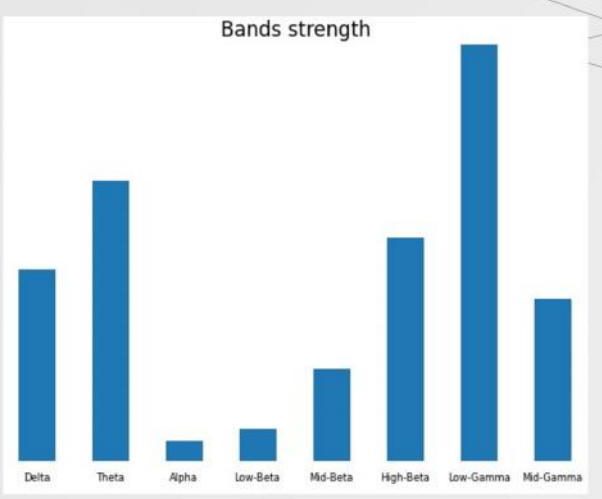




9

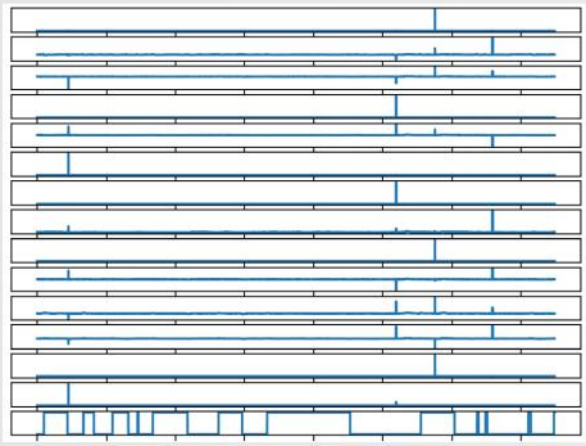
Визначення окремих хвилей мозку

Delta: [0.05, 3]
Theta: [3.05, 8]
Alpha: [8.05, 12]
Low-Beta: [12.05, 15]
Mid-Beta: [15.05, 18]
High-Beta: [18.05, 30]
Low-Gamma: [30.05, 39.75]
Mid-Gamma: [39.80, 49.75]



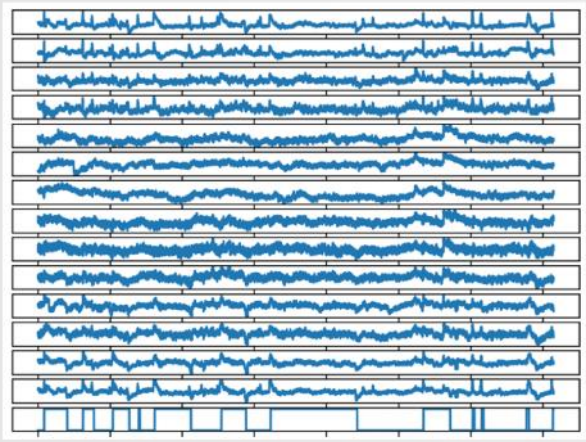
10

Вихідні дані для виявлення артефактів



11

Видалення аномальних викидів



12

Прогнозування стану очей

Для реалізації одного з етапів прогнозування стану очей було використано класифікацію на основі алгоритму K-найближчих сусідів при K=3, яка приймає дані для тренування та прогнозування, використовуючи програмні засоби бібліотеки sklearn.

Наявність лише одного файлу з ЕЕГ-даними призводить до вивчення параметрів функції прогнозування та тестування її на тих самих даних, що призведе до ідеальної оцінки та модель не зможе виконати прогноз на невидимих даних.

Для цього було використано модель KFold з бібліотеки sklearn для розділення вхідних даних на навчальні та тестові вибірки для вирішення завдання класифікації з даними, які не були використані під час навчання.

```
che@DESKTOP-718GH9I:/mnt/c/Users/Che/Projects/eeg_prediction$ python3 prediction_test.py
>0.978
>0.975
>0.970
>0.977
>0.973
>0.979
>0.978
>0.976
>0.974
>0.969
Final Score: 0.975
```

13

Висновки

Результатом роботи є створення методу обробки даних електроенцефалограми з використанням засобів машинного навчання.

Обробка отриманих від апаратної частини даних передбачає їх нормалізацію, отримання періодограми Уелча та визначення хвиль мозку з використанням правила Сімпсона.

При реалізації прогнозування закритого або відкритого стану очей людини на основі зібраних ЕЕГ-даних був використаний метод K-найближчих сусідів. Через відносну обмеженість можливостей апаратної частини на даному етапі для тестування було використано вхідні дані з відкритих джерел.

Результати роботи представлені у рамках Дев'ятої міжнародної науково-технічної конференції "Проблеми інформатизації".

14

ДОДАТОК Б

Основні характеристики модуля TGAM

Безпосередньо підключається до сухого електроду на відміну від звичайних медичних вологих датчиків. Має один канал ЕЕГ з трьома контактами: ЕЕГ, REF та GND. Низький рівень споживання енергії, що підходить для портативних пристроїв з акумуляторами. Максимальна споживана потужність 15 мА при 3,3 В.

Можливість сповіщення користувача про неправильне встановлення через попередження про погану якість сигналу від ASIC, якщо модуль був відключений від голови протягом чотирьох послідовних секунд, або якщо модуль отримує поганий сигнал протягом семи секунд поспіль.

Удосконалена технологія фільтрації з високою стійкістю до поміх.

Видача необроблених даних ЕЕГ зі швидкістю 512 біт у секунду.

Оброблювані метрики модулем TGAM:

- Неопрацьований сигнал мозкових хвиль.
- Обробка та видача спектрів потужності ЕЕГ (альфа, бета та інші).
- Обробка та висновок пропрієтарного лічильника eSense NeuroSky для метрик уваги, медитації та інших метрик майбутнього.
- Аналіз якості сигналу ЕЕГ/ЕКГ, який може використовуватися для визначення поганого контакту і того, чи прикріплений пристрій до голови.

Фізичні показники модуля TGAM:

- Максимальні розміри: 2,79 см x 1,52 см x 0,25 см.
- Максимальна вага: 130 мг.

Технічні характеристики модуля TGAM:

- Частота дискретизації 512 біт у секунду.
- Частотний діапазон 3-100 Гц.
- Захист від електростатичного розряду: контактний розряд 4 кВ, повітряний розряд 8 кВ.

- Максимальна споживана потужність: 15 мА при 3,3 В.
- Робоча напруга від 2,97 до 3,63 В.

Параметри стандартного вихідного інтерфейса UART модуля TGAM:

- Вихідна швидкість: 1200, 9600, 57600 бод.
- Розрядність: 8 біт.
- Парність: немає.
- Стоповий біт: 1.

Параметри фільтра шумів змінного струму модуля TGAM: 50 Гц, 60 Гц.

Вивід та швидкість передачі даних модуля TGAM:

- 1200 бод з даними про увагу, медитацію, можливості ЕЕГ (альфа, бета тощо) та погану якість сигналу.
- 9600 бод з даними про увагу, медитацію, можливості ЕЕГ (альфа, бета тощо) та погану якість сигналу.
- 57600 бод з даними про увагу, медитацію, можливості ЕЕГ (альфа, бета тощо) та погану якість сигналу та необроблені хвилі ЕЕГ.

Модуль TGAM може бути використаний з електродами з наступними характеристиками:

- Максимальна площа поверхні приблизно 150 мм2 (але оптимальна площа поверхні є меншою).
- Ag/AgCl, нержавіюча сталь, золото та/або срібло (працює як з твердого матеріалу, так і з покриттям).
- Електрод ЕЕГ, розташований над лівим або правим оком на лобі.
- Заземлювальні та контрольні електроди, розташовані за вухом або біля мочки вуха.
- Тиск, достатній для запобігання руху, має становити не менше 0,8 PSI.
- Довжина менше 12 дюймів, чим довше, тим вище сприйнятливість до шуму.
- Екранування (для землі не потрібно).
- Тонший, ніж AWG28 [21].

ДОДАТОК В

Вихідний код

Б.1 Файл run.py

```
import os

from scripts.data_output import EXPORT_MAPPING
from scripts.data_reading import (
    brainwave_read_bytes,
    brainwave_parse_bytes,
    brainwave_save_data
)
from scripts.data_processing import (
    brainwave_read_data,
    brainwave_filter_data,
    brainwave_normalize_data,
    brainwave_get_welch_periodogram,
    brainwave_get_band_strength
)
from utils.arg_parser import prepare_arg_parser
from utils.constants import (
    BYTES_TO_READ_PER_SECOND,
    VALUES_PER_SECOND,
    CommandNames,
    ExportNames,
)

if __name__ == '__main__':
    arg_parser = prepare_arg_parser()
    args = arg_parser.parse_args()

    if args.command == CommandNames.COLLECT:
        bytes_to_read = BYTES_TO_READ_PER_SECOND * args.duration
        values_to_read = VALUES_PER_SECOND * args.duration

        result_bytes = brainwave_read_bytes(args.address,
bytes_to_read=bytes_to_read)
        parsed_data = brainwave_parse_bytes(result_bytes,
values_limit=values_to_read)
        brainwave_save_data(
            parsed_data, values_per_file=values_to_read,
path=f'{args.path}/{ExportNames.RAW_DATA}'
        )

    elif args.command == CommandNames.PROCESS:
        read_data =
```

```
brainwave_read_data(f'{args.input_path}/{ExportNames.RAW_DATA}')
        for input_data, filename in read_data:
            filtered_data = brainwave_filter_data(input_data)
            normalized_data =
brainwave_normalize_data(input_data)
            normalized_filtered_data =
brainwave_normalize_data(filtered_data)
            welch_periodogram =
brainwave_get_welch_periodogram(normalized_data)
            band_strength =
brainwave_get_band_strength(welch_periodogram)

            filtered_data.to_csv(

f'{args.output_path}/{ExportNames.FILTERED_DATA}/{filename}',
index=False
            )
            normalized_data.to_csv(

f'{args.output_path}/{ExportNames.NORMALIZED_DATA}/{filename}',
index=False
            )
            normalized_filtered_data.to_csv(

f'{args.output_path}/{ExportNames.NORMALIZED_FILTERED_DATA}/{fil
ename}', index=False
            )
            welch_periodogram.to_csv(

f'{args.output_path}/{ExportNames.WELCH_DATA}/{filename}',
index=False
            )
            band_strength.to_csv(

f'{args.output_path}/{ExportNames.BANDS_DATA}/{filename}',
index=False
            )
    elif args.command == CommandNames.VISUALIZE:
        read_data =
brainwave_read_data(f'{args.input_path}/{args.source}')
        for input_data, filename in read_data:
            filename = os.path.splitext(filename)[0]
            filepath =
f'{args.output_path}/{args.source}/{filename}.jpg'

            export_handler = EXPORT_MAPPING[args.source]
            export_handler(input_data, args.source,
args.output_size, filepath)
```

Б.2 Файл data_preparation.py

```
from enum import Enum
from typing import List, Optional

import numpy as np
import pandas as pd
import serial


class ByteCodes(int, Enum):
    SYNC = 0xaa
    RAW_VALUE = 0x80


def brainwave_read_bytes(
        serial_port: str,
        bytes_to_read: int = 512,
        baud_rate: int = 57600,
        byte_size: int = 8,
        parity: str = 'N',
        stop_bits: int = 1,
        timeout: Optional[int] = None
) -> bytes:
    serial_obj = serial.Serial(serial_port)

    serial_obj.baudrate = baud_rate
    serial_obj.bytesize = byte_size
    serial_obj.parity = parity
    serial_obj.stopbits = stop_bits
    serial_obj.timeout = timeout

    result = serial_obj.read(bytes_to_read)
    serial_obj.close()
    return result


def _get_raw_value(low_byte: int, high_byte: int) -> int:
    raw_value = (high_byte << 8) | low_byte
    return raw_value - 65536 if raw_value > 32768 else raw_value


def brainwave_parse_bytes(
        input_bytes: bytes,
        values_limit: int = 512
) -> List[int]:

    result_data = []
    bytes_iterator = iter(input_bytes)

    try:
        while values_limit > 0:
```

```
            first_byte = next(bytes_iterator)
            second_byte = next(bytes_iterator)

            if first_byte != ByteCodes.SYNC or second_byte !=
ByteCodes.SYNC:
                continue

            packet_length = next(bytes_iterator)
            while packet_length == ByteCodes.SYNC:
                packet_length = next(bytes_iterator)
            if packet_length > ByteCodes.SYNC:
                continue

            packet_data = []
            packet_checksum = 0x00
            while packet_length > 0:
                packet_code = next(bytes_iterator)

                packet_checksum += packet_code
                packet_length -= 1

                if packet_code == ByteCodes.RAW_VALUE:
                    row_length = next(bytes_iterator)
                    low_byte = next(bytes_iterator)
                    high_byte = next(bytes_iterator)

                    packet_data.append(_get_raw_value(low_byte,
high_byte))

                    packet_checksum += row_length + low_byte +
high_byte
                    packet_length -= 3

            packet_checksum &= 0xFF
            packet_checksum = ~packet_checksum & 0xFF

            received_checksum = next(bytes_iterator)
            if received_checksum == packet_checksum:
                result_data.extend(packet_data)

            values_limit -= len(packet_data)

    except StopIteration:
        pass

    return result_data


def brainwave_save_data(data: List[int], values_per_file: int,
path: str):
    files_count = len(data) // values_per_file
    for file_number in range(files_count):
        left = file_number * values_per_file
```

```
        right = (file_number + 1) * values_per_file
        data_per_file = data[left:right]
        time_values = np.arange(len(data_per_file)) /
len(data_per_file)

        df = pd.DataFrame(list(zip(time_values, data_per_file)),
columns=['Time', 'Value'])
        df.to_csv(f'{path}/{file_number +
1:02d}_data_frame.csv', index=False)
```

Б.3 Файл data_processing.py

```
import os
from typing import List, Tuple

import pandas as pd
import numpy as np
from scipy import signal, integrate

BAND_RANGE = {
    'Delta': (0.05, 3),
    'Theta': (3.05, 8),
    'Alpha': (8.05, 12),
    'Low-Beta': (12.05, 15),
    'Mid-Beta': (15.05, 18),
    'High-Beta': (18.05, 30),
    'Low-Gamma': (30.05, 39.75),
    'Mid-Gamma': (39.80, 49.75),
}


def brainwave_read_data(path: str) -> List[Tuple[pd.DataFrame,
str]]:
    filenames = [name for name in os.listdir(path)
                 if os.path.isfile(f'{path}/{name}')]

    return [(pd.read_csv(f'{path}/{filename}'), filename) for
filename in filenames]


def brainwave_filter_data(input_data: pd.DataFrame) ->
pd.DataFrame:
    b, a = signal.butter(5, 0.0005, btype='highpass',
analog=False)
    filtered_data = signal.filtfilt(b, a, input_data['Value'])

    b, a = signal.butter(5, [0.1, 0.5], btype='bandstop')
    filtered_data = signal.filtfilt(b, a, filtered_data)

    return pd.DataFrame(list(zip(input_data['Time'],
filtered_data)),
```

```
                        columns=['Time', 'Value'])


def brainwave_normalize_data(filtered_data: pd.DataFrame) ->
pd.DataFrame:
    filtered_values = np.array(filtered_data['Value'])

    min_value = filtered_values.min()
    max_value = filtered_values.max()

    normalized_values = 2 * (filtered_values - min_value) /
(max_value - min_value) - 1
    return pd.DataFrame(list(zip(filtered_data['Time'],
normalized_values)),
                        columns=['Time', 'Value'])


def brainwave_get_welch_periodogram(normalized_data:
pd.DataFrame) -> pd.DataFrame:
    window_length = normalized_data['Time'].max() *
normalized_data['Time'].size
    frequency_values, power_spectral_density_values =
signal.welch(np.array(normalized_data['Value']),

normalized_data['Time'].size,

nperseg=window_length)
    return pd.DataFrame(list(zip(frequency_values,
power_spectral_density_values)),
                        columns=['Frequency', 'PSD'])


def brainwave_get_band_strength(welch_periodogram: pd.DataFrame)
-> pd.DataFrame:
    power_spectral_density_values = welch_periodogram['PSD']
    frequency_values = welch_periodogram['Frequency']

    band_values = []
    auc_values = []

    for band_name, band_range in BAND_RANGE.items():
        low_value, high_value = band_range
        band_result = (frequency_values >= low_value) &
(frequency_values < high_value)

        band_psd_values =
power_spectral_density_values[band_result]
        band_frequency_values = frequency_values[band_result]

        area_under_curve = 0.0
        if not band_psd_values.empty and not
band_frequency_values.empty:
            area_under_curve =
```

```
integrate.simpson(band_psd_values, band_frequency_values)

        band_values.append(band_name)
        auc_values.append(area_under_curve)

    total_auc = sum(auc_values)
    normalized_auc_values = np.array(auc_values) / total_auc

    normalized_result = pd.DataFrame(list(zip(band_values,
normalized_auc_values)), columns=['Band', 'AUC'])
    return normalized_result
```

Б.4 Файл data_output.py

```
from typing import Tuple, List

import pandas as pd

from utils.constants import ExportNames, SOURCE_NAMES_MAPPING


def _prepare_xticks_tuple(
        input_data: pd.DataFrame,
        dimensions: Tuple[str, str],
        step: int
) -> Tuple[List[int], List[int]]:
    first, second = dimensions
    xticks = [0, 0] + list(range(-1, len(input_data[first]),
step))[1:]
    xticklabels = [round(input_data[second][i], 2) for i in
xticks]
    return xticks, xticklabels


def brainwave_prepare_bands(input_data: pd.DataFrame, source:
str, output_size: int, filepath: str):
    ax = input_data.plot(kind='bar')
    ax.set_title(SOURCE_NAMES_MAPPING[source])
    ax.set_xticklabels(input_data['Band'], fontsize=6,
rotation='horizontal')
    ax.tick_params(bottom=False)
    ax.axes.get_yaxis().set_visible(False)
    for spine in ['top', 'bottom', 'left', 'right']:
        ax.spines[spine].set_visible(False)
    ax.get_legend().remove()

    ax.get_figure().savefig(filepath)
    ax.cla()


def brainwave_prepare_data(input_data: pd.DataFrame, source:
```

```
str, output_size: int, filepath: str):
    xticks, xticklabels = _prepare_xticks_tuple(input_data,

dimensions=('Value', 'Time'), step=128)
    ax = input_data['Value'].plot(figsize=(output_size, 10),
xticks=xticks)
    ax.set_title(SOURCE_NAMES_MAPPING[source])
    ax.set_xlabel('Time')
    ax.set_ylabel('Value')
    ax.set_xticklabels(xticklabels)
    for spine in ['top', 'right']:
        ax.spines[spine].set_visible(False)

    ax.get_figure().savefig(filepath)
    ax.cla()


def brainwave_prepare_welch_data(input_data: pd.DataFrame,
source: str, output_size: int, filepath: str):
    xticks, xticklabels = _prepare_xticks_tuple(input_data,

dimensions=('PSD', 'Frequency'), step=64)
    ax = input_data['PSD'].plot(figsize=(output_size, 5),
xticks=xticks)
    ax.set_title(SOURCE_NAMES_MAPPING[source])
    ax.set_xlabel('Frequency')
    ax.set_ylabel('PSD')
    ax.set_xticklabels(xticklabels)
    for spine in ['top', 'right']:
        ax.spines[spine].set_visible(False)

    ax.get_figure().savefig(filepath)
    ax.cla()


EXPORT_MAPPING = {
    ExportNames.RAW_DATA: brainwave_prepare_data,
    ExportNames.FILTERED_DATA: brainwave_prepare_data,
    ExportNames.NORMALIZED_DATA: brainwave_prepare_data,
    ExportNames.NORMALIZED_FILTERED_DATA:
brainwave_prepare_data,
    ExportNames.WELCH_DATA: brainwave_prepare_welch_data,
    ExportNames.BANDS_DATA: brainwave_prepare_bands,
}
```

Б.4 Файл arg_parser.py

```
import argparse
from utils.constants import CommandNames, ExportNames
```

```
def duration(value):
    parsed = int(value)
    if parsed < 1 or parsed > 30:
        raise ValueError('Duration must be in range [1:30]')
    return parsed


def prepare_arg_parser():
    parser = argparse.ArgumentParser(description='provide
command name')
    subparsers = parser.add_subparsers(dest='command')
    collect_subparser =
subparsers.add_parser(CommandNames.COLLECT)
    collect_subparser.add_argument('address', type=str,
                                   help='Please provide the port
information.')
    collect_subparser.add_argument('duration', type=duration,
                                   help='Please provide time
duration in seconds.'
                                        ' It must be at least
one second and no more than 30 seconds')
    collect_subparser.add_argument('path', type=str,
                                   help='Please provide path to
folder to save result data')
    process_subparser =
subparsers.add_parser(CommandNames.PROCESS)
    process_subparser.add_argument('input_path', type=str,
                                   help='Please provide path to
folder with raw data')
    process_subparser.add_argument('output_path', type=str,
                                   help='Please provide path to
folder to save result data')
    export_subparser =
subparsers.add_parser(CommandNames.VISUALIZE)
    export_subparser.add_argument('source', type=str,
                                  choices=[name for name in
ExportNames],
                                  help='Please provide folder
name with data')
    export_subparser.add_argument('output_size', type=int,
                                  help='Please provide output
graph size')
    export_subparser.add_argument('input_path', type=str,
                                  help='Please provide root
path')
    export_subparser.add_argument('output_path', type=str,
                                  help='Please provide path to
folder to save result data')

    return parser
```