Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних наук

# КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему *Технологія реалізації Діскорд чат-бота з використанням нейронної мережі за допомогою бібліотеки PyTorch*

Галузь знань: *12 – Інформаційні технології* (Шифр і назва галузі знань)

Спеціальність: *122 – Комп'ютерні науки* (Шифр і назва спеціальності)

Освітня програма: *Комп'ютерні науки* (Назва освітньої програми)

Виконав: *студент 4 курсу, група КН-19-1* (Курс, група виконавця) _____ (Підпис) *Д.О. Книш* (Ініціали, прізвище)

Керівник: *к.т.н., доцент кафедри КН* (Науковий ступінь, посада) _____ (Підпис) *Р.О. Багрій* (Ініціали, прізвище)

Нормоконтроль: *к.т.н., доцент кафедри КН* (Науковий ступінь, посада) _____ (Підпис) *Р.О. Багрій* (Ініціали, прізвище)

До захисту допускаю:
*Зав. кафедри КН, д.т.н., професор* _____ (Підпис) *О.В. Бармак* (Ініціали, прізвище)
01 06 2023 р.

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет *інформаційних технологій*
Кафедра *комп'ютерних наук*
Освітній ступінь *бакалавр*
Галузь знань *12 – Інформаційні технології*
Спеціальність *122 – Комп'ютерні науки*
Освітня програма *освітньо-професійна програма підготовки бакалавра*

ЗАТВЕРДЖУЮ
Завідувач кафедри *комп'ютерних наук*

(підпис)
*д.т.н., професор О.В. Бармак*
«*06*» *03* *2023* року

**ЗАВДАННЯ**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА**

1. Тема кваліфікаційної роботи бакалавра: *«Технологія реалізації Діскорд чат-бота з використанням нейронної мережі за допомогою бібліотеки PyTorch»*

2. Завдання видано студенту *Книшу Денису Олександровичу*
(прізвище, ім'я, по батькові)

3. Керівник роботи *доцент кафедри КН Багрій Руслан Олександрович*
(посада, прізвище, ім'я, по батькові)

4. Затверджено наказом університету від «*01*» *03* *202* р. № *5*

5. Зміст пояснювальної записки (перелік задач) та вихідні дані: *провести аналіз засобів штучного інтелекту для створення чат-ботів; підготувати дані до роботи з моделлю та навчити штучну нейронну мережу на основі бібліотеки PyTorch; розробити чат-бот, що розпізнає запитання та дає на них відповіді; підключити чат-бот до соціальної мережі Діскорд з використанням Діскорд API та провести тестування його взаємодії з користувачем. Вихідними даними для навчання нейронної мережі є визначена тема повідомлення для подальшої обробки.*

6. Календарний план виконання кваліфікаційної роботи бакалавра:

| № | Назва етапів (розділів) кваліфікаційної роботи бакалавра | Термін виконання | Примітка |
|---|---|---|---|
| 1 | *Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи бакалавра з керівником* | *грудень.2022* | виконано |
| 2 | *Ознайомлення з предметною областю, формулювання мети та задач дослідження, визначення об'єкта та предмета дослідження* | *січень 2023* | виконано |
| 3 | *Робота над розділом 1 – Характеристика предметної області та постановка задачі* | *січень 2023* | виконано |
| 4 | *Робота над розділом 2 – Проектування чат-бота з використанням нейронної мережі за допомогою бібліотеки PyTorch* | *березень 2023* | виконано |
| 5 | *Робота над розділом 3 – Програмна реалізація чат-бота з використанням нейронної мережі за допомогою бібліотеки PyTorch* | *квітень 2023* | виконано |
| 6 | *Оформлення пояснювальної записки згідно вимог* | *травень 2023* | виконано |
| 7 | *Підготовка статті до журналу, попередній захист кваліфікаційної роботи бакалавра* | *травень 2023* | виконано |
| 8 | *Захист кваліфікаційної роботи бакалавра на засіданні Екзаменаційної комісії* | *червень 2023* | |

Виконавець: *студент 4 курсу, група КН-19-1* (Курс, група виконавця) _____ (Підпис) *Д.О. Книш* (Ініціали, прізвище)

Керівник: *к.т.н., доцент кафедри КН* (Науковий ступінь, посада) _____ (Підпис) *Р.О. Багрій* (Ініціали, прізвище)

# Анотація

Тема кваліфікаційної роботи бакалавра: *Технологія реалізації Діскорд чат-бота з використанням нейронної мережі за допомогою бібліотеки PyTorch*

Виконавець кваліфікаційної роботи бакалавра: *студент групи КН-19-1 Книш Денис Олександрович*

Керівник кваліфікаційної роботи бакалавра: *к.т.н., доцент кафедри КН Багрій Руслан Олександрович*

Кваліфікаційна робота бакалавра містить:

| Пояснювальна записка | | | | Кількість додатків |
|---|---|---|---|---|
| Сторінок | Рисунків | Таблиць | Джерел інформації | |
| 67 | 40 | 2 | 25 | 2 |

Метою кваліфікаційної роботи бакалавра є розробка технології Діскорд чат-бота з використанням нейронної мережі за допомогою бібліотеки PyTorch, що розпізнає запитання та дає на них відповіді.

Результатом виконання кваліфікаційної роботи бакалавра є створення чат-боту до соціальної мережі Діскорд, що розпізнає запитання та дає на них відповіді з використанням штучної нейронної мережі на основі бібліотеки PyTorch.

Ключові слова: штучна нейронна мережа, бібліотека PyTorch, чат-бот, Діскорд.

Виконавець: *студент 4 курсу, група КН-19-1* _____ *Д.О. Книш*

Курс, група виконавця | Підпис | Ініціали, прізвище

# Зміст

Перелік скорочень

| **Скорочення, термін, позначення** | **Пояснення** |
|---|---|
| КРБ | Кваліфікаційна робота бакалавра |
| КН | Комп’ютерні науки |
| ІТ | Інформаційні технології |
| ШІ | Штучний інтелект |
| ПЗ | Програмне забезпечення |

# Вступ

Кваліфікаційна робота бакалавра присвячена розробці технології Діскорд чат-бота з використанням нейронної мережі за допомогою бібліотеки PyTorch, що розпізнає запитання та дає на них відповіді.

**Актуальність теми.**

Нейронні мережі – один із напрямків у розробці систем штучного інтелекту. Ідея полягає в тому, щоб максимально близько змоделювати роботу людського мозку – а саме його здатності до навчання та виправлення помилок. У цьому полягає головна особливість будь-якої нейронної мережі – вона здатна самостійно навчатися і діяти на підставі попереднього досвіду, щоразу роблячи все менше помилок.

Всі завдання, які можуть вирішувати нейронні мережі пов'язані з навчанням. Серед основних галузей застосування нейронних мереж – прогнозування, прийняття рішень, розпізнавання образів, оптимізація, аналіз даних.

Чат-боти на базі штучного інтелекту (ШІ) – це програмний додаток, який взаємодіє з користувачами природною мовою. В основі їх роботи лежать механізми опрацювання природної мови (NLP). Чат-боти можуть розуміти людську мову, імітувати розмови та виконувати прості автоматичні завдання.

**Мета кваліфікаційної роботи бакалавра** полягає у розробці технології Діскорд чат-бота з використанням нейронної мережі за допомогою бібліотеки PyTorch, що розпізнає запитання та дає на них відповіді.

**Об’єкт дослідження** – процес навчання чат-бота на основі бібліотеки PyTorch.

**Предмет дослідження** – методи збору та аналізу інформації, технології та методи проектування чат-бота з використанням нейронної мережі.

**Завдання кваліфікаційної роботи бакалавра** – провести аналіз засобів штучного інтелекту для створення чат-ботів; підготувати дані до роботи з

моделлю та навчити штучну нейронну мережу на основі бібліотеки PyTorch; розробити чат-бот, що розпізнає запитання та дає на них відповіді, підключити чат-бот до соціальної мережі Діскорд з використанням Діскорд API та провести тестування його взаємодії з користувачем.

# Розділ 1 Характеристика предметної області та постановка задачі

## 1.1 Аналіз предметної області

Чат бот – це програма, що автоматично взаємодіє з людьми через текстові повідомлення що отримуються та посилаються через чат-інтерфейс. Частіше всього їх використовують для вирішення таких задач як надання інформації, вирішення проблем або просто для розважальних цілей[1].

Технологія чат-ботів зародилася вже більше 50 років назад. Першим задокументованим чат-ботом стала ELIZA, що створена Джозефом Вайзенбаумом ще аж у 1966 році [2]. ELIZA створили для того щоб імітувати психолога. Вона розпізнавала ключові фрази і відповідала заготовленими фразами (рис 1.1). Хоч ELIZA дуже часто відповідала не правильно, та не враховувала контекст, але на той час це стало великим проривом і надихнуло інших дослідників на створення і модернізацію мовленнєвих інтерфейсів.

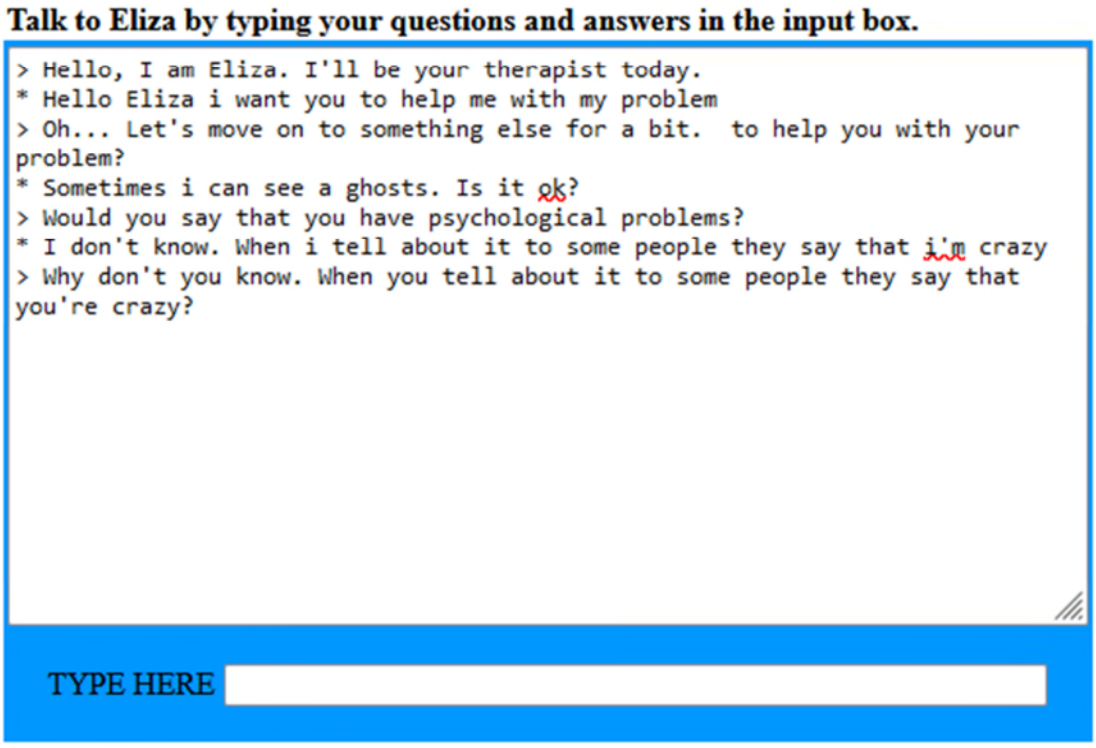



Рисунок 1.1 – Діалог з відтворенням чат-боту ELIZA [2]

Пізніше чат боти стали краще розпізнавати мову за допомогою послідовностей ключових слів і почали відповідати на більш специфічні запитання, але досі доволі часто могли неправильно розпізнати речення і відповісти на нього не так, як це задумано.

Отже, основною перешкодою для розвитку чат-ботів стало правильне розпізнавання отриманих повідомлень. Дану проблему стало можливо розв’язати за допомогою нейромереж, так як вони можуть розпізнавати саме семантичний зміст слів, що дозволило вивести технології чат-ботів на новий рівень і тепер, замість пошуку заготовлених ключових слів, чат-боти почали розуміти саме значення отриманого речення [3].

Основними параметрами чат-ботів є:

– функціональність – можливості та функції, що можуть бути виконані чат-ботом;

– мови – те, на яких мовах може проводитись взаємодія із чат-ботом. Він може бути запрограмованим для однієї або декількох мов;

– інтерфейс – спосіб взаємодії із чат-ботом. Чат-бот може використовувати текстовий інтерфейс, голосовий, а іноді навіть і візуальний;

– платформа – місце, де розташований чат-бот. Він може бути розміщений на таких платформах як Діскорд, Telegram, Facebook, і так далі.

Сьогодні користувачі можуть використовувати чат-боти для багатьох різних видів завдань – від взаємодії із даними мобільних додатків, до регуляції температури розумного термостату. В діловому секторі також можуть використовуватись чат-боти, наприклад маркетологи можуть їх використовувати для персоналізації взаємодії з клієнтами, call-центри використовують чат-ботів для надання відповідей на питання клієнтів, або направлення їх на потрібні ресурси [4].

Отже, враховуючи все вище сказане можна зробити висновок, що технологія чат-ботів дуже поширена на сьогоднішній день, і для найкращих результатів їх роботи використовують нейромережі, що можуть дуже добре

розпізнавати отримані повідомлення та відповідати на них. Їх основними параметрами є: функціональність, мова, інтерфейс та платформа.

## 1.2 Засоби створення нейромереж

### 1.2.1 Аналіз бібліотек для розробки нейромереж

На сьогоднішній день найпопулярнішими бібліотеками для машинного навчання є PyTorch та TensorFlow [5].

Основна відмінність між цими двома бібліотеками полягає в їх підходах до розробки а також архітектурі. PyTorch для обчислень використовує динамічні графи обчислень, що дозволяє більш гнучко та просто розробляти та налагоджувати глибокі нейроні мережі. TensorFlow же базується на статичному графі обчислень. Це забезпечує високу ефективність та можливість оптимізації при роботі із великим набором даних.

Основні переваги PyTorch є наступними:

– бібліотека є дуже простою у використанні;

– динамічна обробка графів, що дає полегшує процес налагодження моделі;

– зручна робота із базовими Python структурами даних;

– набагато більша популярність у порівнянні із TensorFlow станом на 2023 рік (рис 1.2)[6]

Недоліки PyTorch є наступними:

– менша швидкість роботи із великими наборами даних у порівнянні із TensorFlow

В свою чергу плюсами TensorFlow є:

– висока швидкість при обчислені великих наборів даних;

– велика кількість інструментів для розробки та налаштування моделей:

– велика кількість документації та навчальних проєктів.

Мінуси TensorFlow:

– бібліотека є складнішою у використанні у порівнянні з PyTorch;

– через статичний обчислювальний граф ускладняється процес налагодження моделі.

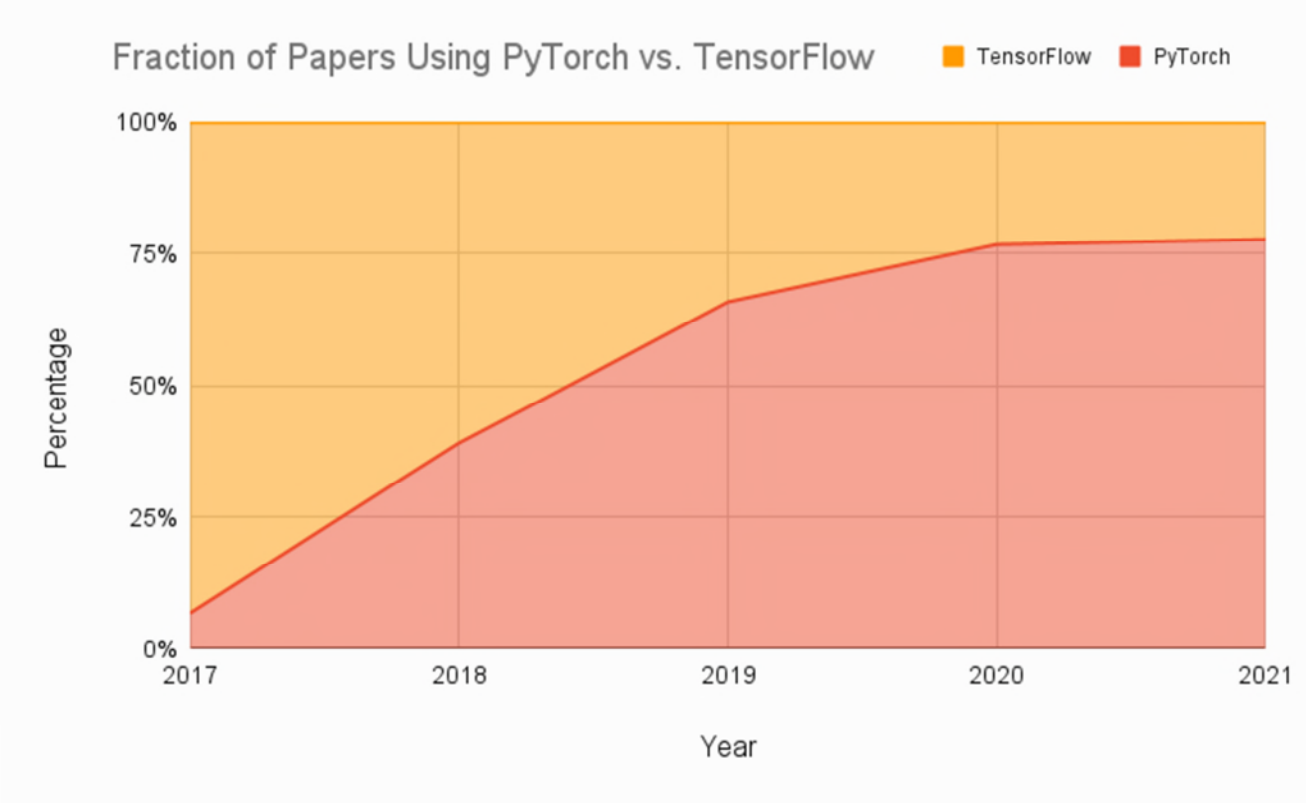



Рисунок 1.2 – Відсоткове співвідношення кількості документації PyTorch та Tensorflow у різні роки [6]

Отже, після порівняння двох бібліотек, створених розробки нейромережі, кращим варіантом виявилась бібліотека PyTorch через простоту розробки, а також через не критичну потребу у швидкості розрахунків.

### 1.2.2 Опис бібліотеки PyTorch

PyTorch – бібліотека з відкритим кодом для машинного навчання побудована на основі бібліотеки Torch, яка створена для таких задач, як комп’ютерний зір або обробка природної мови. За її розробкою стоїть команда з дослідження штучного інтелекту від Facebook [7].

PyTorch забезпечує такі високорівневі функціональності:

- тензорні обчислення із сильним прискоренням завдяки використанню графічних процесорів;
- глибинні нейронні мережі, що побудовані на системі автоматичного диференціювання.

До складу бібліотеки входять три основні модулі: autograd, optim та nn (рис 1.3) [8].

Модуль autograd є основним для бібліотеки PyTorch, він відповідає за автоматичне диференціювання операторів на тензорах. Це означає, що даний модуль може автоматично обчислювати градієнти функцій, що використовуються в нейронних мережах без їх ручного обчислення. Це зменшує кількість коду, який необхідно написати а також полегшує розробку та навчання нейронних мереж. Цей модуль створює динамічний граф обчислень, що зберігає всі операції, що виконуються на тензорах під час тренування нейромережі. Кожен вузол графа представляє собою операцію над тензорами, а кожне ребро, в свою чергу, представляє залежність між вузлами. Алгоритм обчислення градієнтів функцій називається алгоритмом зворотного поширення помилки. Також однією із ключових переваг модуля autograd є обчислення градієнтів для складних функцій, що складаються із декількох операцій над тензорами, що забезпечує пришвидшення розробки складних нейромереж.

Модуль optim відповідає за алгоритми оптимізації параметрів нейромереж. Він складається із набору класів, що відповідають за різні методи оптимізації, такі як SGR, ADAM, RMSProp та інші. Кожен із цих класів містить в собі методи, які дозволяють задавати різні параметри для оптимізації навчання такі як швидкість навчання, коефіцієнт зниження швидкості навчання, імпульс, та інші. За допомогою даних параметрів можна досягти значно кращих результатів при тренування нейронної мережі. Ці класи дозволяють автоматично оновлювати параметри моделі під час її тренування, використовуючи обчислення градієнтів, що здійснюються модулем autograd. Це забезпечує

ефективну оптимізацію параметрів моделі та сприяє зменшенню кількості часу, що необхідно для навчання нейронної мережі.

Модуль nn надає інтерфейс для побудови нейронних мереж. Він містить набір функцій та класів, що дозволяють з легкістю визначати та налаштовувати шари для нейронних мереж. Модуль містить класи-об'єкти для різноманітних шарів таких як повнозв'язні, згорткові, рекурентні та інші. Окрім цього в nn містяться різні види функцій активації такі як MSELoss, CrossEntropyLoss, NLLLoss, і так далі, функції втрат: ReLU, Tanh, Sigmoid, і так далі. Ці функції дозволяють визначити критерії за якими модель оцінюється під час процесу навчання. Цей модуль дозволяє з легкістю побудувати складні нейромережі, складаючи різні шари та операції. Також модуль має підтримку обчислень на графічних картах, що значно прискорює швидкість навчання нейромережі

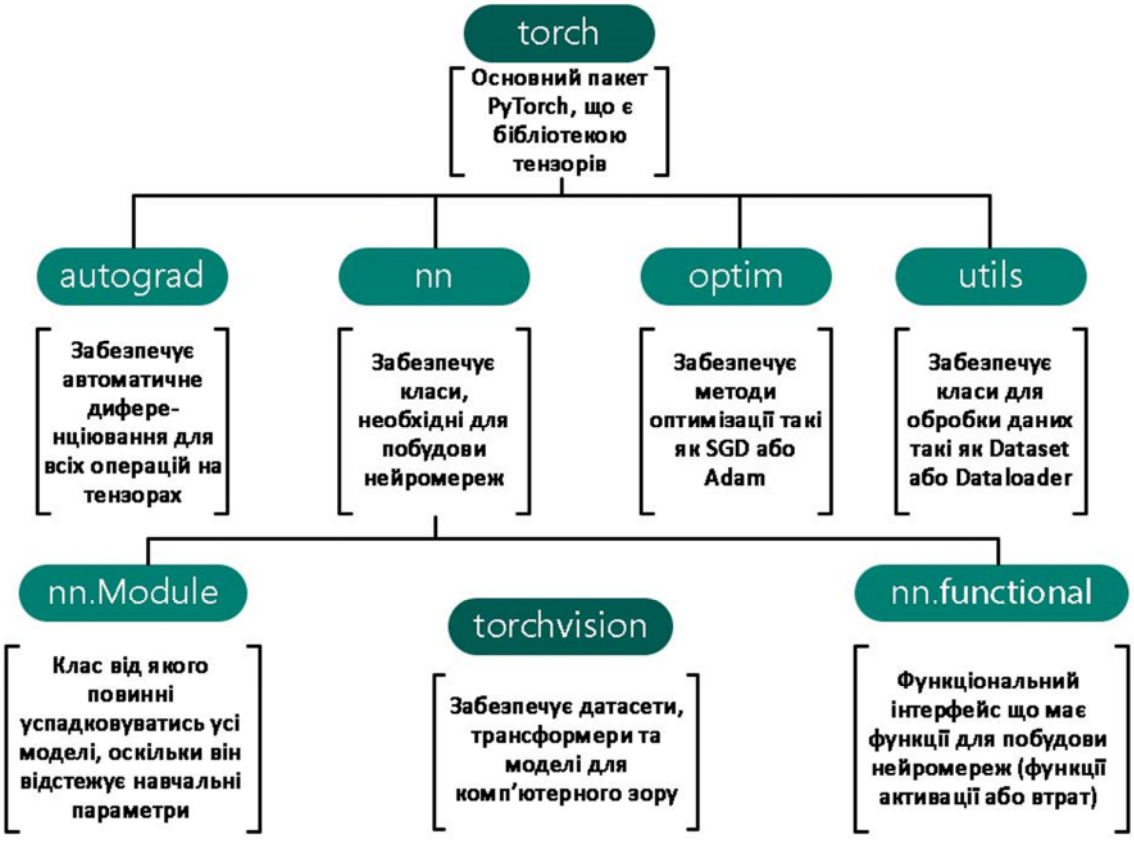



Рисунок 1.3 – Ієрархія модулів PyTorch [8]

Отже, бібліотека PyTorch має багато корисних інструментів, що надають можливість швидко та просто створити та навчити модель нейромережі

### 1.3 Аналіз існуючих чат-ботів

Для соціальної мережі Діскорд створено вже більше мільйона ботів, кожен з яких має різний функціонал. В більшості випадків цей функціонал полягає в модерації серверу і різних «слеш команд» для розваги учасників серверу, але також є деяка кількість ботів, які мають унікальний функціонал, що не зустрічається у інших ботів. Для аналізу вибрано деякі із найпопулярніших Діскорд ботів [9]

Clarity [10] – бот, який має найпоширеніший серед ботів набір функціоналу (рис 1.2), а саме:

- система модерації – бот має набір команд, які допомагають модераторам серверу керувати порядком. Команди дозволяють блокувати користувачів, виганяти їх, закривати доступ до певних чатів, змінювати їм ролі і так далі;
- система підписки – користувачі можуть прописати команду, що підпише їх на розсилку важливих оголошень. Бот надсилатиме підписаним користувачам оголошення у приватні повідомлення;
- розважальні команди – команди які запускають різні міні-ігри по типу хрестики-нулики, камінь ножиці папір, можуть запустити таймер, та інші;
- деякі інші команди.

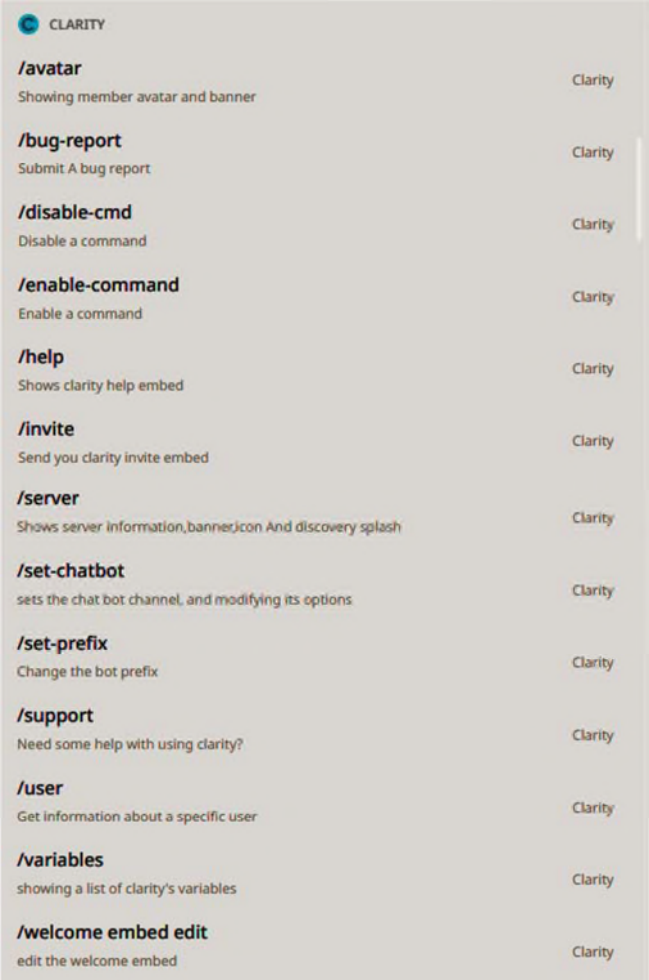



Рисунок 1.2 – Вікно із частиною списку команд Clarity

Cleverbot [11] – бот, основна задача якого – розмовляти із користувачем (рис 1.3). Отримавши повідомлення він обробляє його за допомогою нейромережі та присилає відповідь на нього. Підтримує 12 мов в тому числі і українську.


Рисунок 1.3 – розмова з Cleverbot

Wallu [12] – бот, створений для того, щоб відповідати на часто запитувані питання (рис 1.4). Бот розпізнаватиме запитання користувача і давати на них заготовлену відповідь. Навчається за допомогою історії запитань, і з кожним разом починає все краще розпізнавати запитання.

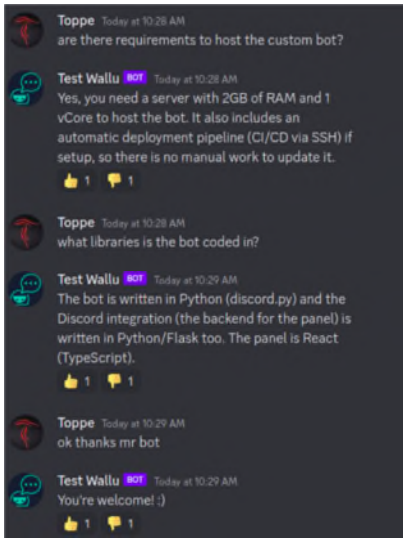



Рисунок 1.4 – Приклад розмови з Wallu із офіційного сайту

Midjourney [13] – бот, створений для генерування зображень із запитів користувача (рис. 1.5). Генерує зображення за допомогою методу випадкового шуму. На сьогоднішній день є одним із кращих ботів для генерації зображень.

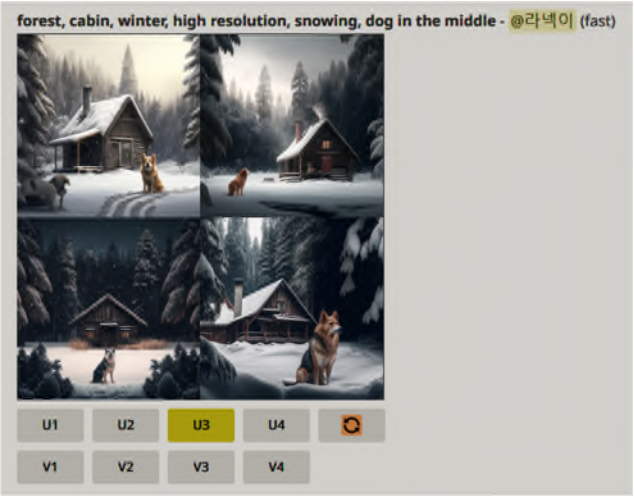



Рисунок 1.5 – Генерація зображення ботом Midjourney по запиту «ліс, хатина, зима, велике розширення, снігопад, собака по центру»

Отже технологія чат-ботів достатньо розвинута на сьогоднішній день. Можна знайти чат-ботів із різноманітними функціями такі як допомога у модерації серверів, відповіді на питання та іншими.

### 1.4 Мета, завдання та вимоги до реалізації інформаційної системи

Метою кваліфікаційної роботи бакалавра є розробка технології Діскорд чат-бота з використанням нейронної мережі за допомогою бібліотеки PyTorch, що може розпізнавати запитання та коректно на них відповідати.

Для досягнення поставленої мети необхідно реалізувати виконання наступних задач:

– провести аналіз засобів штучного інтелекту для створення чат-ботів;

– підготувати дані до роботи з моделлю та навчити штучну нейронну мережу на основі бібліотеки PyTorch;

– спроектувати чат-бот, що розпізнає отримані повідомлення українською мовою та дає на них відповіді;

– підключити чат-бот до соціальної мережі Діскорд з використанням Діскорд API та провести тестування його взаємодії з користувачем.

# Розділ 2 Проектування чат-бота з використанням нейронної мережі за допомогою бібліотеки PyTorch

## 2.1 Підготовка даних для роботи з штучною нейронною мережею

Для тренування нейромережі потрібно їй подавати текстові данні у форматі «зразок – визначення», наприклад зразок тексту «Привіт» має визначення «Привітання». Для зберігання даних у такому форматі ідеально підійде текстовий формат обміну даних JSON, адже дані у ньому зберігаються майже по такому ж принципу: «ключ - значення» [14]. Ключем виступає визначення, а для значень використано зразки тексту (рис 2.1).


Рисунок 2.1 – Приклад збереження зразків для одного визначення у JSON файлі

Формат збереження даних визначено, тепер можна перейти до створення навчального датасету. Зразки для навчального датасету мають добре відповідати

назначеному їм визначенню, також кожен зразок має бути створений в формі запитання звичайного користувача по темі заданого визначення, адже таким чином навчання нейромережі проходитиме найкраще.

Зразки не мають складатися із одного слова. Звичайно, можна додати декілька таких зразків, але якщо весь датасет складатиметься із таких зразків то виникне проблема нестачі контексту, або недонавчання. Звісно ж, нейромережа навчена на таких даних іноді зможе визначити деякі запитання і правильно на них відповідати, але в більшості випадків вона вагатиметься між декількома варіантами замість того, щоб визначати однозначно.

Для прикладу візьмемо наступне речення: «Добрий день, я хочу купити хостинг» - при недонавчена нейромережа вагитеметься одразу між багатьма варіантами – вона може розпізнати це як привітання або як запит про покупку хостингу. Щоб уникнути такої ситуації зразки потрібно подавати лише із правильним контекстом.

З іншого боку – зразки не повинні бути надто довгі, адже це призведе або до перенавчання. Наприклад речення: «Добрий день, я вже давно хотів купити якийсь хостинг адже в мене має з’явитись сайт і тому я почав його шукати в інтернеті і т. д.» - не підійде для навчання, адже тут дуже багато непотрібної інформації яка лише заважатиме нормальному розпізнаванню через створення шуму.

Зразки повинні бути середнього розміру, і повністю охоплювати сенс запиту. Наприклад речення для визначення «покупка хостингу» є таким: «Я хотів би купити хостинг» - це речення повністю охоплює сенс визначення і не має ніякої лишньої інформації [15].

Отже, для створення датасету використовуватиметься json файл, що заповнений стандартними фразами, які використовуються для запитань по різним темам.

## 2.2 Векторизація тексту

### 2.2.1 Вибір методу векторизації

PyTorch не може працювати із текстом напряму, замість цього він потребує числові вектори як вхідні дані. Процес перетворення текстової інформації на числові вектори називається векторизацією. Для векторизації існує декілька різних методів, найпоширенішими із них є: Bag of Words, TF-IDF і Word2Vec [16]. Для подальшої роботи необхідно обрати один із цих методів, для цього потрібно проаналізувати їх переваги та недоліки:

Bag of Words (мішок слів) [17] – це метод репрезентації текстів у вигляді векторів, де кожному слову в тексті відповідає окремий елемент вектора.

Переваги:

- простота та швидкість обчислення;
- дозволяє враховувати унікальність слів в тексті.

Недоліки:

- не враховує порядок слів в тексті;
- не враховує семантичний зміст слів;
- не враховує різницю важливості слів в тексті.

TF-IDF [18] – це метод, що оцінює важливість кожного слова в тексті, порівнюючи частоту вживання цього слова в зразку з частотою вживання цього слова в усіх зразках.

Переваги:

- враховує важливість слів в тексті;
- дозволяє враховувати унікальність слів в тексті.

Недоліки:

- не враховує порядок слів в тексті;
- не враховує семантичний зміст слів.

Word2Vec (слово у вектор) [19] – це метод, що використовує нейронні мережі для представлення слів у вигляді векторів. У векторному просторі, слова з більш подібним контекстом розташовані ближче одне до одного.

Переваги:

– враховує порядок слів в тексті;

– враховує семантичний зміст слів;

Недоліки:

– вимагає великої кількості даних для навчання;

– вимагає багато ресурсів для обчислень.

Метод Word2Vec є найкращим, адже враховує семантичний зміст слів, проте для його роботи потрібна дійсно дуже велика вибірка даних, тому цей метод відкинуто і залишаються методи TF-IDF і Bag of Words. Хоч метод TF-IDF на перший погляд і здається найкращим вибором через те, що він враховує частоту вживання слів у реченні, але його використання є більш доцільним для великих текстів. У поточному випадку для зразків використовується лише одне речення в якому одне і те ж слово зустрічатиметься дуже рідко, і при його використанні якість результату не сильно зміниться від якості результату, що надасть метод Bag of Words, лише з тією відмінністю, що обрахунок проходитиме трохи довше.

Отже, проаналізувавши всі вище вказані методи, для векторизації обрано метод Bag of Words через його простоту в реалізації.

### 2.2.2 Векторизація «Bag of Words»

Векторизація тесту за допомогою методу Bag of Words проходить у декілька етапів:

1) створення масиву всіх слів;
2) створення масивів для зразків;
3) заповнення масивів зразків.

Отже спочатку необхідно створити масив усіх слів, це відбувається наступним чином: усі речення зразків розбивають на слова, після чого кожне слово додають у масив, при цьому якщо слово вже є у масиві то воно відкидається (рис 2.2).


Рисунок 2.2 – Створення масиву всіх слів

На другому етапі для кожного зразку створюємо масив такої ж розмірності, як і розмір масиву всіх слів.

І на останньому етапі заповнюємо масив цифрами. Робиться це наступним чином: проводимо пошук позиції кожного слова із зразку у масиві усіх слів, і якщо це слово знайдено – на цій же позиції у масиві зразку ставиться 1. Після того як у зразку знайдено всі позиції слів із масиву всіх слів, на тих позиціях, на яких немає збігу ставиться 0 (рис 2.3). Такий процес повторюється для кожного зразку.

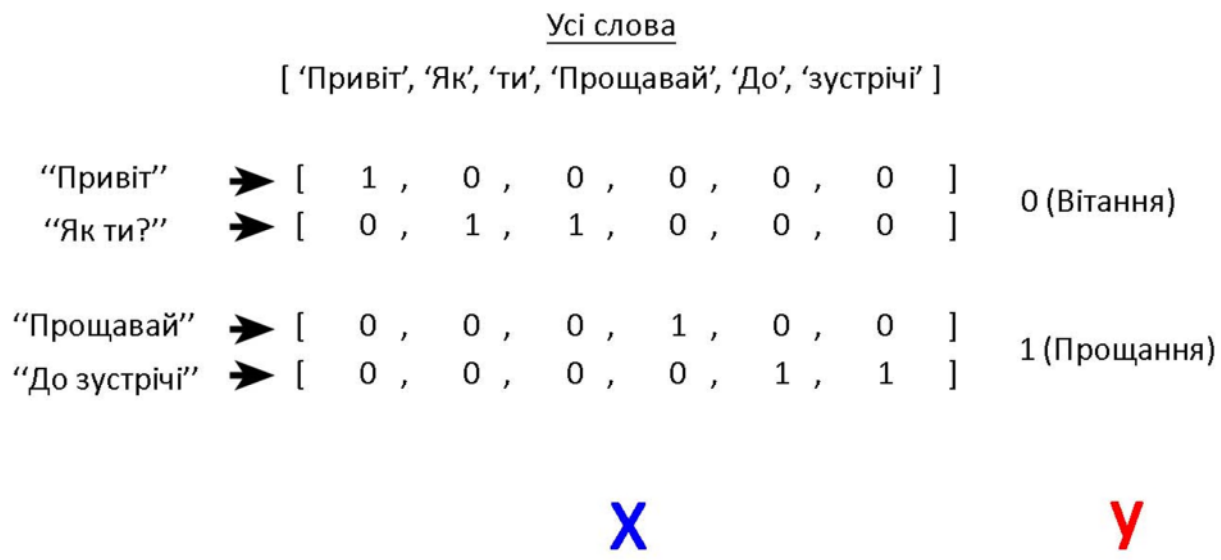



Рисунок 2.3 – Заповнення масивів зразків

## 2.2 Нормалізація тексту

### 2.2.1 Методи нормалізації тексту

Для покращення результатів навчання, а потім і розпізнавання, усі слова в прикладах нормалізують. Для тексту, процес нормалізації виконується за допомогою наступних методів: видалення розділових знаків, зведення всього тексту до нижнього регістру, видалення стоп-слів (займенників, дієприкметників, прийменників, вигуків, тощо), приведення слів до основної форми, також прибирання префіксів, суфіксів та закінчення, тобто зведення слів до кореню [20].

Таким чином це зменшить шум тексту, так як зведе одні і ті ж слова у різних відмінках або з різними префіксами до одного спільного, а також прибере багато слів, які ніяк не допоможуть розпізнаванню (рис 2.4)


Рисунок 2.4 – Процес нормалізації двох речень

### 2.3.2 Лематизація слів

Лематизація слова – це зведення слова до його основної форми. Наприклад слова «їхав», «їхала» та «їхало» - це все форми слова «їхати». Таким чином після лематизації видалиться велика кількість шуму, адже тепер слова у будь-якому відмінку, часі або роді будуть однаковими (рис 2.5).

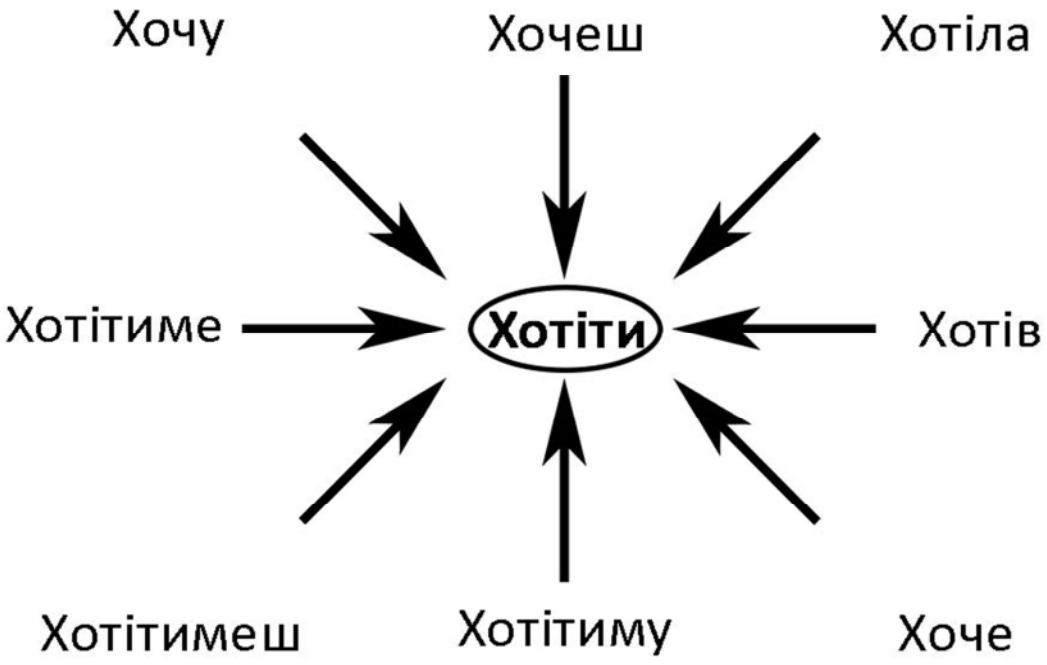



Рисунок 2.5 – Лематизація слів з основною формою «хотіти»

Через те, що в українській мові зведення слів до основи не можна автоматизувати, бо немає спільного правила по якому їх зводять, то для правильної лематизації необхідно мати словник усіх слів та їх форм у вигляді цифрових даних. З цією задачею допоможе бібліотека pymorphy2 [21]. Ця бібліотека використовує словник від OpenCorpora для української мови, а для невідомих слів використовуються різні влаштовані алгоритми. За допомогою цієї бібліотеки можна провести морфологічний аналіз слів – визначити частину мови, рід, час, відмінок і так далі, також є влаштований функціонал для зведення слова до його основи. Швидкість обробки слів в даній бібліотеці становить більше ста тисячі слів за секунду, що повністю задовільнить потребу у лематизації слів.

Проте після лематизації слів може виникнути проблема неправильного розуміння контексту, наприклад слово «бігав» означає, що дія вже завершилась, в той час як «бігатиме» означає, що дія відбуватиметься в майбутньому, проте після зведення цих слів до основи, результатом стане слово «бігати». Так як це може бути важливо для правильного розпізнавання, то необхідно це врахувати. Бібліотека pymorphy2 має не тільки функціонал для морфологічного аналізу слова та його нормалізації, а також і функціонал для зміни роду слова та його часу. Таким чином за допомогою даного функціоналу можна просто звести слова до одного роду та називного відмінку при цьому не змінюючи їх час. Так кількість форм слів зменшиться, проте проблема нерозуміння контексту не виникне.

Отже, для лематизації слів використовуватиметься бібліотека pymorphy2 для зменшення шуму прикладів

### 2.2.3 Стемінг слів

Стемінг – процес визначення кореня слова шляхом відкидання закінчень, префіксів та суфіксів. Це зменшує різноманітність форм одного слова та полегшує навчання та розпізнавання моделі (рис 2.6).

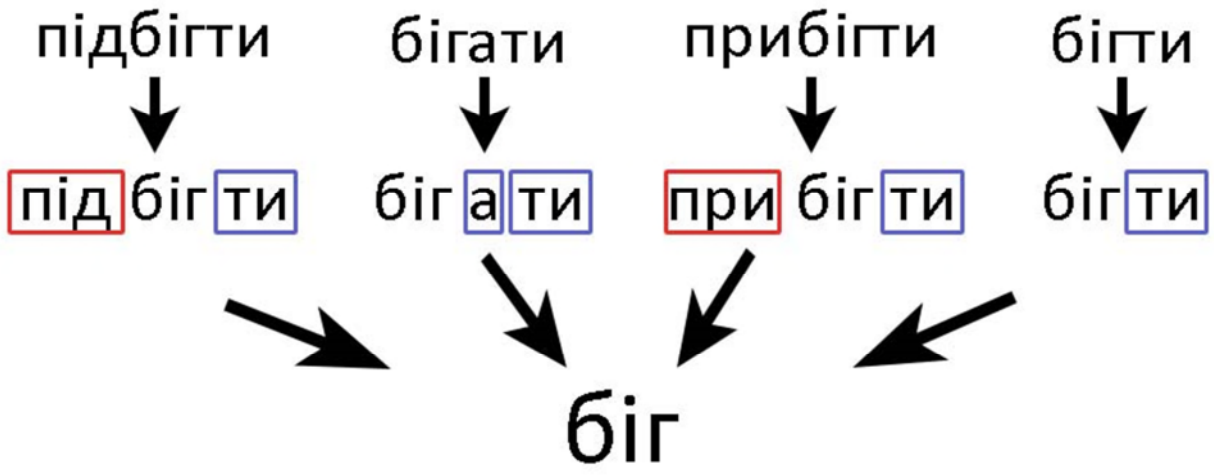



Рисунок 2.6 – Стемінг слів із коренем «біг»

Вже створених стемерів для української мови дуже мало, а для Python їх взагалі одиниці, проте все ж вдалося знайти одну бібліотеку для стемінгу, що працює на Python створену Кирилом Захаровим. Вона добре справляється із видаленням закінчень та суфіксів, але взагалі не працює із префіксами, тому результат стемінгу вийде не настільки ефективний, наскільки міг би. Тому функціонал даної бібліотеки розширюватиметься шляхом додавання функції видалення префіксів слів для того, щоб збільшити ефективність навчання та розпізнавання.

Отже, для стемінгу обрано бібліотеку від Кирила Захарова, функціонал якої доповниться для роботи із префіксами.

## 2.3 Методи створення нейромережі на основі бібліотеки PyTorch

### 2.3.1 Кроки для створення нейромережі

Процес побудови нейромережі складається із наступних кроків:

1) створення моделі;
2) визначення розмірів шарів нейромережі;
3) створення функції втрат;
4) створення оптимізатору;
5) налаштовування параметрів моделі.

На етапі створення моделі необхідно визначити тип нейромережі а також знайти спосіб створити модель для нейромережі обраного типу.

На етапі визначення розмірів та типів шарів нейромережі потрібно підібрати розмір згідно поставленої задачі.

Задачею етапу налаштування параметрів моделі є визначення оптимальних параметрів для найкращих результатів навчання.

На етапі створення функцій втрат необхідно створити функцію, що визначає те, наскільки добре нейромережа справляється із навчанням.

На останньому етапі потрібно створити оптимізатор, що покращуватиме результати навчання за допомогою оптимізації вагових коефіцієнтів нейромережі.

### 2.3.2 Створення моделі.

Для створення моделі необхідно визначитись, якого типу нейромережа використовуватиметься для поставленого завдання. Для задачі класифікації тексту зазвичай використовуються нейромережі типу прямого поширення або рекурентні нейронні мережі.

Нейромережі прямого поширення також відомі як штучні нейронні мережі або перцептрони. Інформація у даного типу нейромереж поширюється лише в один бік. Цей тип складається із штучних нейронів, що згруповані у шари. Зазвичай у даного типу нейромереж є один або декілька прихованих шарів окрім вхідного та вихідного. Кожен нейрон у нейромережі прямого поширення

отримує дані на вхід, обчислює їх вагову суму та використовує функцію активації щоб згенерувати вихід, після чого він з’єднується із входом наступного шару нейронів. Цей процес повторюватиметься до тих пір поки дані не дійдуть до вихідного шару. Найпоширеніший алгоритм навчання даного типу нейромереж є метод зворотного поширення помилки. Даний алгоритм дозволить знайти оптимальні ваги мережі, що дозволить досягнути достатньо високої точності [22].

Нейромережі даного типу мають такі переваги:

– ефективно обробляють великі кількості даних;

– використовують достатньо мало обчислюваних ресурсів;

– результати обробки мережі легко інтерпретуються;

– показують непогані результати навіть при невеликій навчальній вибірці.

Також присутні наступні недоліки:

– не здатні враховувати контекст попередніх кроків вхідних даних;

– мають обмежену здатність моделювати складні залежності між вхідними даними.

Рекурентні нейромережі працюють за допомогою іншого принципу – дані зберігаються на певних етапах для того, щоб в подальшому використовувати їх для обробки кожного наступного елемента послідовності. Дана нейромережа складається із рекурентних шарів компонентами якої є набір нейронів, що повторюється на кожному часовому кроці. Кожен нейрон отримує ваги та стан із попереднього часового кроку. При роботі з послідовністю, на кожному кроці мережа обробляє дані та зберігає стан мережі, який використовується для обробки наступного елемента послідовності [23].

Нейромережі рекурентного типу мають наступні переваги:

– здатні враховувати контекст інформації із попередніх кроків вхідних даних;

– здатні моделювати складні залежності між вхідними даними.

Також наявні такі недоліки:

– при наданні довгих послідовностей може виникнути проблема втрати контексту;

– менш стійкі до шуму;

– для навчання потребує достатньо великий набір навчальних даних для виявлення залежностей.

Отже після аналізу типів нейромереж найоптимальнішим варіантом виявилась нейромережа прямого поширення через те, що вона використовує менше обчислювальних ресурсів, а також показує непогані результати навіть при невеликій навчальній вибірці.

### 2.3.3 Визначення кількості шарів нейромережі

Нейромережі прямого поширення не мають чітко визначеної кількості шарів, тому в залежності від задачі їх кількість може змінюватись

Так як поставлена задача – класифікувати текст, то розмір нейромережі не повинен бути великим так як ця задача не є комплексною, тому повинно вистачити трьох шарів – перший із розміром масиву всіх слів, другий матиме схований розмір, який можна налаштовувати і останній шар має розмір, який дорівнює розміру масиву всіх тегів (рис. 2.5).

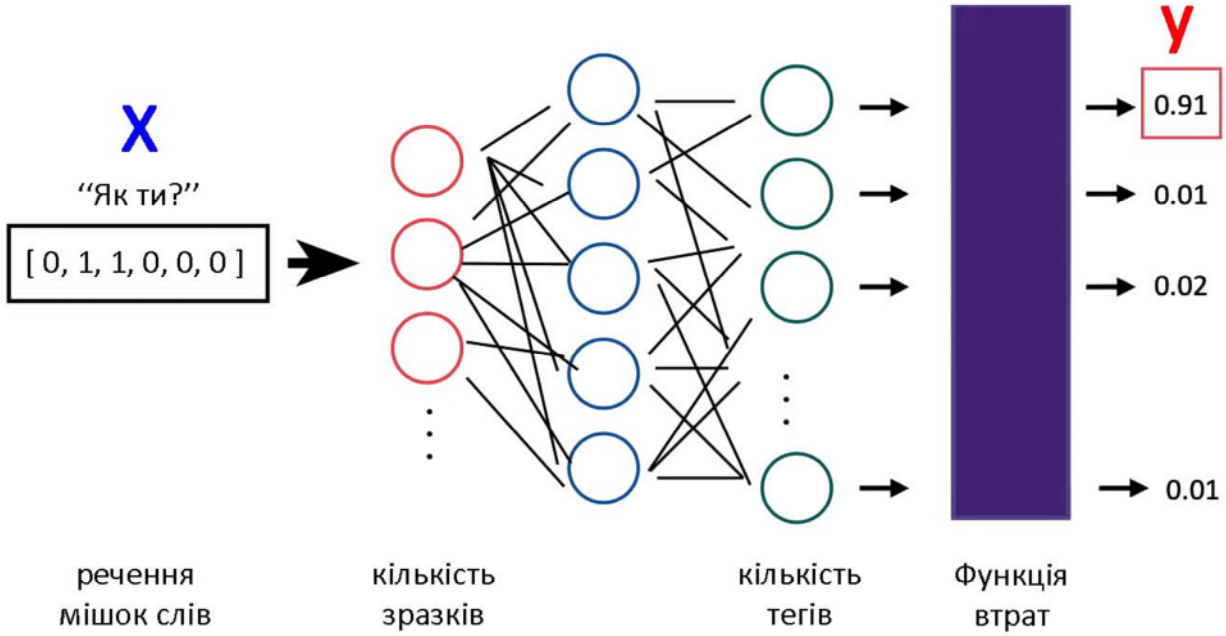



Рисунок 2.5 – Модель нейромережі типу прямого поширення

Схований шар повинен мати не надто великий розмір, але також і не занадто малий. При малому розмірі схованого шару нейромережа недонавчатиметься, і отримані результати не задовільнять потреб. При занадто великому схованому шарі нейромережа просто почне запам’ятовувати самі зразки замість того, щоб визначати їх основні риси. Звісно ж з великим розміром схованого шару нейромережа видаватиме ідеальні результати при розпізнаванні тренувальної вибірки, але при цьому розпізнавання нових прикладів стане проблематичним. Крім того завеликий розмір схованого шару може призвести до збільшення кількості часу, яке необхідне для навчання.

Отже, визначення розміру схованого шару для нейромережі прямого поширення відбуватиметься вже під час тестування моделі.

### 2.3.4 Вибір функції втрат

У глибинному навчанні функція втрат використовується для того, щоб оцінити те, наскільки добре нейромережа навчається. Дана функція розраховує

різницю між прогнозованими та реальними значеннями вихідних даних і їх мінімізація являється основною задачею навчання. У класі nn бібліотеки PyTorch вже є декілька влаштованих функцій втрат. Розберемо функції, що найкраще підходять для класифікації текстів:

– nn.CrossEntropyLoss – для задачі класифікації текстів дана функція застосовується найчастіше. Вона використовується, якщо кількість вихідних класів є незмінною та кожен зразок належить лише до одного класу. Ця функція є об'єднанням логарифмічної та softmax функцій. Формула даної фукції є наступною:

$$CrossEntropyLoss(y,t) = -\sum_{i=1}^{C} t_i \log(y_i) \quad (2.1)$$

де C – кількість класів, i – індекс класу, y – вектор вихідних значень, t – вектор міток класів.

– nn.MultiLabelSoftMarginLoss – ця функція частіше всього використовується, якщо приклад тексту можна віднести до декількох класів одночасно, але її також можна використовувати і для одного. Ця функція об'єднує в собі логістичну та функцію soft margin. Формула даної функції є наступною:

$$\mathrm{M}(y,t) = -\sum_{i=1}^{C}\left[t_i \log\left(\frac{\exp(y_i)}{\sum_{j=1}^{C}\exp(y_i)}\right) + (1+t_i)\log\left(1-\frac{\exp(y_i)}{\sum_{j=1}^{C}\exp(y_i)}\right)\right] \quad (2.2)$$

де M – MultiLabelSoftMarginLoss, C – кількість класів, i – індекс класу, y – вектор вихідних значень, t – вектор міток класів.

Це дві функції які найчастіше використовуються для багатокласової класифікації текстів, але так як nn.CrossEntropyLoss цілеспрямовано створена для текстів, що належать лише до одного класу, то її використання є доцільнішим ніж функції nn.MultiLabelSoftMarginLoss.

Отже, в якості функції втрат обрано nn.CrossEntropyLoss через те, що вона ідеально підходить для поточної ситуації.

### 2.3.5 Вибір оптимізатору

Оптимізатор – це алгоритм, що використовується для того, щоб налаштовувати параметри моделі під час її навчання. Робиться це для того, щоб зменшити функцію втрат для отримання кращих результатів. Розглянемо найпопулярніші оптимізатори SGD, Adam та Adagrad.

SGD – цей оптимізатор являється найпростішим, він працює методом оновлення вагів з кожним прикладом навчання. Це дозволяє швидко навчити модель.

Переваги даного оптимізатору є наступні:

– оптимізатор є простим у реалізації та при цьому ефективним при великому наборі даних;

– може допомогти унікальним мінімумам.

Недоліки:

– може застрягнути у локальному мінімумі.

Adam – найпопулярніший із оптимізаторів, працює методом збільшення коефіцієнта зневаження, що відповідає за швидкість згасання попереднього градієнту у пам’яті оптимізатора. Також використовує збережені оцінки моментів градієнту, що дозволяє навчати модель більш ефективно.

Плюси даного оптимізатору є наступними:

– для кожного параметру використовуються індивідуальні швидкості навчання;

– швидкість збіжності є більшою ніж у SGD.

Мінуси:

– переваги оптимізатора не завжди допомагають при невеликих обсягах даних.

– при різних умовах може виникнути проблема масштабування градієнту

Adagrad – даний оптимізатор забезпечує збільшення розміру кроку для параметрів, що рідко змінюються, а також зменшення розміру кроку для параметрів, які змінюються часто.

Основні переваги даного оптимізатора є наступні:

– є ефективним для невеликої кількості даних та невеликої кількості параметрів;

– швидкість навчання кожного параметра автоматично адаптується, що призводить до більш ефективного пошуку локального мінімуму.

Недоліки:

– погано працює з даними із складними структурами та багатовимірними просторами;

– в результаті накопичення градієнтів може зменшувати швидкість навчання, призводячи до того, що ця швидкість може стати дуже малою, або взагалі нульовою.

В результаті аналізу виявилось, що оптимізатор Adam підходить найкраще через його популярність, а також хороші характеристики.

### 2.3.6 Налаштовування параметрів нейромережі

У нейромереж є багато різних параметрів від яких залежить якість її навчання. Основними параметрами є наступні: розмір партії навчання (batch size), кількість епох навчання (epochs), крок навчання (learning rate).

Параметр batch size відповідає за кількість прикладів в партії, що обробляється за один раз при проходженні через мережу. Кожна партія складається із набору вхідних даних та очікуваних вихідних значень, що подаються до мережі. Однією із головних преваг такого методу подачі інформації є те, що обчислення можуть виконуватись паралельно на більшій частині вхідних даних, що значно зменшує час, необхідний для навчання моделі. Великий розмір партії може призвести до використання занадто великої

кількості пам’яті, особливо якщо модель має багато параметрів. Малий розмір партії в свою чергу може призвести до зменшення стійкості та точності навчання. Зазвичай для розпізнавання текстів розмір партії становить 16 або 32 приклади.

Параметр epochs відповідає за кількість повторних використовувань даних (епох). Наприклад значення 1 означатиме, що навчальні дані використовуватимуться всього один раз. При встановлені великого значення модель зможе краще розпізнавати відповідність між вхідними та вихідними даними, проте встановлення занадто великого значення призведе до того, що модель перенавчиться, тобто почне ідеально розпізнавати приклади із навчальної вибірки, проте у неї будуть виникати труднощі при розпізнавані даних, що не зустрічалися у навчальній вибірці. Для того щоб добре визначити кількість необхідних епох потрібно відслідковувати значення, що повертає функція втрат, при надто високих значеннях необхідно збільшити кількість епох, а при значеннях близьких до нуля необхідно зменшити кількість епох, адже це значить, що модель перенавчилась.

Параметр learning rate є найважливішим параметром для оптимізатора нейромережі. Він відповідає за розмір кроку, що виконується при градієнтному спуску під час оптимізації вагових коефіцієнтів мережі. Градієнтний спуск – це метод оптимізації, що полягає в пошуку мінімальної функції втрат шляхом регулювання вагових коефіцієнтів мережі. Швидкість навчання визначає те, з яким кроком вагові коефіцієнти будуть оновлюватись під час цього процесу. Використання надто великого значення параметру learning rate призведе до нестабільності навчання через те, що градієнт рухається надто швидко, це призводить до втрати точності. Якщо ж встановити цей параметр є надто малим, то навчання проходитиме дуже повільно, і нейромережа зможе застрягнути в локальному мінімумі. Розробники PyTorch рекомендують починати зі значення від 0.1 до 0.01 для невеликого набору даних або простих моделей, для

складніших моделей або великих наборів даних рекомендується встановлювати значення від 0.0001 до 0.00001.

Отже найкращими параметрами для початку навчання будуть batch size = 16, epochs = 3, і learning rate = 0.001, проте ці параметри будуть змінюватись в залежності від результатів навчання.

### 2.4 Засоби взаємодії із Діскорд API

Для запуску бота в Діскорд спочатку потрібно його зареєструвати на Діскорд Developer Portal. Щоб це зробити необхідно заповнити невелику форму та вказати дозволи, які матиме бот (рис 2.6). Боту необхідно надати лише дозвіл на зчитування та відправку повідомлень. На сайті також є калькулятор для розрахунку числа, що відповідатиме набору дозволів, що були вказані (рис 2.7). Цей калькулятор може знадобитися пізніше при запуску бота для вказування дозволів.

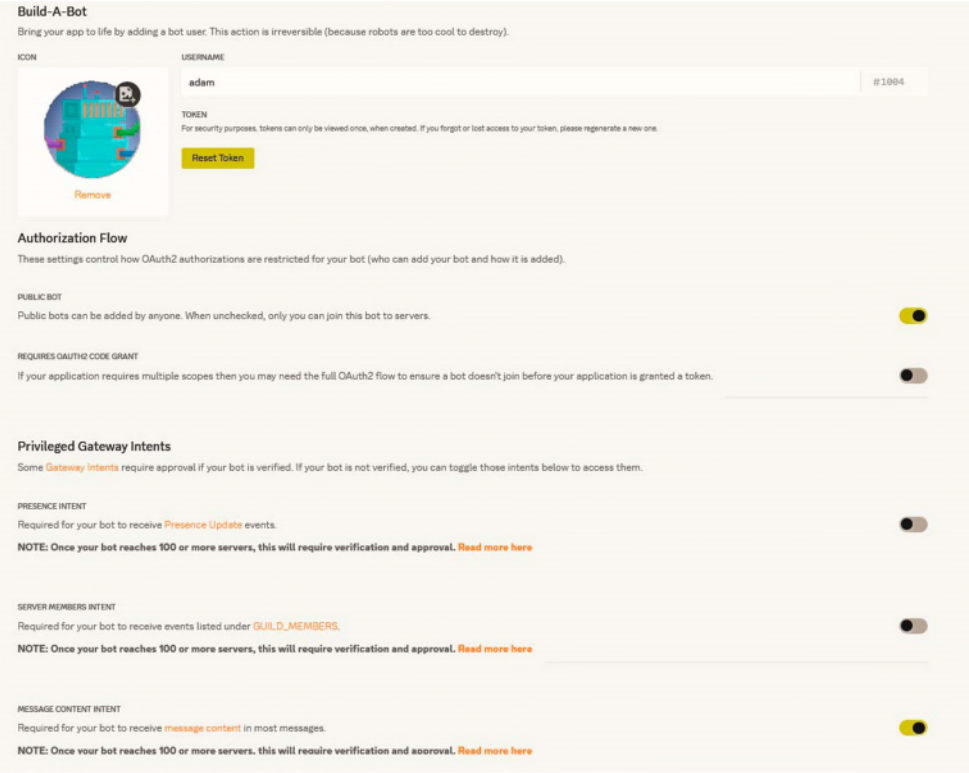


Рисунок 2.6 – Форма для налаштування параметрів бота

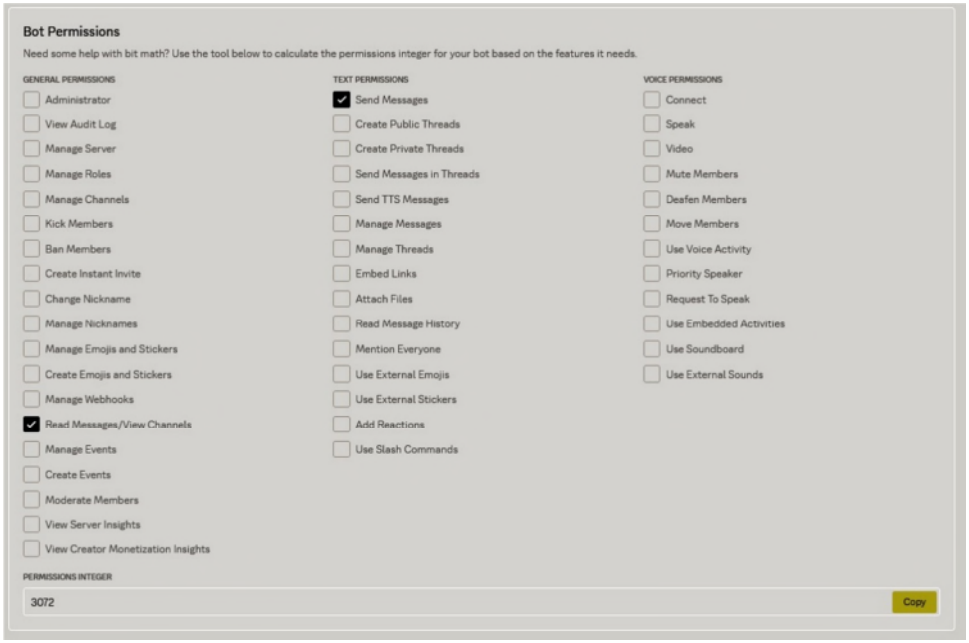



Рисунок 2.7 – Калькулятор для визначення числа, що відповідає набору дозволів Діскорд бота

Після цього видається токен, який використовуватиметься щоб з’єднатися з ботом.

Хоч розробники Діскорд і надають офіційну бібліотеку Діскорд API Library, що дозволяє взаємодіяти із Діскорд API, проте вона написана на мові програмування JavaScript. Хоча і можна створити канал обміну даних між JavaScript та Python, проте це лише створить додаткові складнощі для розробки і краще скористатися неофіційними бібліотеками для взаємодії із Діскорд API за допомогою Python.

Найпопулярнішою Python бібліотекою для взаємодії із Діскорд API є Discord.py. Переваги цієї бібліотеки є наступними:

– велика розширюваність – бібліотека дуже просто модифікується за допомогою різних плагінів та модулів;

– асинхронність - Discord.py створена із використанням асинхронного програмування, що дозволяє їй обробляти декілька запитів одночасно.

– підтримка спільноти - через те, що ця бібліотека є популярною то для неї створено багато різної документації, що допоможе при розробці

Незважаючи на вище перечисленні переваги, дана бібліотека має достатньо складну структуру, що підвищуватиме складність розробки бота. Для вирішення цієї проблеми створили розширення Disnake. Disnake представляє собою набір декораторів, які використовуються для задання поведінки бота при різних подіях. У поточному випадку корисним є обробник подій при отриманні повідомлень, який можна створити за допомогою всього лише однієї стрічки коду.

Отже, для взаємодії із Діскорд API використовуватиметься бібліотека Disnake, що є розширенням до бібліотеки Discord.py, адже це значно пришвидшить розробку бота.

# Розділ 3 Програмна реалізація чат-бота з використанням нейронної мережі за допомогою бібліотеки PyTorch

## 3.1 Середовище розробки нейронних мереж

PyTorch це бібліотека створена для мов програмування Python та C++. Для розробки нейромережі обрано мову програмування Python так як вона дуже добре підходить для розробки нейромереж а також для неї створено багато бібліотек які допомагають у розробці такого типу програм.

Для мови програмування необхідно обрати зручне середовище розробки. Для Python є багато середовищ для розробки, розглянемо переваги і недоліки найпопулярніших середовищ, а саме Sublime Text, Visual Studio Code та IDLE

Розглянемо переваги та недоліки Sublime Text.

Переваги:

– швидкість – sublime text не потребує багато ресурсів для роботи, із-за чого він працює дуже швидко навіть з багатьма файлами;

– велика кількість плагінів – середовище має велику кількість плагінів, що дозволяють розширити можливості редактора;

– наявність автодоповнення – редактор самостійно пропонуватиме різні варіанти для завершення запису.

– вільне використання – середовище є безкоштовним, хоча і є можливість купити повну версію.

Недоліки:

– відсутність влаштованого відладчика;

– відсутність можливості працювати з python із самого початку – для роботи з ним потрібно встановлювати додатковий пакет;

– відсутність терміналу – для роботи із терміналом необхідно встановлювати розширення.

Переваги середовища Visual Studio Code є наступними:

– вбудований відладчик коду – Visual Studio Code має вбудований відладчик, що допомагає у процесі створення програм;

– велика кількість плагінів – середовище має велику кількість плагінів, що дозволяють розширити можливості редактора;

– наявність автодоповнення – редактор самостійно пропонуватиме різні варіанти для завершення запису;

– повністю вільне використання – весь функціонал Visual Studio Code, а також пакети до нього є безкоштовними;

– влаштований термінал – середовище містить влаштований термінал, що дозволяє запускати файли, встановлювати та оновлювати додаткові пакети для середовища, та має інший функціонал.

Недоліки:

– із всіх вище перечислених середовищ Visual Studio Code вимагає найбільше ресурсів для стабільної роботи;

– відсутність можливості працювати з python із самого початку – для роботи з ним потрібно встановлювати додатковий пакет;

Переваги IDLE:

– вбудований відладчик коду – IDLE має вбудований відладчик, що допомагає у процесі створення програм;

– легкий у використанні – середовище має дуже простий інтерфейс, що робить його простим у використанні.

– IDLE створений для роботи із Python, тому в ньому можна працювати одразу після запуску.

Недоліки:

– IDLE має достатньо обмежені можливості у порівнянні із Sublime Text та Visual Studio Code;

– Середовище має не багато розширень для функціоналу.

Враховуючи всі плюси і мінуси представлених вище середовищ для розробки чат-боту обрано середовище Visual Studio Code.

## 3.2 Створення навчального датасету

### 3.2.1 Заповнення навчального датасету

Як вже вказано в розділі 2.1 – для зберігання прикладів та їх визначень використовуватиметься JSON файл. Цей файл складається із масиву, в кожній комірці якого зберігається тег та масив прикладів, що відносяться до цього тегу (рис. 3.1).

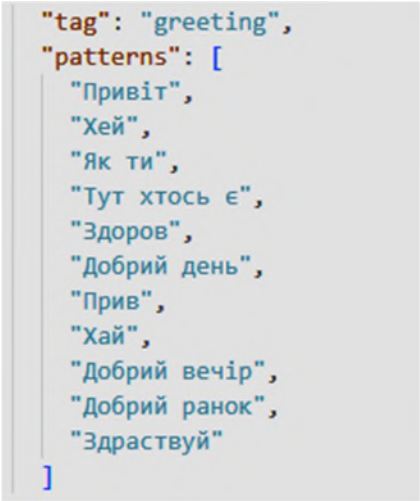



Рисунок 3.1 – Початок JSON файлу з навчальними даними

В прикладах використовуються різні варіації того, як людина може дати запитання по темі тегу (рис. 3.2), і чим більше прикладів надано, тим якісніше модель нейромережі класифікуватиме ці повідомлення.

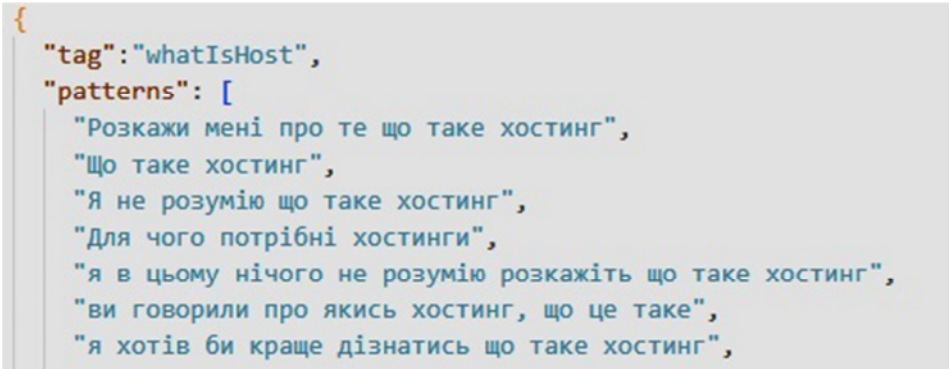



Рисунок 3.2 – Різноманітні способи спитати «що таке хостинг?»

В сумі для 16 тегів вдалося створити близько 600 прикладів, що не є надто великим числом, проте цього може бути достатньо для непоганого розпізнавання повідомлень невеликої комплексності.

### 3.2.2 Оптимізація навчального датасету

Спочатку необхідно вилучити приклади та теги із JSON файлу. У Python для роботи із JSON вже є влаштовані функції, тому це не викликає ніяких проблем. В після виконання, функція в якості результату повертає звичайний словник Python з яким можна надалі працювати (рис. 3.3).

```
with open('intents.json', 'r', encoding='utf-8') as f:
    intents = json.load(f)
```

Рисунок 3.3 – Код для перетворення JSON файлу в словник Python

Наступним кроком для форматування навчального  датасету є оптимізація прикладів. Для даних операцій необхідно створити окремий клас. Для початку краще всього токенізувати приклади попередньо видаливши усі розділові знаки, частки а також звівши речення до нижнього регістру. Після цього використовується функція word_tokenize із бібліотеки nltk (рис. 3.4)

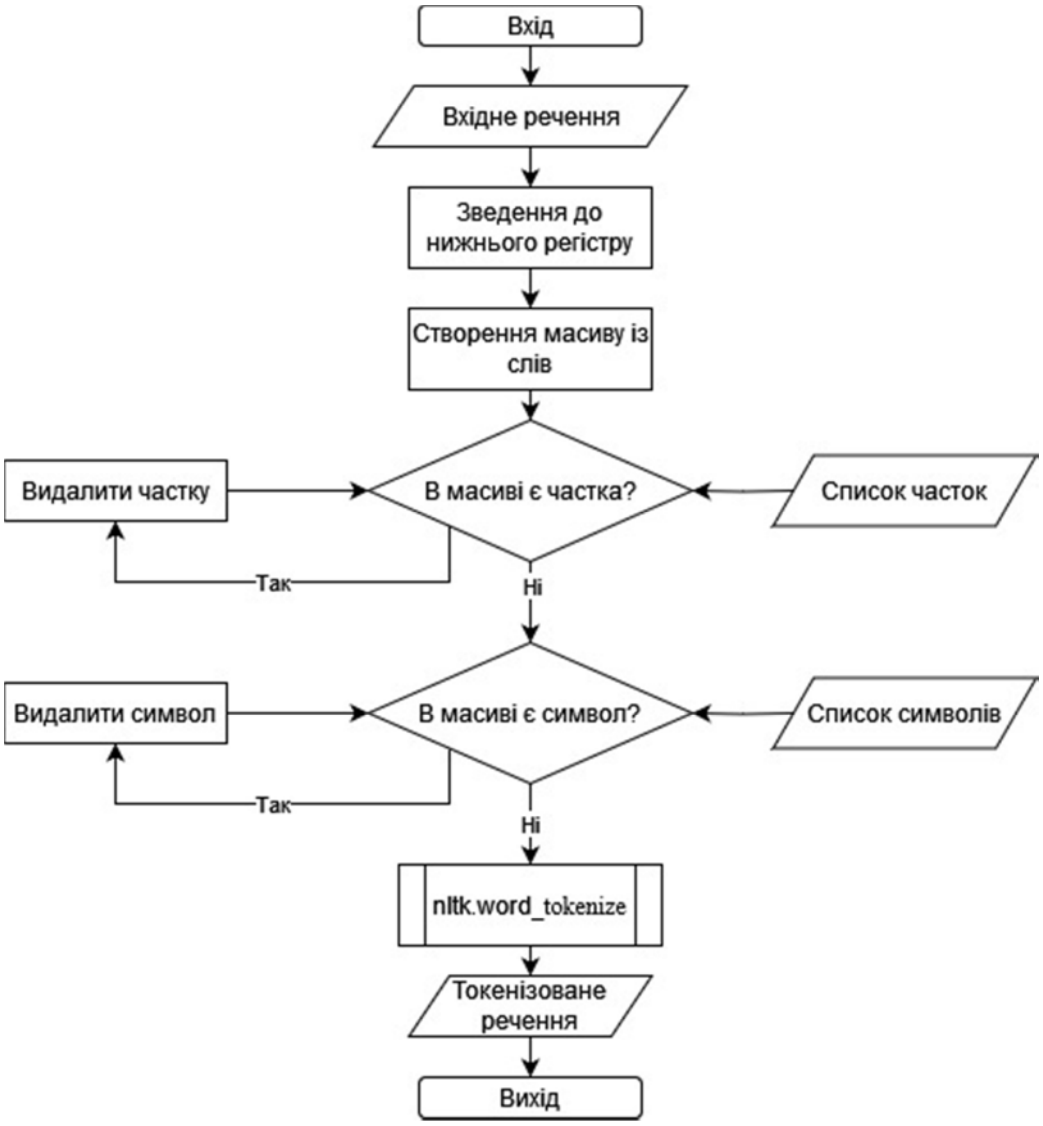



Рисунок 3.4 – Функція токенізації речення

Після токенізації можна лематизувати кожне слово за допомогою бібліотеки pymorphy2. Також слід не забувати про переведення слова до одного роду зі збереженням часу, в якому слово стояло до цього (рис. 3.5).

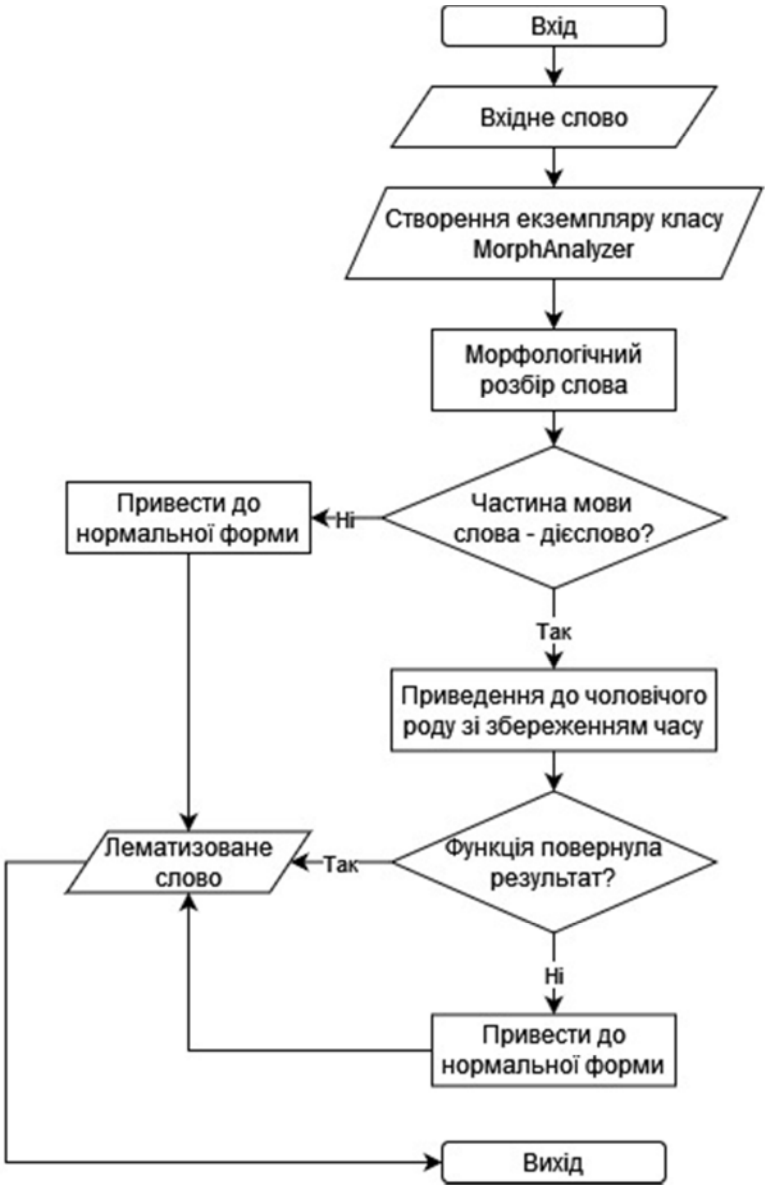



Рисунок 3.5 – Функція для лематизації слова

Спочатку необхідно створити екземпляр класу MorphAnalyzer і параметром вказати українську мову. Всередині функції слово аналізується за допомогою функції parse класу MorphAnalyzer. Потім проводиться перевірка на те, чи являється отримане слово дієсловом. У даній бібліотеці дієприкметники та дієприслівники також класифікуються як дієслова. Ця перевірка робиться, адже лише дані частини мови можуть мати часові форми, всі інші частини просто нормалізуються. Для функції inflect вказується чоловічий рід, а також час, що отримано від слова. Якщо функція не зможе знайти вказану форму для слова, то вона поверне NoneType, тому необхідно перевіряти результат функції, і якщо

вказана форма не знайдена, то це слово просто приводиться до нормальної форми.

Наступним кроком є стемінг слова. Тут використовуватиметься бібліотека Кирила Захарова, яку необхідно модифікувати для роботи із префіксами.

При ініціалізації класу в конструкторі створюються регулярні вирази для стемінгу.

При виклику основної функції бібліотеки stem_word спочатку за допомогою функції __ukstemmer_search_preprocess російські символи замінюються на їм співзвучні (рис. 3.7).

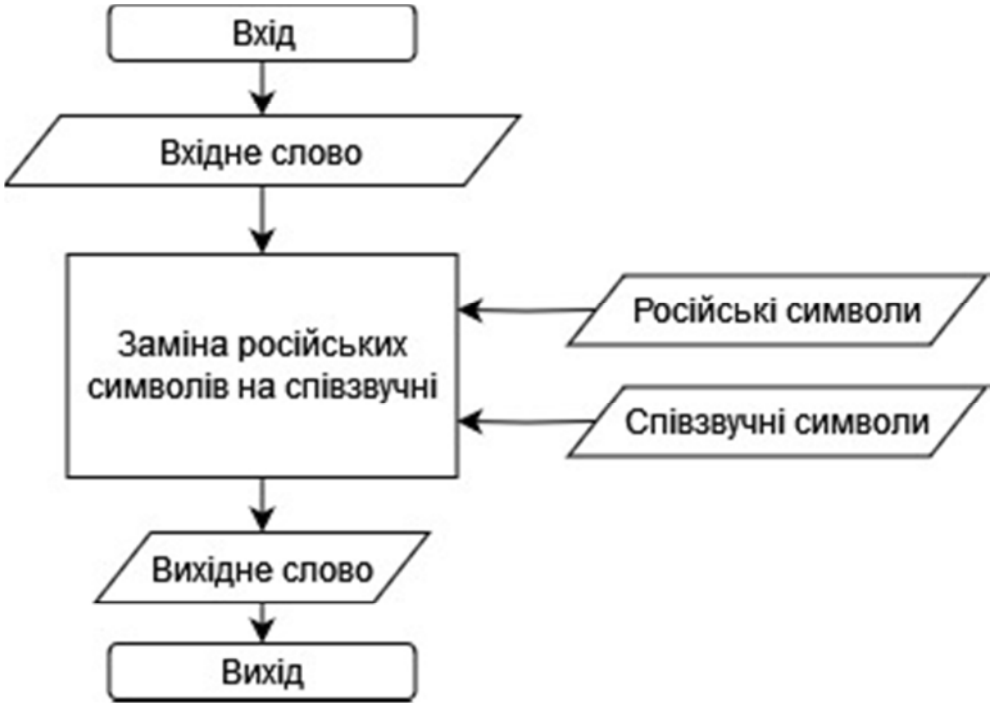



Рисунок 3.7 – Функція __ukstemmer_search_preprocess

Наступним кроком за допомогою модуля re, що відповідає за роботу із регулярними виразами, перевіряється, чи містяться голосні букви в слові, якщо ні, то слово повертається як стема. Скоріше за все це робиться через те, що слово без приголосних вже являється стемою, і не може містити ні закінчення ні суфіксу. Якщо ж голосні букви містяться, то далі йдуть різноманітні перевірки

по виразам, а також видалення знайдених частин слова за допомогою функції sub модуля re (рис. 3.8)


Рисунок 3.8 Функція stem_word

Щоб видалити префікси можна також додати вираз, що міститиме найуживаніші префікси, і по аналогії їх прибрати. Для цього можна скористатися аналогічним методом, тільки використовуючи вираз для пошуку префіксів (рис 3.10). У виразах автора використовуються символи “$”, що означають пошук в кінці тексту, щоб шукати вираз на початку тексту необхідно використати символ “^” (рис 3.9).

```
self.prefix = r'^(без|від|од|між|над|об|під|перед|понад|роз|через|пре|при|прі)'
```

Рисунок 3.9 – Вираз для пошуку префіксів


Рисунок 3.10 – Створене доповнення до функції

Тепер можна об’єднати усі ці функції у один цикл, що оброблятиме усі повідомлення із json файлу (рис 3.11)


Рисунок 3.11 – Цикл для оптимізації прикладів

Після цього для масиву всіх слів можна прибрати слова, що повторюються за допомогою функції set, а також відсортувати його за допомогою функції sorted.

Без оптимізації датасету отримано 312 унікальних слова, а після оптимізації унікальних слів стало 218. Після обробки вдалося видалити 94 різних варіацій одних і тих же слів, що зменшило кількість унікальних слів на 30% (рис 3.12).

```
Кількість унікальних слів до оптимізації: 312
Кількість унікальних слів після оптимізації: 218
Кількість видалених слів: 94 (30.13%)
```

Рисунок 3.12 – Порівняння неоптимізованого та оптимізованого масивів усіх слів

Отже за допомогою оптимізації датасету кількість унікальних слів зменшилась на 30%, що може позитивно відобразитись на результатах навчання а також розпізнавання.

### 3.2.3 Векторизація тексту

Після аналізу методів векторизації, обрано метод «Bag of words». Реалізація цього методу не є дуже складною. Створимо цей метод у вигляді функції, що приймає в якості параметрів токенізоване речення, а також масив усіх слів. Спочатку створимо масив, що має таку ж розмірність, як масив усіх слів, після цього повністю заповнюємо його нулями. При цьому також необхідно вказати тип масиву як float32, тобто 32-бітні числові дані з плаваючою крапкою. Робиться це за допомогою параметру dtype функції np.zeros. Це зв’язано з тим, що модуль nn.Linear, що використовуватиметься для побудови моделі приймає на вхід лише дані типу float32.

Наступним кроком є перевірка того, чи присутні слова із токенізованого речення у масиві всіх слів. Для цього можемо використати функцію enumerate, що створить ітератор з послідовністю пар індекс-значення. Робимо перевірку на

те, чи присутнє слово в реченні, і якщо це так, то на індексі цього слова в масиві нулів ставимо 1 (рис 3.13).


Рисунок 3.13 – Функція Bag of words

Після створення функції векторизації, перетворюємо масив ху із зразками та тегами, що зберігаються у вигляді тексту, на навчальні масиви у вигляді числових даних. Теги також перетворюємо на цифри за допомогою їх індексу в масиві тегів.

Отже, в результаті розробки отримано функцію, що може перетворювати текстові дані у їх векторний аналог.

### 3.2.4 Створення завантажувача даних

Для того, щоб ефективніше використовувати ресурси під час навчання моделі, розробники PyTorch рекомендують використовувати клас DataLoader із модуля utils. Цей клас дозволяє подавати дані за допомогою пакетів. Розмір цих пакетів залежить від параметру batch size, який описано в розділі 2.3.6. Також перевагою DataLoader є використання паралельного завантаження даних, що прискорює роботу при завантажені великих обсягів даних.

Клас DataLoader потребує подавати дані у вигляді посилання на екземпляр класу, що наслідується від ще одного класу із модуля utils – Dataset. У середині класу, що наслідує клас Dataset необхідно реалізувати два методи: «__getitem__» із числовим параметром «index» та «__len__». Метод «__getitem__» повинен повертати елемент датасету за допомогою наданого індексу, метод «__len__» - повертає кількість елементів датасету. Даними для Dataset є два python масиви у вигляді зразок-значення, тобто комірка із значенням прикладу має мати такий же індекс, як і комірка із відповідним прикладом. Такі масиви вже створено, тому просто надаємо їх класу.

Тепер можна створити екземпляр класу DataLoader подавши в якості параметрів створений датасет, параметр batch_size, також для незначного покращення результатів можна перемкнути параметр shuffle на True, що дозволить перемішування прикладів під час навчання. Також можна змінити параметр num_workers, що відповідає за кількість робочих процесів, що використовуються під час навчання. Так як поточний датасет є не надто великим, то можна встановити цей параметр зі значенням 0, яке означає, що завантаження даних відбуватиметься у головному процесі.

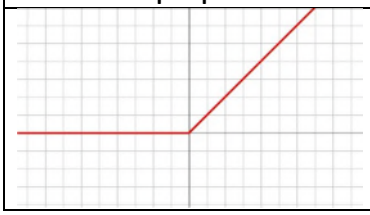
Отже, за допомогою створеного класу, що наслідується від класу Dataset, можна використовувати клас DataLoader, що дозволяє ефективно завантажувати дані до нейромережі

### 3.3 Створення моделі

У PyTorch створення моделі – це в першу чергу написання класу, що відповідає за обчислення вхідних значень моделі. Цей клас повинен успадковуватись від класу nn.Module, що є базовим класом для всіх моделей у PyTorch. Наступним кроком необхідно реалізувати абстрактний метод forward, що успадкований від nn.Module. Цей метод повинен відповідати за передачу даних вперед по шарам мережі. Ці шари необхідно створити у конструкторі класу. Як вже визначено у розділі 2.3.3, нейромережа повинна складатися із трьох лінійних шарів: вхідний має мати розмір кількості слів у масиві всіх слів, вихідний – розмір, що дорівнює розмірності масиву тегів, а середній шар має схований розмір. Для початку цей розмір можна вказати як 5, і потім змінити в залежності від результатів навчання. Створення лінійних шарів у PyTorch відбувається за допомогою класу nn.Linear. Конструктор даного класу приймає два значення – кількість входів та кількість виходів, тому для першого шару вкажемо вхідний розмір – кількість усіх слів, а вихідний – розмір схованого шару, для другого шару кількість входів і виходів - розмір схованого шару, в останнього шару вхід - розмір схованого шару, а вихід – кількість тегів. Для покращень результатів навчання та розпізнавання, між лінійними шарами зазвичай використовують функції активації, для додавання нелінійності в мережу. Це допоможе знаходити більш складні залежності між вхідними та вихідними даними, що важче вдається при звичайній лінійній залежності, що надають лінійні шари нейромережі. Найчастіше в якості функції активації використовують функцію ReLU [24]. Ця функція повертає 0, якщо отриманий сигнал менше 0, або повертає вхідний сигнал, якщо він все ж більший за 0 (табл. 3.1).

Дана функція популярна через свою швидкість та ефективність, а також через забезпечення здатності нейромережі вивчати більш складні залежності завдяки фільтрації вхідного сигналу.

Таблиця 3.1 – Параметри функції ReLU

| Графік | Рівняння | Область |
|---|---|---|
|  | $f(x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$ | $[0, \infty)$ |

При наслідувані від класу nn.Module необхідно реалізувати метод forward. Цей метод повинен приймати на вхід тензори, та виконувати передачу даних через шари та операції моделі, в поточному випадку – три лінійні шари, а також функції активації ReLU між ними.

Отже за допомогою наслідування від класу nn.Module створено модель з трьома лінійними шарами, та функціями активації між ними, що покращують результати навчання.

### 3.4 Вказування обчислювального пристрою

Для прискорення розрахунків PyTorch може використовувати графічний прискорювач NVIDIA CUDA. Це можна зробити за допомогою класу torch.device, вказавши в якості параметру «cuda». Проте не всі пристрої можуть використовувати даний графічний прискорювач, так як для цього необхідно мати відеокарту від компанії NVIDIA, що підтримує технологію CUDA. У цьому випадку можна проводити розрахунки за допомогою процесора, проте це може зайняти значно більше часу (рис 3.15).

За допомогою функції «torch.cuda.is_available» можна перевірити, чи підтримує пристрій cuda, і якщо підтримує – то проводити обчислення за допомогою графічного прискорювача, якщо ж ні – використовувати cpu.

Тепер необхідно передати екземпляр створеної моделі на доступний пристрій за допомогою методу torch.Tensor.to, та вказавши пристрій в якості параметру.

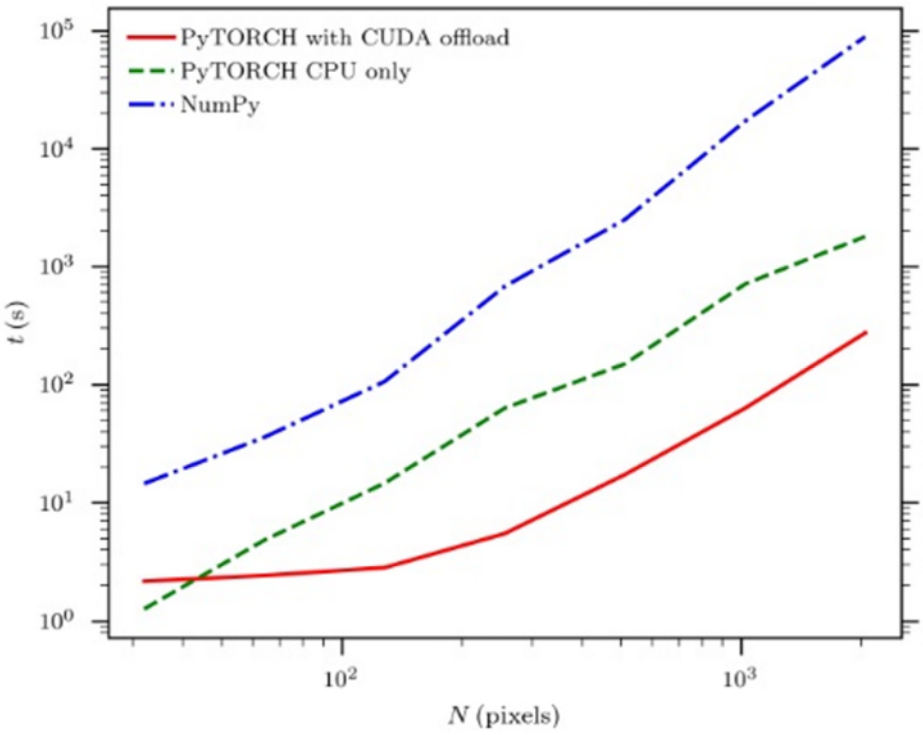



Рисунок 3.15 – Порівняння швидкості роботи CUDA, CPU та звичйного NumPy для різної кількості пікселів на зображенні [25]

Отже, при наявності графічного прискорювача cuda, можна значно прискорити швидкість обчислень за допомогою їх переносу на GPU

### 3.5 Створення функції втрат та оптимізатора

Як вже визначено в розділі 2.3.4 – в якості функції втрат виступатиме функція «Cross entropy loss». В модулі nn бібліотеки PyTorch є багато різних функцій втрат, в тому числі і «Cross entropy loss», тому її можна отримати створивши екземпляр класу nn.CrossEntropyLoss.

У розділі 2.3.5 в якості оптимізатора обрано оптимізатор «Adam». Даний також вже є вбудованим у бібліотеку PyTorch і знаходиться у модулі optim. Отримати даний оптимізатор можна створивши екземпляр класу torch.optim.Adam. В якості параметрів йому необхідно надати параметри моделі, їх можна отримати за допомогою методу «parameters» із батьківського класу

моделі, також необхідно вказати параметр «learning_rate». Початкове значення даного параметру визначено у розділі 2.3.6, і становить воно 0.001. Для запобігання перенавчання можна також вказати параметр «weight_decay». Цей параметр відповідає за зменшення ваг моделі. В якості початкового значення можна встановити 0.001, і потім в змінювати його в залежності від результатів навчання.

Отже, за допомогою модулів бібліотеки PyTorch створено функцію втрат «Cross entropy loss» а також оптимізатор «Adam».

### 3.6 Створення циклу навчання

Цикл навчання – найважливіша частина програми, адже саме тут проводиться навчання моделі нейромережі, яка потім використовуватиметься для розпізнавання повідомлень.

Спочатку необхідно створити цикл, що повторюватиметься таку кількість разів, яка вказана в параметрі «epochs», початковим значенням даного параметру у розділі 2.3.6 обрано 3.

Далі необхідно створити ще один цикл, на цей раз вже для пакетів завантажувача даних. В якості параметрів циклу використовуються тензори речень та тензори тегів цих речень.

Наступним кроком необхідно подати ці тензори на пристрій для прискорення обчислень.

Далі надаємо моделі тензори речень для навчання, та розраховуємо втрати за допомогою функції втрат, подаючи їй в якості параметрів результат обробки даних моделлю, а також правильні значення.

Після цього необхідно обнулити градієнти оптимізатора, це робиться для того, щоб кожна ітерація починала обрахунок заново, не використовуючи результати попередніх ітерацій. Це робиться за допомогою методу «zero_grad» оптимізатора.

Наступним етапом є обчислення градієнтів функції втрат відносно всіх параметрів моделі. Ця дія виконується для того, щоб визначити наскільки необхідно змінити параметри моделі для отримання кращих результатів. Це можна зробити за допомогою методу «backward» функції втрат.

Останньою дією навчального циклу є оновлення значень параметрів моделі після обчислення градієнтів функцією втрат. Робиться це за допомогою методу «step» оптимізатора (рис. 3.16).

Ще однією важливою дією для покращення результатів навчання є вивід результату функції втрат. За допомогою цього можна відслідковувати якість навчання на різних епохах, і змінювати параметри при надто великих, або надто малих значень функції втрат.

Для того, щоб кожен при запуску бота не починати навчання заново модель можна зберегти. Щоб це зробити необхідно створити Python-словник, в який необхідно записати стани всіх параметрів моделі, розмір вхідного шару, розмір схованого шару, розмір вихідного шару, масив усіх слів та масив тегів. Стани параметрів моделі можна отримати використавши метод «state_dict», що повертає словник в якому ключами є значення параметрів а значеннями - їх поточні значення. Після створення словнику його необхідно зберегти за допомогою методу «torch.save», вказавши в якості параметрів створений словник а також шлях збереження файлу. По стандарту файл зберігають із розширенням «.pt» або «.pth», проте за бажанням можна вказати будь-яке інше. Після збереження модель можна легко завантажити за допомогою методу «torch.load», вказавши при цьому шлях до збереженої моделі, а потім відновити стани всіх параметрів моделі за допомогою методу «load_state_dict», та в якості параметру вказати збережений словник станів моделі. При використанні створеної моделі дуже важливо вказати, що в даний момент модель не повинна навчатися, адже якщо це не зробити, то модель може зіпсуватися. Це можна зробити за допомогою методу «eval» екземпляру класу моделі. Якщо ж завантажена модель використовуватиметься для донавчання, то необхідно використати метод «train».

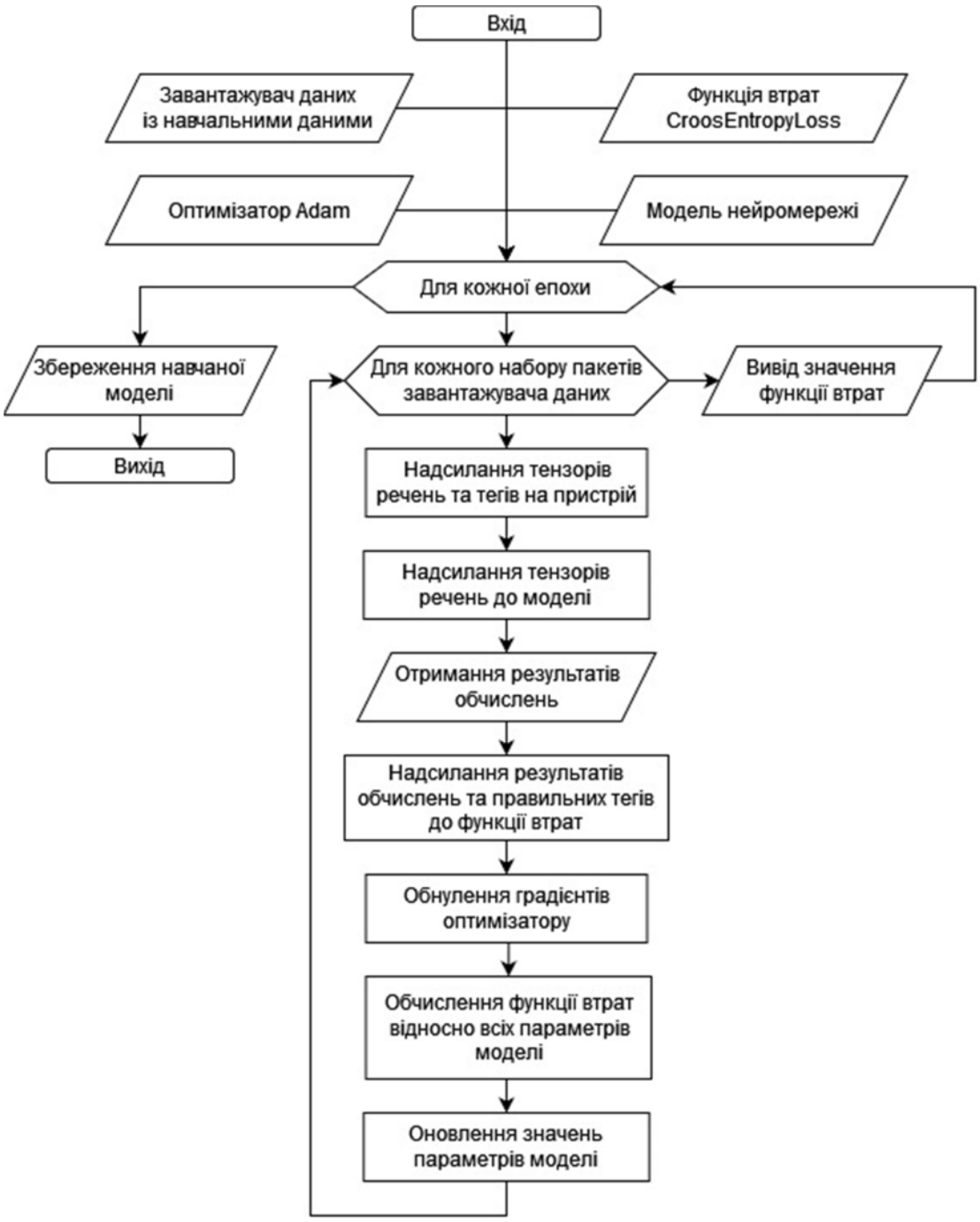



Рисунок 3.16 – Навчальний цикл моделі нейромережі

Отже, в результаті роботи створено цикл навчання для моделі нейромережі використовуючи завантажувач даних, модель, оптимізатор і функцію втрат. Також додано функціонал для збереження навченої моделі.

### 3.7 Створення діскорд боту

Як вже вказано в розділі 2.4, для зв’язку із соціальною мережею Діскорд використовуватиметься бібліотека Disnake. Створення та налаштування бота в даній бібліотеці виконується за допомогою класу «Bot» із модуля «commands». При створенні екземпляру класу бота необхідно вказати дозволи, що матиме бот. Для простоти можна вказати, що бот має всі дозволи, адже їх вже вказано при реєстрації бота на сайті Діскорд Developer Portal, і тому навіть якщо бот намагатиметься використати функціонал, що не вказаний на сайті, то замість помилки від бібліотеки, Діскорд видасть заборону на виконання цієї дії.

Після цього боту необхідно додати його функціонал перед запуском. У Disnake налаштовування функціоналу робиться за допомогою підписок на події. Наприклад можна зробити, щоб при готовності бота до роботи в терміналі з’явилося повідомлення «Бот готовий до роботи». Щоб це зробити необхідно підписати метод із написаннями повідомлення у термінал до події «on_ready». Таким чином можна підписувати різні методи до різних подій для створення функціоналу.

Найважливішою подією для боту є отримання повідомлення. Ця подія має назву «on_message». Дана подія при активації окрім самого повідомлення надає також різну корисну інформацію таку як автора повідомлення, канал повідомлення, та інше. На початку методу для отримання повідомлень необхідно створити перевірку на те, чи не є сам бот автором отриманого повідомлення, а інакше бот зациклиться, адже почне відповідати на свої ж повідомлення. Далі отримане повідомлення необхідно перетворити у вектор так само, як це зроблено при навчанні нейромережі. Наступним кроком необхідно надати моделі, що попередньо завантажена з файлу, векторизоване речення та отримати число, що відповідає тегу, за допомогою методу «torch.max». Після цього можна отримати тег в текстовому вигляді використавши результат розпізнавання в якості індексу для масиву тегів, який отримано із файлу зі збереженою моделлю. Щоб

отримати відсоток того, наскільки модель впевнена у результаті розпізнавання можна використати функцію «torch.softmax», що повертає значення вірогідностей для кожного класу таким чином, щоб усі ці значення в сумі повертали 1. Після цього важливо перевірити те, чи найвища вірогідність класу відповідає мінімальному порогу, для цього можна просто порівняти, чи вірогідність вище значення мінімального порогу, і якщо перевірка не пройшла – то метод поверне невдачу розпізнавання.

Так як створена модель лише класифікує текст, то самостійно створювати відповіді вона не зможе, тому необхідно підготовити json файл із заготовками відповідей для різних тегів. В якості ключа використовуватиметься тег, а в якості значення виступатиме масив із текстовими відповідями. Таким чином при вдалому розпізнаванні можна використати отриманий тег як ключ та отримати заготовлену відповідь для даного тегу.

Після отримання речення його можна надіслати за допомогою методу «send» каналу, з якого надійшло повідомлення, і вказавши в якості параметру повідомлення, що відправлятиметься (рис 3.17).

Підписавши цей метод до події «on_message» можна запустити бота використавши метод «run» екземпляру класу бота, і вказавши в якості параметру токен, що отримано на сайті Діскорд Developer Portal при реєстрації бота.

Для полегшення розробки бота необхідно створити ще один невеликий метод, що писатиме про помилки виконання програми бота. Щоб це зробити необхідно підписатись на подію «on_command_error», що надає контекст виклику помилки а також код помилки. Все, що робитиме підписаний метод – виводити помилку в терміналі.

В доповнення до цього, можна також створити режим відладки для бота, який виводитиме різну корисну для відладки інформації. Наприклад можна зробити так, щоб замість повідомлення, що відповідає тегу, він надсилав би сам тег, а також відсоток впевненості в цьому тегу.

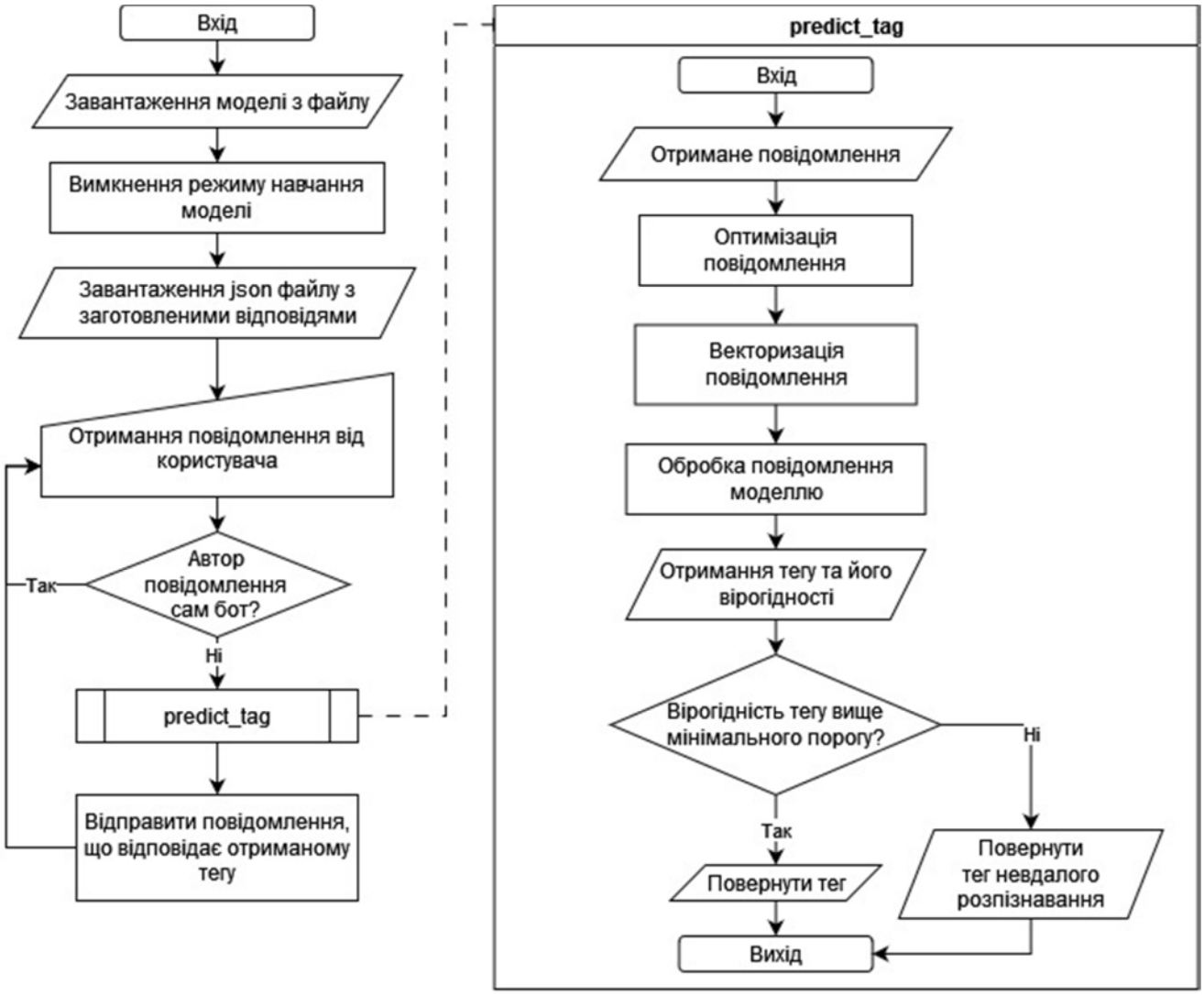



Рисунок 3.17 – Дії при запуску бота

Для зручного перемикання у режим відладки можна створити «слеш-команду» для бота, при введені якої бот перемикатиметься між режимом відладки та звичайним режимом. Для такого функціоналу в бібліотеці Disnake є метод «slash_command». В якості параметру цей метод може приймати багато різних налаштувань такі як опис команди, дозвіл на використання команди для різних учасників, та інше. Також після цього необхідно прописати метод, що викликатиметься при застосуванні команди. Назва методу використовуватиметься в якості назви команди в інтерфейсі користувача, тому метод називатиметься «debug», та в описі напишемо, що ця команда перемикає режим відладки бота. Тепер після запуску бота він синхронізує команди із

сервером Діскорду, та вони будуть відображатися при введені символу «/» разом із описом (рис. 3.18).

Отже, після роботи з бібліотекою Disnake, бота запущено у Діскорді. Також створено функціонал для нього у вигляді розпізнавання повідомлень та використання заготовленої відповіді для нього, і функціонал для відладки його роботи.


Рисунок 3.18 – Відображення «слеш-команди» для відладки у інтерфейсі користувача

### 3.8 Пошук оптимальних параметрів нейромережі

Хоча в розділі 2.3.6 і знайдено оптимальні стартові параметри, проте ці параметри можуть не одразу підійти для створеної моделі, адже на якість навчання впливає багато різних факторів. Проте слід спробувати використати ці параметри, та подивитись на результат (рис. 3.19).

```
Epoch [1/3], Loss: 2.8594
Epoch [2/3], Loss: 2.8243
Epoch [3/3], Loss: 2.6303
final loss: 2.6303
```

Рисунок 3.19 – Результати функції втрат для трьох епох при стартових параметрах

Результат 2.63 є дуже поганим для розміру поточного датасету. Дане число показує середнє значення суми логарифмів вірогідностей класів, або якщо простіше - те, наскільки далеко модель від правильної класифікації. і при такому результаті бот навіть не розпізнає звичайного привітання (рис. 3.20)

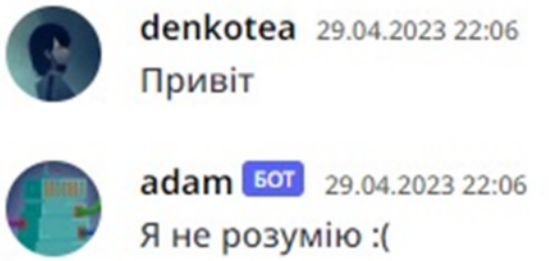



Рисунок 3.20 – Результат розпізнавання при стартових параметрах

Через невеликий розмір датасету найкращим методом покращення якості розпізнавання є підвищення кількості епох, а також зменшення розмірів пакетів, також можна трохи зменшити параметр «learning_rate» для запобігання перенавчання. Зазвичай пошук оптимальних параметрів моделі є процесом експерементування, і потребує аналізу результатів при різних наборах параметрів. Після використання різних параметрів, отримано таблицю 3.2 в якій вказано фінальну функцію втрат та параметри, які були використані для отримання цих результатів.

Таблиця 3.2 – Результати навчання при різних параметрах моделі

| № | batch_size | learning_rate | num_epochs | Фінальний результат функції втрат |
|---|---|---|---|---|
| 1 | 16 | 0.001 | 3 | 2.63 |
| 2 | 12 | 0.08 | 5 | 0.97 |
| 3 | 10 | 0.08 | 8 | 2.28 |
| 4 | 12 | 0.08 | 8 | 0.85 |
| 5 | 12 | 0.05 | 10 | 0.0023 |
| 6 | 8 | 0.01 | 20 | 0.19 |

Хоча результат функції втрат при наборі параметрів №5 і є найменшим, проте він вже є занадто малим, що може свідчити про перенавчання моделі, тому

набір параметрів №6 є найкращим. І тепер якщо привітатись з ботом, то він привітається у відповідь (рис. 3. 21).

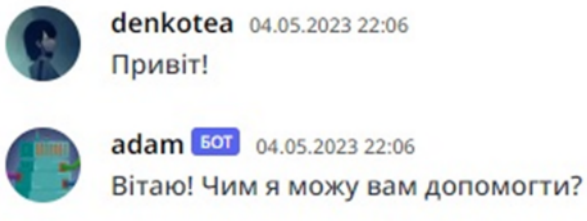



Рисунок 3.21 – Вітання боту

Отже, в результаті пошуку оптимальних параметрів, обрано наступні параметри: batch_size = 8, learning_rate = 0.01 та num_epochs = 20.

### 3.9 Тестування моделі

Після створення датасету, моделі, налаштування її параметрів та запуску бота у діскорді, його можна протестувати. Бот навчений на допомогу користувачеві у покупці хостингу, тому для першого тестування використовуться найвірогідніший шлях користувача, який розуміє які можливості зазвичай є в ботів, тобто використовуватимуться такі формульовки, в яких невелика кількість слів, але достатня для того, щоб бот повністю зрозумів контекст. Приблизно на таких повідомленнях і навчений бот, тому тест повинен пройти успішно (рис. 3.22).

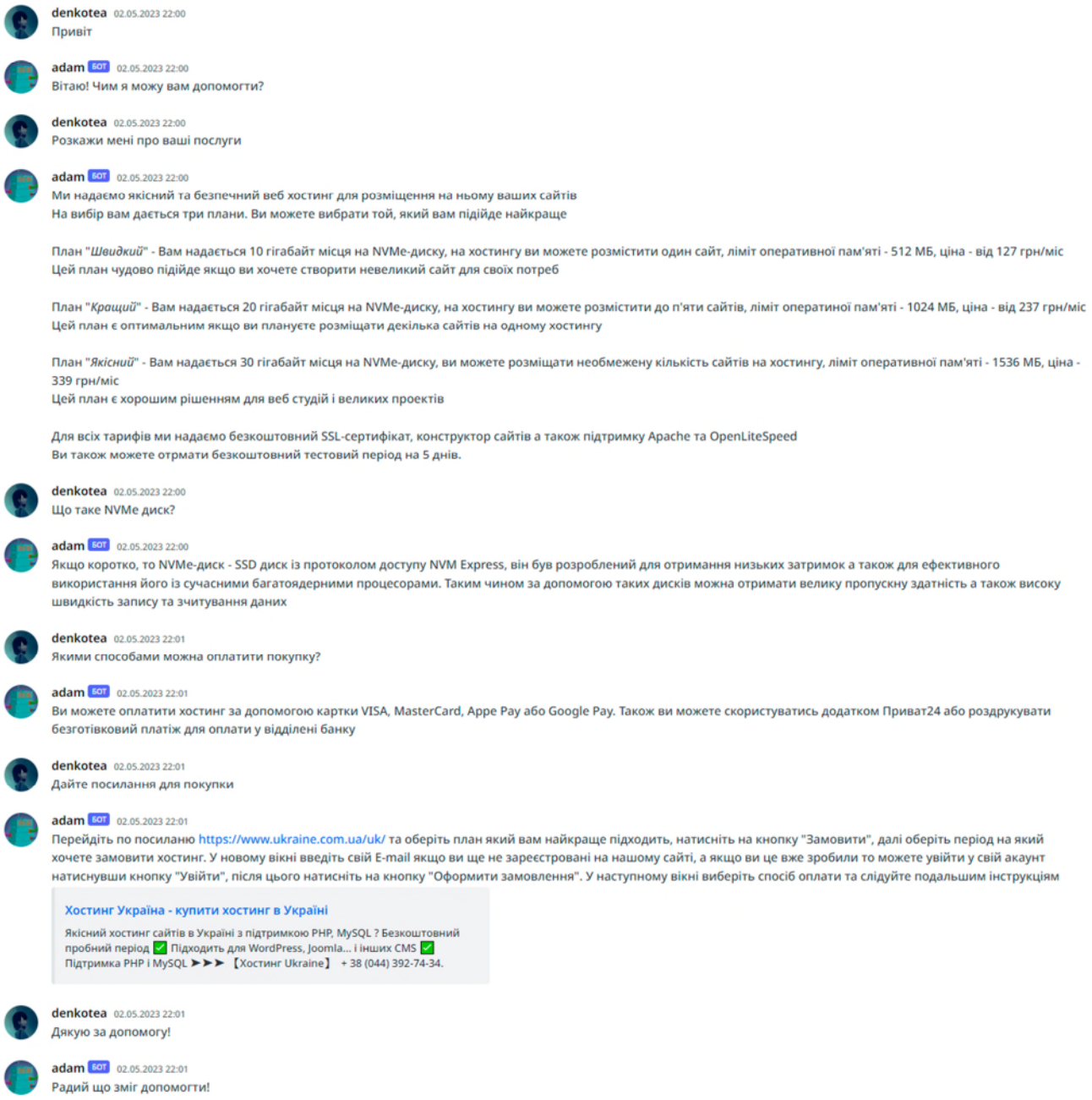



Рисунок 3.22 – Розмова з ботом із використанням простих формульовок

Як видно з малюнку - все пройшло добре і бот з легкістю розповів користувачеві усе, про що він питав. Можна провести ще один тест використовуючи взагалі прості формульовки. Повідомлення складатиметься з одного або двох слів, і в деяких випадках може виникнути проблема нестачі контексту, тому при тестуванні можуть виникнути деякі проблеми (рис. 3.23).

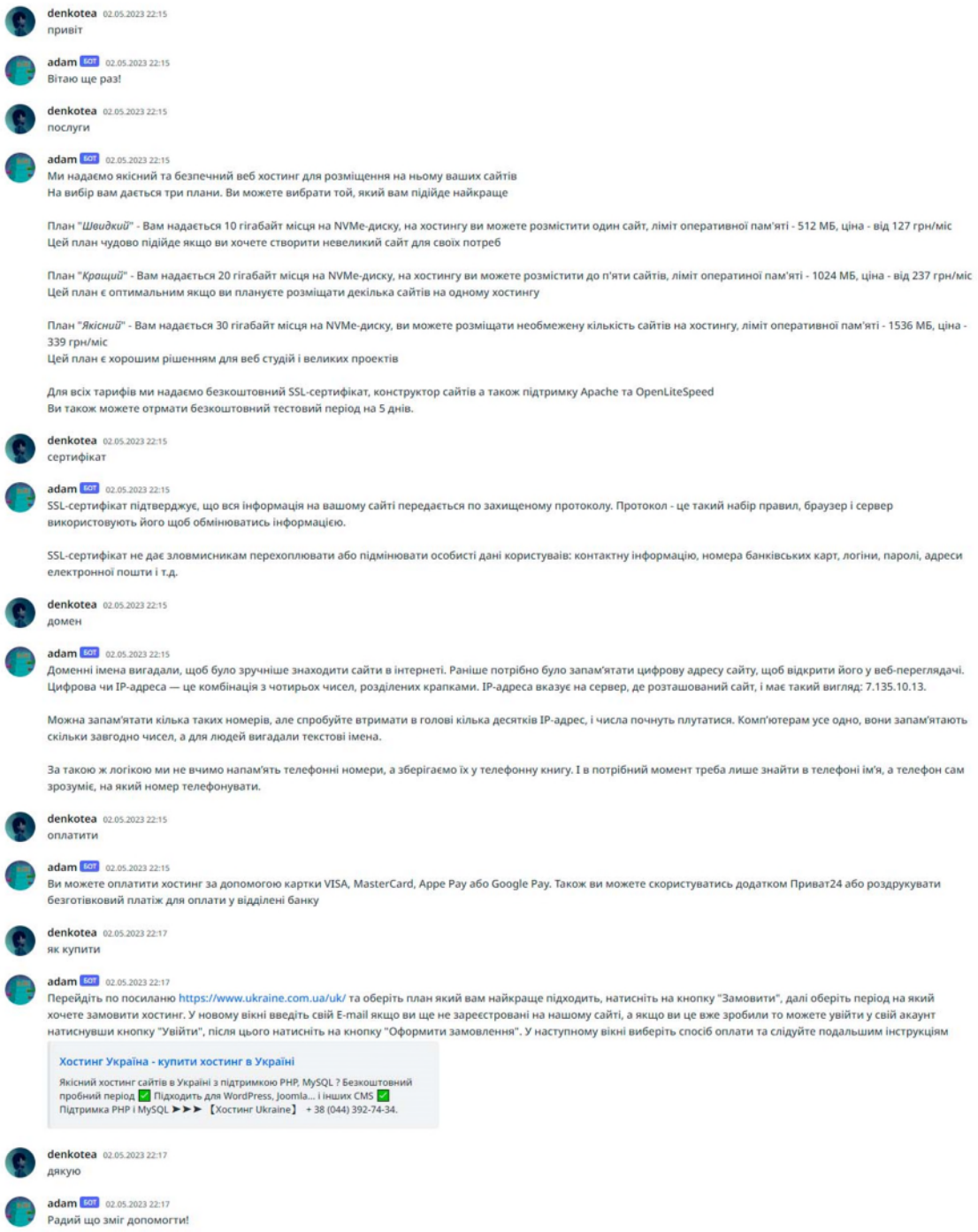



Рисунок 2.23 – Розмова з ботом із використанням дуже простих формульовок

Даний тест також пройшов доволі успішно, лише під повідомленням «оплатити» могло матись на увазі запитання про те, у який спосіб можна придбати хостинг, але через недостачу контексту це неможливо зрозуміти. Тепер проведемо останній тест в якому використовуються складні формульовки (рис. 2.24).

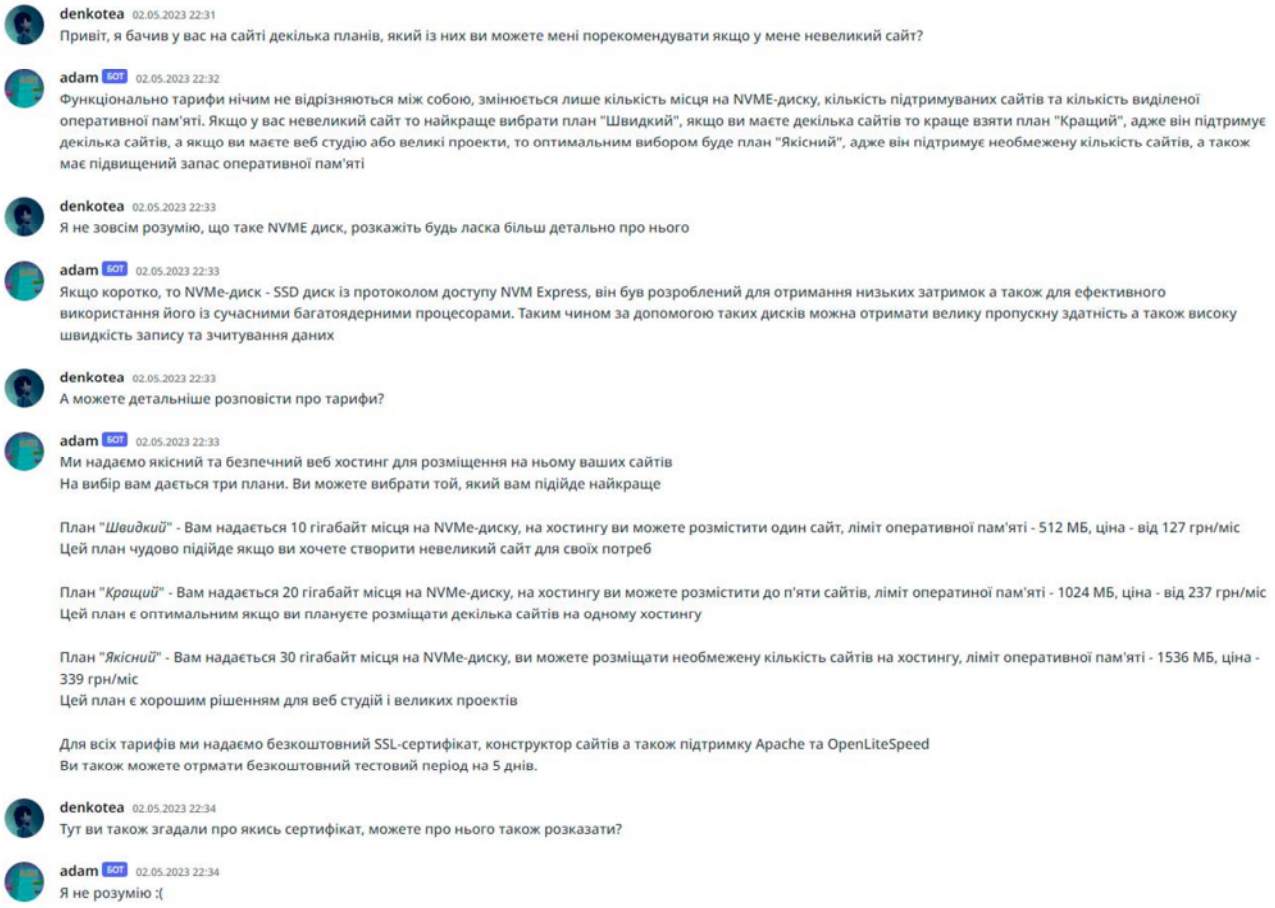



Рисунок 2.24 - Розмова з ботом із використанням складних формульовок

Спочатку бот правильно відповідав на всі запитання, проте в кінці не зміг зрозуміти через велику кількість шуму, проте якщо використати те саме речення, але додати до нього «SSL», то бот вже зможе розпізнати це повідомлення (рис. 2.25). Скоріше за все це зв’язано з тим, що модель запам’ятала саме зв’язку слів «SSL сертифікат», бо в навчальній вибірці у всіх прикладах ці слова завжди використовуються одночасно, і тому коли ці слова використовуються окремо, то боту не вдається правильно класифікувати повідомлення. Якщо використати те саме речення, але написати лише «SSL» замість «сертифікат», то бот також не зможе це розпізнати (рис. 2.26).

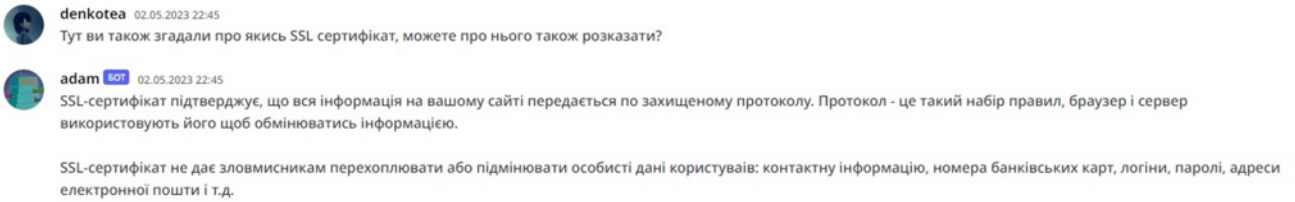



Рисунок 2.25 – Додавання «SSL» до речення, що не вдалось розпізнати

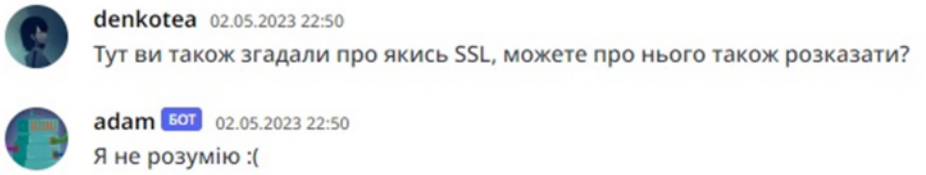



Рисунок 2.26 – Заміна слова «сертифікат» на «SSL»

Дану ситуацію можна вирішити за допомогою додавання прикладів, в яких ці слова використовуються окремо один від одного.

Отже, після тестування моделі, виявлено достатньо високу точність розпізнавання повідомлень різної складності.

### 3.10 Вимоги до розгортання системи

Вимоги для запуску бота у Діскорд:

– Встановлений Python версії 3.8 і вище

– Встановлена бібліотека Disnake версії 2.7 і вище

– Встановлена бібліотека PyTorch з підтримкою встановленої версії Python

– Встановлена бібліотека NLTK Utils версії 3.8.1 і вище

– Встановлена бібліотека pymorphy2 останньої версії

– Встановлена бібліотека UkStemmer

– Наявність середовища для запуску python файлів

## Висновки

Результатом роботи над кваліфікаційною роботою бакалавра став бот на основі нейромережі для соціальної мережі Діскорд, що вміє класифікувати повідомлення від користувача та коректно відповідати на них. Для цього проведено дослідження різних способів створення штучного інтелекту для чат-бота в результаті чого знайдена та в подальшому використана бібліотека PyTorch. Для оптимізації навчання та подальшого розпізнавання, підготовлені для навчання дані лематизувались за допомогою бібліотеки pymorphy2 та простемувались за допомогою модифікації бібліотеки Кирила Захарова. За допомогою методу «мішок слів» створено функцію для векторизації повідомлень. Використовуючи влаштовані інструменти PyTorch створено DataLoader для оптимізації завантаження даних до нейромережі, обрано оптимізатор для зміни ваг нейромережі та функцію втрат для оцінки якості навчання. Саму модель створено за допомогою наслідування від класу nn.Module бібліотеки PyTorch, Отриману нейромережу додано в якості функціоналу бота для Діскорда, за допомогою бібліотеки Disnake, що надає спрощений доступ до Діскорд API, з’єднує програму із ботом, а також має влаштовані події для різних ситуацій. Під час тестувань отриманий бот видавав достатньо хороші результати, і за потреби може дуже легко розширюватись для відповіді на більшу кількість запитань.

## Перелік посилань


1. Що таке чат-бот - https://creativesmm.com.ua/shho-take-chatbot-ta-komu-vonu-potribni/
2. Про першого чат-бота ELIZA - https://web.njit.edu/~ronkowit/eliza.html
3. Проблеми та розвиток чат-ботів https://www.ibm.com/topics/chatbots
4. Для чого використовуються чат-боти - https://gerabot.com/article/sho_take_chatbot_riznovidi_chatbotiv_dlya_biznesu
5. Порівняння бібліотек PyTorch та Tensorflow - https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/
6. Статистика використань PyTorch та TensorFlow - https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/
7. Про бібліотеку PyTorch - https://pytorch.org/
8. Про модулі бібліотеки PyTorch - https://manalelaidouni.github.io/Pytorch%20guide%20101.html
9. Список найпопулярніших Діскорд ботів - https://top.gg/list/top
10. Інформація про Діскорд бота Clarity - top.gg/bot/847238999235493909
11. Інформація про Діскорд бота Cleverbot - top.gg/bot/712635007981846601
12. Інформація про Діскорд бота Wallu - wallubot.com
13. Інформація про Діскорд бота Midjourney - https://www.midjourney.com
14. Про формат JSON - https://www.json.org/json-uk.html
15. Про шум при створені датасету - https://www.madrasresearch.org/post/noise-in-a-dataset
16. Методи векторизації слів - https://neptune.ai/blog/vectorization-techniques-in-nlp-guide

17. Інформація про Bag of words - https://machinelearningmastery.com/gentle-introduction-bag-words-model/
18. Інформація про TF-IDF - https://monkeylearn.com/blog/what-is-tf-idf/
19. Інформація про Word2Vec - https://towardsdatascience.com/word2vec-explained-49c52b4ccb71
20. Нормалізація тексту - https://towardsdatascience.com/text-normalization-7ecc8e084e31
21. Сайт бібліотеки pymorphy2 - https://pymorphy2.readthedocs.io/en/stable/
22. Про нейронні мережі прямого поширення - https://towardsdatascience.com/forward-propagation-in-neural-networks-simplified-math-and-code-version-bbcfef6f9250
23. Про рекурентні нейронні мережі - https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/
24. Про переваги функції активації ReLU - https://builtin.com/machine-learning/relu-activation-function
25. Порівняння швидкості роботи PyTorch з CUDA, CPU Та NumPy - https://arxiv.org/pdf/1805.07439.pdf

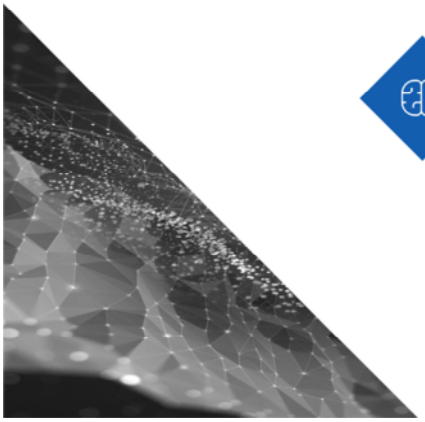
# ДОДАТКИ

# Додаток А

## Слайди презентації


# Завдання

## Підготовка даних
Підготовка та форматування навчальних даних для тренування моделі



## Підготовка моделі
Створення моделі нейромережі та її навчання за допомогою підготовлених даних

## Підключення до Діскорду
Підключення чат-боту до Діскорду з використанням Діскорд API

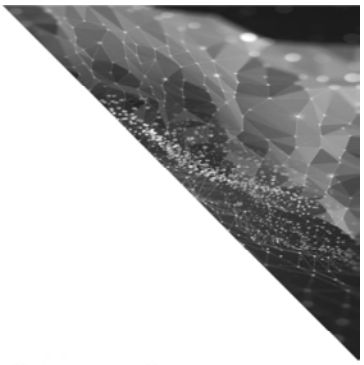
## Підготовка даних
Зберігання даних

Щоб навчити нейромережу необхідно подавати їй набір даних у форматі «зразок - визначення».
Для збереження даних у такому вигляді може допомогти текстовий формат обміну даних JSON, так як він має саме таку структуру яка необхідна, а саме «ключ - значення».

JSON файл

```
{
  "tag": "greeting",
  "patterns": [
    "Привіт",
    "Хей",
    "Як ти",
    "Тут хтось є",
  ]
},
{
  "tag": "goodbye",
  "patterns": [
    "Прощавай",
    "Прощай",
    "Допобачення",
    "Бувай",
  ]
},
{
  "tag": "thanks",
  "patterns": [
    "Дякую",
    "Дякую тобі",
    "Дякую вам",
    "Дякую за допомогу",
  ]
},
```

## Підготовка даних
Нормалізація слів

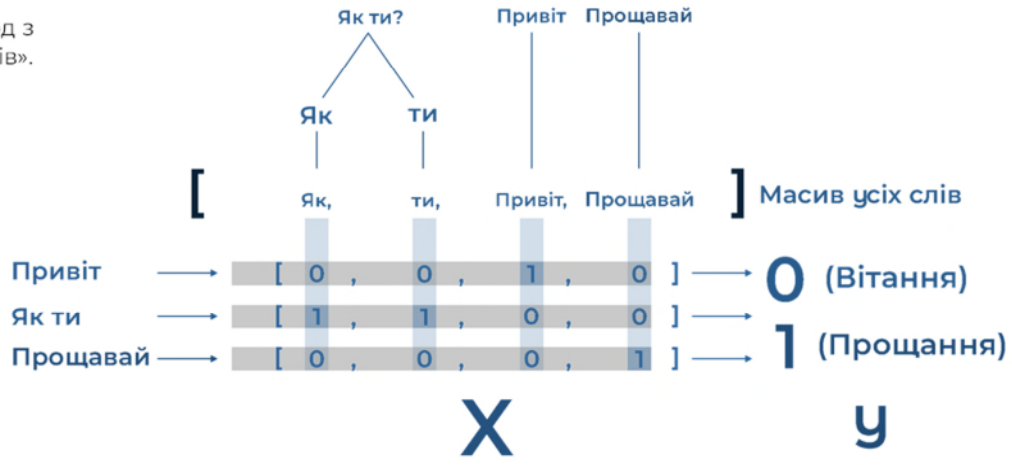


Для покращення якості навчання, а в подальшому і розпізнавання тексту, його можна нормалізувати. Щоб досягти цього можна звести весь текст до нижнього регістру, видалити пунктуаційні знаки, видалити стоп-слова. Також можна скористатися лематизацією та стемінгом.


## Підготовка даних
Векторизація слів

PyTorch не вміє працювати з текстом а лише з цифрами, тому необхідно якимось чином перетворити набір тексту у цифри. Цей процес називається векторизацією.

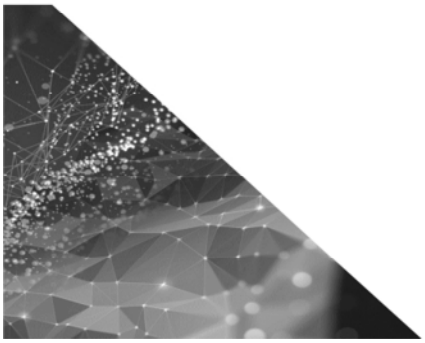
Для цього можна використати метод з назвою «мішок слів».

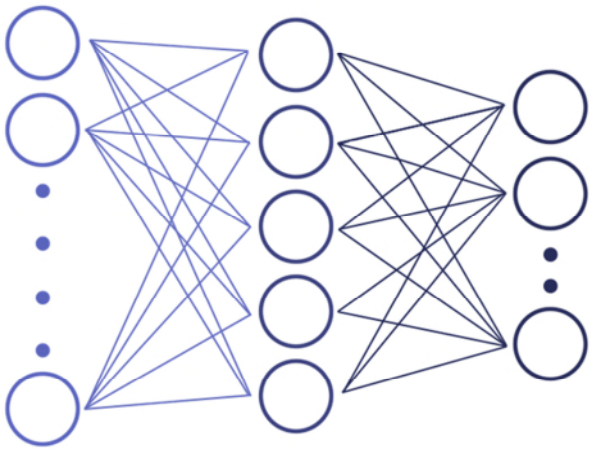


## Підготовка моделі

Створення моделі

Тепер отримавши цифрові масиви необхідно створити нейромережу, яка оброблятиме ці дані. Для цього використовуватиметься бібліотека PyTorch.






## Підключення до Діскорду

Етапи створення та підключення діскорд бота


Діскорд Developer Portal

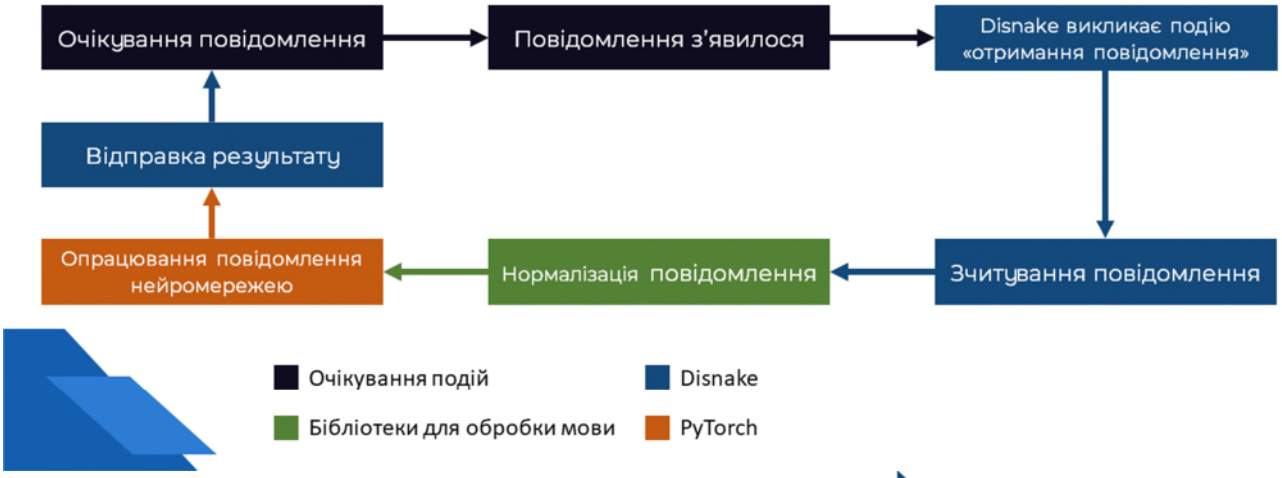
Для того щоб запустити свого бота у Діскорді його необхідно зареєструвати та налаштувати на сайті Діскорд Developer Portal

Disnake

Для створення функціоналу використовуватиметься бібліотека Disnake

## Підключення до Діскорду
Бібліотека Disnake

## Цикл Діскорд бота




## Висновок

- ✓ Проведено дослідження способів створення ШІ для чат бота
- ✓ Створено алгоритми для оптимізації повідомлень
- ✓ Використано метод «мішок слів» для векторизації
- ✓ Створено модель нейромережі за допомогою PyTorch
- ✓ Підключено бота до Діскорду за допомогою Disnake
- ✓ Навчено отриманого чат-боту та проведено його тестування

## Додаток Б

## Лістинг файлу train.py

```
import json
from nltk_utils import tokenize, stem, bag_of_words, lemmatization

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

from model import Neuro

with open('intents.json', 'r', encoding='utf-8') as f:
    intents = json.load(f)

all_words = []
tags = []
xy = []

for intent in intents['intents']:
    tag = intent['tag']
    tags.append(tag)
    for pattern in intent['patterns']:
        optimized_sentence = tokenize(pattern)
        optimized_sentence = [lemmatization(a) for a in optimized_sentence]
        optimized_sentence = [stem(a) for a in optimized_sentence]
        all_words.extend(optimized_sentence)
        xy.append((optimized_sentence, tag))

all_words = sorted(set(all_words))
tags = sorted(set(tags))

X_train = []
y_train = []

for(snt, tag) in xy:
    X_train.append(bag_of_words(snt, all_words))
    y_train.append(tags.index(tag))

class ChatDataset(Dataset):
    def __init__(self):
        self.n_samples = len(X_train)
        self.x_data = X_train
        self.y_data = y_train

    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    def __len__(self):
```

```
        return self.n_samples

output_size = len(tags)
input_size = len(all_words)

batch_size = 8
hidden_size = 18
learning_rate = 0.01
num_epochs = 20

weight_decay = 0.001

dataset = ChatDataset()
train_loader = DataLoader(dataset=dataset, batch_size=batch_size, shuffle=True,
num_workers=0)

print(f"CUDA is available: {torch.cuda.is_available()}")

if(torch.cuda.is_available()):
    device = torch.device('cuda')
else:
    device = torch.device('cpu')
model = Neuro(input_size, hidden_size, output_size).to(device)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate,
weight_decay=weight_decay)

print("Begin")
for epoch in range(num_epochs):
    for (words, labels) in train_loader:
        words = words.to(device)
        labels = labels.to(dtype=torch.long).to(device)

        outputs = model(words)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print (f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')


print(f'final loss: {loss.item():.4f}')

data = {
    "model_state": model.state_dict(),
    "input_size": input_size,
    "output_size": output_size,
```

```
    "hidden_size": hidden_size,
    "all_words": all_words,
    "tags": tags
}

FILE = "data.pth"
torch.save(data, FILE)
print(f'Saved to {FILE}')
```

**Лістинг файлу chat.py**

```
import random
import json
import torch
from model import Neuro
from nltk_utils import bag_of_words, tokenize, stem, lemmatization
from bot_token import token

import disnake
from disnake.ext import commands

bot = commands.Bot(command_prefix="!", help_command=None,
intents=disnake.Intents.all(), test_guilds=[1055839193424732240])

device = torch.device('cpu')

with open('replies.json', 'r', encoding='utf-8') as f:
    replies = json.load(f)
# Chat settings ############################

FILE = "data.pth"
data = torch.load(FILE)
global debugMode
debugMode = False

# Model settings ############################

input_size = data["input_size"]
hidden_size = data["hidden_size"]
output_size = data["output_size"]
all_words = data["all_words"]
tags = data["tags"]
model_state = data["model_state"]

############################

model = Neuro(input_size, hidden_size, output_size).to(device)
model.load_state_dict(model_state)
model.eval()

@bot.event
```

```
async def on_ready():
    global debugMode
    debugMode = False
    print(f"Bot {bot.user} is here")

@bot.event
async def on_command_error(ctx, error):
    print(f"{error}\nContext:\n{ctx}")

@bot.event
async def on_message(message):
    await bot.process_commands(message)
    if message.author == bot.user: return
    await give_reaction(predict(message.content), message.channel)

@bot.slash_command(description="Перемикає режим відладки бота")
async def debug(inter):
    global debugMode
    debugMode = not debugMode
    if(debugMode):
        await inter.response.send_message("Включаю режим дебага")
    else:
        await inter.response.send_message("Виключаю режим дебага")

async def send_message(message, channel):
    await channel.send(message)

async def give_reaction(message, channel):
    global debugMode
    if message == "debug":
        debugMode = not debugMode
        if(debugMode):
            await send_message("Включаю режим дебага", channel)
        else:
            await send_message("Виключаю режим дебага", channel)
    elif debugMode:
        await send_message(message, channel)
    elif message == "Я не розумію :(":
        await send_message("Я не розумію :(", channel)
    else:
        await send_message_fromList(channel, message)


async def send_message_fromList(channel, tag):
    await send_message(random.choice(replies[tag]), channel)

def predict(sentence):
    global debugMode
    sentence = tokenize(sentence)
    sentence = [lemmatization(w) for w in sentence]
```

```
    sentence = [stem(w) for w in sentence]
    X = bag_of_words(sentence, all_words, debug=True)
    X = X.reshape(1, X.shape[0])
    X = torch.from_numpy(X)

    output = model(X)
    _, predicted = torch.max(output, dim=1)
    tag = tags[predicted.item()]

    probs = torch.softmax(output, dim=1)
    prob = probs[0][predicted.item()]

    if debugMode and tag != "debug":
        return f"{tag} - {round(prob.item() * 100, 2)}%"

    if prob.item() > 0.85:
        return(tag)
    else:
        return "Я не розумію :("

bot.run(token)
```

## Лістинг файлу model.py

```
import torch
import torch.nn as nn

class Neuro(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(Neuro, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_size)
        self.l2 = nn.Linear(hidden_size, hidden_size)
        self.l3 = nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        out = self.l1(x)
        out = self.relu(out)
        out = self.l2(out)
        out = self.relu(out)
        out = self.l3(out)

        return out
```

Mon May 29 16:34:31 EEST 2023, Петровський Сергій Степанович, Хмельницький національний університет, ХНУ

# Anti-Plagiarism v-15.257

**Максимальне співпадіння з одним документом 1.0%**

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 12%**

| | Документ | | Сумарний збіг по Базі Даних | |
|---|---|---|---|---|
| ID: 114244<br>Назва: КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА<br>Додано в БД: 2023-05-29<br>Автора: Д.О. Книш<br>Керівники: Р.О. Багрій<br>Консультанти:<br>Опоненти: | Символи | Лексеми | Символи | Лексеми |
| | 58462 | 891 | 1351 (2%) | 21 (2%) |

Джерело плагіату

| ID | Опис | Наявність плагіату в документі | |
|---|---|---|---|
| | | Символи | Лексеми |

UNICHECK

Ім'я користувача:
**Кафедра КН**

ID перевірки:
**1015302486**

Дата перевірки:
**29.05.2023 17:29:39 EEST**

Тип перевірки:
**Doc vs Internet + Library**

Дата звіту:
**29.05.2023 17:32:05 EEST**

ID користувача:
**100005671**

---

Назва документа: **КН-19-1 Книш**

Кількість сторінок: **63** Кількість слів: **9660** Кількість символів: **70223** Розмір файлу: **2.56 MB** ID файлу: **1014974109**

---

**Виявлено модифікації тексту (можуть впливати на відсоток схожості)**

## 3.52%
## Схожість

Найбільша схожість: **1.92%** з джерелом з Бібліотеки (ID файлу: **1011209601**)

| | | |
|---|---|---|
| 3.26% Джерела з Інтернету | 218 | Сторінка 65 |
| 2.43% Джерела з Бібліотеки | 101 | Сторінка 67 |

## 0% Цитат

**Вилучення цитат вимкнене**

**Вилучення списку бібліографічних посилань вимкнене**

## 0%
## Вилучень

**Немає вилучених джерел**

## Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

| | |
|---|---|
| Замінені символи | 6 |
| Підозріле форматування | 19 сторінок |

**РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ КАФЕДРИ КОМП'ЮТЕРНИХ НАУК**
**ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ**

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: *Технологія реалізації Діскорд чат-бота з використанням нейронної мережі за допомогою бібліотеки PyTorch*

Автор: *студент гр. КН-19-1 Книш Денис Олександрович*

Спеціальність: *122 – Комп'ютерні науки*

Освітня програма: *освітньо-професійна*

Науковий керівник: *к.т.н., доц. Багрій Р.О.*

Після аналізу звіту подібності зроблено такий висновок:

| № | Висновок | Позначка про відповідність |
|---|---|---|
| 1 | Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту. | ***відповідає*** |
| 2 | Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи | |
| 3 | Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат. | |
| 4 | Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту. | |

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) за програмою Anti-Plagiarism виявлені 1% є фрагментарними – містять поширені конструкції, загальновідомі терміни, скорочення та визначення.
2) За програмою UNICHECK виявлені 3.52%, що є запозиченнями, які розміщені в розділах аналізу існуючих технологій та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 1% і 3.52% відповідно, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Руслан Багрій

Гарант ОП _____ Олександр Мазурець

Завідувач кафедри КН _____ Олександр Бармак

**ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**МОН УКРАЇНИ**

**Кафедра комп'ютерних наук**

# ВІДГУК НАУКОВОГО КЕРІВНИКА

## на кваліфікаційну роботу бакалавра

студента *гр. КН-19-1 Книша Дениса Олександровича*

за темою *Технологія реалізації Діскорд чат-бота з використанням нейронної мережі за допомогою бібліотеки PyTorch*

**1. Актуальність теми**

*Актуальність теми достатньо обґрунтована, оскільки технологія реалізації Діскорд чат-бота з використанням нейронної мережі дозволяє імітувати розмови та виконувати прості автоматичні завдання – відповідати на запитання користувачів та надавати необхідну інформацію. Особливістю теми є використання нейронної мережі та бібліотеки PyTorch, що дозволяє створити більш складні та гнучкі моделі чат-ботів. Ця технологія може бути корисною в різних сферах, включаючи онлайн-комерцію, технічну підтримку, освіту та розваги.*

**2. Відповідність роботи предметній області Стандарту спеціальності 122 Комп'ютерні науки**

*Теми кваліфікаційної роботи "Технологія реалізації Діскорд чат-бота з використанням нейронної мережі за допомогою бібліотеки PyTorch" відповідає предметній області спеціальності 122 Комп'ютерні науки та вимогам до кваліфікаційної роботи бакалавра, оскільки результатом роботи є створення чат-боту до соціальної мережі Діскорд, що розпізнає запитання та дає на них відповіді з використанням штучної нейронної мережі на основі бібліотеки PyTorch. При вирішенні поставленої задачі використано методи збору та аналізу інформації, нейронні мережі, фреймворки та бібліотеки для розробки чат-ботів з використанням нейронної мережі.*

**3. Професійні та особистісні якості бакалавра**

*Книш Д. О. під час роботи над кваліфікаційною роботою бакалавра продемонстрував розуміння теорії та практичних аспектів використання нейронних мереж для створення інтелектуальних систем, зокрема чат-ботів.*

**4. Ступінь самостійності під час виконання кваліфікаційної роботи**

*Робота виконана самостійно, академічного плагіату не виявлено, стосовно всіх запозичень наведено відповідні посилання на джерела.*

**5. Ступінь оволодіння методами дослідження**

*При реалізації кваліфікаційної роботи показав достатній рівень компетентності та володіння необхідними інструментами, обладнанням, методами, методиками та технологіями предметної області комп'ютерних наук.*

**6. Повнота та якість розкриття теми роботи**

*Тема роботи повністю розкрита, виконані усі поставлені задачі та розроблено програмне забезпечення для підтвердження запропонованої способу.*

**7. Логічність, послідовність, аргументованість, літературна грамотність викладення матеріалу**

*Викладення матеріалу логічне, послідовне та аргументоване. Мова і стиль викладення кваліфікаційної роботи відповідають стандартам, що забезпечує доступність сприймання матеріалу і відповідає вимогам до сучасних наукових робіт.*

**8. Можливість практичного застосування кваліфікаційної роботи бакалавра, окремих її частин**

*Розроблений у роботі Діскорд чат-бот може бути використаний у різних сферах, включаючи онлайн-комерцію, технічну підтримку, освіту та розваги.*

**9. Висновок про можливість допуску кваліфікаційної роботи бакалавра до захисту, на яку оцінку заслуговує робота**

*Враховуючи високий рівень виконання та забезпечення усіх необхідних вимог, робота може бути допущена до захисту. Рекомендована оцінка «**відмінно**» .*

Керівник _____ *к.т.н., доц. Руслан Багрій*

**ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**МОН УКРАЇНИ**

**Кафедра комп'ютерних наук**

# РЕЦЕНЗІЯ

## на кваліфікаційну роботу бакалавра

студента *гр. КН-19-1 Книш Денис Олександрович*
за темою: *Технологія реалізації Діскорд чат-бота з використанням нейронної мережі за допомогою бібліотеки PyTorch*

1. Актуальність обраної теми

*Тема кваліфікаційна роботи бакалавра є дуже актуальною на сьогоднішній день, адже Діскорд є однією із найпопулярніших платформ для спілкування, і створення чат-боту для даної платформи може допомогти автоматизувати деякі завдання такі як обробка запитів користувачів, комерційна діяльність, відповіді на запитання та інше. Використана для цього бібліотека PyTorch надає для цього потужні інструменти, а також є найпопулярнішою бібліотекою для глибинного навчання на даний момент, що показує актуальність використаних технологій кваліфікаційної роботи.*

2. Повнота розкриття мети та завдань роботи

*Мета кваліфікаційної роботи полягала у аналізі технологій розробки чат-ботів з використанням нейронної мережі, а також у подальшому створені такого чат-боту на основі проаналізованих даних. Завданнями роботи є підготовка даних для навчання нейромережі, створення та навчання моделі штучної нейронної мережі а також підключення отриманої моделі в якості чат-боту до соціальної мережі Діскорд. Мета та завдання повністю розкривають тему кваліфікаційної роботи бакалавра*

3. Зміст кожного розділу роботи

*У першому розділі кваліфікаційної роботи бакалавра розписано характеристику предметної області а також постановку задачі. У ньому проведено аналіз предметної області, бібліотек, що можуть використовуватись при виконанні роботи, а також переглянуто інші програмні засоби предметної області. Другий розділ детально описує кроки, які необхідно зробити для досягнення поставлених задач: описано метод зберігання даних, способи векторизації та оптимізації навчальних прикладів, метод створення та налаштування моделі, а також технології, за допомогою яких можна зв'язати модель із соціальною мережею Діскорд. У третьому розділі розказано, як використати теоретичну частину з другого розділу на практиці: описано класи, які необхідно використовувати, специфіку їх роботи, зображено блок-схеми, що показують як працюють різні модулі програми. У кінці проведено тестування отриманої моделі, яке показало достатньо хороші результати*

4. Оцінка розробленої інформаційної системи, її практична цінність

*Розроблену інформаційну систему можна використовувати сфері обслуговування клієнтів. Її можна навчити розпізнавати і відповідати на повідомлення різної тематики, що підвищить якість обслуговування кожного окремого клієнта, адже тепер він одразу зможе задати питання та отримати на нього відповідь замість того, щоб шукати необхідний пункт в ієрархії розділів.*

5. Якість оформлення кваліфікаційної роботи бакалавра

*Кваліфікаційна робота бакалавра відповідає стандартам. Матеріал викладено доступно, але при цьому деталізовано. В роботі є достатньо велика кількість ілюстрацій, що полегшують сприйняття інформації. При порівняннях в наглядній формі наведено переваги та недоліки аналізованих об'єктів.*

6. Недоліки кваліфікаційної роботи бакалавра

*Через невеликий розмір датасету використовується не найсучасніший метод векторизації, який враховує семантичний зміст слів.*

7. Загальний висновок (допускається чи не допускається до захисту), та оцінка на яку заслуговує кваліфікаційна робота.

*Враховуючи рівень виконання та забезпечення усіх необхідних вимог, робота може бути допущена до захисту. Рекомендована оцінка «* відмінно *».*

Рецензент к.т.н., доцент каф. АКІТпаР
Л. Корецька