Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп’ютерних систем та мереж

Пояснювальна записка
до кваліфікаційної роботи бакалавра
за спеціальністю 123 «Комп’ютерна інженерія»

на тему: AI АСИСТЕНТ ДЛЯ ПЕРЕКЛАДУ ТА СИНТЕЗУ МОВЛЕННЯ

Проектував _____ А. Є. Єрмаченков

Керівник роботи _____ Д. І. Кузнєцов

Нормоконтроль _____ Д. І. Кузнєцов

Завідувач кафедри _____ А. І. Купін

Кривий Ріг
2024

Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

Ступінь вищої освіти бакалавр
Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри, голова циклової комісії

_____ А. І. Купін

"____" _____ 20__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

_____
(прізвище, ім'я, по батькові)

1. Тема роботи _____
_____
_____

керівник роботи _____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "___"_____ 20__року №__

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_____
_____
_____
_____

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка: 54 сторінки, 13 рисунків, 1 таблиця, 20 вокористаних джерел, 2 додатки.

Об’єкт проєктування – AI асистент для перекладу та синтезу мовлення.

Проект складається з трьох розділів.

Перший розділ присвячений опису теми штучного інтелекту та його використання в контексті перекладу та синтезу мовлення.

У другому розділі розкриті питання вибору технологій реалізації проєкту а саме: мова програмування, система дизайну, модель штучного інтелекту і т. д.

Третій розділ присвячений моделюванню AI асистента, впровадженню технологій описаних в другому розділі, тестуванню отриманої програми та аналізу її роботи.

ШТУЧНИЙ ІНТЕЛЕКТ, АСИСТЕНТ, ПЕРЕКЛАД, СИНТЕЗ МОВДЕННЯ, AI МОДЕЛЬ.

Explanatory note: 54 pages, 13 figures, 1 table, 20 used sources, 2 appendices.

The design object is an AI assistant for translation and speech synthesis.

The project consists of three sections.

The first chapter is devoted to the description of the topic of artificial intelligence and its use in the context of translation and speech synthesis.

In the second chapter, the issues of choosing project implementation technologies are revealed, namely: programming language, design system, artificial intelligence model, etc.

The third section is devoted to modeling the AI assistant, implementing the technologies described in the second section, testing the resulting program and analyzing its operation.

ARTIFICIAL INTELLIGENCE, ASSISTANT, TRANSLATION, SPEECH SYNTHESIS, AI MODEL.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AI – artificial intelligence;
API – Application Programming Interface;
CSS – cascading style sheets;
DOM – Document Object Model;
DS – Data Science;
DSRPAI – Dartmouth Summer Research Project on Artificial Intelligence;
ES – Ecmascript;
FGCP – FortiGate Clustering Protocol;
HTML – HyperText Markup Language;
JSX – JavaScript Syntax eXtension;
RAND – research and development;
RBMD – Rule-based machine translation;
ML – machine learning;
NMT – neural machine translation;
NLP – Natural language processing;
PHP – Hypertext Preprocessor;
TTS – Text-to-speech;
UI – User Interface;
UX – User Experience;
ШІ – штучний інелект;
МП – машинний переклад.

ВСТУП

У сучасну епоху зростаючих глобальних зв'язків потреба в постійному та ефективному спілкуванні між різними мовами та культурами є важливішою, ніж будь-коли. Поява штучного інтелекту (ШІ) в мовному перекладі передбачає подолання комунікаційного розриву і відкриває нові можливості для більш плідних міжкультурних зв'язків.Цей проект досліджує глибокий вплив штучного інтелекту на мовний переклад шляхом дослідження того, як технологія штучного інтелекту змінює індустрію перекладу.

Технологічні досягнення значно покращили ефективність та стандарти перекладу мови, сприяючи контактам у всьому світі та вирішуючи проблеми, пов'язані з мовними бар'єрами та обмеженнями.

Останнім часом було досягнуто значного прогресу в галузі МП, популярності МП, необхідності розуміння великої інформації, доступної в Інтернеті кількома мовами, та високої ефективності МП у міжнародній торгівлі, зумовленої швидкодією роботи комп'ютерів внаслідок розвитку апаратних компонентів, а також крім широкої доступності одномовних та двомовних даних, AI та MP - це перекладацький бізнес, який вперше був розглянутий автором у контексті бізнесу. У зв'язку з цим вам потрібні зручні технології, здатні забезпечити якісний переклад в необхідних ситуаціях [1].

Метою цього проекту є створення Програми для перекладу з використанням штучного інтелекту, яка могла б задовольнити потреби користувачів.

AI Assistant для перекладу і синтезу мови-це інноваційний продукт, що поєднує в собі передові технології машинного навчання, обробки природної мови і технології глибоких нейронних гонок. Інструмент не тільки забезпечує автоматичний переклад тексту між мовами, але і дозволяє синтезувати мову, відтворюючи природні розмовні вирази і інтонації. Він був створений для сприяння спілкуванню між людьми з різних культурних та мовних верств, зменшення мовних бар'єрів та сприяння міжнародному співробітництву.

Одним з головних переваг AI assistant для перекладу і синтезу мови є те, що він може швидко і точно переводити Тексти різної складності. Завдяки використанню передових алгоритмів машинного навчання інструмент забезпечує високоякісний переклад при збереженні семантичної і граматичної точності вихідного тексту. Крім того, асистенти зі штучним інтелектом постійно вдосконалюють свої навички перекладу та здатність адаптуватися до потреб користувачів, постійно адаптуючись до нових мовних та культурних контекстів.

Ще однією важливою особливістю цього інструменту є здатність синтезувати мову. Штучний інтелект-Асистент може не тільки переводити текст, але і відтворювати його у вигляді мови. Це дозволяє отримати більш природне і зрозуміле рішення для спілкування. Ця функція особливо корисна при створенні аудіовізуального контенту, навчальних матеріалів та інтерактивних додатків, що вимагають голосової взаємодії з користувачами.

Програми-помічники зі штучним інтелектом для перекладу та синтезу мовлення можуть мати значний вплив на різні сфери життя, такі як бізнес, освіта, ЗМІ та Міжнародні відносини. У сучасному глобалізованому світі, де міжнародне спілкування стає невід'ємною частиною повсякденного життя, дуже важливо мати ефективні інструменти для перекладу та синтезу мови.

ТЕОРЕТИЧНА ЧАСТИНА

1.1. Історія штучного інтелекту

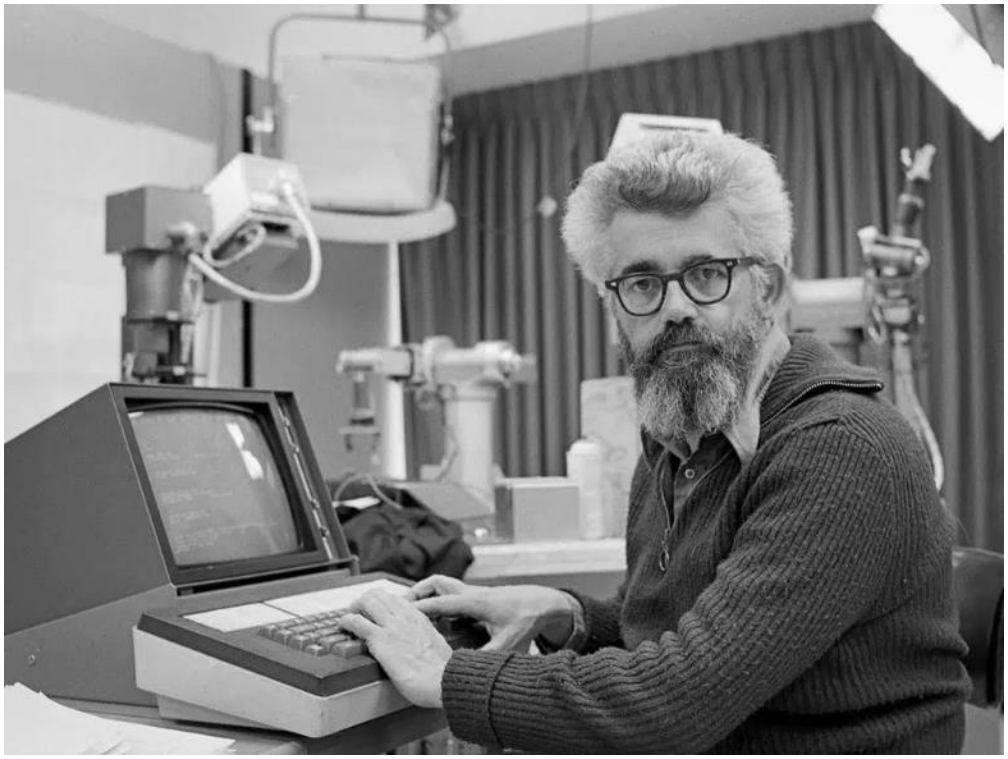

Рисунок 1.1 – Автор терміну «штучний інтелект» Джон МакКарті

У першій половині 20 століття наукова фантастика познайомила світ з концепцією роботів зі штучним інтелектом. Все почалося з «безсердечного» олов'яного чоловіка з «Чарівника країни Оз» і продовжилося людиноподібним роботом, який уособлював Марію в Метрополісі. До 1950-х років у нас було покоління вчених, математиків і філософів, у свідомості яких концепція штучного інтелекту (або ШІ) була культурно засвоєна. Одним із таких людей був Алан Тюрінг, молодий британський ерудит, який досліджував математичну можливість штучного інтелекту. Тьюрінг припустив, що люди використовують наявну інформацію, а також розум для вирішення проблем і прийняття рішень, тож чому машини не можуть робити те саме? Це була логічна структура його статті 1950 року «Обчислювальна техніка та інтелект», в якій він обговорював, як будувати розумні машини та як перевіряти їхній інтелект.

Що завадило Тьюрингу взятися за роботу саме тоді? По-перше, комп'ютери потребували фундаментальних змін. До 1949 року комп'ютерам не вистачало ключової передумови для інтелекту: вони не могли зберігати команди, а лише їх виконувати. Іншими словами, комп'ютерам можна було сказати, що робити, але вони не могли запам'ятати, що вони зробили. По-друге, обчислення були надзвичайно дорогими. На початку 1950-х років вартість оренди комп'ютера сягала 200 000 доларів на місяць. Лише престижні університети та великі технологічні компанії могли дозволити собі подолати ці незвідані води. Щоб переконати джерела фінансування в тому, що машинний інтелект варто шукати, знадобилося підтвердження концепції, а також пропаганда високопоставлених людей.

Через п'ять років доказ концепції було ініціалізовано Алленом Ньюеллом, Кліффом Шоу та Гербертом Саймоном, теоретиком логіки. Logic Theorist була програмою, розробленою для імітації людських навичок вирішення проблем і фінансувалася Корпорацією досліджень і розвитку (RAND). Багато хто вважає її першою програмою штучного інтелекту, і вона була представлена на Дартмутському літньому дослідницькому проекті зі штучного інтелекту (DSRPAI), організованому Джоном МакКарті та Марвіном Мінським у 1956 році. На цій історичній конференції Маккарті, уявляючи великі спільні зусилля, приніс об'єднати провідних дослідників із різних галузей для відкритої дискусії про штучний інтелект, термін, який він ввів під час цієї події. На жаль, конференція не виправдала очікувань Маккарті; люди приходили та йшли, як їм заманеться, і не вдалося узгодити стандартні методи для цієї галузі. Незважаючи на це, усі щиро приєдналися до того, що штучний інтелект є досяжним. Важливість цієї події не можна применшувати, оскільки вона стала каталізатором наступних двадцяти років досліджень ШІ.

З 1957 по 1974 рік ШІ процвітав. Комп'ютери могли зберігати більше інформації та стали швидшими, дешевшими та доступнішими. Алгоритми машинного навчання також покращилися, і люди стали краще знати, який алгоритм застосувати до своєї проблеми. Ранні демонстрації, такі як «Загальний розв'язник проблем» Ньюелла та Саймона та «ELIZA» Джозефа Вайзенбаума, показали перспективи щодо цілей вирішення проблем та інтерпретації розмовної мови відповідно. Ці успіхи, а також пропаганда провідних дослідників (а саме учасників DSRPAI) переконали державні установи, такі як Агентство передових оборонних дослідницьких проектів (DARPA), фінансувати дослідження ШІ в кількох установах. Уряд був особливо зацікавлений у машині, яка могла б транскрибувати та перекладати усну мову, а також високопродуктивну обробку даних. Оптимізм був високий, а очікування ще більші. У 1970 році Марвін Мінскі сказав журналу Life: «Через три-вісім років у нас буде машина із загальним інтелектом середньої людини». Однак, незважаючи на наявність основного доказу принципу, попереду ще довгий шлях до досягнення кінцевих цілей обробки природної мови, абстрактного мислення та самопізнання.

Порушення початкового туману ШІ виявило гору перешкод. Найбільшою була відсутність обчислювальної потужності, щоб зробити щось суттєве: комп'ютери просто не могли зберігати достатньо інформації або обробляти її досить швидко. Для того, щоб спілкуватися, наприклад, потрібно знати значення багатьох слів і розуміти їх у багатьох комбінаціях. Ганс Моравек, докторант Маккарті того часу, заявив, що «комп'ютери все ще були в мільйони разів надто слабкими, щоб демонструвати інтелект». У міру того, як терпіння закінчувалося, фінансування зменшувалося, і дослідження повільно розвивалися протягом десяти років.

У 1980-х роках штучний інтелект відродився завдяки двом джерелам: розширенню набору алгоритмічних інструментів і збільшенню коштів. Джон Гопфілд і Девід Румелхарт популяризували методи «глибокого навчання», які дозволяли комп'ютерам навчатися, використовуючи досвід. З іншого боку, Едвард Фейгенбаум представив експертні системи, які імітували процес прийняття рішень людиною-експертом. Програма запитувала експерта в певній галузі, як реагувати в тій чи іншій ситуації, і як тільки це було вивчено практично для кожної ситуації, неексперти могли отримати поради від цієї програми. Експертні системи широко використовувалися в промисловості. Японський уряд значно профінансував експертні системи та інші заходи, пов'язані зі штучним інтелектом, у рамках свого проекту комп'ютерів п'ятого покоління (FGCP). З 1982 по 1990 рік вони інвестували 400 мільйонів доларів, щоб революціонізувати комп'ютерну обробку, реалізувати логічне програмування та вдосконалити штучний інтелект. На жаль, більшість амбітних цілей не було досягнуто. Проте можна стверджувати, що опосередкований вплив FGCP надихнув талановите молоде покоління інженерів і вчених. Незважаючи на це, фінансування FGCP припинилося, і ШІ випав з поля уваги.

За іронією долі, за відсутності державного фінансування та суспільного галасу ШІ процвітав. Протягом 1990-х і 2000-х років було досягнуто багатьох визначних цілей штучного інтелекту. У 1997 році чинний чемпіон світу з шахів і гросмайстер Гарі Каспаров зазнав поразки від Deep Blue від IBM, комп'ютерної програми для гри в шахи. Цей широко розрекламований матч став першим випадком, коли чинний чемпіон світу з шахів програв комп'ютеру, і став величезним кроком до створення програми прийняття рішень зі штучним інтелектом. У тому ж році програмне забезпечення для розпізнавання мови, розроблене Dragon Systems, було реалізовано на Windows. Це був ще один великий крок вперед, але в напрямку усного усного перекладу. Здавалося, немає проблеми, з якою машини не могли б впоратися. Навіть людські емоції були чесною грою, про що свідчить Кісмет, робот, розроблений Синтією Брізіл, який міг розпізнавати та демонструвати емоції [2].

Наприкінці 2010-х та на початку 2020-х років компанії AGI почали надавати програми, які викликали величезний інтерес. У 2015 році AlphaGo, розроблений DeepMind, переміг чемпіона світу з гри Go. Програма вчила лише

правилам гри та сама розробляла стратегію. GPT-3 – це велика мовна модель, яка була випущена в 2020 році OpenAI і здатна генерувати високоякісний текст, схожий на людину [3]. Ці та інші програми надихнули агресивний бум ШІ, коли великі компанії почали інвестувати мільярди в дослідження ШІ. За даними «AI Impacts», близько 50 мільярдів доларів щорічно інвестувалося в «ШІ» близько 2022 року тільки в США, і близько 20% нових випускників американських докторів наук з комп'ютерних наук спеціалізувалися на «ШІ» [4]. У 2022 році в США було близько 800 000 вакансій, пов'язаних із штучним інтелектом [5].

1.2. Майбутнє штучного інтелекту

Рисунок 1.2 – Обличчя людей згенеровані штучним інтелектом

У штучного інтелекту (ШІ) світле майбутнє, але він також стикається з кількома труднощами. Прогнозується, що штучний інтелект стане все більш поширеним із розвитком технологій, революціонізуючи сектори, зокрема охорону здоров'я, банківську справу та транспорт. Ринок праці зміниться в результаті автоматизації, керованої штучним інтелектом, що потребуватиме нових посад і навичок.

Майбутнє штучного інтелекту в різних галузях життя:

1) Охорона здоров'я:

На Індію припадає 17,7% світової циркуляції, що робить її другою за величиною країною після Китаю за рівнем обороту. Усі громадяни країни не мають доступу до закладів охорони здоров'я. Це пов'язано з нестачею кваліфікованих лікарів, недостатньою інфраструктурою та іншими факторами. Деякі люди не можуть отримати доступ до лікарів або лікарень.

Навіть якщо ви не йдете до лікаря, я можу діагностувати захворювання на основі симптомів, читаючи дані з фітнес-браслета або історії хвороби людини, аналізуючи картину та пропонуючи відповідне лікування, яке можна легко замовити по мобільному телефону.

Галузь охорони здоров'я в цілому зосереджена на зборі точних і доречних даних про пацієнтів і тих, хто починає лікування. Як наслідок, штучний інтелект чудово підходить для великої кількості даних галузі охорони здоров'я. Крім того, існує кілька застосувань ШІ в галузі охорони здоров'я.

AI легко розширюється, адаптується та застосовується до багатьох бізнес-процесів. Ми можемо почати розуміти можливе використання цієї технології, коли згадаємо, що ШІ — це лише комп'ютерна програма. Завдяки своїй здатності надавати інтелектуальні дані роботам, на яких раніше їх не вистачало, ШІ використовується у величезних масштабах.

2) Освіти:

Рівень освіти, яку отримує молодь, визначає прогрес країни. Ми бачимо, що зараз на AI доступно багато курсів. Проте в майбутньому я зміню традиційну школу. Виробничі галузі більше не потребують кваліфікованих робітників, оскільки роботи та технології здебільшого замінили їх.

Освітня система має потенціал бути дуже ефективною та адаптованою до індивідуальних особливостей і здібностей людини. Це дало б шанси для яскравішого блиску, а також допомогло б учням, які мають труднощі, покращити їх. З одного боку, правильна освіта може зміцнити окремих осіб і нації, а неправильна освіта може мати згубні наслідки.

3) Фінанси:

Економічна та фінансова ситуація в будь-якій країні безпосередньо пов'язана з кількісним показником її зростання. Оскільки я маю таку велику роль практично в кожній галузі, вона має багато можливостей для покращення економічного здоров'я людини та економічного здоров'я країни. Алгоритм AI зараз використовується в управлінні фондами акцій.

Визначаючи оптимальний підхід до обробки коштів, система AI може враховувати велику кількість змінних. У світі фінансів тактики, керовані AI, мають намір порушити традиційну практику торгівлі та інвестування. Це може бути катастрофічним для організацій з управління фондами, які не можуть дозволити собі такі можливості, і це може мати великі наслідки для бізнесу, тому що вибір буде зроблено швидко та різко. Боротьба завжди буде запеклою та напруженою.

Можна очікувати, що майбутні робо-консультанти, керовані ШІ, будуть більш поширеними у фінансовому секторі. Наприклад, нове дослідження Wealthramp показує, що міленіали мають більш цілеспрямоване та технологічно орієнтоване бачення майбутнього фінансового керівництва. Третина заможних інвесторів, за даними Wealthramp, «використовують робо-консультанти та цифрові інструменти для здійснення інвестицій». Біонічне консультування – це

ще одна галузь, що розвивається, яка поєднує комп'ютерні обчислення з людською інтуїцією, щоб покращити клієнтські зв'язки ефективніше, ніж будь-яка з них може це зробити самостійно.

4) Військова галузь та кібербезпека

Військові технології за допомогою ШІ створили автономні системи зброї, які не потребують людей, що є найбезпечнішим способом покращити безпеку нації. У найближчому майбутньому ми можемо стати свідками роботи-військового, який настільки ж розумний, як солдат/командир, і здатний виконувати різноманітні завдання.

Методи за допомогою штучного інтелекту покращили б ефективність місії, а також забезпечили б найбезпечніше виконання. Елемент системи з підтримкою штучного інтелекту, який викликає невелике занепокоєння, полягає в тому, що алгоритм, який вона виконує, не є цілком зрозумілим. Ключовою проблемою тут був би з'ясовний AI, оскільки глибинні нейронні мережі ростуть швидше та продовжують розвиватися. Коли технологія потрапляє в погані руки або приймає рішення самостійно, це може мати катастрофічні наслідки [6].

1.3. Переклад за допомогою штучного інтелекту

Рисунок 1.3 – Штучний інтелект в галузі перекладу

Фахівці з перекладу вже багато років використовують у своїй професії цифрові рішення, такі як комп'ютерний переклад.

Машинний переклад з'явився приблизно п'ятнадцять років тому, з появою Google Translation, яка поклала початок перекладу за допомогою ШІ. Однак незабаром з'ясувалося, що цей метод має значні обмеження: переклад був дослівним і часто містив грубі мовні помилки.

Слід розрізняти машинний переклад RBMD (на основі лінгвістичних правил цей метод використовує словники для перекладу вмісту) і статистичний машинний переклад. Останній використовує машинне навчання для роботи з алгоритмами, здатними аналізувати великі обсяги існуючих перекладів і витягувати статистичні моделі.

У 2017 році з DeepL штучний інтелект справді зарекомендував себе у сфері перекладу. Це програмне забезпечення нейронного перекладу базується на даних веб-сайту перекладу Linguee, багатомовного словника, який порівнює переклади майже 20 мовами.

Це поклало початок нейронному машинному перекладу (NMT). Ця інтелектуальна технологія, заснована на штучних нейронах, враховує весь текст і його контекст. Крім того, NMT може постійно вдосконалюватися та вдосконалюватися завдяки потоку даних, які він отримує.

Нейронний машинний переклад — це серйозний технологічний прогрес, який дозволяє отримувати набагато якісніший і плавніший текст, ніж за допомогою машинного перекладу.

Переваги штучного інтелекту в перекладі:

AI дозволяє виконувати надзвичайно швидкі, великі обсяги перекладів, які стають все більш точними та точними. Програмне забезпечення для перекладу з підтримкою штучного інтелекту, яке використовує машинне навчання, має здатність самостійно виправляти та покращувати якість створених перекладів.

Крім того, багато інструментів перекладу з підтримкою штучного інтелекту здатні перекладати декілька текстів різними мовами одночасно.

Нарешті, перевага цих методів для користувачів полягає в тому, що вони дуже дешеві (або навіть безкоштовні) і охоплюють дуже широкий діапазон мов.

Недоліки штучного інтелекту в перекладі:

Незважаючи на численні переваги, ШІ далеко не безпомилковий і все ще має багато обмежень у сфері перекладу. Дійсно, ця технологія не в змозі адаптувати переклад до цільової аудиторії.

Він також не може враховувати місцеві культурні норми та звичаї, очікування клієнтів, стиль, наміри перекладу... Це важливі елементи перекладу, щоб отримати тексти, які поважають місцеву культуру, адаптовані до цільової аудиторії та вірні вихідний текст.

Крім того, хоча ШІ-переклад може бути ефективним для найпоширеніших мов (англійська, французька, іспанська, німецька, голландська, італійська, арабська тощо), він набагато менш ефективний для рідкісних мов або діалектів, для яких існує мало даних. У таких випадках штучному інтелекту дуже часто доведеться використовувати англійський переклад як проміжний крок, що може спричинити значні помилки та непорозуміння [7].

1.4. Синтез мовлення за допомогою штучного інтелекту

Синтез мовлення, також відомий як перетворення тексту в мовлення (TTS), — це технологія штучного інтелекту (ШІ), яка перетворює написаний

текст у вимовлені слова. Ця потужна функція революціонізувала наш спосіб взаємодії з машинами, дозволивши пристроям і програмам спілкуватися з користувачами природним і схожим на людину способом.

Витоки синтезу мови можна віднести до початку 18 століття, коли такі винахідники, як Вольфганг фон Кемпелен і Чарльз Вітстон, розробили механічні пристрої, здатні імітувати людську мову. Однак лише в середині 20 століття в цій галузі було досягнуто значних успіхів.

Однією з піонерських робіт у синтезі мовлення була розробка вокодеру Гомером Дадлі в 1930-х роках. Вокодер був пристроєм, здатним синтезувати мову шляхом аналізу та відтворення спектральних характеристик людської мови. Це заклало основу для подальших досліджень у цій галузі.

Поява цифрової обробки сигналів і прогрес обчислювальної потужності в 1960-х і 1970-х роках привели до розробки систем синтезу мови на основі правил. Ці системи використовували заздалегідь визначені правила та лінгвістичні бази даних для створення мови. Однак синтетичній продукції часто бракувало природності та виразності.

З появою машинного та глибокого навчання за останнє десятиліття синтез мовлення досяг значного прогресу. Моделі на основі нейронних мереж, такі як WaveNet і Tacotron, продемонстрували надзвичайні можливості у створенні високоякісної мови з природним звучанням.

Синтез мовлення відіграє вирішальну роль у різноманітних програмах штучного інтелекту/ML та наукових програмах, покращуючи взаємодію з користувачем та забезпечуючи більш інтуїтивно зрозумілу взаємодію. Ось кілька ключових сфер, де широко використовується синтез мовлення:

1) Доступність і допоміжні технології

Синтез мовлення значно покращив доступність для людей із вадами зору або труднощами з читанням. Програми зчитування з екрана, навігаційні системи та інші допоміжні технології покладаються на синтез мовлення для перетворення тексту на вимовлені слова, що дозволяє користувачам легко використовувати інформацію.

2) Віртуальні помічники та чат-боти

Віртуальні помічники, такі як Amazon Alexa, Google Assistant і Apple Siri, значною мірою покладаються на синтез мовлення, щоб надавати відповіді, схожі на людину. Генеруючи природне звучання мови, ці віртуальні помічники сприяють безперебійному спілкуванню між користувачами та пристроями, дозволяючи виконувати такі завдання, як голосові команди, пошук інформації та керування розумним будинком.

3) Вивчення мови та практика вимови

Синтез мовлення широко використовується в програмах для вивчення мови, щоб надати учням точні моделі вимови. Генеруючи розмовні слова та фрази, учні можуть практикувати свої навички вимови та покращувати вільне володіння іноземною мовою.

4) Виробництво аудіокниг

Видавнича галузь охопила синтез мовлення для виробництва аудіокниг. Замість того, щоб покладатися виключно на дикторів, видавці можуть використовувати системи TTS для перетворення письмового тексту в аудіо, таким чином скорочуючи час і витрати на виробництво.

5) Персоналізована реклама та маркетинг

Синтез мовлення дозволяє створювати персоналізовану рекламу та маркетингові кампанії. Динамічно генеруючи аудіовміст на основі вподобань користувачів або демографічних показників, компанії можуть доносити цільові повідомлення своїй аудиторії, підвищуючи залучення та коефіцієнти конверсії [8].

В проєкті синтез мовлення використовується для озвучування перекладеного тексту.

Висновки

В ході першого розділу було розглянуто історію та перспективи розвитку штучного інтелекту та проведено аналіз теми штучного інтелекту в контексті перекладу та синтезу мовлення.

ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Frontend

Для реалізації фронтенду веб-додатку, в якості основної мови прогрмування, було обрано Typescript, а також для побудови проєкту використано React.

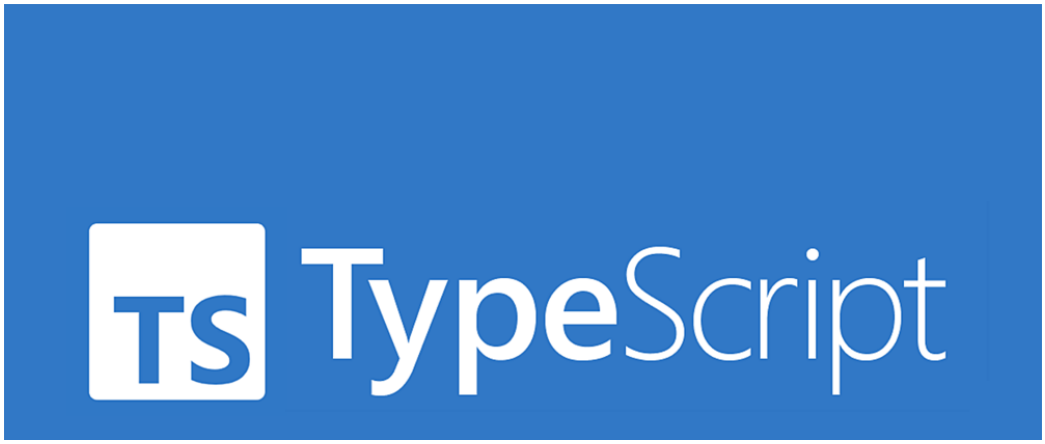

Рисунок 2.1 – Typescript лого

TypeScript — це мова програмування, яка є надмножиною JavaScript. Вона була створена компанією Microsoft і вперше випущена в 2012 році. Основною метою TypeScript є розширення можливостей JavaScript за рахунок додавання статичної типізації, що дозволяє знизити кількість помилок під час розробки і зробити код більш зрозумілим та підтримуваним.

Основні переваги TypeScript:

- TypeScript дозволяє розробникам явно вказувати типи змінних, функцій, параметрів та повертаючих значень. Це допомагає виявляти помилки на етапі компіляції, що значно підвищує надійність коду.
- TypeScript підтримує всі сучасні функції JavaScript (ES6/ES7 та новіші), включаючи стрілкові функції, класи, модулі, деструктуризацію тощо. Код на TypeScript компілюється в чистий JavaScript, який можна запускати в будь-якому середовищі, що підтримує JS.
- TypeScript інтегрується з різними інструментами та редакторами коду, такими як Visual Studio Code, забезпечуючи автозавершення, навігацію по коду, рефакторинг і інші зручності, які значно спрощують процес розробки.

- Завдяки строгій типізації та зрозумілій структурі коду, TypeScript особливо корисний при роботі над великими проектами, де важливо зберігати високу якість коду та легко здійснювати рефакторинг [9].

Рисунок 2.2 – React лого

React — це структура, яка використовує Webpack для автоматичної компіляції коду React, JSX і ES6 під час обробки префіксів файлів CSS. React — це бібліотека розробки інтерфейсу користувача на основі JavaScript. Хоча React є бібліотекою, а не мовою, він широко використовується у веб-розробці. Бібліотека вперше з'явилася в травні 2013 року і зараз є однією з найбільш часто використовуваних інтерфейсних бібліотек для веб-розробки.

У порівнянні з іншими технологіями на ринку, React є новою технологією. Джордан Волке, інженер-програміст Facebook, заснував бібліотеку в 2011 році, давши їй життя. Подібні до XHP, простому фреймворку HTML-компонентів для PHP, впливають на React. Стрічка новин React була його дебютним додатком у 2011 році. Пізніше Instagram підбирає його та включає в свою платформу.

Популярність React сьогодні затьмарила популярність усіх інших інтерфейсних фреймворків розробки. Ось чому:

- Просте створення динамічних додатків: React полегшує створення динамічних веб-додатків, оскільки вимагає менше кодування та пропонує більше функціональних можливостей, на відміну від JavaScript, де кодування часто стає складним дуже швидко.
- Покращена продуктивність: React використовує Virtual DOM, завдяки чому швидше створюються веб-додатки. Віртуальний DOM порівнює попередні стани компонентів і оновлює лише ті елементи в реальному DOM, які були змінені, замість того, щоб оновлювати всі компоненти знову, як це роблять звичайні веб-програми.

- Багаторазові компоненти: компоненти є будівельними блоками будь-якої програми React, і одна програма зазвичай складається з кількох компонентів. Ці компоненти мають свою логіку та елементи керування, і їх можна повторно використовувати в усій програмі, що, у свою чергу, значно скорочує час розробки програми.
- Односпрямований потік даних: React слідує за односпрямованим потоком даних. Це означає, що під час розробки програми React розробники часто вкладають дочірні компоненти в батьківські компоненти. Оскільки дані передаються в одному напрямку, стає легше виправляти помилки та знати, де виникає проблема в програмі в даний момент.
- Невелика крива навчання: React легко освоїти, оскільки він здебільшого поєднує базові концепції HTML і JavaScript з деякими корисними доповненнями. Проте, як і у випадку з іншими інструментами та фреймворками, вам доведеться витратити деякий час, щоб правильно зрозуміти бібліотеку React.
- Його можна використовувати для розробки як веб-, так і мобільних додатків: ми вже знаємо, що React використовується для розробки веб-додатків, але це ще не все, що він може зробити. Існує фреймворк під назвою React Native, похідний від самого React, який надзвичайно популярний і використовується для створення красивих мобільних додатків. Отже, насправді React можна використовувати для створення як веб-, так і мобільних додатків.
- Спеціальні інструменти для легкого налагодження: Facebook випустив розширення Chrome, яке можна використовувати для налагодження програм React. Це робить процес налагодження веб-додатків React швидшим і простішим.

Наведені вище причини з лишком виправдовують популярність бібліотеки React і чому її приймає велика кількість організацій і компаній. Тепер давайте познайомимося з функціями React [10].

Рисунок 2.3 – Material UI лого

Для побудови UI/UX інтерфейсу було використано бібліотеку React компонентів Material UI.

Material UI – це бібліотека компонентів React з відкритим кодом, яка реалізує Material Design від Google. Вона містить повну колекцію попередньо зібраних компонентів, які готові до використання у виробництві одразу після вилучення, а також містить набір параметрів налаштування, які спрощують реалізацію власної системи дизайну на основі наших компонентів.

Переваги Material UI:

- Доставляйте швидше: понад 2500 учасників з відкритим кодом витратили незліченні години на ці компоненти. Зосередьтеся на своїй основній бізнес-логіці замість того, щоб заново винаходити велосипед — ми подбаємо про ваш інтерфейс користувача.
- Красиво за замовчуванням: ми ретельно підходимо до впровадження Material Design, гарантуючи, що кожен компонент Material UI відповідає найвищим стандартам форми та функцій, але відхиляємось від офіційних специфікацій, де необхідно, щоб надати кілька чудових варіантів.
- Настроюваність: бібліотека містить широкий набір інтуїтивно зрозумілих функцій настроювання. Шаблони в нашому магазині демонструють, наскільки далеко ви можете зайти з налаштуванням.
- Міжкомандна співпраця: інтуїтивно зрозумілий інтерфейс розробника Material UI зменшує бар'єри для входу на роботу для бекенд-розробників і менш технічних дизайнерів, дозволяючи командам співпрацювати ефективніше. Набори дизайну спрощують ваш робочий процес і підвищують узгодженість між дизайнерами та розробниками.
- Довіряють тисячі організацій: Material UI має найбільшу спільноту UI в екосистемі React. Він майже такий же старий, як і сам React — його історія тягнеться з 2014 року — і ми в цьому довго. Ви можете розраховувати на

підтримку спільноти протягом багатьох років (наприклад, Stack Overflow) [11].

2.2. Backend

Рисунок 2.4 – Python лого

Python — це мова програмування, якій надають перевагу для програмування завдяки широким можливостям, можливості застосування та простоті. Мова програмування Python найкраще підходить для машинного навчання завдяки своїй незалежній платформі та популярності в спільноті програмістів.

Машинне навчання – це розділ штучного інтелекту (ШІ), який спрямований на те, щоб змусити машину навчатися на досвіді та автоматично виконувати роботу без обов'язкового програмування на виконання завдання. З іншого боку, штучний інтелект – це ширше значення машинного навчання, коли комп'ютери сприйнятливі до людського рівня шляхом візуального розпізнавання, мовлення, мовного перекладу та, як наслідок, прийняття критичних рішень.

Переваги використання Python:

1) Незалежність між платформами

Завдяки його здатності працювати на кількох платформах без необхідності змін, розробники віддають перевагу Python, на відміну від інших мов програмування. Python працює на різних платформах, таких як Windows, Linux і macOS, тому не потребує змін або не потребує жодних змін. Платформи повністю сумісні з мовою програмування Python, що означає, що експерту з Python для пояснення коду програми практично немає потреби.

Простота виконання дозволяє легко розповсюджувати програмне забезпечення, дозволяючи створювати та запускати автономне програмне забезпечення за допомогою Python. Програмне забезпечення можна програмувати від початку до кінця, використовуючи Python як єдину мову. Це плюс для розробників, оскільки інші мови програмування вимагають доповнення іншими мовами до того, як проект буде повністю завершено. Незалежність

Python від різних платформ економить час і ресурси для розробників, які інакше потребували б багато ресурсів для завершення одного проекту.

2) Послідовність і простота

Мова програмування Python є притулком для більшості розробників програмного забезпечення, які шукають простоти та послідовності у своїй роботі. Код Python є лаконічним і читабельним, що спрощує процес презентації. Розробник може легко написати код і лаконічно порівняти його з іншими мовами програмування. Це дозволяє розробникам отримувати інформацію від інших розробників у спільноті, щоб допомогти покращити програмне забезпечення чи програму.

Простота мови Python дозволяє початківцям легко освоїти її швидко та з меншими зусиллями порівняно з іншими мовами програмування. Крім того, досвідченим розробникам легко створювати стабільні та надійні системи, і вони можуть зосередити свої зусилля на розвитку своєї творчості та вирішенні реальних проблем за допомогою машинного навчання.

3) Різноманітність фреймворків і бібліотек

Бібліотеки та фреймворки є життєво важливими для підготовки відповідного середовища програмування. Фреймворки та бібліотеки Python пропонують надійне середовище, яке значно скорочує час розробки програмного забезпечення. Бібліотека в основному містить попередньо написаний код, який розробники можуть використовувати для прискорення кодування під час роботи над складними проектами.

Python містить модульну бібліотеку машинного навчання, відому як PyBrain, яка надає прості у використанні алгоритми для використання в завданнях машинного навчання. Найкращі та найнадійніші рішення кодування вимагають належної структури та перевіреного середовища, яке доступне у фреймворках і бібліотеках Python.

Python найбільше підходить для машинного навчання:

Машинне навчання та ШІ як єдине ціле все ще розвиваються, але їх використання швидко зростає через потребу в автоматизації. Штучний інтелект дає змогу створювати інноваційні рішення для типових проблем, таких як виявлення шахрайства, персональні помічники, фільтри спаму, пошукові системи та системи рекомендацій.

Попит на інтелектуальні рішення реальних проблем зумовлює необхідність подальшого розвитку штучного інтелекту, щоб автоматизувати завдання, програмування яких утомливо без штучного інтелекту. Мова програмування Python вважається найкращим алгоритмом для автоматизації таких завдань, і вона пропонує більшу простоту та послідовність, ніж інші мови програмування.

Крім того, присутність зацікавленої спільноти python дозволяє розробникам легко обговорювати проекти та вносити ідеї щодо вдосконалення свого коду [12].

2.3. Hugging Face

Рисунок 2.5 – Hugging Face лого

В проєкті використовуються AI моделі які розміщені на платформі Hugging Face.

Hugging Face — це DS та ML платформа з відкритим кодом. Вона діє як центр для експертів та ентузіастів зі штучного інтелекту, як GitHub для ШІ.

Спочатку запущений як чат-бот для підлітків у 2017 році, Hugging Face з роками перетворився на місце, де ви можете розміщувати власні моделі штучного інтелекту, тренувати їх і при цьому співпрацювати зі своєю командою. Він забезпечує інфраструктуру для запуску всього: від першого рядка коду до розгортання штучного інтелекту в активних програмах або службах. Крім цих функцій, ви також можете переглядати та використовувати моделі, створені іншими людьми, шукати та використовувати набори даних і тестувати демонстраційні проекти.

Однією з головних особливостей Hugging Face є можливість створювати власні моделі ШІ. Ця модель буде розміщена на платформі, що дозволить вам додавати більше інформації про неї, завантажувати всі необхідні файли та відстежувати версії. Ви можете контролювати, чи будуть ваші моделі загальнодоступними чи приватними, тож ви можете вирішувати, коли їх оприлюднити чи навіть запустити взагалі.

Це також дозволяє створювати обговорення безпосередньо на сторінці моделі, що зручно для співпраці з іншими та обробки запитів на отримання (їх роблять, коли учасники пропонують оновлення коду). Коли модель буде готова до використання, вам не потрібно розміщувати модель на іншій платформі: ви можете запускати її безпосередньо з Hugging Face, надсилати запити та завантажувати результати в будь-які програми, які ви створюєте.

Якщо ви не хочете починати з нуля, ви можете переглянути бібліотеку моделей Hugging Face. З понад 200 000 доступних моделей ви зможете працювати з такими речами, як:

- Обробка природної мови, включаючи такі завдання, як переклад, реферування та створення тексту. Ці функції є основою того, що, наприклад, пропонує OpenAI GPT-3 у ChatGPT.
- Аудіо дозволяє виконувати такі завдання, як автоматичне розпізнавання мовлення, виявлення голосової активності або перетворення тексту в мовлення.
- Комп'ютерний зір — це все, що допомагає комп'ютерам бачити реальний світ і розуміти його. Ці завдання включають оцінку глибини, класифікацію зображень і зображення від зображення до зображення. Це ключове значення, наприклад, для безпілотних автомобілів.
- Мультимодальні моделі працюють із кількома типами даних (текст, зображення, аудіо), а також можуть виводити кілька видів виводу.

Бібліотека Transformer від Hugging Face дає змогу підключатися до цих моделей, надсилати завдання та отримувати результати без необхідності їх налаштування самостійно. Ви також можете завантажувати моделі, тренувати їх за допомогою власних даних або швидко створювати простір. Це полегшує пошук моделей для виконання будь-якого завдання, зв'язує їх із власним кодом і починає отримувати результати [13].

На Hugging Face також можна знайти:

1) Набори даних

Хаб містить понад 5000 наборів даних більш ніж 100 мовами, які можна використовувати для широкого спектру завдань у сфері NLP, комп'ютерного зору та аудіо. Hub спрощує пошук, завантаження та завантаження наборів даних. Набори даних супроводжуються розширеною документацією у формі карток наборів даних і попереднього перегляду наборів даних, щоб ви могли переглядати дані безпосередньо у своєму браузері. Хоча багато наборів даних є загальнодоступними, організації та окремі особи можуть створювати приватні набори даних, щоб відповідати вимогам ліцензування чи конфіденційності.

Бібліотека наборів даних дозволяє вам програмно взаємодіяти з наборами даних, тож ви можете легко використовувати набори даних із Hub у своїх проектах. За допомогою одного рядка коду можна отримати доступ до наборів даних; навіть якщо вони настільки великі, що не поміщаються у комп'ютері, можна використовувати потокове передавання для ефективного доступу до даних.

2) Простори

Простори – це простий спосіб розміщення демонстраційних додатків ML на Hub. Вони дозволяють створювати портфоліо машинного навчання,

демонструвати свої проекти на конференціях або зацікавленим сторонам і співпрацювати з іншими людьми в екосистемі машинного навчання.

Наразі ми підтримуються два пакети SDK для Python (Gradio та Streamlit), які дозволяють створювати програми за лічені хвилини. Користувачі також можуть створювати статичні простори, які є простою сторінкою HTML/CSS/JavaScript усередині простору.

3) Організації

Компанії, університети та некомерційні організації є важливою частиною спільноти Hugging Face. Hub пропонує організації, які можна використовувати для групування облікових записів і керування наборами даних, моделями та просторами. Викладачі також можуть створювати спільні організації для учнів, використовуючи Hugging Face for Classrooms. Репозиторії організації будуть представлені на сторінці організації, і кожен член організації матиме можливість зробити свій внесок у репозиторій. На додаток до зручного групування всієї роботи організації, хаб дозволяє адміністраторам встановлювати ролі для контролю доступу до сховищ, а також керувати способом оплати своєї організації та платіжною інформацією [14].

Висновки

В ході другого розділу було детально розглянуто програмне забезпечення та сторонні сервіси, які використовуються для побудови frontend та backend частин проєкту.

НАПИСАННЯ ПРОГРАМИ ТА АНАЛІЗ

3.1. Структура проєкту

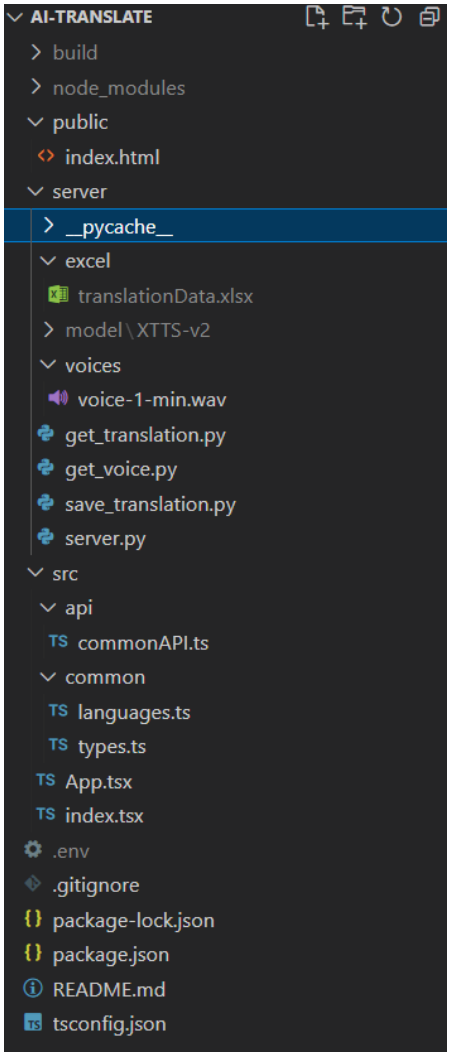

Рисунок 3.1 – Структура додатку

Програма має структуру стандартного React проєкту. Після створення проєкту було реалізовано tsconfig файл у якому міститься конфігурація Typescript та змінено розширення програмних файлів: js на ts, jsx на tsx. Також було створено папку server, у якій міститься серверна частина програми.

В цілому проєкт містить:

- /src – папка що містить файли з кодом програми;
- /src/api/commonAPI.ts – файл з функціями для API запитів;
- /src/common/languages.ts – файл, що містить необхідні для Select компонентів об’єкти;
- /src/common/types.ts – файл, у якому знаходяться кастомні typescript інтерфейси;
- /src/App.tsx – основний файл з кодом програми;
- /src/index.tsx – файл є точкою входу для скриптів react start/build програми;
- /public/index.html – цей файл HTML є шаблоном. Якщо ви відкриєте його безпосередньо в браузері, ви побачите порожню сторінку;
- /server – папка що містить серверну частину програми написану на Python;
- /server/__pycache__ – папка для зберігання скомпільованого байт-коду імпортованих модулів у проекті;
- /server/excel/translationData.xlsx – Excel файл, у якому зберігається результат роботи програми;
- /server/model – папка для зберігання моделі синтезу мовлення;
- /server/voices – папка для аудіофайлів, які використовуються в ході роботи моделі синтезу мовлення;
- /server/server.py – цей файл містить код для веб-сервера на базі Flask;
- /server/get_translation.py – файл містить функцію перекладу;
- /server/get_voice.py – файл містить функцію синтезу мовлення;
- /server/save_translation.py – файл містить функцію яка зберігає результат роботи програми в файл translationData.xlsx;
- /node_modules – папка використовується для зберігання залежностей (зовнішніх бібліотек або пакетів), які є в проєкті;
- /build – каталог збірки всередині кореневого каталогу, який об’єднує програму React і мінімізує її в прості файли HTML, CSS і JavaScript;
- /.env – містить змінні середовища;
- /.gitignore – містить директорії та файли проігноровані Git;
- /package.json – файл метаданих, який описує залежності проекту, сценарії, конфігурацію та інші деталі [15];
- /package-lock.json – автоматично генерується для будь-яких операцій, де npm змінює або дерево node_modules, або package.json [15];
- /tsconfig.json – файл з конфігурацією Typescript.

3.2. UI

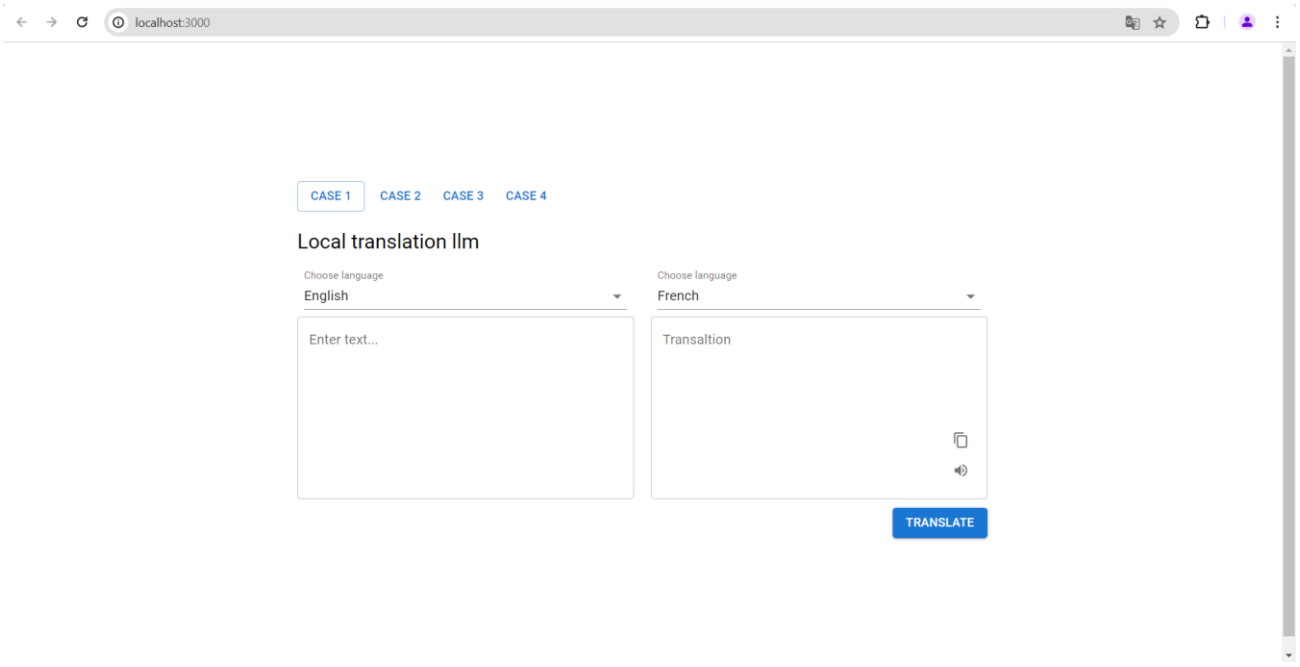

Рисунок 3.2 – UI додатку

Щоб запустити програму необхідно в консолі виконати команду npm start, після чого у браузері відкриється локальна сторінка з frontend частиною програми.

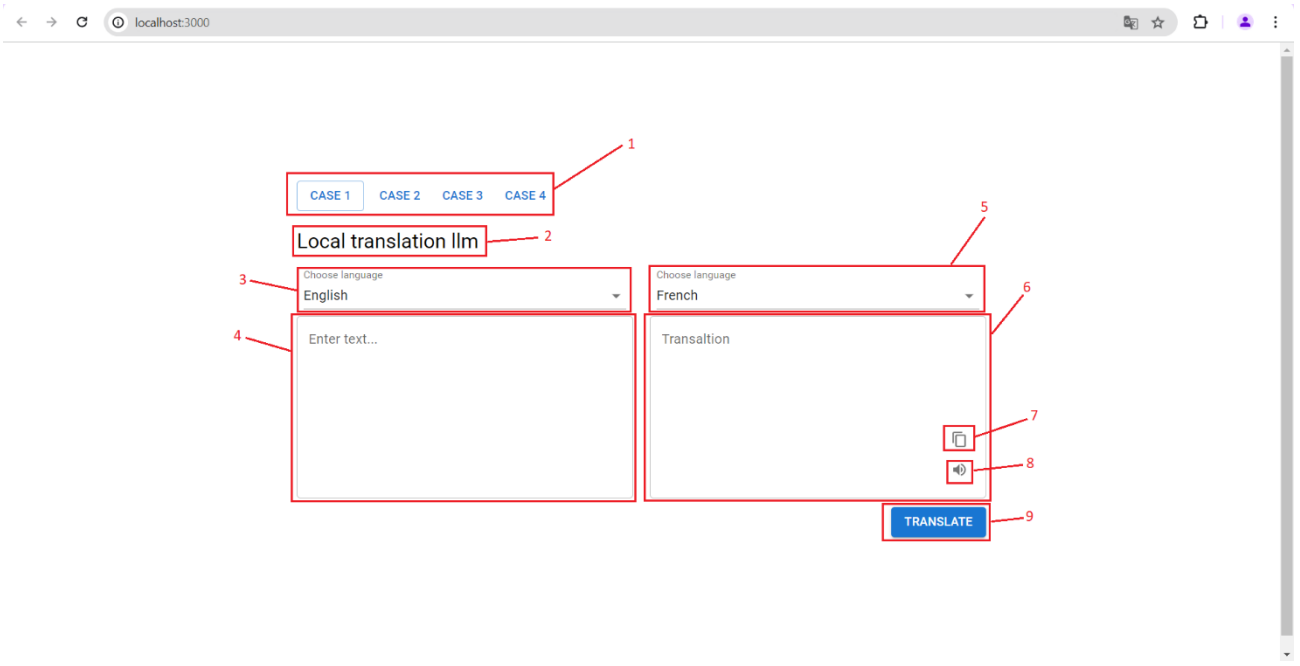

Рисунок 3.3 – Компоненти додатку

Компоненти програми включають в себе:

1) Опис:

Кнопки вибору AI моделі яка робить переклад.

Код:

```
<div style={{ display: 'flex', columnGap: '10px', marginBottom: '20px' }}>
          <Button onClick={() => handleChangeCase(1)} variant={caseNum === 1 ? "outlined" : "text"}>Case 1</Button>
          <Button onClick={() => handleChangeCase(2)} variant={caseNum === 2 ? "outlined" : "text"}>Case 2</Button>
          <Button onClick={() => handleChangeCase(3)} variant={caseNum === 3 ? "outlined" : "text"}>Case 3</Button>
          <Button onClick={() => handleChangeCase(4)} variant={caseNum === 4 ? "outlined" : "text"}>Case 4</Button>
        </div>
```

2) Опис:

Коротка назва методу перекладу, який зараз використовується юзером. Змінюється при переключенні кнопок з пункту 1.

Код:

```
<Typography variant="h5" gutterBottom>{caption}</Typography>
```

3) Опис:

Вибір мови тексту, який має бути перекладений. Варіанти вибору залежать від обраного Case та знаходяться в файлі languages.ts.

Код:

```
<FormControl variant="standard" sx={{ m: 1, minWidth: 120 }}>
              <InputLabel key="input1-label">Choose language</InputLabel>
              <Select
                labelId="select1-label"
                id="select1"
                value={sourceLang}
```

```
                onChange={(e)                                    =>
setSourceLang(e.target.value)}
                label="Language1"
                MenuProps={{
                  PaperProps: {
                    style: {
                      maxHeight: '200px'
                    },
                  },
                }}
              >
                {languageArr
                  .sort((a, b) => a.name < b.name ? -1 :
1)
                  .map((item, i) => (
                    <MenuItem        key={`menu-item${i}`}
value={item.index}>{item.name}</MenuItem>
                  ))
                }
              </Select>
            </FormControl>
```

4) Опис:

Текстове поле для введення тексту, який буде перекладений. При введенні тексту було додано кнопку для очищеня поля.

Код:

```
<TextField
              id="field1"
              label="Enter text..."
              variant="outlined"
              multiline
              rows={8}
              style={{ width: '400px' }}
              value={fieldValue}
              onChange={(e)                                    =>
setFieldValue(e.target.value)}
              InputProps={{
                endAdornment: (
                  <>
```

```
                    {fieldValue !== '' &&
                      <InputAdornment   style={{   height:
'100%',  maxHeight:  '100%',  alignItems:  'flex-start'  }}
position="end">
                        <IconButton    aria-label="clear"
onClick={() => setFieldValue('')}>
                          <ClearIcon fontSize='small' />
                        </IconButton>
                      </InputAdornment>
                    }
                  </>
                )
              }}
            />
```

5) Опис:

Вибір мови, на яку має бути перекладений введнеий текст. Варіанти вибору залежать від обраного Case та знаходяться в файлі languages.ts.

Код:

```
<FormControl variant="standard" sx={{ m: 1, minWidth: 120
}}>
              <InputLabel        key="input2-label">Choose
language</InputLabel>
              <Select
                labelId="select2-label"
                id="select2"
                value={targetLang}
                onChange={(e)                              =>
setTargetLang(e.target.value)}
                label="Language2"
                MenuProps={{
                  PaperProps: {
                    style: {
                      maxHeight: '200px'
                    },
                  },
                }}
              >
                {languageArr
```

```
                .sort((a, b) => a.name < b.name ? -1 :
1)
                .map((item, i) => (
                  <MenuItem        onClick={()        =>
setTranslateLan(item.lan)}          key={`menu-item2${i}`}
value={item.index}>{item.name}</MenuItem>
                ))
              }
            </Select>
          </FormControl>
```

6) Опис:

Текстове поле для відображення результатів перекладу введеного тексту. Перекладений текст не може бути стертий або змінений користувачем.

Код:

```
<TextField
            id="field2"
            label="Transaltion"
            variant="outlined"
            multiline
            rows={8}
            style={{ width: '400px' }}
            value={translatedText}
            InputProps={{
              endAdornment: (
                <InputAdornment style={{ height: '100%',
maxHeight: '100%', alignItems: 'flex-end', flexDirection:
'column', justifyContent: 'flex-end' }} position="end">
                  <IconButton          aria-label="copy"
disabled={isLoading}            onClick={()             =>
navigator.clipboard.writeText(translatedText)}>
                    <ContentCopyIcon fontSize='small' />
                  </IconButton>
                  {isLoadingVoice ?
                    <IconButton aria-label="spin">
                      <CircularProgress size={20} />
                    </IconButton>
                    :  <Tooltip  title={audioLoaded  ?
'Reproduce' : 'Get voice'}>
```

```
                        <IconButton     aria-label="copy"
disabled={isLoading}   onClick={()   =>   audioLoaded   ?
startAudio() : handleGetVoice()}>
                          <VolumeUpIcon
color={audioLoaded ? 'info' : 'inherit'} fontSize='small'
/>
                        </IconButton>
                      </Tooltip>
                    }
                  </InputAdornment>
                )
              }}
            />
```

7) Опис:

Кнопка копіювання перекладеного тексту. Після натискання текст зберігається в буфер обміну користувача.

Код:

```
<IconButton     aria-label="copy"     disabled={isLoading}
onClick={()                                              =>
navigator.clipboard.writeText(translatedText)}>
                      <ContentCopyIcon fontSize='small' />
                    </IconButton>
```

8) Опис:

Кнопка озвучки перекладеного тексту. Після натискання запускається функція синтезу мовлення, під час виконання якої замість кнопкаи користувач може побачит спінер. Коли результат роботи функції буде готовий користувач зможе натиснути кнопки ще раз та почути озвучений текст.

Код:

```
{isLoadingVoice ?
                      <IconButton aria-label="spin">
                        <CircularProgress size={20} />
                      </IconButton>
                      :   <Tooltip   title={audioLoaded   ?
'Reproduce' : 'Get voice'}>
                        <IconButton    aria-label="volume"
disabled={isLoading}   onClick={()   =>   audioLoaded   ?
startAudio() : handleGetVoice()}>
```

```
                            <VolumeUpIcon
color={audioLoaded ? 'info' : 'inherit'} fontSize='small'
/>
                          </IconButton>
                        </Tooltip>
                      }
```

9) Опис:

Кнопка перекладу. В залежності від обраного Case запускає одну з чотирьох функцій, які приймають обрані юзером занечення елеменів 3, 4 і 5 , а в результаті роботи повертають перекладений текст.

Код:

```
<div style={{ display: 'flex', justifyContent: 'right' }}>
          <LoadingButton
            loading={isLoading}
            variant="contained"
            onClick={() => handleClick()}
            style={{ marginTop: '10px' }}
          >
            Translate
          </LoadingButton>
        </div>
```

Основні Material UI компоненти використані в побудові UI частини додатку:

- Button: Кнопки дозволяють користувачам виконувати дії та робити вибір одним дотиком [16].
- LoadingButton: Кнопка яка має пропс загрузки. Якщо значення цього пропса true, відображається індикатор завантаження, а кнопка стає недоступною [16].
- FormControl: Надає контекст, як-от заповнений/фокусований/помилка/потрібний для введення даних форми. Покладання на контекст забезпечує високу гнучкість і гарантує, що стан завжди залишається узгодженим для дочірніх елементів FormControl [16].
- InputLabel: Відображає текст над полем вводу.
- InputAdornment: Використовується для додавання до input компонентів іконок або тексту, які можна розмістити перед (початок) або після (кінець) поля введення [16].

- MenuItem: Компонент, який зазвичай використовується в меню, наприклад у спадних меню або вхідних даних. Він представляє окремий пункт у цих меню та надає користувачам спосіб вибору або переходу між параметрами. [16]
- ClearIcon: іконка очистки;
- VolumeUpIcon: іконка гучності;
- ContentCopyIcon: іконка копіювання;
- IconButton: Компонент, який використовується для створення кнопок, які містять іконки. [16]
- Select: Вибрані компоненти використовуються для збору наданої користувачем інформації зі списку параметрів. [16]
- TextField: Текстові поля дозволяють користувачам вводити та редагувати текст. [16]
- Tooltip: Підказки відображають інформативний текст, коли користувачі наводять курсор на елемент, фокусують його або торкаються елемента. [16]
- CircularProgress: Компонент, який відображає круговий завантажувач, який часто використовується для вказівки того, що виконується завдання чи операція. [16]
- Typography: Використовуйте типографіку, щоб якомога чіткіше та ефективніше представити свій дизайн і вміст. [16]

Файл commonAPI.ts експортує API функції які комунікують з сервером або роблять API реквести до сторонних сервісів та повертають певний результат.

API функції:

1) getTranslation:

Ця функція приймає три параметри: input (рядок для перекладу), src (вихідна мова) і tgt (цільова мова). Вона надсилає запит POST на кінцеву точку локального сервера ("/getTranslation") із введеним текстом і мовною парою, отримує відповідь перекладу і повертає перекладений текст. Якщо під час процесу виникає помилка, вона реєструється та повертає порожній рядок.

2) getTranslation2:

Ця функція, є асинхронною функцією, яка перекладає вхідний текст із зазначеної вихідної мови на цільову мову за допомогою API Hugging Face. Спочатку він надсилає запит до API та чекає на відповідь. Якщо відповідь не є масивом і містить помилку, запит повторюється до 10 разів із затримкою 3 секунди між повторами. Після отримання успішної відповіді він зберігає дані перекладу та повертає перекладений текст. Якщо під час процесу виникає помилка, вона реєструється та повертає порожній рядок.

Варто зазначити що специфіка цієї функціїї полягає в тому, що Hugging Face організація Helsinki-NLP є автором багатьох окремих моделей перекладу, і, на відміну від інших функцій, getTranslation2 використовує декілька моделей, специфікованих для перекладу з однієї мови на іншу, замість однієї мультимовної моделі.

Функцію було реалізовано за допомогою технології Hugging Face Inference API, яка включає в себе наступні фічі:

- Можливість використання понад 150 000 моделей Transformer, Diffusers або Timm (T5, Blenderbot, Bart, GPT-2, Pegasus...)
- Завантаження, керування та обслуговування своїх власних моделей приватно
- Виконання завдання класифікації, NER, розмовного спілкування, узагальнення, перекладу, відповідей на запитання, вилучення вставок
- Робота з великими моделями, які важко розгорнути у виробництві
- Масштабування до 1000 запитів на секунду за допомогою вбудованого автоматичного масштабування
- Надсилання нових функцій NLP, CV, Audio або RL швидше, оскільки з'являються нові моделі

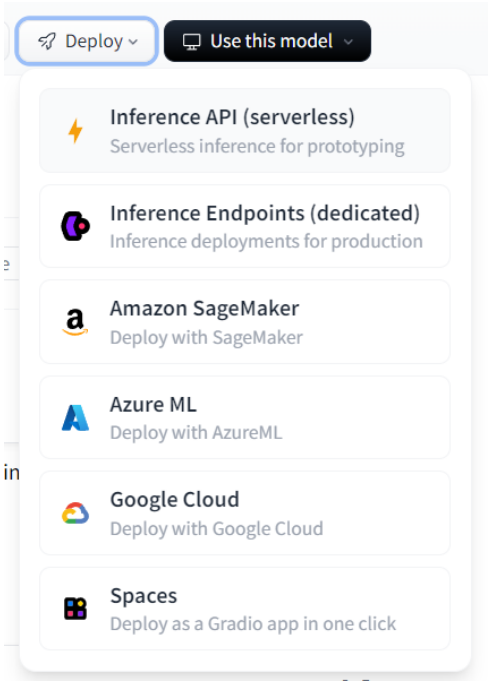

Рисунок 3.4 – Inference API знаходиться на сторінці моделі

Щоб почати роботу з Inference API, потрібно:
Зареєструватися або увійти.

Отримати маркер доступу користувача або API в налаштуваннях профілю Hugging Face.

Побачити токен hf_xxxxx (старі токени — api_XXXXXXXX або api_org_XXXXXXX).

Першим кроком є вибір моделі, яку ви збираєтеся використовувати. Перейдіть до Model Hub і виберіть модель, яку хочете використовувати.

Приклад ендпоінту:

```
ENDPOINT = https://api-inference.huggingface.co/models/<MODEL_ID>
```

Щоб зробити API реквест можна використати цей Javascript код (у прикладі наведено ендпоінт моделі gpt2, розміщеної на Hugging Face):

```
import fetch from "node-fetch";
async function query(data) {
    const response = await fetch(
        "https://api-inference.huggingface.co/models/gpt2",
        {
            headers: { Authorization: `Bearer ${API_TOKEN}` },
            method: "POST",
            body: JSON.stringify(data),
        }
    );
    const result = await response.json();
    return result;
}
query("Can you please let us know more details about your ").then((response) => {
    console.log(JSON.stringify(response));
});
```

Залежно від завдання, для якого налаштована модель, запит прийматиме певні параметри. Під час надсилання запитів на запуск будь-якої моделі параметри API дозволяють вказати кешування та поведінку завантаження моделі. Усі опції та параметри API детально описано тут, https://huggingface.co/docs/api-inference/detailed_parameters.

Так як ви є клієнтом API, ваш маркер API автоматично вмикатиме inference із прискоренням процесора для ваших запитів, якщо тип моделі підтримується.

Наприклад, якщо ви порівнюєте inference моделі gpt2 через наш API із прискоренням процесора, порівнюючи із запущеним inference моделі з коробки в локальному налаштуванні, ви повинні виміряти ~10-кратне прискорення. Конкретне підвищення продуктивності залежить від моделі та вхідного корисного навантаження (і вашого локального обладнання).

Щоб переконатися, що ви використовуєте версію моделі з процесорним прискоренням, ви можете перевірити заголовок типу x-compute у своїх запитах, який має бути оптимізованим cpu+. Якщо ви його не бачите, це просто означає, що не всі оптимізації ввімкнено. Це може бути пов’язано з різними факторами; модель може бути нещодавно додана до transformers, або модель можна оптимізувати кількома різними способами, і найкращий залежить від вашого випадку використання.

Inference API може обслуговувати прогнози на вимогу з понад 100 000 моделей, розгорнутих у Hugging Face Hub, які динамічно завантажуються в спільну інфраструктуру. Якщо запитану модель не завантажено в пам’ять, API безсерверного висновку почне із завантаження моделі в пам’ять і повернення відповіді 503, перш ніж він зможе відповісти прогнозом [17].

При написанні функції getTranslation2 було реалізовано цикл з 10 ітерацій і затримкою кожної ітерації в 3 секунди саме через те, що при зміні мови перекладу функція звератється до нового Inference API ендпоінту. При цьому замість перекладу у результаті повертається обьект, який повідомляє про те що модель знаходиться у стані загрузки.

Але після завантаження нової моделі через декілька ітерацій повертаеться переклад. У програмі Inference API є тимчасовим бесплатним рішенням, тому може давати нестабільний результат. Для покращення роботи програми у майбутньому Inference API може бути замінено на Inference Endpoints.

3) getTranslation3:

Функція має ту ж структуру що і getTranslation2, але замість специфікованих моделей перекладу використовує Inference API більш універсальної моделі під назвою google/flan-t5-base.

Модель google-t5/t5-base – це трансформерна мовна модель, розроблена компанією Google і є частиною серії моделей T5 (Text-To-Text Transfer Transformer). На відміну від багатьох інших моделей, які вирішують специфічні завдання, T5 є універсальним підходом, де всі завдання, пов'язані з обробкою тексту, формулюються як завдання перетворення тексту. Це означає, що незалежно від завдання — це переклад, сумаризація, класифікація або інше завдання обробки природної мови — вхідні дані та очікувані результати подаються у вигляді тексту.

Модель t5-base, будучи однією з версій T5, має збалансовану кількість параметрів, що робить її досить потужною для вирішення багатьох завдань, зберігаючи при цьому розумні вимоги до обчислювальних ресурсів. Це робить її привабливою для використання в прикладних системах, де потрібен баланс між якістю та продуктивністю.

4) getTranslation4:

Функція getTranslation4 використовує OpenAI API для перекладу тексту.

OpenAI – це дослідницька лабораторія та компанія ШІ, яка прагне розвивати ШІ та керувати ним у спосіб, який «принесе користь всьому людству». Спочатку вона була створена як некомерційна організація через занепокоєння її засновників щодо можливого зловживання та катастрофи через використання ШІ «в дикій природі». Кілька інвесторів об'єднали 1 мільярд доларів, щоб забезпечити дослідження та ресурси, які залишаються відкритими для громадськості.

OpenAI було засновано в 2015 році з фокусом на розробці штучного інтелекту та інструментів машинного навчання для різних видів діяльності. Його першою пропозицією був набір інструментів з відкритим вихідним кодом для розробки алгоритмів навчання з підкріпленням (OpenAI Gym), що спонукало його зосередитися на дослідженнях ШІ для більш загальних цілей.

У 2018 році OpenAI випустив концепцію Generative Pre-trained Transformer (GPT), яка є нейронною мережею (модель машинного навчання), яка моделює людський мозок і навчається на наборах даних. У 2021 році було випущено DALL-E, графічну версію ChatGPT, де люди можуть підказувати генеративній моделі ШІ створювати зображення. ChatGPT був випущений у листопаді 2022 року та став найпопулярнішим чат-ботом і генеративним інструментом штучного інтелекту – для створення будь-чого: від відповідей чат-бота до опитувань до резюме.

Продукти OpenAI:

- ChatGPT: чат-бот зі штучним інтелектом, який створює текст і відповідає на підказки та запитання користувачів. Він навчається на великих наборах даних і моделює досвід розмови з людиною та її слухання.
- DALL-E 2: це платформа, яка аналізує описи та підказки зображень, які потрібні користувачам, і створює їх, як описано. Наприклад, «намалюй кота в сюрреалістичному стилі».
- Codex: Codex схожий на ChatGPT, але для коду. Він навчений на тоннах коду на різних мовах програмування, щоб спростити процес кодування для розробників.

- Whisper: Whisper — це автоматичний інструмент розпізнавання мовлення, навчений на аудіоданих десятками мов, щоб він міг транскрибувати та перекладати мовлення.
- Scholar: програма, яка підтримує дослідників і студентів у проектах, пов'язаних зі штучним інтелектом, а іноді й фінансову допомогу.
- OpenAI Gym: Gym — це набір інструментів, який забезпечує основу для розробки алгоритмів навчання з підкріпленням.
- OpenAI API: Платформа розробника – це набір служб, включаючи вищезазначені, які допомагають створювати та розгортати програми ШІ.

Переваги OpenAI:

- Економія часу: алгоритми машинного навчання можуть автоматизувати такі завдання, як розпізнавання тексту, зображень і голосу, що може заощадити користувачам багато часу. Звичайний користувач може використати час, щоб зосередитися на редагуванні питань опитування, а не витрачати зусилля на розробку 20 оригінальних. Розробники програмного забезпечення можуть зосередитися на тому, щоб зробити мобільний додаток функціональним.
- Економте гроші: OpenAI може заощадити компаніям гроші на оплаті праці, оскільки для ручного позначення фотографій, завантажених користувачами (наприклад, у Facebook), потрібна величезна команда.
- Отримайте статистику: прогнозна аналітика OpenAI може аналізувати великі набори даних і надавати статистику, яку можна використовувати для сприяння взаємодії з продуктом або послугою. Він може ідентифікувати моделі поведінки користувачів, щоб перетворити хороший продукт або послугу на чудову.

Недоліки OpenAI:

- Проблема етики: OpenAI отримав критику за перехід від свого «некомерційного» статусу в 2019 році. Це спонукало людей повірити, що вони беруть участь у змаганнях за розробку найпередовіших технологій на основі досліджень, які він зібрав як некомерційна організація, і використовуючи щоб отримати прибуток.
- Проблема точності: Продукти OpenAI, такі як ChatGPT, зазнали критики за те, що вони навчаються на новинах і даних із відкритого Інтернету, які можуть бути застарілими або, що ще гірше, упередженими. Його здатність відокремлювати факти від вигадки від стереотипів заслуговує на увагу.

- Проблема безпечності: Токсичний вміст уже з’явився в службах OpenAI, зокрема про те, як створити бомбу та як вкрасти у людей, які нічого не підозрюють.
- Проблема законності: Ризики послуг OpenAI досягли рівня федерального уряду, де чиновники подають позови проти законності джерела даних і захищених авторським правом матеріалів. Були вжиті заходи щодо захисту оригінальних творів авторів і художників [18].

OpenAI API:

API OpenAI дозволяє користувачам використовувати потужність своїх моделей ШІ. API дозволяє надсилати запити до моделей OpenAI і отримувати інформацію у відповідь. Моделі, до яких наразі можна отримати доступ за допомогою API OpenAI, це GPT, DALL-E та Whisper, модель розпізнавання мовлення. Завдяки API ви можете по суті об’єднати технологію OpenAI із власною програмою.

Використання API OpenAI має багато переваг, включаючи, але не обмежуючись, живлення чат-ботів штучного інтелекту, обслуговування широкого кола клієнтів без мовних обмежень, а також використання постійних оновлень і прогресу в технології для «перспективної» вашої програми.

ШІ Інтеграція API OpenAI у ваш чат-бот надає автоматизовану підтримку клієнтам і потенційним клієнтам, що підвищує ефективність і покращує взаємодію з користувачем, оскільки спілкування більш схоже на людину та персоналізоване. Оскільки чат-боти на основі штучного інтелекту стають кращими з кожним днем, не є надуманим думати про використання чат-ботів штучного інтелекту замість справжньої цілодобової живої підтримки від реальних людей. Звичайно, чат-бот зі штучним інтелектом не буде настільки надійним чи універсальним, як справжній експерт у цій галузі, але він може зменшити тиск, пов’язаний із типовими проблемами.

Open AI API також можна використовувати для аналізу на кількох рівнях. Це може бути щось на зразок аналізу даних у контексті поведінки користувачів, щоб визначити тенденції та надати кращі рекомендації своїм клієнтам. Або контекст може бути навіть пов’язаним із безпекою, використовуючи той самий аналіз поведінки для виявлення підозрілої активності та відповідного реагування. Якою б не була мета аналізу, Open AI API може надавати інформацію, керовану даними, і автоматизовані відповіді, щоб значно підвищити продуктивність та ефективність додатків.

Хоча у світі технологій може й не бути такого поняття, як «забезпечення майбутнього», використання OpenAI стає досить близьким. Впроваджуючи технологію OpenAI, ви отримуєте вигоду від будь-яких оновлень і прогресу в цій

технології. OpenAI наразі оцінюється приблизно від 27 до 29 мільярдів доларів та має понад мільярд користувачів на місяць і продовжує швидко зростати [19].

В API реквесті функції getTranslation4 в body додатково передаються такі параметри:

```
{
    model: "gpt-3.5-turbo",
    messages: [
        {
            role: "system",
            content: "You are helpful assistant for text
translating."
        },
        {
            role: "user",
            content: `You should return only translated
text. Translate the following text from ${srcLang} into
${tgtLang}: ${text}`
        }
    ],
    temperature: 0.1
}
```

У параметрі model вказано Open AI модель, що буде використана для обробки запиту. В даному випадку обрано модель gpt-3.5-turbo. В залежності від використаної моделі зміниться ціна обробки запиту. Наступним пармаетром є messages, у ньому ми вказуємо моделі її поведінку та робимо текстовий запит. Останнім параметром є temperature, значення температури коливається від 0 до 2, причому нижчі значення вказують на більший детермінізм, а вищі значення вказують на більшу випадковість.

5) getVoice:

Функція getVoice — це асинхронна функція, яка надсилає запит POST на локальний сервер за адресою "http://127.0.0.1:5000/getVoice". Вона приймає два параметри, text і lan, які включає в тіло запиту як JSON. Очікується, що сервер поверне відповідь JSON, що містить аудіодані, які функція витягує та повертає. Якщо під час запиту виникає помилка, функція записує її на консоль і повертає порожній масив.

6) saveData:

Ця функція, saveData, є асинхронною функцією, розробленою для надсилання запиту POST до кінцевої точки сервера за адресою

http://127.0.0.1:5000/saveExcel. Він створює об'єкт JSON із метаданими, включаючи поточний час, назву моделі, вихідну та цільову мови, вихідний та цільовий тексти, і надсилає ці дані в тілі запиту. Якщо запит не виконується, він перехоплює та записує помилку на консоль.

Детальніше код програми та функціїї можна подивитися в додатках А, Б.

3.3. Server

Файл server.py – це проста веб-програма, створена за допомогою Flask, легкої веб-платформи для Python. Ця програма встановлює сервер із трьома основними маршрутами, кожен з яких призначений для виконання певних завдань, пов'язаних із перекладом і обробкою аудіо.

Код файлу server.py:

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import logging
from get_translation import get_translation
from save_translation import save_translation
from get_voice import get_voice

app = Flask(__name__)
CORS(app)

@app.route('/getTranslation', methods=['POST'])
def get_translation_route():
  query = request.json.get("query")
  source_lang = request.json.get("source_lang")
  target_lang = request.json.get("target_lang")
  return {'answer': get_translation(query, source_lang,
target_lang)}

@app.route('/saveExcel', methods=['POST'])
def save_excel_route():
  data = request.json.get("data")
  app.logger.info(data)
  save_translation(data)
  return 'saved'

@app.route('/getVoice', methods=['POST'])
def get_voice_route():
  text = request.json.get("text")
```

```
  lan = request.json.get("lan")
  audio_data = get_voice(text, lan)
  return jsonify({'audio_data': audio_data.tolist()})

if __name__ == "__main__":
    app.run(debug=True, use_reloader=False)
```

Додаток імпортує необхідні модулі, зокрема Flask для створення веб-сервера, request і jsonify для обробки та форматування HTTP-запитів і відповідей, а також CORS для спільного використання ресурсів між джерелами. Крім того, він імпортує три спеціальні функції з окремих модулів: get_translation для отримання перекладів, save_translation для збереження даних перекладу та get_voice для створення аудіо з тексту.

Програма Flask ініціалізується за допомогою app = Flask(__name__), а CORS увімкнено за допомогою CORS(app), щоб дозволити серверу обробляти запити з різних джерел.

У сценарії визначено три основні маршрути:

/getTranslation: цей маршрут приймає запити POST і призначений для обробки запитів на переклад. Він витягує параметри «query», «source_lang» і «target_lang» із корисного навантаження JSON запиту. Потім він викликає функцію get_translation із цими параметрами та повертає перекладений текст як відповідь JSON.

/saveExcel: цей маршрут також приймає запити POST і використовується для збереження даних перекладу. Він витягує параметр «дані» з корисного навантаження JSON, реєструє дані для цілей налагодження, а потім викликає функцію save_translation для збереження даних. Він повертає просте «збережено» повідомлення як відповідь.

/getVoice: цей маршрут відповідає за створення аудіоданих із тексту. Він приймає запити POST, витягує параметри «текст» і «лан» (мова) із корисного навантаження JSON і викликає функцію get_voice для створення аудіо. Потім аудіодані повертаються як відповідь JSON.

Щоб запустити цю програму, потрібно встановити Flask і його залежності, а також спеціальні модулі get_translation, save_translation і get_voice, та, у консолі, виконати команду: python server.py.

1) Функція get_translation:

Ця функція використовує модель MBart із бібліотеки Hugging Face Transformers для перекладу тексту між мовами. Вона починається з імпорту двох ключових класів: `MBartForConditionalGeneration` і `MBart50TokenizerFast`. Функція `get_translation` визначена для прийняття трьох параметрів: тексту для

перекладу (`query`), коду вихідної мови (`source_lang`) і коду цільової мови (`target_lang`).

Усередині функції налаштовано два шляхи до каталогу для кешування файлів моделі та токенізера, щоб уникнути їх завантаження під час кожного запуску сценарію. Потім сценарій завантажує попередньо навчену модель MBart і токенайзер із зазначених каталогів кешу або завантажує їх, якщо вони ще не кешовані.

Токенізатор налаштований на розпізнавання вихідної мови, а вхідний текст розрізняється на тензори, придатні для обробки моделі. Функція `model.generate` генерує перекладений текст із `forced_bos_token_id`, який забезпечує початок перекладу потрібною цільовою мовою. Згенеровані токени потім декодуються назад у зрозумілий людині текст за допомогою токенізера, за винятком спеціальних токенів, які використовуються під час процесу генерації.

Перекладений текст друкується та повертається функцією. Цей процес дозволяє перекладати текст з однієї мови на іншу за допомогою найсучаснішої багатомовної моделі.

Код функції:

```
from transformers import MBartForConditionalGeneration,
MBart50TokenizerFast

def get_translation(query, source_lang, target_lang):
    cache_dir = "/home/transformers_files/"
    cache_dir_models = cache_dir + "default_models/"
    cache_dir_tokenizers = cache_dir + "tokenizers/"

    model                                                =
MBartForConditionalGeneration.from_pretrained("facebook/m
bart-large-50-many-to-many-mmt",
cache_dir=cache_dir_models)
    tokenizer                                            =
MBart50TokenizerFast.from_pretrained("facebook/mbart-
large-50-many-to-many-mmt",
cache_dir=cache_dir_tokenizers)

    tokenizer.src_lang = source_lang
    encoded_lan = tokenizer(query, return_tensors="pt")
    generated_tokens = model.generate(
        **encoded_lan,
```

```
forced_bos_token_id=tokenizer.lang_code_to_id[target_lang
]
    )
    translation = tokenizer.batch_decode(generated_tokens,
skip_special_tokens=True)
    print(translation)

    return translation
```

2) Функція get_translation:

Функція призначена для створення синтетичного мовлення з тексту, введеного за допомогою вказаної мови. Вона починається з імпорту необхідних компонентів із бібліотеки синтезу мовлення (TTS), включаючи конфігурації та моделі, специфічні для XTTS, а також модуль звукових файлів для обробки аудіоданих.

Модель XTTS-v2 завантажен для локального користування та знаходиться в папці server/model/XTTS-v2.

XTTS-v2 – це модель генерації голосу, яка дозволяє клонувати голоси на різні мови за допомогою лише швидкого 6-секундного аудіозапису. Немає потреби в надмірній кількості тренувальних даних, які охоплюють незліченні години [20].

Функція get_voice визначена для прийняття двох параметрів: text, який є вхідним текстом, який потрібно перетворити на мовлення, і lan, який визначає мову вхідного тексту. Усередині функції за допомогою XttsConfig створюється об’єкт конфігурації та ініціалізується налаштуваннями, завантаженими з файлу JSON, розташованого за адресою «server/model/XTTS-v2/config.json».

Далі створюється та ініціалізується екземпляр моделі за допомогою параметрів конфігурації. Параметри моделі завантажуються з каталогу контрольних точок, а модель переміщується до GPU для прискореного обчислення.

Функція намагається синтезувати мовлення з введеного тексту за допомогою методу синтезу моделі. Для цього методу потрібні кілька параметрів: текст, який потрібно синтезувати, об’єкт конфігурації, еталонний аудіофайл для характеристик голосу мовця, довжина умови для моделі GPT і мова. Якщо процес синтезу стикається з винятковою ситуацією, друкується повідомлення про помилку, і функція повертається до генерації повідомлення про помилку за замовчуванням англійською мовою.

Функція повертає дані форми сигналу синтезованого мовлення, які витягуються з виходу методу synthesize. Дані форми хвилі потім можна використовувати для збереження або відтворення згенерованого звуку мови.

Код функції:

```
from TTS.tts.configs.xtts_config import XttsConfig
from TTS.tts.models.xtts import Xtts
import soundfile as sf

def get_voice(text, lan):
    config = XttsConfig()
    config.load_json("server/model/XTTS-v2/config.json")
    model = Xtts.init_from_config(config)
    model.load_checkpoint(config,
checkpoint_dir="server/model/XTTS-v2/", eval=True)
    model.cuda()
    outputs = None
    try:
        outputs = model.synthesize(
            text,
            config,
            speaker_wav="server/voices/voice-1-min.wav",
            gpt_cond_len=3,
            language=lan,
        )
    except Exception as e:
        print("An error occurred:", e)
        outputs = model.synthesize(
            'Sorry, this language is not available',
            config,
            speaker_wav="server/voices/voice-1-min.wav",
            gpt_cond_len=3,
            language='en',
        )
    return outputs['wav']
```

3) Функція save_translation:

Ця функція використовується для збереження даних перекладу у файл Excel за допомогою бібліотеки pandas. Сценарій починається з імпорту

необхідних бібліотек: pandas для обробки операцій з даними та os для взаємодії з файловою системою.

Функція save_translation приймає дані словника як параметр. Спочатку перевіряється, чи існує файл translationData.xlsx у вказаному шляху server/excel/. Якщо файл не існує, функція перетворює словник на pandas DataFrame, гарантуючи, що кожна пара ключ-значення у словнику стає стовпцем, а її відповідне значення у DataFrame. Потім цей DataFrame зберігається як файл Excel під назвою translationData.xlsx за вказаним шляхом, а DataFrame друкується на консолі для перевірки.

Якщо файл уже існує, функція читає наявний файл Excel у DataFrame. Він визначає кількість рядків у DataFrame, щоб визначити, куди слід додати нові дані. Нові дані, передані у функцію як словник, потім додаються як новий рядок у кінці DataFrame. Нарешті, оновлений DataFrame зберігається назад у той самий файл Excel, а оновлений DataFrame друкується на консолі для перевірки. Це гарантує, що нові дані перекладу можна постійно додавати до існуючого файлу без перезапису попередніх записів.

Код функції:

```
import pandas as pd
import os

def save_translation(data):
    if not os.path.exists('server/excel/translationData.xlsx'):
        data_lists = {key: [value] for key, value in data.items()}
        df = pd.DataFrame(data_lists)
        df.to_excel('server/excel/translationData.xlsx', index=False)
        print(df)
    else:
        df = pd.read_excel('server/excel/translationData.xlsx')
        num_rows = df.shape[0]
        df.loc[num_rows] = data
        df.to_excel('server/excel/translationData.xlsx', index=False)
        print(df)
```

3.4 Аналіз роботи програми

При тестуванні було досліджено швидкість роботи функцій getTranslation1-4 також швидкість роботи функції синтезу мовлення getVoice. Отримані дані можна представити у вигляді таблиці:

Таблиця 3.1. – Результати роботи функцій

Кількість символів в тексті	Мова тексту	Мова перекладу	getTranslation1, мс	getTranslation2, мс	getTranslation3, мс	getTranslation4, мс	getVoice, мс
88	en	fr	41806	15065	693	1306	34890
309	en	fr	70203	1545	581	3022	54230
962	en	fr	165047	13717	725	4779	72678
3296	en	fr	159053	35621	10757	11789	32828
87	fr	en	35395	12736	10379	741	63575
362	fr	en	64062	14613	768	1482	44894
1214	fr	en	141337	14367	876	2808	31822
2950	fr	en	157778	-	11890	6925	34811
3296	en	uk	154623	18740	1376	23124	42340
2536	uk	zh	166746	-	4119	10053	43054

Результати перекладу зберігаються окремо в Excel файлі translationData.xlsx:

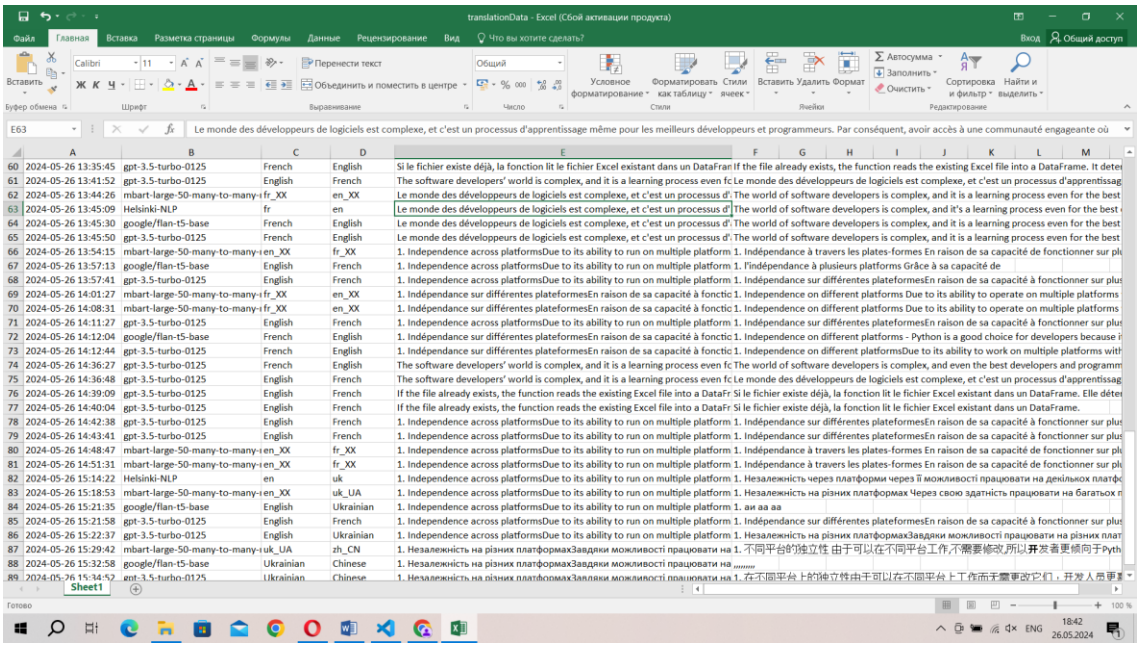

Рисунок 3.5 – Результати перекладу

За результатами тестування можна всиновки по роботі кожної з функцій:

1) getTranslation1:

AI модель, яка перекладає текст, встановлена локально і працює за рахунок ресурсів комп'ютера, тому ця функція має найбільший час роботи. Результат перекладу функції є нестабільним при великих обьемах тексту. Однією з переваг є те, що для робота функція не потребує підключення до інтернету, так як модель встановлена локально.

2) getTranslation2:

В таблиці 3.1. можна побачити нестабільність часу роботи функції через специфіку Hugging Face Inference API, але загалом її можна охарактеризувати як відносно швидку. Функція дає стабільний переклад, але деякі використані моделі мають ліміт і при спробі перекаду повертають помилку.

3) getTranslation3:

Через специфіку використовуваної моделі, функція не повертає очікуваний переклад та не є працездатною.

4) getTranslation4:

Функція працює швидко та повертає точний переклад. Функція використовує OpenAI API, який є платним сервісом і характеризується високою якістю. Для перекладу тексту у програмі рекомендується використовати самец ю функцію.

5) getVoice:

Як і getTranslation1, функція синтезу мовлення працює локально, тому має середню швидкість, при цьому повертає стабільний результат перекладу.

В результаті тестування було визначено що, серед обраних моделей, найкращою накращою є Open AI, так як видає найточніший результат перекладу за найкоротший проміжок часу.

Висновки

В ході третього розділу було виконано реалізацію проєкту за допомогою обраних інструментів та технологій, розглянуто роботу основних функцій програми, протестовано роботу програми та проаналізонвано отримані в ході тестування показники.

ВИСНОВОК

При завершенні роботи над статтею була розглянута тема використання штучного інтелекту в контексті перекладу і синтезу мови. Робота складається з 3 основних розділів, які логічно, в свою чергу, охоплюють різні аспекти досліджуваної теми.

У першому розділі роботи були розглянуті історичний розвиток і сучасний стан технологій штучного інтелекту, особливу увагу приділено їх застосуванню в області перекладу текстів і синтезу мови. Історичний огляд включав етапи розвитку від перших спроб автоматизації перекладу до новітніх методів, заснованих на машинному навчанні та нейронних мережах. Хоча ранні системи мали значні обмеження, сучасні технології, такі як нейронний машинний переклад (NMT), значно покращили якість перекладів, наблизивши їх до людського рівня.

Другий розділ присвячений вибору технологій для реалізації проекту. Було обґрунтовано вибір мови програмування, архітектури системи та моделей штучного інтелекту. Ми порівняли різні інструменти та платформи, включаючи hugging Face для обробки природної мови та OpenAI API для синтезу мови. Вибір технології був зроблений з урахуванням таких критеріїв, як ефективність, точність і простота інтеграції в загальну архітектуру системи.

У третьому розділі описується процес моделювання та впровадження помічника зі штучним інтелектом.Це включає розробку інтерфейсу користувача та серверної системи. Ми протестували основні функції Програми, зокрема переклад і синтез мови. В результаті тестування було показано, що обрана модель OpenAI забезпечує найвищу якість перекладу за мінімальну кількість часу. Аналіз показників роботи системи підтвердив її працездатність і ефективність при виконанні поставлених завдань.

Таким чином, результати роботи показують важливий потенціал використання штучного інтелекту для автоматизації процесу перекладу і синтезу мови. Запропонований асистент AI може широко використовуватися в різних областях, де потрібні швидкі та якісні мовні послуги. Використання сучасних технологій штучного інтелекту дозволяє значно підвищити ефективність і точність перекладу, сприяючи поліпшенню міжкультурної комунікації і співпраці.

На підставі проведених досліджень і тестів можна зробити висновок, що подальший розвиток і вдосконалення технології штучного інтелекту в області перекладу і синтезу мови є перспективним напрямком. Впровадження такої системи дозволить знизити вартість мовних послуг і забезпечить доступ до високоякісних послуг з перекладу для більш широкої аудиторії.

Загальний огляд і результати роботи свідчать про досягнення цілей і завдань дослідження, підтверджують важливість і актуальність теми в сучасному світі. Результати дослідження можуть бути використані в якості основи для подальшого наукового розвитку цієї області і впровадження практичних рішень в реальних умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) The Impact of Artificial Intelligence on Language Translation: A Review. URL: https://www.researchgate.net/publication/378284156_The_Impact_of_Artificial_Intelligence_on_Language_Translation_A_review (дата звернення: 29.04.2024)
2) The History of Artificial Intelligence. URL: https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/ (дата звернення: 29.04.2024)
3) OpenAI Releases GPT-3, The Largest Model So Far. URL: https://analyticsindiamag.com/open-ai-gpt-3-language-model/
4) AI has already changed the world. URL: https://www.sfchronicle.com/tech/article/ai-artificial-intelligence-report-stanford-17869558.php (дата звернення: 29.04.2024)
5) Here's where the A.I. jobs are. URL: https://www.cnbc.com/2023/04/05/ai-jobs-see-the-state-by-state-data-from-a-stanford-study.html (дата звернення: 29.04.2024)
6) Future of AI (Artificial Intelligence): What Lies Ahead? URL: https://www.simplilearn.com/future-of-artificial-intelligence-article (дата звернення: 29.04.2024)
7) The impact of AI on the future of translation. URL: https://www.alphatrad.com/news/future-impact-ai-translation (дата звернення: 29.04.2024)
8) Speech synthesis explained. URL: https://ai-jobs.net/insights/speech-synthesis-explained/#:~:text=Speech%20synthesis%2C%20also%20known%20as,natural%20and%20human-like%20manner (дата звернення: 18.05.2024)
9) TypeScript for the New Programmer. URL: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html (дата звернення: 19.05.2024)
10) The Best Guide to Know What Is React. URL: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs (дата звернення: 19.05.2024)
11) Frequently Asked Questions. URL: https://mui.com/material-ui/getting-started/faq/ (дата звернення: 19.05.2024)

12) Python (in Machine Learning). URL: https://corporatefinanceinstitute.com/resources/data-science/python-in-machine-learning/ (дата звернення: 20.05.2024)
13) What is Hugging Face? URL: https://zapier.com/blog/hugging-face/ (дата звернення: 20.05.2024)
14) Hugging Face Hub documentation. URL: https://huggingface.co/docs/hub/en/index (дата звернення: 20.05.2024)
15) Configuring npm. URL: https://docs.npmjs.com/cli/v10/configuring-npm (дата звернення: 20.05.2024)
16) Material UI components. URL: https://mui.com/material-ui/all-components/ (дата звернення: 20.05.2024)
17) Overview. URL: https://huggingface.co/docs/api-inference/quicktour (дата звернення: 21.05.2024)
18) What Is OpenAI? Everything You Need to Know. URL: https://www.coursera.org/articles/what-is-openai (дата звернення: 21.05.2024)
19) What is OpenAI's API? [+ How to Start Using It]. URL: https://blog.hubspot.com/website/what-is-open-ai-api (дата звернення: 21.05.2024)
20) XTTS. URL: https://huggingface.co/coqui/XTTS-v2 (дата звернення: 21.05.2024)

ДОДАТОК А
Код програми файл App.tsx

```
import React, { useState, useEffect } from 'react';
import { Button } from '@mui/material';
import LoadingButton from '@mui/lab/LoadingButton';
import FormControl from '@mui/material/FormControl';
import InputLabel from '@mui/material/InputLabel';
import InputAdornment from '@mui/material/InputAdornment';
import MenuItem from '@mui/material/MenuItem';
import ClearIcon from '@mui/icons-material/Clear';
import VolumeUpIcon from '@mui/icons-material/VolumeUp';
import ContentCopyIcon from '@mui/icons-
material/ContentCopy';
import Select from '@mui/material/Select';
import TextField from '@mui/material/TextField';
import Tooltip from '@mui/material/Tooltip';
import CircularProgress from
'@mui/material/CircularProgress';
import Typography from '@mui/material/Typography';
import { getTranslation, getTranslation2, getTranslation3,
getTranslation4, getVoice } from './api/commonAPI';
import IconButton from '@mui/material/IconButton';
import { case1Languages, case2Languages, case3Languages }
from './common/languages';
import { ILanguage } from './common/types';

function App() {
  const [fieldValue, setFieldValue] = useState<string>('')
  const [translatedText, setTranslatedText] =
useState<string>('')
  const [sourceLang, setSourceLang] =
useState<string>('en_XX')
  const [targetLang, setTargetLang] =
useState<string>('fr_XX')
  const [translateLan, setTranslateLan] =
useState<string>('fr')
  const [isLoading, setIsLoading] =
useState<boolean>(false)
  const [caseNum, setCaseNum] = useState<number>(1)
  const [languageArr, setLanguageArr] =
useState<Array<ILanguage>>(case1Languages)
```

```
  const [caption, setCaption] = useState<string>('Local
translation llm')
  const [isLoadingVoice, setIsLoadingVoice] =
useState<boolean>(false)
  const [audioLoaded, setAudioLoaded] =
useState<boolean>(false)
  const [voice, setVoice] = useState<Array<number>>([])

  const handleClick = async () => {
    setIsLoading(true)
    if (fieldValue === '' || sourceLang === targetLang)
setTranslatedText(fieldValue)
    else {
      let res = ''
      switch (caseNum) {
        case 1:
          res = await getTranslation(fieldValue,
sourceLang, targetLang)
          break
        case 2:
          res = await getTranslation2(fieldValue,
sourceLang, targetLang)
          break
        case 3:
          res = await getTranslation3(sourceLang,
targetLang, fieldValue)
          break
        case 4:
          res = await getTranslation4(fieldValue,
sourceLang, targetLang)
          break
      }
      setTranslatedText(res)
      setAudioLoaded(false)
    }
    setIsLoading(false)
  }

  const handleChangeCase = (num: number) => {
    if (num !== caseNum) {
      setCaseNum(num)
      setAudioLoaded(false)
```

```
      setIsLoadingVoice(false)
      setVoice([])
      setFieldValue('')
      setTranslatedText('')
      setLanguageArr(num === 1 ? case1Languages : num ===
2 ? case2Languages : case3Languages)
      setSourceLang(num === 1 ? 'en_XX' : num === 2 ? 'en'
: 'English')
      setTargetLang(num === 1 ? 'fr_XX' : num === 2 ? 'fr'
: 'French')
      setCaption(
        num === 1 ? 'Local translation llm' :
        num === 2 ? 'API transaltion llm' :
        num === 3 ? 'API chat llm' :
        'OpenAI API'
      )
    }
  }

  const handleGetVoice = async () => {
    if (translatedText !== '') {
      setIsLoadingVoice(true)
      const audio = await getVoice(translatedText,
translateLan)
      setVoice(audio)
      setIsLoadingVoice(false)
      setAudioLoaded(true)
    }
  }

  const startAudio = async () => {
    if (voice.length > 0) {
      const context = new AudioContext();
      const audioBuffer = context.createBuffer(1,
voice.length, 22050);
      const audioBufferData =
audioBuffer.getChannelData(0);
      audioBufferData.set(voice);

      const source = context.createBufferSource();
      source.buffer = audioBuffer;
      source.connect(context.destination);
```

```
      source.onended = () => context.close();
      source.start();
    }
  }

  return (
    <div style={{ width: '100%', height: '100vh', display:
'flex', justifyContent: 'center', alignItems: 'center' }}>
      <div  style={{  display:  'flex',  flexDirection:
'column' }}>
        <div style={{ display: 'flex', columnGap: '10px',
marginBottom: '20px' }}>
          <Button  onClick={()  =>  handleChangeCase(1)}
variant={caseNum  ===  1  ?  "outlined"  :  "text"}>Case
1</Button>
          <Button  onClick={()  =>  handleChangeCase(2)}
variant={caseNum  ===  2  ?  "outlined"  :  "text"}>Case
2</Button>
          <Button  onClick={()  =>  handleChangeCase(3)}
variant={caseNum  ===  3  ?  "outlined"  :  "text"}>Case
3</Button>
          <Button  onClick={()  =>  handleChangeCase(4)}
variant={caseNum  ===  4  ?  "outlined"  :  "text"}>Case
4</Button>
        </div>
        <Typography                              variant="h5"
gutterBottom>{caption}</Typography>
        <div style={{ display: 'flex', columnGap: '20px'
}}>
          <div  style={{  display:  'flex',  flexDirection:
'column' }}>
            <FormControl  variant="standard"  sx={{  m:  1,
minWidth: 120 }}>
              <InputLabel       key="input1-label">Choose
language</InputLabel>
              <Select
                labelId="select1-label"
                id="select1"
                value={sourceLang}
                onChange={(e)                              =>
setSourceLang(e.target.value)}
                label="Language1"
```

```
                MenuProps={{
                  PaperProps: {
                    style: {
                      maxHeight: '200px'
                    },
                  },
                }}
              >
                {languageArr
                  .sort((a, b) => a.name < b.name ? -1 :
1)
                  .map((item, i) => (
                    <MenuItem        key={`menu-item${i}`}
value={item.index}>{item.name}</MenuItem>
                  ))
                }
              </Select>
            </FormControl>
            <TextField
              id="field1"
              label="Enter text..."
              variant="outlined"
              multiline
              rows={8}
              style={{ width: '400px' }}
              value={fieldValue}
              onChange={(e)                                  =>
setFieldValue(e.target.value)}
              InputProps={{
                endAdornment: (
                  <>
                    {fieldValue !== '' &&
                      <InputAdornment   style={{   height:
'100%',  maxHeight:  '100%',  alignItems:  'flex-start'  }}
position="end">
                        <IconButton    aria-label="clear"
onClick={() => setFieldValue('')}>
                          <ClearIcon fontSize='small' />
                        </IconButton>
                      </InputAdornment>
                    }
                  </>
```

```
                )
              }}
            />
          </div>
          <div style={{ display: 'flex', flexDirection:
'column' }}>
            <FormControl variant="standard" sx={{ m: 1,
minWidth: 120 }}>
              <InputLabel       key="input2-label">Choose
language</InputLabel>
              <Select
                labelId="select2-label"
                id="select2"
                value={targetLang}
                onChange={(e)                              =>
setTargetLang(e.target.value)}
                label="Language2"
                MenuProps={{
                  PaperProps: {
                    style: {
                      maxHeight: '200px'
                    },
                  },
                }}
              >
                {languageArr
                  .sort((a, b) => a.name < b.name ? -1 :
1)
                  .map((item, i) => (
                    <MenuItem        onClick={()        =>
setTranslateLan(item.lan)}          key={`menu-item2${i}`}
value={item.index}>{item.name}</MenuItem>
                  ))
                }
              </Select>
            </FormControl>
            <TextField
              id="field2"
              label="Transaltion"
              variant="outlined"
              multiline
              rows={8}
```

```
              style={{ width: '400px' }}
              value={translatedText}
              InputProps={{
                endAdornment: (
                  <InputAdornment style={{ height: '100%',
maxHeight: '100%', alignItems: 'flex-end', flexDirection:
'column', justifyContent: 'flex-end' }} position="end">
                    <IconButton          aria-label="copy"
disabled={isLoading}            onClick={()            =>
navigator.clipboard.writeText(translatedText)}>
                      <ContentCopyIcon fontSize='small' />
                    </IconButton>
                    {isLoadingVoice ?
                      <IconButton aria-label="spin">
                        <CircularProgress size={20} />
                      </IconButton>
                      :  <Tooltip  title={audioLoaded  ?
'Reproduce' : 'Get voice'}>
                        <IconButton   aria-label="volume"
disabled={isLoading}   onClick={()   =>   audioLoaded   ?
startAudio() : handleGetVoice()}>
                          <VolumeUpIcon
color={audioLoaded ? 'info' : 'inherit'} fontSize='small'
/>
                        </IconButton>
                      </Tooltip>
                    }
                  </InputAdornment>
                )
              }}
            />
          </div>
        </div>
        <div  style={{  display:  'flex',  justifyContent:
'right' }}>
          <LoadingButton
            loading={isLoading}
            variant="contained"
            onClick={() => handleClick()}
            style={{ marginTop: '10px' }}
          >
            Translate
```

```
          </LoadingButton>
        </div>
      </div>
    </div>
  );
}

export default App;
```

ДОДАТОК Б
Код програми файл commonAPI.ts

```
import { ITranslate1, ITranslate4, IVoiceResponse } from
"../common/types";

export const getVoice = async (text: string, lan: string)
=> {
    try {
        const          response          =          await
fetch("http://127.0.0.1:5000/getVoice", {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                text: text,
                lan: lan
            }),
            redirect: 'follow'
        });
    const res: IVoiceResponse = await response.json()
    return res.audio_data
    } catch (error) {
        console.log('error', error)
    return []
    }
}

export  const  saveData  =  async  (model:  string,  sLang:
string, tLang: string, sText: string, tText: string) => {
    try {
        await fetch("http://127.0.0.1:5000/saveExcel", {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                data: {
                Time:                                    new
Date().toISOString().split('.')[0].split('T').join(' '),
                Model: model,
```

```
                SourceLanguage: sLang,
                TargetLanguage: tLang,
                SourceText: sText,
                TargetText: tText,
                }
            }),
            redirect: 'follow'
        });
    } catch (error) {
        console.log('error', error)
    }
}

export const getTranslation = async (input: string, src:
string, tgt: string) => {
    try {
        const          response          =          await
fetch("http://127.0.0.1:5000/getTranslation", {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                query: input,
                source_lang: src,
                target_lang: tgt
            }),
            redirect: 'follow'
        });
        const res: ITranslate1 = await response.json()
    saveData('mbart-large-50-many-to-many-mmt', src, tgt,
input, res.answer[0])
        return res.answer[0]
    } catch (error) {
        console.log('error', error)
        return ''
    }
}

export  const  getTranslation2  =  async  (input:  string,
source: string, target: string) => {
    try {
```

```
        const response = await fetch(
            `https://api-
inference.huggingface.co/models/Helsinki-NLP/opus-mt-
${source}-${target}`,
            {
                headers: {
                    Authorization:                `Bearer
${process.env.REACT_APP_HUGGING_FACE_TOKEN}`,
                    "Content-Type": "application/json"
                },
                method: "POST",
                body:  JSON.stringify({  "inputs":  input
}),
            }
        );
        let result = await response.json();
        let i = 0
        while          (!Array.isArray(result)          &&
result.hasOwnProperty('error') && i <= 10) {
            i++
            const newResponse = await fetch(
                `https://api-
inference.huggingface.co/models/Helsinki-NLP/opus-mt-
${source}-${target}`,
                {
                    headers: {
                        Authorization:            `Bearer
${process.env.REACT_APP_HUGGING_FACE_TOKEN}`,
                        "Content-Type":
"application/json"
                    },
                    method: "POST",
                    body:   JSON.stringify({   "inputs":
input }),
                }
            );
            result = await newResponse.json();
            if         (!Array.isArray(result)         &&
result.hasOwnProperty('error')) await new Promise(resolve
=> setTimeout(resolve, 3000));
        }
```

```
        if           (!Array.isArray(result)           &&
result.hasOwnProperty('error')) return '';
        else {
      saveData('Helsinki-NLP',  source,  target,  input,
result[0].translation_text)
      return result[0].translation_text
        }
    } catch (error) {
        console.log('error', error)
        return ''
    }
}

export const getTranslation3 = async (sLang: string, tLang:
string, input: string) => {
    try {
        const response = await fetch(
            "https://api-
inference.huggingface.co/models/google/flan-t5-base",
            {
                headers: {
                    Authorization:                `Bearer
${process.env.REACT_APP_HUGGING_FACE_TOKEN}`,
                    "Content-Type": "application/json"
                },
                method: "POST",
                body:     JSON.stringify({     "inputs":
`Translate ${sLang} text to ${tLang}: ${input}` }),
            }
        );
        let result = await response.json();
        let i = 0
        while          (!Array.isArray(result)          &&
result.hasOwnProperty('error') && i <= 10) {
            i++
            const newResponse = await fetch(
                "https://api-
inference.huggingface.co/models/google/flan-t5-base",
                {
                    headers: {
                        Authorization:           `Bearer
${process.env.REACT_APP_HUGGING_FACE_TOKEN}`,
```

```
                        "Content-Type":
"application/json"
                    },
                    method: "POST",
                    body:   JSON.stringify({   "inputs":
`Translate ${sLang} text to ${tLang}: ${input}` }),
                }
            );
            result = await newResponse.json();
            if         (!Array.isArray(result)         &&
result.hasOwnProperty('error')) await new Promise(resolve
=> setTimeout(resolve, 3000));
        }
        if           (!Array.isArray(result)           &&
result.hasOwnProperty('error')) return '';
        else {
      saveData('google/flan-t5-base', sLang, tLang, input,
result[0].generated_text)
      return result[0].generated_text
    }
    } catch (error) {
        console.log('error', error)
        return ''
    }
}

export  const  getTranslation4  =  async  (text:  string,
srcLang: string, tgtLang: string) => {
    try {
        const          response          =          await
fetch("https://api.openai.com/v1/chat/completions", {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                Authorization:                      `Bearer
${process.env.REACT_APP_OPENAI_API_KEY}`
            },
            body: JSON.stringify({
                model: "gpt-3.5-turbo",
                messages: [
                    {
                        role: "system",
```

```
                        content: "You are helpful
assistant for text translating."
                    },
                    {
                        role: "user",
                        content: `You should return only
translated text. Translate the following text from
${srcLang} into ${tgtLang}: ${text}`
                    }
                ],
        temperature: 0.1
            }),
        });
        const res: ITranslate4 = await response.json()
    saveData(res.model,    srcLang,    tgtLang,    text,
res.choices[0].message.content)
        return res.choices[0].message.content
    } catch (error) {
        console.log('error', error)
        return ''
    }
}
```