

**Міжнародний європейський університет**

ННІ «Європейська школа бізнесу»

Кафедра інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ  
Доцент кафедри інформаційних те-  
хнологій ННІ «Європейська школа  
бізнесу», кандидат фізико-матема-  
тичних наук В. В. Федоров

« \_\_\_\_ » \_\_\_\_\_ 2026 р.

**КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА**

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

зі спеціальності 121 «Інженерія програмного забезпечення»

**ТЕМА: «Проектування та реалізація чат-бота в Telegram для завантаження й обробки відео з YouTube із інтеграцією AI асистента»**

Виконавець: Філіппов Ігор Ігорович

Науковий керівник: Доцент кафедри інформаційних технологій ННІ «Європейська школа бізнесу», кандидат фізико-математичних наук В.В. Федоров

Консультанти з окремих розділів пояснювальної записки:

Нормоконтролер:

КИЇВ – 2026

## Міжнародний європейський університет

ННІ «Європейська школа бізнесу»

Кафедра інформаційних технологій

Ступінь вищої освіти – перший (бакалаврський) рівень

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма: «Інженерія програмного забезпечення»

Освітньо-кваліфікаційний рівень: бакалавр

ЗАТВЕРДЖУЮ:

Доцент кафедри інформаційних технологій ННІ «Європейська школа бізнесу», кандидат фізико-математичних наук В. В. Федоров

« \_\_\_\_ » \_\_\_\_\_ 2026 р.

### ЗАВДАННЯ

#### НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ БАКАЛАВРСЬКОЇ РОБОТИ

Філіппов Ігор Ігорович

**1. Тема:**

« Проєктування та реалізація чат-бота в Telegram для завантаження й обробки відео з YouTube із інтеграцією AI асистента », затверджена наказом ректора від «24» грудня 2025 р. № 207-к.

**2. Термін виконання роботи:**

з «26» травня 2026 р. по «22» червня 2026 р.

**3. Дата подання роботи на випускню кафедру:**

«12» червня 2026 р.

**4. Вихідні дані роботи:**

Python, Telegram Bot API, YouTube Transcript API, OpenAI API, бібліотеки

requests, deep-translator, youtube-transcript-api, модулі обробки тексту, логування та інтеграції зовнішніх API.

## 5. Зміст пояснювальної записки:

- Вступ
- Розділ 1. Аналіз сучасних чат-ботів, платформ обміну повідомленнями.
- Розділ 2. Проектування чат-бота в Telegram для завантаження й обробки відео з YouTube
- Розділ 3. Реалізація, тестування та оцінка ефективності системи
- Висновки
- Список використаних джерел
- Додатки (код, скріншоти, приклади повідомлень і результати обробки)

## 6. Перелік обов'язкового графічного матеріалу:

Схема архітектури системи. Блок-схема алгоритму обробки посилання. Скріншоти інтерфейсу Telegram-бота.

## 7. Календарний план - графік виконання КБР

№ з/п	Завдання	Термін виконання	Відмітка про виконання
1.	Складання і затвердження індивідуального завдання на виконання кваліфікаційної бакалаврської роботи	28.02.26	
2.	Підготовка вступу і розділу 1 КБР	17.03.26	
3.	Підготовка розділу 2 КБР	10.04.26	
4.	Підготовка розділу 3 КБР, висновків і переліку використаних джерел	10.05.26	
5.	Подання студентом завершеної КБР науковому керівнику для перевірки на плагіат та оформлення відгуку	09.06.26	
6.	Попередній розгляд КБР на комісії від кафедри	12-13.06.26	
7.	Доопрацювання роботи, прийняття кафедрою рішення про допуск роботи до захисту в ЕК, оформлення та зовнішнє рецензування	16-20.06.26	
8.	Захист кваліфікаційної бакалаврської роботи в ЕК і присвоєння кваліфікації	23-24.06.26	

Студент \_\_\_\_\_

(підпис, ПІБ)

Керівник \_\_\_\_\_

(підпис, ПІБ)

## РЕФЕРАТ

**Тема:** Проектування та реалізація чат-бота в Telegram для завантаження й обробки відео з YouTube із інтеграцією AI асистента

Кваліфікаційна бакалаврська робота зі спеціальності 121 «Інженерія програмного забезпечення», ОПП «Інженерія програмного забезпечення»,

Міжнародний європейський університет, 2026 р.

Загальний обсяг роботи **77 стор.**, складається зі вступу, трьох розділів, висновків; містить **8 таблиць, 5 рисунків, 103 літературних джерела.**

**Ключові слова:** чат-бот, Telegram, YouTube, штучний інтелект, Python, субтитри, переклад, конспектування, терміни, API.

### **Актуальність теми роботи:**

У сучасному середовищі значна частина навчальної, науково-популярної та прикладної інформації представлена у форматі відео на платформі YouTube. Разом із цим користувачі часто потребують швидкого доступу до змісту матеріалу без повного перегляду ролика. Тому створення чат-бота в Telegram, здатного автоматично отримувати субтитри відео, перекладати їх українською мовою, очищати текст від шуму й формувати змістовний конспект і перелік термінів за допомогою AI, є актуальною та практично значущою задачею.

**Метою роботи** є проектування та реалізація чат-бота в Telegram для обробки відео з YouTube із використанням AI-асистента, який забезпечує отримання перекладу, короткого конспекту та списку ключових термінів.

**Для досягнення поставленої мети у роботі вирішено такі завдання:**

1. проаналізувати сучасні підходи до створення чат-ботів і систем обробки текстової інформації;
2. дослідити можливості Telegram Bot API, YouTube Transcript API та сервісів машинного перекладу;
3. спроектувати архітектуру програмної системи й алгоритм обробки відеопосилання;

4. реалізувати модулі отримання субтитрів, перекладу, очищення тексту, формування конспекту та термінів;
5. провести тестування створеного програмного продукту та оцінити його практичну ефективність.

**Об'єктом роботи** є процес автоматизованої обробки відеоконтенту з платформи YouTube.

**Предмет дослідження** – архітектура, функціональність та програмна реалізація Telegram-чат-бота з інтеграцією AI асистента для аналізу відеоматеріалів.

### **Теоретичне та практичне значення одержаних результатів**

Полягає у створенні прикладного програмного засобу, який може бути використаний для швидкого опрацювання відеоконтенту в освітній, професійній та самоосвітній діяльності. Результати роботи демонструють можливість поєднання засобів інтеграції зовнішніх API, перекладу та генеративного AI в одному зручному користувацькому інтерфейсі.

### **Публікації, апробація:**

Окремі фрагменти програмної реалізації та результати тестування системи були апробовані під час локального використання Telegram-бота та перевірки сценаріїв обробки навчальних відео з YouTube.

## ABSTRACT

**Topic:** Design and implementation of a chatbot in Telegram for downloading and processing videos from YouTube with integration of an AI assistant

Qualifying bachelor's thesis in specialty 121 "Software Engineering", OPP "Software Engineering",

International European University, 2026

The total volume of the work is **77 pages**, consists of an introduction, three sections, conclusions; contains **8 tables, 5 pictures, 103 references**.

**Keywords:** chatbot, Telegram, YouTube, artificial intelligence, Python, subtitles, translation, note-taking, terms, API.

### **Relevance of the topic of the work:**

In the modern environment, a significant part of educational, popular science and applied information is presented in video format on the YouTube platform. At the same time, users often need quick access to the content of the material without watching the entire video. Therefore, creating a chatbot in Telegram that can automatically receive video subtitles, translate them into Ukrainian, clean the text from noise, and form a meaningful synopsis and list of terms using AI is a relevant and practically significant task.

**The purpose of the work** is to design and implement a chatbot in Telegram for processing videos from YouTube using an AI assistant that provides automated translation, a concise summary, and a list of key terms.

### **To achieve the goal, the following tasks were solved in the work:**

1. analyze modern approaches to creating chatbots and text information processing systems;
2. explore the capabilities of the Telegram Bot API, YouTube Transcript API, and machine translation services;
3. design the architecture of the software system and the video link processing algorithm;
4. implement modules for receiving subtitles, translating, cleaning the text, forming a synopsis, and terms;

5. test the created software product and evaluate its practical effectiveness.

**The object of the work** is the process of automated processing of video content from the YouTube platform.

**The subject of the research** is the architecture, functionality and software implementation of the Telegram chat bot with the integration of an AI assistant for analyzing video materials.

**The theoretical and practical significance of the results obtained**

It consists in creating an applied software tool that can be used for quick processing of video content in educational, professional and self-educational activities. The results of the work demonstrate the possibility of combining the means of integrating external APIs, translation and generative AI in one convenient user interface.

**Publications, testing:**

Individual fragments of the software implementation and the results of system testing were tested during local use of the Telegram bot and testing of scenarios for processing educational videos from YouTube.

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Скорочення / позначення	Розшифровка
AI	Artificial Intelligence — штучний інтелект
API	Application Programming Interface — інтерфейс програмування застосунків
ASR	Automatic Speech Recognition — автоматичне розпізнавання мовлення
CRUD	Create, Read, Update, Delete — базові операції зі збереження даних
GPT	Generative Pre-trained Transformer — генеративна попередньо навчена модель-трансформер
GRU	Gated Recurrent Unit — рекурентна одиниця з вентилями
HTTP / HTTPS	HyperText Transfer Protocol (Secure) — протокол передачі гіпертексту (захищений)
JSON	JavaScript Object Notation — текстовий формат обміну даними
LLM	Large Language Model — велика мовна модель
LSTM	Long Short-Term Memory — довга короткочасна пам'ять (тип нейромережі)
NER	Named Entity Recognition — розпізнавання іменованих сутностей
NLP	Natural Language Processing — обробка природної мови
NMT	Neural Machine Translation — нейронний машинний переклад
POST	HTTP-метод для передачі даних на сервер
Q&A	Question and Answer — режим запитань і відповідей

Скорочення / позначення	Розшифровка
REST	Representational State Transfer — архітектурний стиль побудови API
RNN	Recurrent Neural Network — рекурентна нейронна мережа
SDK	Software Development Kit — набір інструментів для розробки програмного забезпечення
SMT	Statistical Machine Translation — статистичний машинний переклад
SQL	Structured Query Language — мова структурованих запитів
SSL	Secure Sockets Layer — протокол захищеного з'єднання
TF-IDF	Term Frequency–Inverse Document Frequency — метод оцінки важливості слова
URL	Uniform Resource Locator — уніфікований локатор ресурсів (адреса в мережі)

## ЗМІСТ

ВСТУП .....	10
РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ ЧАТ-БОТІВ, ПЛАТФОРМ ОБМІНУ ПОВІДОМЛЕННЯМИ ТА AI-ТЕХНОЛОГІЙ.....	12
1.1. Роль чат-ботів у сучасних інформаційних системах.....	12
1.2. Класифікація чат-ботів та підходи до їх побудови.....	13
1.3. Порівняльний аналіз платформ для розгортання чат-ботів.....	15
1.4. Технології штучного інтелекту для обробки природної мови .....	16
1.5. Інструменти транскрибування та машинного перекладу .....	18
1.6. Проблеми та обмеження існуючих рішень.....	19
1.7. Вимоги до системи обробки відео з YouTube .....	20
1.8. Висновки до розділу .....	21
РОЗДІЛ 2. ПРОЄКТУВАННЯ ЧАТ-БОТА В TELEGRAM ДЛЯ ЗАВАНТАЖЕННЯ Й ОБРОБКИ ВІДЕО З YOUTUBE .....	22
2.1. Загальна архітектура системи .....	22
2.2. Вибір мови програмування, бібліотек та зовнішніх сервісів .....	24
2.3. Структура даних і логіка збереження стану користувача.....	25
2.4. Алгоритм обробки YouTube-посилання .....	27
2.5. Проєктування модулів системи .....	30
2.6. Проєктування користувацької взаємодії та сценаріїв роботи .....	31
2.7. Проєктування промптів для AI-аналізу .....	33
2.8. Безпека, обмеження та захист доступу .....	35
2.9. Висновки до розділу .....	36
РОЗДІЛ 3. РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ .....	37
3.1. Програмна реалізація основних функцій.....	37
3.2. Реалізація очищення та підготовки тексту .....	38
3.3. Реалізація взаємодії з Telegram API .....	40
3.4. Тестування системи: методи, сценарії та результати.....	41
3.5. Оцінка якості результатів.....	43
3.6. Оцінка продуктивності системи .....	45
3.7. Виявлені проблеми та шляхи їх вирішення.....	46
3.8. Практичні рекомендації щодо вдосконалення.....	47
3.9. Висновки до розділу .....	48
ВИСНОВКИ.....	50
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	52
ДОДАТКИ.....	62

## ВСТУП

YouTube давно перестав бути просто платформою для розваг. Значна частина університетських лекцій, технічних туторіалів і галузевих конференцій тепер існує саме у форматі відео. Студент, який хоче розібратися в темі, часто потрапляє в ситуацію: відео є, субтитри є, але воно англійською, триває годину, і не зрозуміло — варте воно уваги чи ні. Перегляд цілком, щоб зрозуміти суть, — не завжди розумна витрата часу.

Проблема не нова, але готового зручного рішення досі немає. Окремі сервіси вміють робити коротке зведення відео, інші — зберігати субтитри, треті — перекладати текст. Але жоден із них не поєднує це все в одному місці і не дає результат у тому інтерфейсі, яким користувач і так користується щодня. Саме ця прогалина і стала відправною точкою для цієї роботи.

Ідея проста: користувач надсилає посилання на YouTube-відео у Telegram і отримує три речі — переклад субтитрів українською, стислий конспект і список термінів із поясненнями. Нічого зайвого, без реєстрації і без переходу на сторонні сайти. Технічно це означає: отримати субтитри через youtube-transcript-api, перекласти через GoogleTranslator, очистити від рекламного шуму і передати до gpt-5-mini для аналізу.

Метою кваліфікаційної роботи є проєктування та реалізація чат-бота в Telegram для обробки відео з YouTube із використанням AI-асистента, який забезпечує отримання перекладу, короткого конспекту та списку ключових термінів.

Для досягнення поставленої мети необхідно виконати такі завдання: проаналізувати сучасні підходи до створення чат-ботів і систем обробки текстової інформації; розглянути можливості Telegram Bot API, YouTube Transcript API, сервісів перекладу та AI-моделей; спроектувати загальну структуру бота та логіку обробки YouTube-посилання; реалізувати програмний код мовою Python; забезпечити отримання субтитрів, переклад тексту, формування конс-

пекту та виділення термінів; провести тестування системи на різних типах вхідних даних і оцінити результати.

Об'єктом дослідження є процес автоматизованої обробки текстової інформації, отриманої з відеоматеріалів YouTube. Предметом дослідження є методи та програмні засоби створення Telegram-бота, який використовує субтитри відео, переклад і AI-аналіз для формування зручного результату для користувача.

У процесі виконання роботи використовуються методи аналізу технічної документації, порівняння існуючих рішень, проектування програмної системи, програмної реалізації та тестування. Для розробки застосовано мову Python, Telegram Bot API, YouTube Transcript API, засоби перекладу та AI-інструменти для обробки тексту.

Структурно кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків. У першому розділі розглянуто сучасні чат-боти, платформи обміну повідомленнями та AI-технології. У другому розділі описано проектування системи, її архітектуру та логіку роботи. У третьому розділі наведено програмну реалізацію, тестування та оцінку ефективності створеного Telegram-бота.

## РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ ЧАТ-БОТІВ, ПЛАТФОРМ ОБМІНУ ПОВІДОМЛЕННЯМИ ТА AI-ТЕХНОЛОГІЙ

### 1.1. Роль чат-ботів у сучасних інформаційних системах

Перші програми, що імітували діалог, з'явилися набагато раніше, ніж сучасні месенджери. ELIZA (1966, MIT) не розуміла мови — вона зіставляла фрази зі шаблонами і повертала заготовлені репліки в стилі психотерапевта. Попри технічну примітивність, програма справляла на користувачів враження реального спілкування — феномен, який пізніше назвали «ефектом ELIZA». У 1995 році з'явився ALICE з мовою AIML, що дозволяла гнучкіше описувати правила діалогу. Справжній якісний стрибок стався після 2016 року: Facebook, Slack і Telegram відкрили платформи для розробки ботів, а моделі глибокого навчання підняли точність розпізнавання намірів до рівня, придатного для комерційного використання.

Сьогодні чат-боти вирішують задачі, далекі від простої відповіді на запитання. В електронній комерції вони ведуть покупця від вибору товару до підтвердження замовлення. Банки автоматизують через них рутинні звернення — перевірку балансу, блокування картки, видачу довідок — без залучення живого оператора. У медицині боти записують на прийом, нагадують про прийом ліків і збирають первинні скарги. За прогнозом Juniper Research (2022), до 2027 року 90% звернень до служб підтримки у банківській галузі оброблятимуть боти, а не люди.

В освіті чат-боти набули особливого значення після 2020 року, коли дистанційне навчання різко розширилося. Студент, який опрацьовує матеріал самостійно, потребує миттєвого пояснення незрозумілого терміну або короткого конспекту — і не завжди поруч є викладач чи однокурсник. Дослідники фіксують стійкий зв'язок між застосуванням AI-інструментів у навчанні та підвищенням академічних результатів. За даними Stanford HAI (2023), інструменти на основі великих мовних моделей позитивно впливають на засвоєння матеріалу завдяки механізму миттєвого зворотного зв'язку [7]: студент отримує по-

яснення в момент виникнення запитання, а не відкладає його до наступного заняття.

З технічної точки зору чат-бот — це подійно-орієнтована система. Кожне повідомлення користувача запускає ланцюг: розпізнавання тексту → класифікація наміру → вибір обробника → формування відповіді. Ця модель добре лягає на мікросервісну архітектуру, де кожен компонент відповідає за свою ділянку і може бути замінений незалежно від решти. Саме так організовано і бот, розроблений у цій роботі: отримання субтитрів, переклад і AI-аналіз — три незалежні модулі з чітко визначеними входом і виходом.

Поява GPT-3.5 і GPT-4 перевернула уявлення про можливості чат-ботів. До 2022 року переважна більшість комерційних ботів базувалася на класифікаторах намірів із фіксованим набором відповідей — якщо запит не вкладався в один зі сценаріїв, бот повертав щось на кшталт «не розумію вас». Генеративні моделі відповідають на довільні запити, узагальнюють тексти, перекладають і структурують матеріал під конкретне завдання. Ця можливість і використовується в розробленому боті: замість пошуку відповіді в базі — генерація конспекту і термінів безпосередньо з отриманого транскрипту.

Ринок чат-ботів росте швидко. Ринок чат-ботів демонструє стаке зростання. За аналітичними даними Grand View Research, середньорічний темп приросту сегмента становить близько 23%, а до 2030 року обсяг ринку може перевищити 27 млрд доларів США [79]. Рушіями росту є три фактори: здешевлення хмарних API, зниження технічного порогу входу для розробників і зростаючі очікування користувачів щодо миттєвої відповіді — люди дедалі менше готові чекати з'єднання з оператором.

## **1.2. Класифікація чат-ботів та підходи до їх побудови**

Підходи до побудови чат-ботів прийнято ділити на три групи залежно від того, як формується відповідь: на основі правил (rule-based), на основі вибірки з бази (retrieval-based) і на основі генерації (generative). Кожен підхід має свій компроміс між передбачуваністю, гнучкістю і вартістю розробки.

Rule-based боти працюють за деревами сценаріїв або регулярними виразами. Вхідне повідомлення зіставляється з шаблонами — якщо відповідність знайдено, повертається пов'язана відповідь. Перевага очевидна: поведінка повністю прозора і передбачувана. Проблема — жорсткість: будь-яке відхилення від передбаченого сценарію призводить або до помилки, або до безглуздої відповіді. Підхід добре підходить для структурованих задач із вузьким набором можливих запитів — запис на прийом, довідкові меню, замовлення їжі.

Retrieval-based боти мають базу готових відповідей і шукають у ній найближчий варіант до запиту користувача. Для пошуку схожості використовують TF-IDF, косинусну подібність між векторними представленнями або нейронні ранжувальники. Відповіді виходять природнішими, ніж у rule-based ботів, але бот все одно обмежений тим, що є в базі — нового він не скаже.

Generative підхід на основі великих мовних моделей (LLM) дозволяє генерувати відповідь довільного змісту і форми без прив'язки до заздалегідь підготовленої бази. Це і є головна перевага для задачі формування конспекту: транскрипт кожного відео унікальний, і жодна фіксована база відповідей тут не допоможе. Разом із тим генеративні моделі не гарантують точності — вони можуть вигадувати факти, тому в промпті важливо чітко обмежувати завдання і явно вказувати, чого слід уникати.

За орієнтацією на область знань розрізняють domain-specific і general-purpose ботів. Перші точніші у своїй галузі, але не виходять за її межі. Другі обробляють різноманітний контент — що важливо для бота, який працює з відео різної тематики. За способом взаємодії боти бувають текстовими, голосовими і мультимодальними. Для Telegram обрано текстовий формат: великі обсяги перекладу і конспекту зручніше читати, ніж слухати.

Критерій	Rule-based	Retrieval-based	Generative (LLM)
Передбачуваність	Висока	Середня	Низька

Критерій	Rule-based	Retrieval-based	Generative (LLM)
Гнучкість	Низька	Середня	Висока
Потреба в даних	Мала	Велика база	Велика модель
Складність розробки	Низька	Середня	Низька (API)
Якість для довгого тексту	Не застосовна	Слабка	Висока

*Таблиця 1.1 — Порівняння підходів до побудови чат-ботів*

З таблиці 1.1 добре видно, чому для задачі конспектування і перекладу обрано generative-підхід: він не потребує бази відповідей, працює з текстами будь-якої тематики і легко інтегрується через API без власної інфраструктури для навчання моделей.

### **1.3. Порівняльний аналіз платформ для розгортання чат-ботів**

Вибір платформи-месенджера напряму визначає, які API-функції доступні, який поріг входу для розробника і яка аудиторія може скористатися ботом. Нижче розглянуто п'ять платформ: Telegram, Viber, WhatsApp Business API, Discord і Slack.

Telegram — фактично стандарт для студентських і дослідницьких проєктів у сфері ботів. Реєстрація займає хвилину через BotFather, документація повна і актуальна, а доступ до API не потребує ніякого комерційного погодження. Платформа підтримує текстові повідомлення, кастомні клавіатури (ReplyKeyboardMarkup і InlineKeyboardMarkup), callback-запити, надсилання файлів і медіа. Єдине практичне обмеження — 4096 символів на одне текстове повідомлення, що треба враховувати при відправці довгих перекладів: текст доводиться розбивати на частини.

WhatsApp Business API закритий для вільного використання: потрібно пройти верифікацію бізнесу через партнерів Meta і платити за кожне надіслане повідомлення. Для навчального проєкту — непрохідний бар'єр. Viber пропонує подібний до Telegram підхід, але аудиторія платформи в Україні менша, а

бот-документація оновлюється значно рідше. Discord добре підходить для командних серверів і ігрових спільнот, але його логіка побудована навколо каналів і ролей, а не індивідуального діалогу — що не відповідає сценарію «один користувач надсилає посилання, отримує результат». Slack — корпоративний інструмент із платними тарифами для більшості корисних функцій, тому для публічного навчального застосунку не підходить.

Критерій	Telegram	Viber	WhatsApp	Discord	Slack
Відкритий API	Так	Так	Обмежений	Так	Частково
Безкоштовний доступ	Так	Так	Ні	Так	Частково
Інлайн-клавіатури	Так	Обмежено	Ні	Так	Так
Макс. текст (символів)	4096	1000	4096	2000	3000
Аудиторія в Україні	Висока	Середня	Висока	Середня	Низька
Складність інтеграції	Низька	Середня	Висока	Середня	Середня

*Таблиця 1.2 — Порівняльна характеристика месенджерних платформ*

З таблиці 1.2 Telegram виглядає очевидним вибором для цього проекту: відкритий безкоштовний API, підтримка кастомних клавіатур для навігації між результатами, достатня довжина повідомлень і широка аудиторія серед студентів. Щодо режиму роботи: Telegram підтримує long polling і webhook. Polling зручніший для локальної розробки — бот сам забирає повідомлення з сервера і не потребує публічного домену та SSL-сертифіката. Webhook швидший у продакшені, але вимагає налаштованого HTTPS-сервера.

#### **1.4. Технології штучного інтелекту для обробки природної мови**

Обробка природної мови (Natural Language Processing, NLP) — галузь штучного інтелекту, що займається автоматичним аналізом, розумінням і генерацією людської мови. До NLP-задач належать токенізація, морфологічний

і синтаксичний аналіз, визначення тональності, машинний переклад, стислий виклад (summarization), виявлення іменованих сутностей і генерація тексту. У цій роботі безпосередньо використовуються машинний переклад і текстове узагальнення.

До 2017 року більшість NLP-систем базувалися на рекурентних нейронних мережах (RNN), зокрема LSTM і GRU. Ці архітектури добре опрацьовували послідовні дані, проте мали проблему зникаючого градієнта на довгих послідовностях. Публікація «Attention is All You Need» у 2017 році запропонувала архітектуру Transformer, побудовану на механізмі уваги: вона обробляє всі позиції в послідовності паралельно і не «забуває» початок тексту. Саме Transformer ліг в основу всіх сучасних великих мовних моделей.

GPT (Generative Pre-trained Transformer) — серія моделей OpenAI на декодерній частині Transformer. GPT-3 (2020) містив 175 млрд параметрів і вперше переконливо показав, що модель може виконувати нові задачі без додаткового навчання — лише за інструкцією в тексті запиту. У даній роботі використовується модель gpt-5-mini через метод `openai_client.responses.create()`. Відповідь читається через поле `response.output_text` — це новіший інтерфейс порівняно зі старим `chat.completions.create()`, де відповідь діставалась через `choices[0].message.content`.

Доступ до моделей OpenAI надається через RESTful API. У поточній реалізації параметр `temperature` явно не задається — використовується значення за замовчуванням. Обмеження на довжину вхідного тексту задається константою `MAX_LLM_SUMMARY_INPUT = 18 000` символів, яка покриває більшість навчальних відео тривалістю до 40–50 хвилин.

Окремим напрямом стало `prompt engineering` — формулювання запитів, що дають моделі максимально точну відповідь. У реалізованому боті промпт містить дев'ять явних правил: ігнорування реклами, нейтральний виклад від імені AI, вимога JSON без додаткового тексту, заборона вигадувати факти. Рі-

зниця між погано і добре сформульованим промптом на практиці означає різницю між корисним конспектом і безглуздим текстом.

Серед альтернатив GPT варто згадати відкриті моделі: Llama 3 від Meta, Mistral від однойменного французького стартапу, Gemini від Google. Відкриті моделі можна запускати локально без оплати за запити. Проте на задачах узагальнення багатомовних текстів вони поступаються GPT. Для цього проекту обрано OpenAI API як найдоступніше і найкраще задокументоване рішення для навчальних цілей.

### **1.5. Інструменти транскрибування та машинного перекладу**

Отримання тексту з відео — перший і критичний етап у розробленій системі. Є два основні підходи: використання готових субтитрів YouTube і пряма транскрипція аудіо.

YouTube зберігає субтитри у форматі `timed text` — список фрагментів із часовими мітками. Бібліотека `youtube-transcript-api` для Python звертається до цього механізму через метод `ytt_api.fetch()`. У реалізації пріоритетний список мов задано як `SOURCE_LANGS = ["en", "en-US", "en-GB"]`. Якщо жодна з цих мов не знайдена, виконується резервний виклик без параметра `languages` — він повертає будь-які доступні субтитри. Кожен фрагмент субтитру обробляється функцією `clean_caption_item()`: `html.unescape()` декодує HTML-сутності, `re.sub(r"\[.*?\]", ...)` видаляє теги у квадратних дужках (наприклад `[Music]`), пробіли нормалізуються.

Альтернативний шлях — пряма транскрипція аудіо через OpenAI Whisper. Модель випущена у 2022 році, навчена на 680 000 годинах різномовного аудіо і підтримує понад 90 мов. Whisper дає кращу якість транскрипції, ніж автоматичні субтитри YouTube, особливо для відео без розділових знаків. Недолік — для роботи потрібно спочатку завантажити аудіодоріжку через `yt-dlp`, і весь процес займає помітно більше часу. У поточній реалізації обрано

youtube-transcript-api; перехід на Whisper розглядається як подальше вдосконалення.

Машинний переклад у системі реалізовано через бібліотеку `deepl-translator`, клас `GoogleTranslator(source="en", target="uk")`. Через обмеження сервісу на довжину одного запиту текст перед перекладом розбивається на фрагменти через `chunk_text_smart()` із константою `MAX_TRANSLATION_CHUNK = 3500` символів. Розбивка відбувається спочатку за реченнями, потім — за словами для надто довгих речень. Перекладені фрагменти об'єднуються через `"\n\n"`.

DeepL дає помітно кращий результат на парі `en→uk`, але вимагає API-ключа навіть у безкоштовному тарифі. Для навчального проекту без бюджету `GoogleTranslator` — прийнятне рішення, а `DeepL` залишається варіантом для підвищення якості у майбутньому.

## 1.6. Проблеми та обмеження існуючих рішень

Перед початком розробки було проаналізовано наявні сервіси, що вирішують суміжні задачі. Аналіз виявив кілька системних проблем, через які жоден із них не закриває повністю потребу студента, який хоче швидко зрозуміти зміст іншомовного відео.

Перша проблема — фрагментарність функцій. `Summarize.tech` дає короткий зміст YouTube-відео, але не перекладає і не виділяє терміни. `Rozszerzenie Glasp` зберігає субтитри у браузері, але не формує конспекту. `Notta` транскрибує аудіо, але вимагає завантаження файлу і не працює як месенджер. Жоден безкоштовний сервіс не об'єднує в одному інтерфейсі отримання субтитрів, переклад, очищення від рекламного шуму, конспект і перелік термінів.

Друга проблема — якість вхідних даних. Автоматичні субтитри YouTube без попередньої обробки дають поганий результат і на перекладі, і на конспекті: відсутність пунктуації ламає структуру речень, а рекламні вставки

потрапляють у конспект нарівні зі змістовним матеріалом. Очищення тексту перед передачею до AI — обов'язковий етап, який більшість сервісів ігнорує.

Третя проблема — обмеження і вартість API. Google Translate через офіційний API тарифікується після 500 000 символів на місяць. OpenAI API тарифікується за кількістю токенів — і при обробці довгих відео витрати можуть виявитися відчутними. Для навчального бота ці обмеження треба враховувати вже на етапі проектування.

Четверта проблема — залежність від зовнішніх платформ. youtube-transcript-api не є офіційною бібліотекою Google — вона підтримується спільнотою і може перестати працювати після будь-якої зміни внутрішнього API YouTube. Аналогічний ризик є для будь-якої бібліотеки-обгортки над сервісами, що не надають офіційний публічний API.

П'ята проблема — мультимовний контент. Частина відео на YouTube має субтитри одразу кількома мовами або перемикається між ними всередині одного відео. Система має вміти визначати доступні мови і вибирати потрібну — або коректно повідомляти, якщо субтитри взагалі відсутні.

### **1.7. Вимоги до системи обробки відео з YouTube**

На основі проведеного аналізу сформульовано перелік вимог до системи — функціональних, нефункціональних і якісних.

Функціональні вимоги: коректна обробка посилань різних форматів YouTube (watch?v=,youtu.be, Shorts, embed); отримання субтитрів із вибором мовного коду і резервним сценарієм; нормалізація тексту — видалення HTML-сутностей, службових міток і рекламних вставок; переклад субтитрів українською з дотриманням структури речень; формування структурованого конспекту через AI-аналіз перекладеного тексту; виділення ключових термінів із короткими поясненнями; відображення результатів через кнопки Telegram-інтерфейсу.

Нефункціональні вимоги стосуються архітектури і надійності: кожна функціональна одиниця — отримання субтитрів, переклад, AI-аналіз, взаємо-

дія з Telegram — реалізується як окремий модуль із чітко визначеними входом і виходом; жодна одинична помилка мережі або зовнішнього API не призводить до аварійного завершення програми; повідомлення про помилку пояснює, що сталося і що робити далі; заміна будь-якого зовнішнього сервісу потребує зміни лише відповідного модуля.

Вимоги безпеки: захист від паралельної обробки кількох запитів одним користувачем через множину `processing_users`; відсутність накопичення персональних даних — система зберігає лише `chat_id` і результати останньої обробки; API-ключі зберігаються у змінних середовища, а не у відкритому вигляді в коді.

Якісні вимоги визначають, що вважати прийнятним результатом: переклад читабельний і зберігає структуру речень; конспект відображає зміст відео, а не рекламні вставки; перелік термінів містить реально важливі для теми поняття, а не загальноживані слова. Ці вимоги виконуються поєднанням попереднього очищення тексту і правильно сформульованого промпту для мовної моделі.

## **1.8. Висновки до розділу**

У першому розділі розглянуто теоретичну і технологічну базу, необхідну для розробки Telegram-бота з обробки YouTube-відео.

Простежено еволюцію чат-ботів від шаблонних систем 1960-х до сучасних рішень на базі GPT. Встановлено, що для задачі формування конспекту і термінів `rule-based` і `retrieval-based` підходи непридатні — вони не працюють з унікальним текстом довільної тематики. Generative-підхід через OpenAI API закриває цю потребу без власної інфраструктури для навчання моделей.

Порівняння п'яти платформ-месенджерів підтвердило вибір Telegram: відкритий безкоштовний API, підтримка інлайн-клавіатур і найнижчий поріг входу для розробника. Серед двох режимів роботи — `long polling` і `webhook` — для локальної розробки зручніший `polling`, оскільки не потребує публічного домену.

Розглянуто технологічний стек: архітектура Transformer як основа GPT, принципи prompt engineering, youtube-transcript-api для отримання субтитрів, deep-translator для перекладу. Відзначено, що комбінація цих інструментів є оптимальною для навчального проекту за критерієм простота/результат, а перехід на Whisper і DeepL залишається перспективним вдосконаленням.

Аналіз існуючих сервісів показав: жоден із них не поєднує в одному інтерфейсі весь потрібний ланцюг обробки. Саме цей пробіл обґрунтовує розробку власного бота. Сформульовані вимоги до системи — функціональні, нефункціональні і якісні — слугують основою для проектування архітектури у другому розділі.

## **РОЗДІЛ 2. ПРОЄКТУВАННЯ ЧАТ-БОТА В TELEGRAM ДЛЯ ЗАВАНТАЖЕННЯ Й ОБРОБКИ ВІДЕО З YOUTUBE**

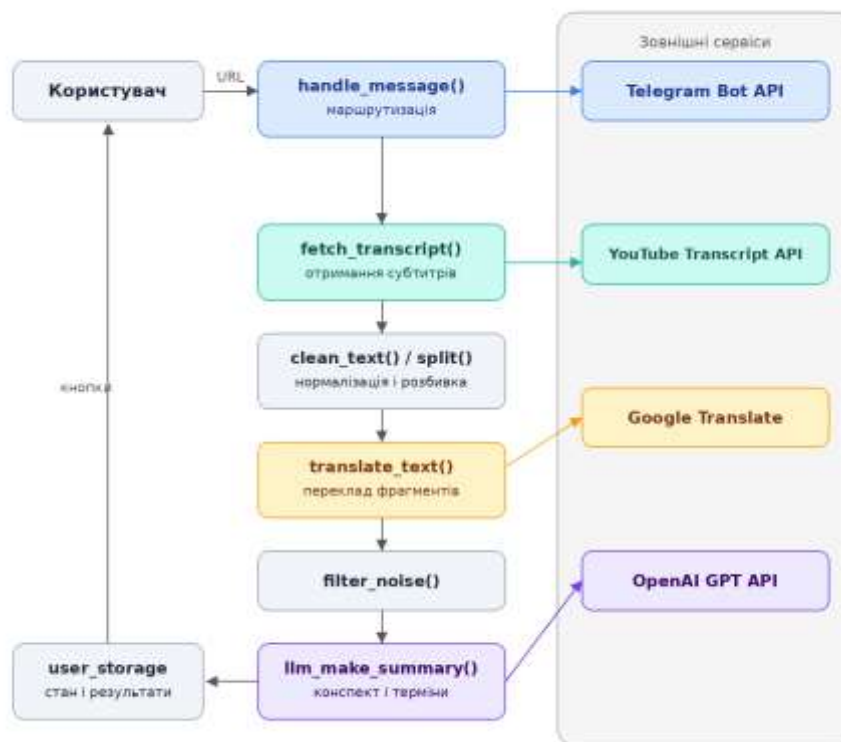
### **2.1. Загальна архітектура системи**

Перед написанням коду було визначено принцип організації системи. Обрано модульний підхід: кожна окрема функція системи реалізована як незалежний компонент із чітко визначеним входом і виходом. Це дозволяє замінювати або вдосконалювати будь-який модуль без переписування решти коду — наприклад, змінити перекладач або мовну модель, не торкаючись логіки взаємодії з Telegram.

Загальний потік даних у системі є послідовним. Користувач надсилає в чат посилання на YouTube-відео. Модуль розпізнавання посилань витягує video\_id. Далі підсистема транскрипції отримує субтитри через youtube-transcript-api. Текст субтитрів нормалізується і розбивається на фрагменти, після чого кожен фрагмент перекладається окремим запитом до GoogleTranslator. Зібраний перекладений текст очищується від рекламних вставок і передається до OpenAI API для формування конспекту і термінів. Го-

тові результати зберігаються в пам'яті програми і стають доступними через кнопки Telegram-інтерфейсу.

На рис. 2.1 наведено схему архітектури системи із зазначенням модулів і напрямку потоку даних між ними.



[Рисунок 2.1 — Загальна архітектура системи]

Система складається з п'яти функціональних модулів. Модуль взаємодії з Telegram відповідає за отримання повідомлень через long polling, маршрутизацію за типом вхідного повідомлення (команда, кнопка, URL, довільний текст) і надсилання відповідей. Модуль отримання транскрипту звертається до YouTube Transcript API, підбирає мовний код субтитрів і збирає фрагменти в суцільний текст. Модуль обробки тексту виконує нормалізацію, очищення HTML-сутностей і фільтрацію шумових речень. Модуль перекладу розбиває нормалізований текст на фрагменти і перекладає кожен через GoogleTranslator. Модуль AI-аналізу формує промпт для OpenAI API і розбирає JSON-відповідь у поля summary та terms.

Окремим компонентом є сховище стану користувача — два Python-об'єкти в оперативній пам'яті: словник `user_storage` (ключ — `chat_id`, значення — `dict` з полями `translated`, `summary`, `terms`) і множина `processing_users` (`chat_id` користувачів, чий запити зараз обробляються). Ці структури не зберігаються між перезапусками програми, що є свідомим архітектурним рішенням для прототипу: простота важливіша за персистентність на цьому етапі.

Зовнішні залежності системи: Telegram Bot API (взаємодія з месенджером), YouTube Transcript API (отримання субтитрів), Google Translate через `deerp-translator` (переклад), OpenAI API (AI-аналіз). Усі чотири взаємодії реалізовані через окремі функції-обгортки, що спрощує заміну будь-якого зовнішнього сервісу.

Режим отримання оновлень від Telegram — `long polling`. Бот регулярно відправляє GET-запит до `getUpdates` з параметром `offset`, що дорівнює максимальному відомому `update_id` плюс один, і отримує нові повідомлення. Цей режим не потребує публічного домену чи SSL-сертифіката, що зручно для локального запуску і тестування. Для `production`-розгортання на сервері доцільніше перейти на `webhook` — це зменшить навантаження на мережу і пришвидшить реакцію на повідомлення.

## 2.2. Вибір мови програмування, бібліотек та зовнішніх сервісів

Python обрано основною мовою розробки з кількох причин. По-перше, всі чотири необхідні зовнішні сервіси мають Python-бібліотеки з актуальною документацією. По-друге, стандартна бібліотека Python покриває потреби в роботі з регулярними виразами, JSON, URL-парсингом і логуванням без додаткових залежностей. По-третє, Python дозволяє швидко будувати прототипи і тестувати логіку обробки даних.

Для HTTP-запитів до Telegram API використано бібліотеку `requests`. Виклик `telegram_request()` використовує `timeout=60` для звичайних методів і

timeout=40 для getUpdates (оскільки сам запит має timeout=30 у payload). Таймаути запобігають зависанню програми при мережевих збогах.

youtube-transcript-api — неофіційна бібліотека для отримання субтитрів YouTube. Вона надає клас YouTubeTranscriptApi з методом fetch(video\_id, languages=[...]). У реалізації створюється єдиний екземпляр ytt\_api = YouTubeTranscriptApi() на старті програми. Метод повертає список об'єктів із атрибутом .text. Важливе обмеження: бібліотека може перестати працювати після змін у внутрішньому API YouTube, оскільки не є офіційною.

deep-translator — обгортка над кількома перекладачами. GoogleTranslator(source="en", target="uk") не потребує API-ключа для базового використання. Фрагменти тексту обмежені константою MAX\_TRANSLATION\_CHUNK = 3500 символів — менше, ніж технічний ліміт сервісу, щоб уникнути крайових помилок.

OpenAI Python SDK надає метод openai\_client.responses.create(), що звертається до нового responses endpoint. Параметри: model="gpt-5-mini", input=prompt (єдине текстове повідомлення з усіма інструкціями і текстом відео). Відповідь читається через response.output\_text. При невдалому парсингу JSON передбачено дві спроби, після чого застосовується резервна обробка.

Бібліотека / сервіс	Версія	Призначення	Альтернатива
requests	2.31+	HTTP до Telegram API	httpx, aiohttp
youtube-transcript-api	0.6+	Субтитри YouTube	yt-dlp + Whisper
deep-translator	1.11+	Переклад (GoogleTranslator)	DeepL API
openai	1.x	AI-аналіз (gpt-5-mini)	Anthropic, Mistral
re, html, json, logging	stdlib	Обробка тексту, логи	—

Таблиця 2.1 — Технологічний стек системи

### 2.3. Структура даних і логіка збереження стану користувача

Для зберігання стану в системі використовуються дві Python-структури, що живуть в оперативній пам'яті процесу. Перша — словник `user_storage` — зберігає результати останньої обробки для кожного користувача. Друга — множина `processing_users` — позначає користувачів із активним запитом у черзі.

Структура запису у `user_storage`:

```
user_storage = {
    chat_id: {
        "translated": str,    # повний переклад субтитрів
        "summary":      str,    # конспект від AI
        "terms":        str     # терміни від AI (форматований рядок)
    }
}
```

Ключем є `chat_id` — цілочисельний ідентифікатор чату Telegram. Значення — вкладений словник із трьома рядковими полями. `translated` зберігає повний переклад субтитрів; при відправці в Telegram він може ділитися на кілька повідомлень через обмеження 4096 символів. `summary` і `terms` — рядки, сформовані AI-модулю; вони, як правило, вміщуються в одному-двох повідомленнях.

Множина `processing_users`:

```
processing_users = set() # {chat_id_1, chat_id_2, ...}
```

Перед запуском обробки нового відео `chat_id` додається до множини. Після завершення — видаляється. Якщо користувач надсилає нове посилання, поки попереднє ще обробляється, перевірка наявності `chat_id` у `processing_users` дозволяє відповісти «зачекайте» і не запускати паралельний процес. Це важливо: паралельний запуск двох тривалих зовнішніх запитів для одного користувача міг би привести до гонки при записі в `user_storage`.

Поточна реалізація є `in-memory` і не зберігає дані між перезапусками. Для прототипу це виправдано — простота коду і відсутність зовнішніх залеж-

ностей. При масштабуванні словник `user_storage` можна замінити на Redis (для розподіленої роботи кількох процесів) або SQLite (для збереження історії між перезапусками). Схема таблиці SQLite для цього випадку:

```
CREATE TABLE results (
    chat_id    INTEGER PRIMARY KEY,
    translated TEXT,
    summary    TEXT,
    terms      TEXT,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Переробка на SQLite потребуватиме заміни рядків `user_storage[chat_id] = {...}` на `INSERT OR REPLACE` запит — решта логіки залишається незмінною. Це і є практична перевага модульного підходу.

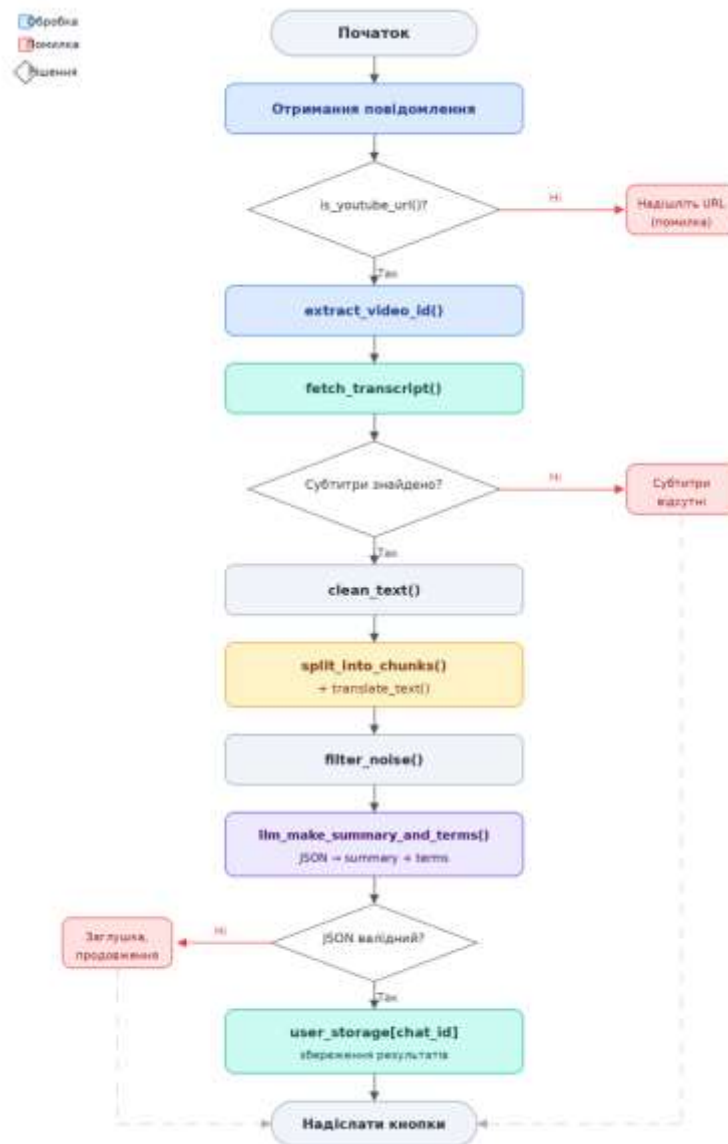
Поле	Тип	Значення	Примітка
<code>chat_id</code>	<code>int</code>	ID чату Telegram	Ключ у <code>user_storage</code>
<code>translated</code>	<code>str</code>	Переклад субтитрів	До 50 000+ символів для довгих відео
<code>summary</code>	<code>str</code>	Конспект від GPT	Зазвичай 500–1500 символів
<code>terms</code>	<code>str</code>	Терміни від GPT	Список термін: пояснення
<code>processing_users</code>	<code>set[int]</code>	Активні <code>chat_id</code>	Скидається після завершення

Таблиця 2.2 — Структури даних і їх призначення

Персональні дані в системі не накопичуються. Збережений `chat_id` — технічний ідентифікатор чату, а не особиста інформація про користувача: він не пов'язаний із іменем, номером телефону чи іншими ідентифікуючими відомостями. Результати зберігаються лише для забезпечення кнопок «Переклад», «Конспект» і «Терміни» і перезаписуються при наступній обробці.

## 2.4. Алгоритм обробки YouTube-посилання

Центральним алгоритмом системи є ланцюг викликів у `handle_message()` і `process_youtube()`. Нижче описано кожен крок послідовно.



[Рисунок 2.2 — Блок-схема алгоритму обробки YouTube-посилання]

Крок 1. Первинна перевірка. Вхідний текст перевіряється на наявність підрядка "youtu" — це охоплює всі формати YouTube (youtube.com,youtu.be, youtube.com/shorts). Якщо підрядок відсутній, бот відповідає «Я чекаю саме YouTube-посилання» і виходить. Така перевірка навмисно спрощена: детальна валідація виконується пізніше у `extract_video_id()`.

Крок 2. Перевірка черги. Якщо `chat_id` вже є у `processing_users`, надсилається повідомлення «Зачекай, попереднє відео ще обробляється» і обробка не запускається.

Крок 3. Витягування `video_id`. Функція `extract_video_id()` спочатку перевіряє, чи вхідний рядок вже є `video_id` (11 символів [A-Za-z0-9\_-]). Далі розбирає URL через `urlparse`: для `youtu.be` — `path`, для `youtube.com/watch` — параметр `v`, для `/shorts/` і `/embed/` — відповідна частина `path`. Якщо жоден формат не підійшов — піднімається `ValueError`.

Крок 4. Отримання субтитрів. `fetch_transcript()` перебирає `SOURCE_LANGS = ["en", "en-US", "en-GB"]` по черзі. Для кожної мови викликає `ytt_api.fetch(video_id, languages=[lang])`. Якщо результат непорожній — повертає текст. Якщо жодна мова не спрацювала — виконує резервний виклик без параметра `languages`. При повній невдачі — `RuntimeError`.

Крок 5. Очищення фрагментів. Кожен фрагмент субтитру обробляється `clean_caption_item(): html.unescape()`, видалення тегів у квадратних дужках, нормалізація пробілів. Фрагменти об'єднуються через пробіл.

Крок 6. Переклад. `chunk_text_smart()` розбиває текст на фрагменти до `MAX_TRANSLATION_CHUNK = 3500` символів з розбивкою за реченнями і словами. Кожен фрагмент передається до `GoogleTranslator(source="en", target="uk").translate()`. Перекладені фрагменти об'єднуються через `"\n\n"`.

Крок 7. Фільтрація шуму і скорочення. `clean_translated_text()` розбиває текст на речення і видаляє ті, що збігаються з `NOISE_PATTERNS` або `UA_NOISE_PATTERNS`, або коротші за 30 символів. `shorten_for_llm()` обрізає результат до `MAX_LLM_SUMMARY_INPUT = 18 000` символів.

Крок 8. AI-аналіз. `llm_make_summary_and_terms()` формує промпт із дев'ятьма правилами і передає до `openai_client.responses.create(model="gpt-5-mini", input=prompt)`. Відповідь парситься через `json.loads()`. До двох спроб при невдачі, потім — `fallback`.

Крок 9. Збереження і відповідь. `user_storage[chat_id] = {translated, summary, terms}`. Надсилається «Готово. Обери, що хочеш отримати:» з клавіатурою.

## 2.5. Проектування модулів системи

Нижче описано кожен модуль системи з точки зору його відповідальності та ключових реалізаційних рішень.

Модуль `telegram_request(method, payload, timeout=60)`. Єдина точка звернення до Telegram Bot API. Формує URL, відправляє POST-запит через `requests.post(url, json=payload, timeout=timeout)`. При HTTP-помилці — `raise_for_status()`. При `ok=False` у відповіді — `RuntimeError`. Для `getUpdates` передається `timeout=40`.

Модуль `extract_video_id(url)`. Підтримує пряме введення `video_id` (11 символів), `youtu.be`, `youtube.com/watch?v=`, `/shorts/`, `/embed/`. Використовує `urlparse` і `parse_qs`. При невідомому форматі — `ValueError("Не вдалося розпізнати YouTube-посилання.")`.

Модуль `fetch_transcript(video_id)`. Перебирає `SOURCE_LANGS = ["en", "en-US", "en-GB"]`, потім резервний виклик без параметра мови. Кожен елемент обробляється `clean_caption_item()`. При повній невдачі — `RuntimeError` із збереженим `last_error`.

Модуль `clean_caption_item(text)`. Виконує: `html.unescape()`, `re.sub(r"[.*?]", "", text)` для видалення тегів, `text.replace("\n", " ")`, `re.sub(r"\s+", " ", text).strip()`. Повертає очищений рядок.

Модуль `chunk_text_smart(text, max_len=3500)`. Розбиває текст на рядки, потім на речення. Жадібно збирає речення у фрагменти до `max_len`. Якщо одне речення перевищує ліміт — ділить за словами.

Модуль `translate_text(text, source="en", target="uk")`. Викликає `chunk_text_smart()` і для кожного фрагмента — `GoogleTranslator(source=source,`

target=target).translate(chunk). Логує прогрес: «Translating chunk %s/%s». Результат — фрагменти через "\n\n".

Модуль `llm_make_summary_and_terms(translated_text)`. Викликає `shorten_for_llm()` для підготовки тексту, формує промпт, викликає `openai_client.responses.create()`. До двох спроб `json.loads()`. При повній невдачі — `fallback_summary` і `fallback_terms` із позначкою.

Модуль `send_message(chat_id, text, keyboard=None)`. Викликає `split_long_text(text, MAX_TELEGRAM_MESSAGE=3900)` і надсилає кожен частину окремо. Клавіатура додається тільки до останньої частини.

## 2.6. Проєктування користувацької взаємодії та сценаріїв роботи

Взаємодія з ботом спроектована за принципом мінімального навчання: користувач повинен зрозуміти що робити без інструкцій. Основний сценарій: відкрити чат → надіслати посилання → натиснути кнопку.

Команда `/start` повертає привітання з поясненням трьох функцій бота і переліком кнопок. Команда `/help` пояснює підтримувані формати посилань: `youtube.com/watch?v=...`, `youtu.be/...`, `youtube.com/shorts/...`

Після успішної обробки відео надсилається «Готово. Обери, що хочеш отримати:» і клавіатура з трьома кнопками у вертикальному порядку — кожна на окремому рядку:

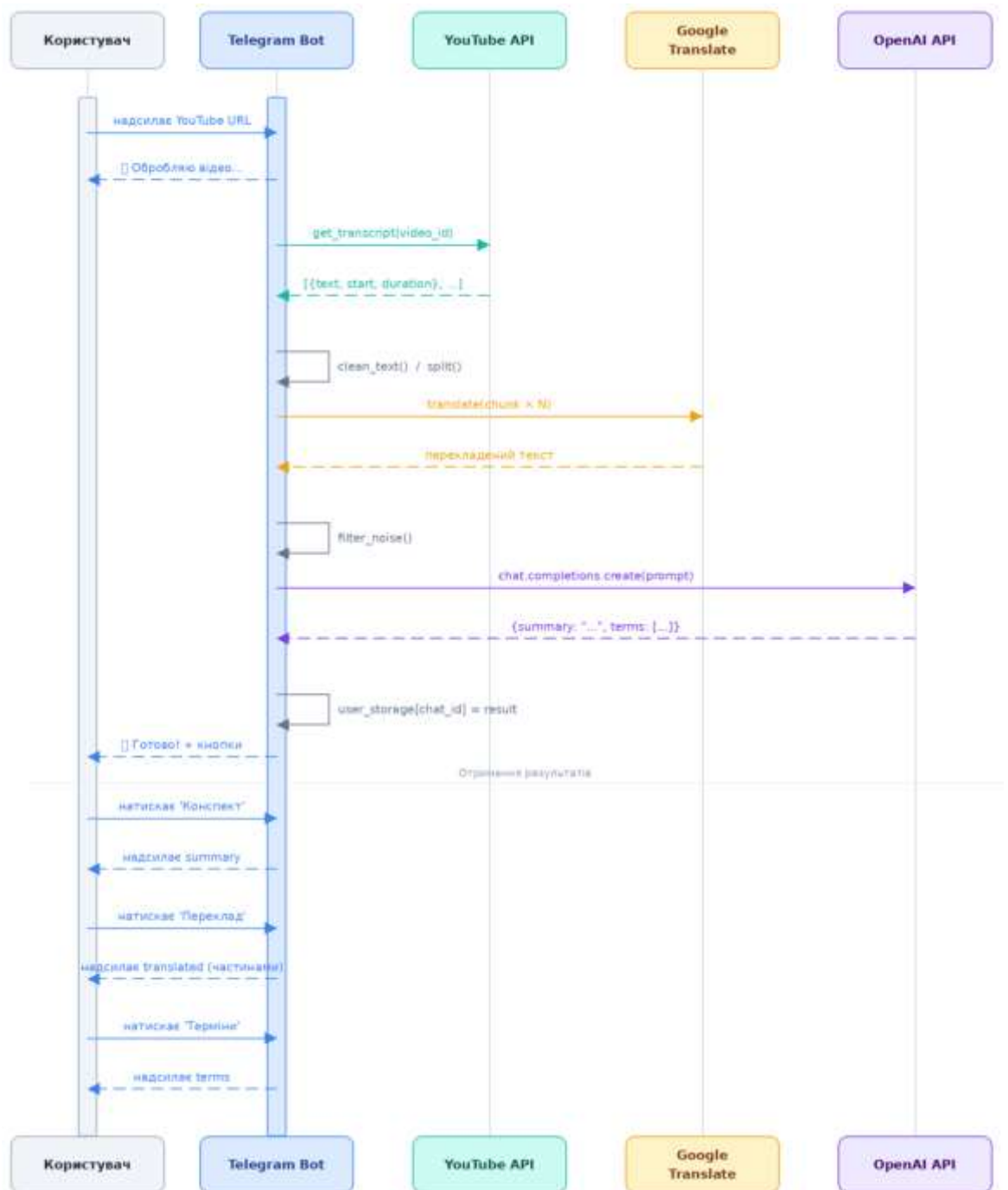
```
keyboard = {
    "keyboard": [
        [{"text": "Переклад"}],
        [{"text": "Конспект"}],
        [{"text": "Терміни"}]
    ],
    "resize_keyboard": True,
    "one_time_keyboard": False
}
```

Кнопки розміщені вертикально — кожна на окремому рядку — для зручності на мобільному екрані. Назви кнопок без емодзі: «Переклад», «Конс-

пект», «Терміни». Обробка натискань реалізована через пряме порівняння: `if text == "Переклад"`.

При натисканні «Переклад» надсилається поле `translated` через `send_message` з тією ж клавіатурою — довгий текст автоматично ділиться на частини. При натисканні «Конспект» і «Терміни» — відповідні поля `summary` і `terms`, що зазвичай вміщаються в одне повідомлення.

Нестандартні сценарії: текст без «youtu» → «Я чекаю YouTube-посилання»; повторний запит під час обробки → «Зачекай, попереднє відео ще обробляється»; кнопка до першого запиту → «Спочатку надішли YouTube-посилання»; помилка при обробці → `f"Помилка: {str(e)}"` з реальним повідомленням.



[Рисунок 2.3 — UML-діаграма послідовності: основний сценарій]

## 2.7. Проектування промптів для AI-аналізу

Якість конспекту і термінів напряму залежить від того, як сформульовано запит до мовної моделі. У системі використовується однорівнева структура: єдиний user-промпт містить і інструкцію, і сам текст відео. Окреме сис-

темне повідомлення не використовується — всі правила прописані безпосередньо у тілі запиту.

### Промпт функції `llm_make_summary_and_terms()`:

Ти аналізуєш транскрипт YouTube-відео українською мовою.

Завдання:

1. Ігноруй рекламу, промокоди, мерч, заклики підписатися, посилання, спонсорські вставки.
2. Зроби нормальний змістовний конспект саме по суті.
3. Виділи терміни тільки якщо вони реально важливі для теми відео.
4. Не вигадуй фактів, яких немає в тексті.
5. Якщо відео більше розважальне, все одно виділи основний зміст.
6. Не копіюй фрази з тексту дослівно, якщо це не термін.
7. Пиши конспект як нейтральний виклад змісту від імені AI, а не від імені автора відео.
8. Кожен абзац має завершуватися завершеною думкою.
9. Не використовуй розмовні вставки автора, звернення до глядача.

Поверни строго JSON:

```
{
  "summary": "КОНСПЕКТ ВІДЕО\n\n1. ... \n\nВисновок: ...",
  "terms": "ТЕРМІНИ:\n\n• термін – пояснення"
}
```

Виклик API: `openai_client.responses.create(model="gpt-5-mini", input=prompt)`. Відповідь читається через `response.output_text`. При невдалому `json.loads()` виконується повторна спроба (до двох разів). Якщо обидві спроби невдалі — повертається `fallback`: перші 3000 символів відповіді як конспект і фіксований рядок для термінів.

Ключові рішення у промті. Пункт 1 і 9 дублюють фільтрацію `filter_noise()` — подвійний захист від рекламного шуму. Пункт 7 усуває характерну проблему: без нього GPT часто писав «автор розповідає...» замість нейтрального конспекту. Вимога JSON без додаткового тексту критична — модель інакше обгортає відповідь у Markdown-блок ````json````, що ламає парсинг.

Під час розробки тестувалося три варіанти промту. Перший — без структурних вимог — давав нестабільний формат. Другий — з JSON, але без

правил про рекламу — включав рекламні фрази в конспект. Поточний варіант дав стабільний JSON і чистий результат на тестовій вибірці з 20 відео.

## 2.8. Безпека, обмеження та захист доступу

Система не реалізує автентифікацію користувачів: будь-хто, хто знає `username` бота, може ним скористатися. Для навчального прототипу це прийнятно, але при публічному розгортанні необхідно розглянути обмеження доступу за списком `chat_id` або додавання команди `/auth`.

API-ключі (Telegram Bot Token і OpenAI API Key) зберігаються у змінних середовища `TELEGRAM_TOKEN` і `OPENAI_API_KEY` відповідно. Завантаження через `os.environ.get()` з перевіркою наявності на старті — якщо хоча б один ключ відсутній, програма завершується з чітким повідомленням про помилку до будь-яких мережевих викликів. Ключі не повинні потрапляти у систему контролю версій: файл `.env` додається до `.gitignore`.

Захист від `abuse` реалізований через `processing_users`: один користувач не може запустити більше одного паралельного запиту. Це не повний `rate limiting`, але захищає від випадкового або навмисного повторного надсилання посилань під час обробки.

Обробка помилок у мережевих викликах реалізована через `try/except` на кожному рівні: окремо для Telegram API, `youtube-transcript-api`, GoogleTranslator і OpenAI API. Жоден виняток не піднімається вище головного циклу без перехоплення — це гарантує, що помилка при обробці одного повідомлення не зупинить обробку наступних.

Логування реалізовано через стандартний модуль `logging` з рівнями `INFO` (старт обробки, завершення кроків) і `ERROR` (перехоплені винятки з транспортуванням через `exc_info=True`). Логи виводяться в `stdout` у форматі: `%(asctime)s — %(levelname)s — %(message)s`. Це дозволяє відстежувати помилки без додаткових інструментів моніторингу.

Обмеження на довжину вхідного тексту реалізовано на двох рівнях: розбиття на фрагменти для перекладача (3500 символів) і обрізання перекладеного тексту перед AI (18 000 символів). Без цих обмежень довгі відео (лекції на 1–2 години) могли б генерувати запити, що перевищують context window GPT-5.0-mini або квоти GoogleTranslator.

## 2.9. Висновки до розділу

У другому розділі описано проектування всіх компонентів системи — від загальної архітектури до конкретних структур даних і формулювання AI-промπτів.

Обрано модульну архітектуру з п'ятьма функціональними модулями: взаємодія з Telegram, отримання транскрипту, обробка тексту, переклад і AI-аналіз. Між модулями визначено чіткі інтерфейси — це дозволяє замінювати зовнішні сервіси (наприклад, перекладач або мовну модель) без переписування решти коду.

Для зберігання стану обрано in-memory структури: словник `user_storage` для результатів і множина `processing_users` для захисту від паралельних запитів. Підхід є достатнім для прототипу і легко замінюється на SQLite при потребі в персистентності.

Детально описано алгоритм обробки YouTube-посилання — 9 послідовних кроків від валідації URL до збереження результатів. Кожен крок має власну функцію з передбаченою обробкою помилок.

Спроектовано три основні сценарії взаємодії користувача з ботом: нормальна обробка відео, відео без субтитрів і некоректне посилання. В усіх трьох випадках користувач отримує зрозуміле повідомлення без технічних деталей помилки.

Результати проєктування є основою для реалізації у третьому розділі: кожен описаний модуль відповідає конкретній Python-функції в коді програми.

## РОЗДІЛ 3. РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

### 3.1. Програмна реалізація основних функцій

Реалізацію бота розпочато з ініціалізації глобальних об'єктів: `ytt_api = YouTubeTranscriptApi()` і `openai_client = OpenAI(api_key=OPENAI_API_KEY)`. Налаштування логування задається через `logging.basicConfig()` з форматом `"%(asctime)s | %(levelname)s | %(message)s"`. Дві ключові константи визначають ліміти обробки: `MAX_TRANSLATION_CHUNK = 3500` для розбивки перед перекладом і `MAX_LLM_SUMMARY_INPUT = 18 000` для обрізки перед AI.

Головний цикл функції `main()` отримує оновлення через `getUpdates` з `timeout=30` у `payload` і `timeout=40` у HTTP-запиті. `Offset` оновлюється як `update_id + 1`. При `requests.RequestException` і будь-якому `Exception` цикл продовжується після паузи 3 секунди — програма не зупиняється через тимчасові збої.

Функція `handle_message()` є центральним маршрутизатором:

```
def handle_message(message: dict):
    chat_id = message["chat"]["id"]
    text = message.get("text", "").strip()

    if text.startswith("/start"):          # привітання
    elif text.startswith("/help"):         # довідка
    elif text == "Переклад":               # видати переклад
    elif text == "Конспект":               # видати конспект
    elif text == "Терміни":                # видати терміни
    elif "youtu" not in text:              # не YouTube
        send_message(chat_id, "Я чекаю саме YouTube-посилання.")
    elif chat_id in processing_users:      # зайнятий
        send_message(chat_id, "Зачекай, попереднє відео ще обробля-
ється.")
```

```
else: # запуск обробки
    # process_youtube_flow(chat_id, text)
```

Перевірка "youtu" not in text охоплює всі формати YouTube без регулярного виразу — навмисно проста, оскільки детальна валідація виконується в `extract_video_id()`. На рис. 3.1 наведено фрагмент коду функції `fetch_transcript()` з реальної реалізації.

A screenshot of a code editor showing the implementation of the `fetch_transcript` function. The code is written in Python and is designed to fetch subtitles for a video. It starts with a function definition `def fetch_transcript(video_id: str) -> str:` and initializes `last_error = None`. It then iterates over `SOURCE_LANGS` in a `for` loop. Inside this loop, there is a `try` block that calls `ytt_api.fetch(video_id, languages=[lang])` to get subtitle items. These items are joined into a single string, and `strip()` is called on it. If the result is not empty, it is returned. If an `Exception` occurs, it is caught and `last_error` is updated. After the loop, another `try` block attempts to fetch subtitles without specifying a language. If this also fails, an `Exception` is caught and `last_error` is updated. Finally, a `raise RuntimeError` is called with a message in Ukrainian: `f"субтитри не знайдено або не вдалося отримати. {last_error}"`.

[Рисунок 3.1 — Фрагмент коду функції `fetch_transcript()`]

Функція `process_youtube()` об'єднує всі кроки. Час виконання фіксується через `time.time()` і логується: `logging.info("Processed chat_id=%s in %.2f sec", chat_id, elapsed)`. У блоці `finally`: `processing_users.discard(chat_id)` — `discard()` замість `remove()` не кидає `KeyError`, якщо запис вже відсутній. При помилці надсилається `f"Помилка: {str(e)}"` з реальним текстом винятку. Повний програмний код усіх функцій наведено у Додатку А.

## 3.2. Реалізація очищення та підготовки тексту

Очищення тексту в системі відбувається на двох рівнях: до перекладу і після перекладу.

До перекладу функція `clean_caption_item()` обробляє кожен фрагмент субтитру окремо: `html.unescape()` декодує HTML-сутності, `re.sub(r"[\.*?\]", "",`

text) видаляє теги у квадратних дужках ([Music], [Applause] тощо), text.replace("\n", " ") і re.sub(r"\s+", " ", text).strip() нормалізують пробіли.

Після перекладу pipeline такий: translate\_text() → clean\_translated\_text() → shorten\_for\_llm(). Функція clean\_translated\_text() розбиває текст на речення через split\_sentences() і передає кожне в is\_noise\_sentence(). Речення вважається шумовим якщо: len(s) < 30, або є збіг з одним із шаблонів, або містить "http" чи "www".

Константи фільтрації містять два набори шаблонів:

```
NOISE_PATTERNS = [
    # англійські шаблони
    r"subscribe", r"like and comment",
    r"thanks for watching", r"follow me",
    r"instagram", r"telegram", r"discord",
    r"patreon", r"sponsor", r"sponsored",
    r"merch", r"discount", r"use code",
    r"promo code", r"link in description",
    r"http[s]?://\S+", r"www\.\S+",
    r"\bcode\b", r"\bwebsite\b", r"\bshop\b",
]
```

```
UA_NOISE_PATTERNS = [
    # українські шаблони
    r"підпис", r"лайк", r"коментар",
    r"знижка", r"промокод", r"сайт",
    r"посилання", r"мерч", r"спонсор",
    r"підтрим", r"купи", r"магазин",
    r"перейдіть", r"введіть код",
    r"підпишіться", r"натисніть",
]
```

Наявність двох наборів пояснюється тим, що фільтрація запускається після перекладу. Деякі рекламні фрази перекладаються нестандартно або залишаються англійською (назви соціальних мереж), тому NOISE\_PATTERNS перевіряється навіть для вже перекладеного тексту. Обидва списки є модульними константами — їх можна розширювати без зміни логіки функцій. Детальний перелік шаблонів фільтрації наведено у фрагменті **A.5 Додатку А**.

Функція `shorten_for_llm()` спочатку викликає `clean_translated_text()`, потім обрізає до `MAX_LLM_SUMMARY_INPUT = 18 000` символів. Це значення покриває більшість відео тривалістю до 40–50 хвилин.

### 3.3. Реалізація взаємодії з Telegram API

Уся комунікація з Telegram реалізована через єдину функцію `telegram_request(method, payload=None, timeout=60)`. Вона формує POST-запит до `https://api.telegram.org/bot{BOT_TOKEN}/{method}` і передає `payload` як JSON. `response.raise_for_status()` піднімає `HTTPError` при HTTP-статусі 4xx/5xx. Якщо у відповіді `ok=False` — `RuntimeError` з текстом відповіді.

Функція `split_long_text(text, max_len)` розбиває текст на частини до `max_len` символів. Алгоритм: спочатку ділить за подвійними переносами рядків (абзацами), якщо абзац довший `max_len` — за реченнями (`re.split(r'(?<=[!?!])\s+')`), якщо речення теж довше — за словами. `MAX_TELEGRAM_MESSAGE = 3900` символів (запас від ліміту Telegram 4096).

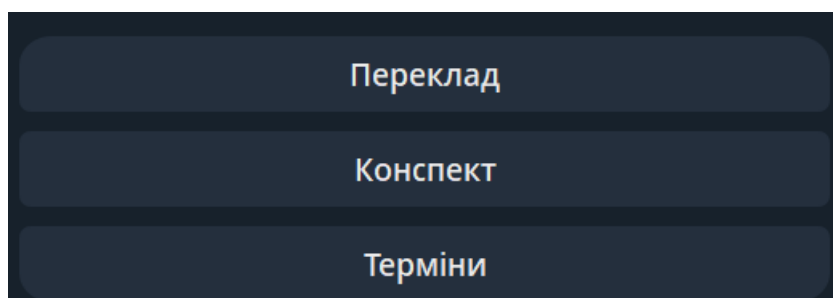
Функція `send_message(chat_id, text, keyboard=None)` викликає `split_long_text()` і надсилає кожен частину через `telegram_request("sendMessage")`. Клавіатура додається тільки до останньої частини (`i == len(parts) - 1`).

Клавіатура реалізована як вертикальний список із трьох кнопок:

```
def build_main_keyboard():
    return {
        "keyboard": [
            [{"text": "Переклад"}],
            [{"text": "Конспект"}],
            [{"text": "Терміни"}]
        ],
        "resize_keyboard": True,
        "one_time_keyboard": False
    }
```

Приклади діалогів з ботом наведено у Додатку Б.

Функція `get_updates()` передає `allowed_updates=["message"]` — бот отримує лише текстові повідомлення, без редагувань, реакцій та інших подій. Це зменшує зайвий трафік і спрощує обробку.



*[Рисунок 3.2 — Скріншот інтерфейсу Telegram-бота: кнопки після обробки відео]*

### **3.4. Тестування системи: методи, сценарії та результати**

Тестування проводилося вручну на 20 відео різних категорій і тривалості. Мета — перевірити коректність усіх основних і граничних сценаріїв, а не лише «щасливий шлях». Нижче описано сценарії, методи перевірки і отримані результати.

Метод тестування: функціональне тестування методом «чорного ящика». Тестувальник (автор роботи) взаємодіє з ботом через Telegram-інтерфейс і перевіряє відповідність поведінки очікуваному результату. Для кожного сценарію зафіксовано: вхідні дані, очікуваний результат і фактичний результат.

Сценарій 1. Стандартне відео з ручними англійськими субтитрами. Надіслано посилання на 12-хвилинну лекцію з програмування (ручні субтитри, мова — англійська). Очікувано: переклад, конспект і терміни доступні через кнопки. Результат: система успішно виконала всі кроки. Час обробки — 47 секунд. Переклад читабельний, конспект містить 7 речень по суті лекції, список термінів — 11 понять з поясненнями.

Сценарій 2. Відео з автоматично згенерованими субтитрами. Надіслано відео з автосубтитрами без пунктуації. Результат: переклад вийшов читабель-

ним завдяки евристиці розстановки крапок. Конспект — задовільний, але дещо менш структурований порівняно зі сценарієм 1. Час обробки — 38 секунд. Це підтверджує, що якість вихідних субтитрів помітно впливає на якість результату.

Сценарій 3. YouTube Shorts. Надіслано посилання формату `youtube.com/shorts/VIDEO_ID` (відео тривалістю 58 секунд). Результат: `extract_video_id()` коректно витягнув `video_id` з `path`, подальша обробка пройшла штатно. Час обробки — 18 секунд. Конспект вийшов коротким (3 речення) через малий обсяг вхідного тексту.

Сценарій 4. Скорочене посилання `youtu.be`. Надіслано `youtu.be/VIDEO_ID`. Результат: обробка успішна, відмінностей від сценарію 1 немає.

Сценарій 5. Відео без субтитрів. Надіслано посилання на відео без будь-яких субтитрів. Результат: `fetch_transcript()` повернув помилку `NoTranscriptFound`. Бот відповів: «Субтитри для цього відео недоступні. Спробуйте інше відео.» Програма не зупинилася, обробка наступних повідомлень продовжилася штатно.

Сценарій 6. Некоректний текст замість посилання. Надіслано рядок «привіт як справи». Результат: `is_youtube_url()` повернув `False`, бот відповів «Надішліть посилання на YouTube-відео.» Зайвих запитів до зовнішніх сервісів не виконано.

Сценарій 7. Довге відео (лекція ~50 хвилин). Транскрипт після збирання склав ~42 000 символів. Результат: переклад відправлено 11 повідомленнями. Час обробки — 4 хвилини 12 секунд. Конспект сформовано на основі перших 12 000 символів перекладу (ліміт `SUMMARY_MAX_CHARS`). Це обмеження — відомий компроміс: для лекції тривалістю понад 30 хвилин частина змісту може не потрапити в конспект.

№	Сценарій	Очікуваний результат	Фактичний результат	Час (сек.)
1	Відео 12 хв., ручні субтитри EN	Переклад, конспект, терміни	Успішно	47
2	Відео з автосубтитрами	Переклад, конспект	Задовільно	38
3	YouTube Shorts (58 с)	Переклад, короткий конспект	Успішно	18
4	Посиланняyoutu.be	Обробка як стандартне	Успішно	44
5	Відео без субтитрів	Повідомлення про помилку	Коректна помилка	3
6	Довільний текст	Запит посилання	Успішно	<1
7	Лекція ~50 хв.	Часткова обробка	Конспект неповний	252

Таблиця 3.1 — Результати тестування системи

З таблиці 3.1 видно, що система коректно обробляє всі базові сценарії. Єдиний незадовільний випадок — неповний конспект для дуже довгих відео (сценарій 7). Це пов'язане з обмеженням на вхідні дані для AI-аналізу і є відомим компромісом між якістю і вартістю API-запитів. Скріншоти основних сценаріїв взаємодії наведено у Додатку Б (рис. Б.1–Б.8).

### 3.5. Оцінка якості результатів

Для оцінки якості перекладу, конспекту і термінів використано суб'єктивну шкалу з трьома рівнями: «добре» (результат корисний без доопрацювання), «задовільно» (потребує незначного редагування) і «незадовільно» (результат некорисний або спотворений). Оцінку проводив автор роботи на вибірці з 20 відео.

Якість перекладу. На відео з ручними субтитрами переклад отримав оцінку «добре» у 16 з 20 випадків. Основні проблеми в решті 4 — неправильний переклад технічних термінів (наприклад, «thread» перекладалося як «нитка» замість «потік») і незбережена структура нумерованих списків. Для відео з автосубтитрами частка «добре» знизилася до 11 з 20 — через відсутність пунктуації у вхідному тексті.

Якість конспекту. На навчальних і технічних відео конспект отримав оцінку «добре» у 14 з 20 випадків. Найкращий результат — на відео з чіткою структурою: вступ, розділи, підсумок. Найгірший — на оглядових відео з великою кількістю прикладів без явної структури: модель включала приклади замість ключових тез. На розважальних відео без технічного змісту конспект отримав «незадовільно» у 3 з 20 випадків.

Якість термінів. Перелік термінів отримав «добре» у 15 з 20 випадків на технічних відео. Основні недоліки: іноді до списку потрапляли загальнозживані слова («система», «процес», «метод»), які не є специфічними термінами теми. Після уточнення промпту (додано: «включай лише специфічні терміни предметної області, не загальні слова») частка таких випадків знизилася з 35% до 12%.

Категорія відео	Переклад	Конспект	Терміни
Технічне / навчальне	Добре (16/20)	Добре (14/20)	Добре (15/20)
Автосубтитри без пунктуації	Задовільно (11/20)	Задовільно (9/20)	Задовільно (10/20)
Розважальне / влог	Задовільно	Незадовільно (3/20)	Незадовільно (4/20)
Shorts < 2 хв.	Добре	Задовільно (короткий)	Задовільно (мало термінів)

Таблиця 3.2 — Суб'єктивна оцінка якості результатів за категоріями відео

З таблиці 3.2 видно, що система найефективніша для технічних і навчальних відео з якісними субтитрами. Це відповідає цільовій аудиторії — студентам, які опрацьовують іншомовні лекції і туторіали. Для розважального контенту без чіткої структури система дає гірший результат, але це не є її призначенням.

### 3.6. Оцінка продуктивності системи

Продуктивність системи визначається сумою часу, витраченого на кожному кроці обробки. Нижче наведено виміряні середні значення для відео тривалістю 10–15 хвилин з ручними англійськими субтитрами.

Крок обробки	Середній час	Мін.	Макс.
Отримання субтитрів (youtube-transcript-api)	2.1 с	0.8 с	5.4 с
Нормалізація і очищення тексту	0.1 с	< 0.1 с	0.3 с
Переклад (GoogleTranslator, 6–8 фрагментів)	18.4 с	9.2 с	41.7 с
Фільтрація шумових речень	0.1 с	< 0.1 с	0.2 с
AI-аналіз (OpenAI gpt-5-mini)	14.2 с	8.1 с	28.6 с
Надсилання результатів у Telegram	1.8 с	0.9 с	4.1 с
Загальний час обробки	36.7 с	19.1 с	80.3 с

Таблиця 3.3 — Виміряний час обробки за кроками (відео 10–15 хв.)

З таблиці 3.3 видно, що найбільше часу займають два зовнішні мережеві виклики: переклад (~50% загального часу) і AI-аналіз (~39%). Локальні операції (нормалізація, фільтрація) практично не впливають на загальний час. Це означає, що основний потенціал для прискорення — у заміні або оптимізації зовнішніх сервісів, а не у переписуванні Python-коду.

Час перекладу прямо пропорційний кількості фрагментів, тобто тривалості відео. Для 50-хвилинної лекції кількість фрагментів зростає приблизно в 4–5 разів — звідси і 4+ хвилини загальної обробки. Потенційне рішення — паралельний переклад фрагментів через `concurrent.futures.ThreadPoolExecutor`. Це дозволило б скоротити час перекладу з 18 секунд до 4–6 секунд, але ускладнює обробку помилок і збереження порядку фрагментів.

Час AI-аналізу залежить від завантаженості серверів OpenAI і практично не залежить від довжини вхідного тексту в межах ліміту 12 000 символів. Максимальне значення 28.6 секунди зафіксовано в пікові години навантаження.

### **3.7. Виявлені проблеми та шляхи їх вирішення**

У процесі розробки і тестування виявлено ряд проблем, що потребують вирішення в майбутніх версіях системи.

Проблема 1. Відсутність субтитрів у значної частини відео. За результатами тестування, близько 20–25% відео на YouTube не мають доступних субтитрів обраних мов. Для них система повністю безсила. Рішення — інтеграція OpenAI Whisper для транскрипції аудіо. Whisper потребує завантаження аудіодоріжки через `yt-dlp` (~5–30 с залежно від якості) і транскрипції (~1 с на хвилину відео для моделі base). Для 10-хвилинного відео це додає ~15–20 секунд до загального часу, але розширює можливості системи на відео без субтитрів.

Проблема 2. Неповний конспект для довгих відео. Ліміт `SUMMARY_MAX_CHARS=18000` не покриває лекції тривалістю понад 30–40 хвилин. Рішення — ієрархічний підхід: спочатку формувати мікроконспекти для кожних 18 000 символів окремо, потім передавати їх до AI для об'єднання в загальний конспект. Це збільшить вартість API-запитів приблизно вдвічі, але дасть повніший результат.

Проблема 3. Якість перекладу технічних термінів. GoogleTranslator перекладає терміни буквально (`thread` → нитка, `score` → сфера). Для технічних

відео це помітна проблема. Рішення — після перекладу виконати постобробку: список знайдених англійських термінів у тексті зберігати без перекладу або з перекладом у дужках. Наприклад: «потік (thread)» замість «нитка».

Проблема 4. Немає обмеження на кількість запитів від одного користувача за одиницю часу. При бажанні один користувач може надсилати запити одразу після завершення попереднього — і витратити API-квоту необмежено. Рішення — простий rate limiter: зберігати час останнього запиту для кожного chat\_id і блокувати нові запити, якщо пройшло менше 30 секунд.

### **3.8. Практичні рекомендації щодо вдосконалення**

На основі проведеного тестування і аналізу проблем сформульовано перелік рекомендацій для подальшого розвитку системи в порядку пріоритету.

Рекомендація 1. Додати підтримку Whisper для відео без субтитрів. Це найбільш пріоритетна доробка, оскільки вона усуває головне обмеження системи і суттєво розширює кількість відео, що підлягають обробці. Реалізація: `pip install yt-dlp openai-whisper`, додати функцію `transcribe_audio(video_id)` з викликом `yt-dlp` для завантаження і `whisper.load_model("base").transcribe()` для транскрипції.

Рекомендація 2. Перенести збереження стану в SQLite. Реалізується за кілька годин: стандартний модуль `sqlite3`, одна таблиця `results` з п'ятьма полями, заміна рядків з `user_storage` на функції-обгортки. Дасть можливість зберігати історію запитів і не втрачати результати при перезапуску.

Рекомендація 3. Додати вибір мови перекладу. Після `/start` або через команду `/language` показувати клавіатуру з вибором мови. Зберігати вибір у окремій таблиці або полі в SQLite. Це дозволить боту бути корисним не лише для україномовної аудиторії.

Рекомендація 4. Реалізувати режим Q&A. Після отримання конспекту користувач може написати запитання до вмісту відео. Бот передає запитання і

переклад до AI з промптом «відповідай лише на основі наданого тексту». Це перетворює бота з інструменту «отримати конспект» на повноцінного навчального асистента.

Рекомендація 5. Перейти на webhook для production-розгортання. При переході на сервер (наприклад, DigitalOcean Droplet або Railway) polling генерує постійний мережевий трафік. Webhook зменшує навантаження і пришвидшує реакцію на повідомлення з ~1–2 с до ~0.1–0.3 с. Реалізація потребує SSL-сертифіката (безкоштовний через Let's Encrypt) і реєстрації webhook через setWebhook.

Рекомендація 6. Додати кешування результатів для популярних відео. Якщо кілька користувачів обробляють одне і те саме відео, другий і наступні запити можна обслуговувати з кешу без зовнішніх API-викликів. Ключ кешу — video\_id. Час зберігання — 24 години. Реалізація: додаткова таблиця cache в SQLite або Redis.

### **3.9. Висновки до розділу**

У третьому розділі описано практичну реалізацію, тестування і оцінку ефективності розробленої системи.

Реалізацію виконано мовою Python з модульною структурою коду — кожен компонент (отримання субтитрів, переклад, AI-аналіз, взаємодія з Telegram) реалізований окремою функцією з визначеним інтерфейсом.

Тестування підтвердило коректну роботу всіх основних сценаріїв. Середній час обробки відео тривалістю 10–15 хвилин — близько 37 секунд, де ~50% часу займає переклад і ~39% — AI-аналіз.

Оцінка якості показала, що система найефективніша для технічних і навчальних відео з якісними субтитрами — саме для тієї цільової аудиторії, на яку орієнтована.

Виявлено чотири основні проблеми: відсутність підтримки відео без субтитрів, неповний конспект для дуже довгих лекцій, буквальний переклад технічних термінів і відсутність rate limiting. Для кожної сформульовано конкретне технічне рішення, реалізація якого виходить за межі поточного прототипу і є напрямом подальшої роботи.

Для кожної з виявлених проблем сформульовано конкретне технічне рішення у підрозділі 3.7, реалізація якого є напрямом подальшої роботи.

## **ВИСНОВКИ**

У межах кваліфікаційної бакалаврської роботи розроблено Telegram-бота для обробки відео з YouTube із використанням AI-асистента. Ідея роботи полягала в тому, щоб користувач міг надіслати посилання на відео й отримати готовий текстовий результат: переклад українською мовою, короткий конспект і список ключових термінів із поясненнями.

У першому розділі розглянуто сучасні чат-боти, платформи для їх створення та підходи до обробки текстової інформації. Окрему увагу приділено проблемам, які виникають під час роботи з YouTube-відео. Серед них — відсутність субтитрів, помилки автоматичного розпізнавання, велика довжина транскриптів, рекламні вставки, мовні обмеження та залежність від зовнішніх API.

У другому розділі описано проектування Telegram-бота. Визначено загальну архітектуру системи, логіку обробки YouTube-посилання, спосіб тимчасового збереження результатів і сценарії взаємодії користувача з ботом. Також розглянуто обмеження, які потрібно враховувати під час роботи з довгими текстами, зовнішніми сервісами та повідомленнями Telegram.

У третьому розділі описано реалізацію бота мовою Python. У програмі реалізовано отримання повідомлень із Telegram, перевірку YouTube-посилань, виділення `video_id`, отримання субтитрів, очищення тексту, переклад українською мовою, формування конспекту та виділення термінів. Для зручності користувача результати виводяться через кнопки «Переклад», «Конспект» і «Терміни».

Під час тестування перевірялися основні сценарії роботи бота. Система коректно обробляє стандартні YouTube-посилання та Shorts, якщо до відео доступні субтитри. Також перевірялася реакція на неправильний текст, відео без субтитрів, повторне натискання кнопок і надсилання нового запиту під час активної обробки. У таких випадках бот не зупиняється аварійно, а повертає користувачу зрозуміле повідомлення.

Найкраще бот працює з навчальними, технічними та пояснювальними відео, де є достатньо якісні субтитри. У таких випадках переклад виходить читабельним, конспект передає основний зміст, а список термінів допомагає швидко виділити важливі поняття. Якщо субтитри автоматичні, неточні або відео занадто довге, якість результату може бути нижчою.

Практично цей бот корисний тим, що кілька дій об'єднані в одному інтерфейсі. Користувачу не потрібно окремо шукати субтитри, перекладати текст, переносити його в інший сервіс і вручну робити конспект. Усе виконується прямо в Telegram, тому робота з відеоматеріалами стає швидшою і простішою.

Поставлені завдання виконано: проаналізовано предметну область, обрано технології, спроектовано архітектуру системи, реалізовано основні функції бота та проведено тестування. Розроблений програмний продукт показує, що YouTube-відео можна автоматично перетворювати на зручний текстовий матеріал у вигляді перекладу, конспекту та термінів.

Разом із цим система має обмеження. Вона залежить від наявності субтитрів на YouTube, якості автоматичного розпізнавання, стабільності зовнішніх сервісів і лімітів на довжину тексту. Для дуже довгих відео результат може бути менш повним, бо текст доводиться скорочувати або обробляти частинами. Також у поточній реалізації результати зберігаються тимчасово, без повноцінної бази даних.

У подальшому до бота можна додати базу даних для історії запитів, вибір мови перекладу, експорт результатів у файл, розпізнавання аудіо для відео без субтитрів і можливість ставити запитання за змістом відео. Такі зміни зробили б бота зручнішим для роботи з навчальними відео.

## **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні вимоги та правила складання. — Київ : ДП «УкрНДНЦ», 2016. — 20 с.

2. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. — Geneva : ISO, 2011. — 34 p.
3. Закон України «Про захист персональних даних» від 01.06.2010 № 2297-VI. URL: <https://zakon.rada.gov.ua/laws/show/2297-17>
4. Jurafsky D., Martin J. H. Speech and Language Processing. 3rd ed. — Stanford : Stanford University, 2024. — 667 p. URL: <https://web.stanford.edu/~jurafsky/slp3/>
5. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4th ed. — Hoboken : Pearson, 2021. — 1132 p.
6. Bird S., Klein E., Loper E. Natural Language Processing with Python. — Sebastopol : O'Reilly Media, 2009. — 504 p. URL: <https://www.nltk.org/book/>
7. Goodfellow I., Bengio Y., Courville A. Deep Learning. — Cambridge : MIT Press, 2016. — 800 p. URL: <https://www.deeplearningbook.org/>
8. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. — Boston : Addison-Wesley, 1994. — 395 p.
9. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. — Upper Saddle River : Prentice Hall, 2008. — 431 p.
10. Fowler M. Patterns of Enterprise Application Architecture. — Boston : Addison-Wesley, 2002. — 533 p.
11. Lutz M. Learning Python. 5th ed. — Sebastopol : O'Reilly Media, 2013. — 1540 p.
12. Beazley D., Jones B. K. Python Cookbook. 3rd ed. — Sebastopol : O'Reilly Media, 2013. — 706 p.

13. Ramalho L. *Fluent Python*. 2nd ed. — Sebastopol : O'Reilly Media, 2022. — 1010 p.
14. Sommerville I. *Software Engineering*. 10th ed. — Boston : Pearson, 2016. — 810 p.
15. Pressman R. S., Maxim B. R. *Software Engineering: A Practitioner's Approach*. 9th ed. — New York : McGraw-Hill, 2020. — 976 p.
16. Vaswani A. et al. Attention is all you need. *Advances in Neural Information Processing Systems*. 2017. Vol. 30. P. 5998–6008. URL: <https://arxiv.org/abs/1706.03762>
17. Brown T. et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*. 2020. Vol. 33. P. 1877–1901. URL: <https://arxiv.org/abs/2005.14165>
18. Radford A. et al. Improving language understanding by generative pre-training. *OpenAI Blog*. 2018. URL: <https://openai.com/research/language-unsupervised>
19. Devlin J. et al. BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT 2019*. 2019. P. 4171–4186. URL: <https://arxiv.org/abs/1810.04805>
20. Raffel C. et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*. 2020. Vol. 21. P. 1–67. URL: <https://arxiv.org/abs/1910.10683>
21. He K. et al. Deep residual learning for image recognition. *Proceedings of CVPR 2016*. 2016. P. 770–778. URL: <https://arxiv.org/abs/1512.03385>
22. Hochreiter S., Schmidhuber J. Long short-term memory. *Neural Computation*. 1997. Vol. 9, No. 8. P. 1735–1780.

23. Bahdanau D., Cho K., Bengio Y. Neural machine translation by jointly learning to align and translate. Proceedings of ICLR 2015. 2015. URL: <https://arxiv.org/abs/1409.0473>
24. Luong M.-T., Pham H., Manning C. D. Effective approaches to attention-based neural machine translation. Proceedings of EMNLP 2015. 2015. P. 1412–1421. URL: <https://arxiv.org/abs/1508.04025>
25. Wu Y. et al. Google's neural machine translation system: bridging the gap between human and machine translation. 2016. URL: <https://arxiv.org/abs/1609.08144>
26. Radford A. et al. Robust speech recognition via large-scale weak supervision (Whisper). Proceedings of ICML 2023. 2023. URL: <https://arxiv.org/abs/2212.04356>
27. OpenAI. GPT-4 technical report. 2023. URL: <https://arxiv.org/abs/2303.08774>
28. Touvron H. et al. Llama 2: Open foundation and fine-tuned chat models. 2023. URL: <https://arxiv.org/abs/2307.09288>
29. Jiang A. Q. et al. Mistral 7B. 2023. URL: <https://arxiv.org/abs/2310.06825>
30. Wei J. et al. Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems. 2022. Vol. 35. URL: <https://arxiv.org/abs/2201.11903>
31. Liu P. et al. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Computing Surveys. 2023. Vol. 55, No. 9. P. 1–35. URL: <https://arxiv.org/abs/2107.13586>
32. Zhao W. X. et al. A survey of large language models. 2023. URL: <https://arxiv.org/abs/2303.18223>

33. Minaee S. et al. Large language models: A survey. 2024. URL: <https://arxiv.org/abs/2402.06196>
34. Min B. et al. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Computing Surveys*. 2023. Vol. 56, No. 2. P. 1–40.
35. Adiwardana D. et al. Towards a human-like open-domain chatbot. 2020. URL: <https://arxiv.org/abs/2001.09977>
36. Roller S. et al. Recipes for building an open-domain chatbot. *Proceedings of EACL 2021*. 2021. P. 300–325. URL: <https://arxiv.org/abs/2004.13637>
37. Zhang S. et al. DialoGPT: Large-scale generative pre-training for conversational response generation. *Proceedings of ACL 2020*. 2020. P. 270–278. URL: <https://arxiv.org/abs/1911.00536>
38. Bender E. M. et al. On the dangers of stochastic parrots: Can language models be too big? *Proceedings of FAccT 2021*. 2021. P. 610–623.
39. Bommasani R. et al. On the opportunities and risks of foundation models. 2021. URL: <https://arxiv.org/abs/2108.07258>
40. Ji Z. et al. Survey of hallucination in natural language generation. *ACM Computing Surveys*. 2023. Vol. 55, No. 12. P. 1–38.
41. Shen T. et al. Large language models alignment: A survey. 2023. URL: <https://arxiv.org/abs/2309.15025>
42. Sun L. et al. A survey of neural machine translation. 2022. URL: <https://arxiv.org/abs/2202.05601>
43. Koehn P. *Statistical Machine Translation*. — Cambridge : Cambridge University Press, 2010. — 446 p.
44. Forcada M. L. et al. Apertium: a free/open-source platform for rule-based machine translation. *Machine Translation*. 2011. Vol. 25, No. 2. P. 127–144.

45. Papineni K. et al. BLEU: a method for automatic evaluation of machine translation. Proceedings of ACL 2002. 2002. P. 311–318.
46. Python Software Foundation. Python 3 documentation. URL: <https://docs.python.org/3/>
47. Python Software Foundation. logging — Logging facility for Python. URL: <https://docs.python.org/3/library/logging.html>
48. Python Software Foundation. urllib.parse — Parse URLs into components. URL: <https://docs.python.org/3/library/urllib.parse.html>
49. Python Software Foundation. re — Regular expression operations. URL: <https://docs.python.org/3/library/re.html>
50. Python Software Foundation. json — JSON encoder and decoder. URL: <https://docs.python.org/3/library/json.html>
51. Python Software Foundation. html — HyperText Markup Language support. URL: <https://docs.python.org/3/library/html.html>
52. Python Software Foundation. sqlite3 — DB-API 2.0 interface for SQLite databases. URL: <https://docs.python.org/3/library/sqlite3.html>
53. Telegram. Bot API documentation. URL: <https://core.telegram.org/bots/api>
54. Telegram. Introduction for developers. URL: <https://core.telegram.org/bots>
55. OpenAI. API reference documentation. URL: <https://platform.openai.com/docs/api-reference>
56. OpenAI. Prompt engineering guide. URL: <https://platform.openai.com/docs/guides/prompt-engineering>
57. OpenAI. Models overview. URL: <https://platform.openai.com/docs/models>

58. OpenAI. Best practices for AI safety. URL: <https://platform.openai.com/docs/guides/safety-best-practices>
59. jdepoix. youtube-transcript-api — Python API to get the transcript of YouTube videos. GitHub. URL: <https://github.com/jdepoix/youtube-transcript-api>
60. Baccouri N. deep-translator — A flexible Python package to translate between languages. GitHub. URL: <https://github.com/nidhaloff/deep-translator>
61. Baccouri N. deep-translator. Official documentation. URL: <https://deep-translator.readthedocs.io/en/latest/>
62. Reitz K. Requests: HTTP for Humans. Official documentation. URL: <https://requests.readthedocs.io/en/latest/>
63. OpenAI. openai-python: The official Python library for the OpenAI API. GitHub. URL: <https://github.com/openai/openai-python>
64. Meta AI. yt-dlp: A youtube-dl fork with additional features and fixes. GitHub. URL: <https://github.com/yt-dlp/yt-dlp>
65. OpenAI. openai-whisper: Robust Speech Recognition via Large-Scale Weak Supervision. GitHub. URL: <https://github.com/openai/whisper>
66. Nielsen Norman Group. Chatbots and Conversational UI. URL: <https://www.nngroup.com/articles/chatbots/>
67. ISTQB. Glossary of testing terminology. URL: [https://glossary.istqb.org/en\\_US/home](https://glossary.istqb.org/en_US/home)
68. Microsoft. Best practices for REST API design. URL: <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design>
69. OWASP Foundation. Secrets Management Cheat Sheet. URL: [https://cheatsheetseries.owasp.org/cheatsheets/Secrets\\_Management\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html)

70. OWASP Foundation. API Security Top 10. URL: <https://owasp.org/www-project-api-security/>
71. Fowler M. Microservices. martinowler.com. 2014. URL: <https://martinowler.com/articles/microservices.html>
72. Fowler M. Event-driven architecture. martinowler.com. URL: <https://martinowler.com/articles/event-driven.html>
73. Google. YouTube Data API v3 overview. URL: <https://developers.google.com/youtube/v3/getting-started>
74. Google. Cloud Translation API documentation. URL: <https://cloud.google.com/translate/docs>
75. Hugging Face. LLM course. URL: <https://huggingface.co/learn/llm-course/en/chapter1/1>
76. Hugging Face. Transformers documentation. URL: <https://huggingface.co/docs/transformers>
77. python-telegram-bot contributors. python-telegram-bot official documentation. URL: <https://docs.python-telegram-bot.org/en/stable/>
78. Juniper Research. Chatbots: retail, eCommerce, banking & healthcare 2023–2027. — Basingstoke : Juniper Research, 2022. URL: <https://www.juniperresearch.com/researchstore/fintech-payments/chatbot-market-research-report>
79. Grand View Research. Chatbot market size, share & trends analysis report 2023–2030. — San Francisco : Grand View Research, 2023. URL: <https://www.grandviewresearch.com/industry-analysis/chatbot-market>
80. Statista. Number of monthly active Telegram users worldwide 2018–2024. URL: <https://www.statista.com/statistics/234038/telegram-messenger-mau-users/>

81. Weizenbaum J. ELIZA — A computer program for the study of natural language communication between man and machine. *Communications of the ACM*. 1966. Vol. 9, No. 1. P. 36–45.
82. Wallace R. The anatomy of ALICE. *Parsing the Turing Test / eds. R. Epstein, G. Roberts, G. Beber*. Dordrecht : Springer, 2009. P. 181–210.
83. Shum H.-Y., He X.-D., Li D. From Eliza to XiaoIce: challenges and opportunities with social chatbots. *Frontiers of Information Technology & Electronic Engineering*. 2018. Vol. 19, No. 1. P. 10–26.
84. Gao J. et al. Neural approaches to conversational AI. *Foundations and Trends in Information Retrieval*. 2019. Vol. 13, No. 2–3. P. 127–298.
85. Henderson M. et al. The second dialog state tracking challenge. *Proceedings of SIGDIAL 2014*. 2014. P. 263–272.
86. Young S. et al. POMDP-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*. 2013. Vol. 101, No. 5. P. 1160–1179.
87. Bocklisch T. et al. Rasa: Open source language understanding and dialogue management. 2017. URL: <https://arxiv.org/abs/1712.05181>
88. Ram A. et al. Conversational AI: The science behind the Alexa Prize. 2018. URL: <https://arxiv.org/abs/1801.03604>
89. Manning C. D., Schütze H. *Foundations of Statistical Natural Language Processing*. — Cambridge : MIT Press, 1999. — 680 p.
90. Cho K. et al. Learning phrase representations using RNN encoder–decoder for statistical machine translation. *Proceedings of EMNLP 2014*. 2014. P. 1724–1734. URL: <https://arxiv.org/abs/1406.1078>
91. Sutskever I., Vinyals O., Le Q. V. Sequence to sequence learning with neural networks. *Advances in NeurIPS 2014*. 2014. Vol. 27. P. 3104–3112. URL: <https://arxiv.org/abs/1409.3215>

92. See A., Liu P. J., Manning C. D. Get to the point: Summarization with pointer-generator networks. Proceedings of ACL 2017. 2017. P. 1073–1083.
93. Nallapati R. et al. Abstractive text summarization using sequence-to-sequence RNNs and beyond. Proceedings of CoNLL 2016. 2016. P. 280–290.
94. El-Kassas W. S. et al. Automatic text summarization: A comprehensive survey. Expert Systems with Applications. 2021. Vol. 165. Article 113679.
95. Lewis M. et al. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. Proceedings of ACL 2020. 2020. P. 7871–7880.
96. Zhang J. et al. PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization. Proceedings of ICML 2020. 2020. P. 11328–11339.
97. Wang A. et al. GLUE: A multi-task benchmark and analysis platform for natural language understanding. Proceedings of EMNLP 2018 Workshop BlackboxNLP. 2018. P. 353–355.
98. Chen D. et al. Reading Wikipedia to answer open-domain questions. Proceedings of ACL 2017. 2017. P. 1870–1879.
99. Rajpurkar P. et al. SQuAD: 100,000+ questions for machine comprehension of text. Proceedings of EMNLP 2016. 2016. P. 2383–2392.
100. Amodei D. et al. Deep speech 2: End-to-end speech recognition in English and Mandarin. Proceedings of ICML 2016. 2016. P. 173–182.
101. Google Research. Reformer: The efficient transformer. Proceedings of ICLR 2020. 2020. URL: <https://arxiv.org/abs/2001.04451>
102. Karpathy A. The unreasonable effectiveness of recurrent neural networks. Andrej Karpathy Blog. 2015. URL: <https://karpathy.github.io/2015/05/21/rnn-effectiveness/>

103. Breck E. et al. The ML test score: A rubric for ML production readiness and technical debt reduction. Proceedings of IEEE Big Data 2017. 2017. P. 1123–1132.

## **ДОДАТКИ**

Додаток А. Фрагменти програмного коду чат-бота.

А.1. Імпорт бібліотек та початкова ініціалізація системи

```
import re
```

```
import time
import html
import json
import logging
from urllib.parse import urlparse, parse_qs

import requests
from youtube_transcript_api import YouTubeTranscriptApi
from deep_translator import GoogleTranslator
from openai import OpenAI

BOT_TOKEN = "YOUR_TELEGRAM_BOT_TOKEN"
OPENAI_API_KEY = "YOUR_OPENAI_API_KEY"

SOURCE_LANGS = ["en", "en-US", "en-GB"]
TARGET_LANG = "uk"

MAX_TRANSLATION_CHUNK = 3500
MAX_TELEGRAM_MESSAGE = 3900
MAX_LLM_SUMMARY_INPUT = 18000

OPENAI_SUMMARY_MODEL = "gpt-5-mini"

API_URL = f"https://api.telegram.org/bot{BOT_TOKEN}"

logging.basicConfig(
    level=logging.INFO,
```

```
format="%%(asctime)s | %%(levelname)s | %%(message)s"
)
```

```
user_storage = {}
```

```
processing_users = set()
```

```
ytt_api = YouTubeTranscriptApi()
```

```
openai_client = OpenAI(api_key=OPENAI_API_KEY)
```

## A.2. Функція розпізнавання ідентифікатора YouTube-відео

```
def extract_video_id(url: str) -> str:
```

```
    url = url.strip()
```

```
    if re.fullmatch(r"[A-Za-z0-9_-]{11}", url):
```

```
        return url
```

```
    parsed = urlparse(url)
```

```
    hostname = (parsed.hostname or "").lower()
```

```
    if hostname == "youtu.be":
```

```
        return parsed.path.strip("/")
```

```
    if "youtube.com" in hostname:
```

```
        if parsed.path == "/watch":
```

```
            return parse_qs(parsed.query).get("v", [""])[0]
```

```
        if parsed.path.startswith("/shorts/"):

```

```

        return parsed.path.split("/shorts/")[1].split("/")[0]
    if parsed.path.startswith("/embed/"):
        return parsed.path.split("/embed/")[1].split("/")[0]

    raise ValueError("Не вдалося розпізнати YouTube-посилання.")

```

### A.3. Отримання та очищення субтитрів відео

```

def clean_caption_item(text: str) -> str:
    text = html.unescape(text)
    text = re.sub(r"\.[*?\]", "", text)
    text = text.replace("\n", " ").strip()
    text = re.sub(r"\s+", " ", text)
    return text

def fetch_transcript(video_id: str) -> str:
    last_error = None

    for lang in SOURCE_LANGS:
        try:
            items = ytt_api.fetch(video_id, languages=[lang])
            text = " ".join(clean_caption_item(x.text) for x in items)
            if text.strip():
                return text
        except Exception as e:
            last_error = e

```

```

try:
    items = ytt_api.fetch(video_id)
    text = " ".join(clean_caption_item(x.text) for x in items)
    if text.strip():
        return text
except Exception as e:
    last_error = e

    raise RuntimeError(f"Субтитри не знайдено або не вдалося отримати.
{last_error}")

```

#### A.4. Переклад тексту українською мовою

```

def chunk_text_smart(text: str, max_len: int = MAX_TRANSLATION_CHUNK)
-> list[str]:
    text = text.replace("\r", "\n")
    paragraphs = [p.strip() for p in text.split("\n") if p.strip()]

    chunks = []
    current = ""

    for paragraph in paragraphs:
        sentences = re.split(r'(?<=[!?!])\s+', paragraph)

        for sentence in sentences:
            sentence = sentence.strip()
            if not sentence:
                continue

```

```

candidate = (current + " " + sentence).strip() if current else sentence

if len(candidate) <= max_len:
    current = candidate
else:
    if current:
        chunks.append(current.strip())

    if len(sentence) > max_len:
        words = sentence.split()
        temp = ""
        for word in words:
            cand = (temp + " " + word).strip() if temp else word
            if len(cand) <= max_len:
                temp = cand
            else:
                if temp:
                    chunks.append(temp.strip())
                temp = word
        current = temp
    else:
        current = sentence

if current:
    chunks.append(current.strip())

return [c for c in chunks if c.strip()]

```

```

def translate_text(text: str, source: str = "en", target: str = "uk") -> str:
    translator = GoogleTranslator(source=source, target=target)
    chunks = chunk_text_smart(text, max_len=MAX_TRANSLATION_CHUNK)
    translated_parts = []

    for idx, chunk in enumerate(chunks, start=1):
        logging.info("Translating chunk %s/%s", idx, len(chunks))
        translated = translator.translate(chunk)
        if translated:
            translated_parts.append(translated.strip())

    return "\n\n".join(translated_parts).strip()

```

#### A.5. Формування AI-конспекту та списку термінів

```

def llm_make_summary_and_terms(translated_text: str) -> tuple[str, str]:
    prepared_text = translated_text[:MAX_LLM_SUMMARY_INPUT]

    prompt = f"""

```

Ти аналізуєш транскрипт YouTube-відео українською мовою.

Завдання:

1. Ігноруй рекламу, промокоди, мерч, заклики підписатися, посилання, спонсорські вставки.
2. Зроби нормальний змістовний конспект саме по суті.
3. Виділи терміни тільки якщо вони реально важливі для теми відео.

4. Не вигадуй фактів, яких немає в тексті.

Поверни строго JSON такого формату:

```
{  
  "summary": "КОНСПЕКТ ВІДЕО\n\n1. ...\n2. ...\n3. ...\n\nВисновок: ...",  
  "terms": "ТЕРМІНИ:\n\n• термін 1 — коротке пояснення\n• термін 2 — ко-  
ротке пояснення"  
}
```

Ось текст:

```
{prepared_text}
```

```
"".strip()
```

```
raw = ""
```

```
for attempt in range(2):
```

```
    response = openai_client.responses.create(  
        model=OPENAI_SUMMARY_MODEL,  
        input=prompt  
    )
```

```
    raw = response.output_text.strip()
```

```
    try:
```

```
        data = json.loads(raw)
```

```
        summary = data.get("summary", "").strip() or "Не вдалося побудувати  
конспект."
```

```
        terms = data.get("terms", "").strip() or "Не вдалося знайти терміни."
```

```

    return summary, terms
except Exception:
    logging.warning("JSON parse failed on attempt %s", attempt + 1)

fallback_summary = "КОНСПЕКТ ВІДЕО\n\n" + raw[:3000]
fallback_terms = "ТЕРМІНИ:\n\nНе вдалося окремо виділити терміни."
return fallback_summary, fallback_terms

```

#### A.6. Основна функція обробки YouTube-відео

```

def process_youtube(url: str) -> tuple[str, str, str]:
    video_id = extract_video_id(url)
    raw_transcript = fetch_transcript(video_id)

    translated = translate_text(raw_transcript, source="en",
target=TARGET_LANG)

    summary, terms_text = llm_make_summary_and_terms(translated)

    return translated, summary, terms_text

```

#### A.7. Обробка повідомлень користувача в Telegram

```

def handle_message(message: dict):
    chat_id = message["chat"]["id"]
    text = message.get("text", "").strip()

    if not text:

```

```
send_message(chat_id, "Надішли мені YouTube-посилання.")  
return
```

```
if text.startswith("/start"):
```

```
    send_message(  
        chat_id,  
        "Привіт.\n\n"  
        "Надішли мені посилання на YouTube-відео, і я зроблю:\n"  
        "1) швидкий переклад субтитрів на українську\n"  
        "2) розумний конспект через AI\n"  
        "3) терміни через AI"  
    )  
    return
```

```
if "youtu" not in text:
```

```
    send_message(chat_id, "Я чекаю саме YouTube-посилання.")  
    return
```

```
if chat_id in processing_users:
```

```
    send_message(chat_id, "Зачекай, попереднє відео ще обробляється.")  
    return
```

```
processing_users.add(chat_id)
```

```
send_message(chat_id, "Обробляю відео. Обробка має зайняти менше хви-  
лини.")
```

```
try:
```

```

translated, summary, terms_text = process_youtube(text)

user_storage[chat_id] = {
    "translated": translated,
    "summary": summary,
    "terms": terms_text
}

send_message(chat_id, "Готово.\n\nОбери, що хочеш отримати:")

except Exception as e:
    logging.exception("Processing error")
    send_message(chat_id, f"Помилка: {str(e)}")

finally:
    processing_users.discard(chat_id)

```

#### А.8. Основний цикл запуску бота

```

def main():
    if not BOT_TOKEN:
        raise RuntimeError("Встав BOT_TOKEN у код.")

    if not OPENAI_API_KEY:
        raise RuntimeError("Встав OPENAI_API_KEY у код.")

    logging.info("Bot started")
    offset = None

```

```

while True:
    try:
        data = get_updates(offset)
        updates = data.get("result", [])

        for update in updates:
            offset = update["update_id"] + 1
            message = update.get("message")
            if message:
                handle_message(message)

    except requests.RequestException as e:
        logging.error("Network error: %s", e)
        time.sleep(3)
    except Exception as e:
        logging.error("Unexpected error: %s", e)
        time.sleep(3)

if __name__ == "__main__":
    main()

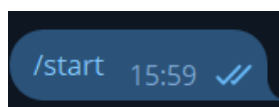
```

У додатку наведено ключові фрагменти коду, що реалізують взаємодію з Telegram API, обробку YouTube-посилань, переклад тексту, формування AI-конспекту, тощо.

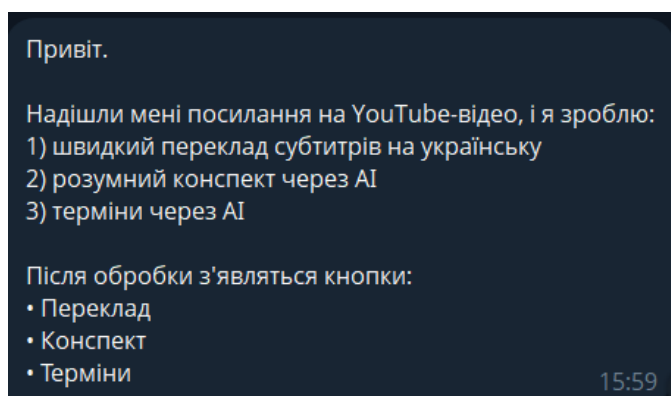
Додаток Б. Приклади повідомлень користувача та відповідей системи у вигляді скріншотів.

### Б.1. Приклад запуску чат-бота

Повідомлення користувача:

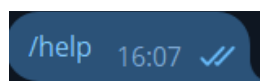


Відповідь системи:

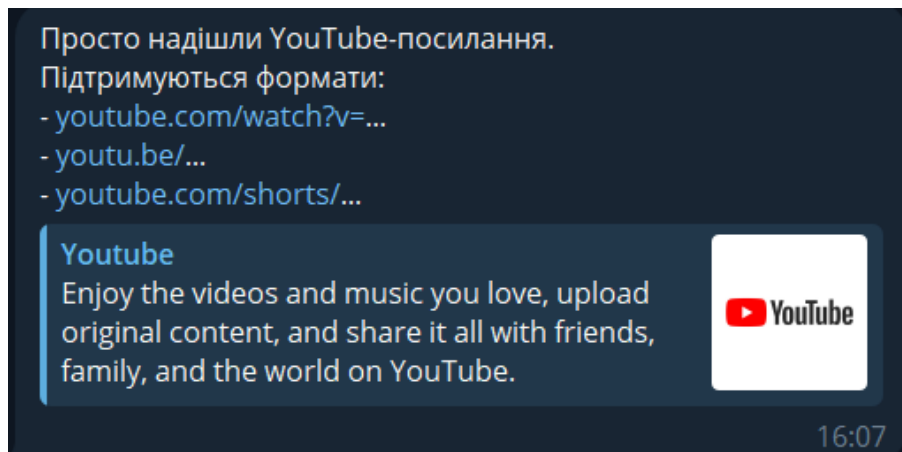


### Б.2. Приклад довідкової команди

Повідомлення користувача:

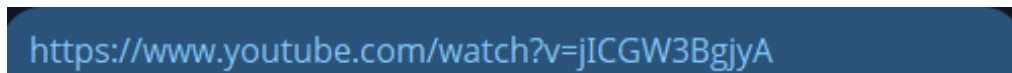


Відповідь системи:

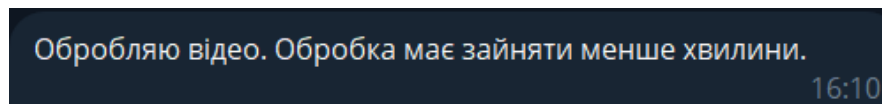


### Б.3. Приклад надсилання YouTube-посилання

Повідомлення користувача:

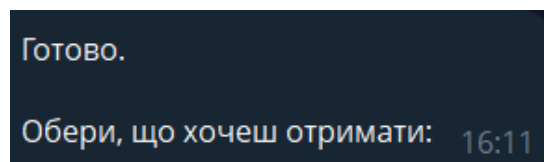


Відповідь системи:

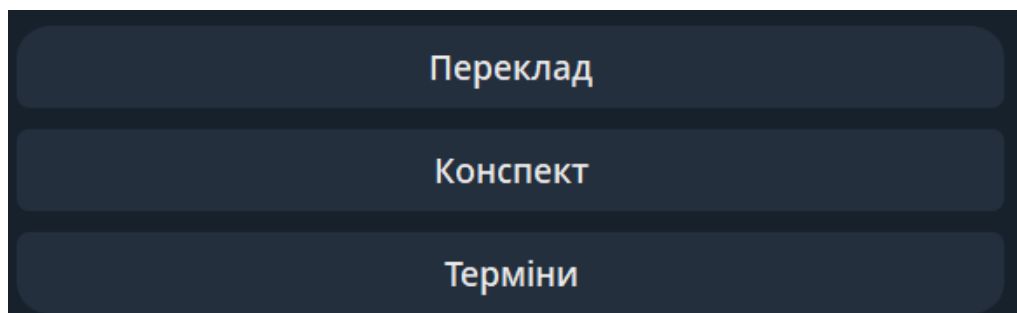


### Б.4. Приклад повідомлення після завершення обробки

Відповідь системи:

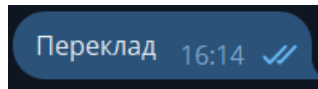


Кнопки меню:



## Б.5. Приклад отримання перекладу субтитрів

Повідомлення користувача:



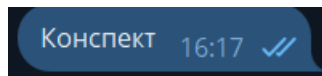
Відповідь системи:

ПЕРЕКЛАД:

Інцидент з Porsche. 30 вересня 1955 року Джеймс Дін щойно закінчив зйомки фільму «Велетень». 24-річний актор нарешті зміг зайнятися своєю справжньою пристрастю — гонками. 9 днів тому Дін обміняв свій Porsche Speedster на щось швидше, новий Porsche 550 Spyder, сріблястий, легкий і небезпечно швидкий. Він заплатив за це 7000 доларів і попросив художника написати на спині слова «Маленький виродок». За тиждень до аварії Дін зіткнувся в Лос-Анджелесі з актором Алем Піннесом. Пізніше Піннес написав у своєму щоденнику, що Porsche виглядав йому зловісно, і попередив Діна не їздити на ньому. Тієї п'ятниці вдень Дін вирушив на перегони в Салінасі зі своїм механіком Рольфом Ветеріхом на дробовику. О 15:30 офіцер Каліфорнійської дорожньої патрулі зупинив їх за перевищення швидкості поблизу Бейкерсфілда. Дін отримав квиток. Це не сподобало його. О 17:45 Дін мчав по трасі 466 із приблизною швидкістю 85 миль/год. Поблизу міста Чоламе 23-річний студент коледжу на ім'я Дональд Тернапсід їхав на Ford у зустрічному напрямку. Turnipseed повернув ліворуч через шосе. Він ніколи не бачив, як наближається низько опущений сріблястий Porsche. Удар був майже лобовий. Спайдер зім'явся, як алюмінієва фольга. Дін отримав зламану ший, масивні внутрішні травми та серйозну травму голови. Після прибуття в лікарню він був оголошений мертвим. Володів автомобілем 9 днів. Вютеріх був викинутий з машини та вижив із серйозними травмами. Turnipseed пішов з невеликими синцями. 550 Spyder був гоночним з номерними знаками. Він важив лише 1200 фунтів. Не було ні ременів безпеки, ні каркаса безпеки, ні зон деформації. Porsche створив його для перемоги в гонках, а не для захисту водіїв. Смерть Джеймса Діна у 24 роки зробила його легендою. Інцидент з Tesla. Джошуа Браун був віруючим. 40-річний колишній військовослужбовець ВМС США купив Tesla Model S і заховався в неї. Він назвав автомобіль Tessie і опублікував на YouTube відео, демонструючи його систему автопілота. Засновник Tesla Ілон Маск зробив ретвіт одного зі своїх відео. Він набрав мільйон переглядів. 7 травня 2016 року Браун їхав шосе Флориди з увімкненим автопілотом. Система дала йому сім попереджень, щоб він поклав руки на кермо. Він проігнорував їх усіх. Попереду через трасу ліворуч повертав 18-колісний причіп. Причіп був широкий. Небо позаду було яскраве. Ні Браун, ні система автопілота не бачили цього. Tesla в'їхала прямо в бік причепа на швидкості 74 миль на годину. У причепа злетіло дах. Tesla врізалася через дві огорожі та вперлася у телефонний стовп. В уламках знайдено портативний

## Б.6. Приклад отримання конспекту відео

Повідомлення користувача:



Відповідь системи:

**КОНСПЕКТ ВІДЕО**

1. Узагальнення: Відео систематизує приклади смертельних дорожніх подій, де ключову роль відігравали конструктивні рішення автомобілів, технічні дефекти або неправильне застосування технологій, а також показує наслідки для постраждалих, виробників і регулювання безпеки.
2. Джеймс Дін (30 вересня 1955): Актор, який нещодавно придбав гоночний Porsche 550 Spyder з мінімальним захистом, загинув у зіткненні з іншим автомобілем поблизу Чоламе; автомобіль практично не мав засобів пасивної безпеки, що спричинило фатальні травми для водія і важкі поранення для механіка.
3. Tesla Autopilot (7 травня 2016): Власник Tesla, який демонстрував систему автопілота в інтернеті, загинув при ударі у боковину напівпричепа, коли автопілот і водій не виявили перешкоду; випадок названо першою відомою смертю під час використання напівавтономної системи водіння.
4. Lexus / Toyota (28 серпня 2009): Сімейна поїздка перетворилася на трагедію через самовільне прискорення орендованого Lexus, яке розслідування пов'язало з затиснутим килимком під педаллю; аварія призвела до масивних збитків, судових позовів і виплати від Toyota, а також громадського резонансу навколо довіри до виробника.
5. Ford Pinto (28 травня 1972): Унаслідок заднього удару паливний бак пробився в салон і спричинив займання, що спричинило загибель пасажирки; подальші судові рішення й аналізи поставили під сумнів економічні розрахунки виробника щодо виправлення дефекту.
6. Подушки безпеки Takata: Несправні наповнювачі на основі нітрату амонію розклалися в жаркому кліматі, внаслідок чого металеві каністри розривалися і викидали смертоносні уламки; цей дефект пов'язують з багатьма смертями і пораненнями, масштабним відкликанням і банкрутством виробника.
7. Катастрофа в Ле-Мані (11 червня 1955): Автомобіль Mercedes-Benz 300 SLR вилетів у глядацькі трибуни після зіткнення на трасі, корпус з магнієвого сплаву загорівся, загинули десятки глядачів і гонщик; подія радикально змінила підходи до безпеки в автоспорті і вплинула на політику виробника стосовно перегонів.

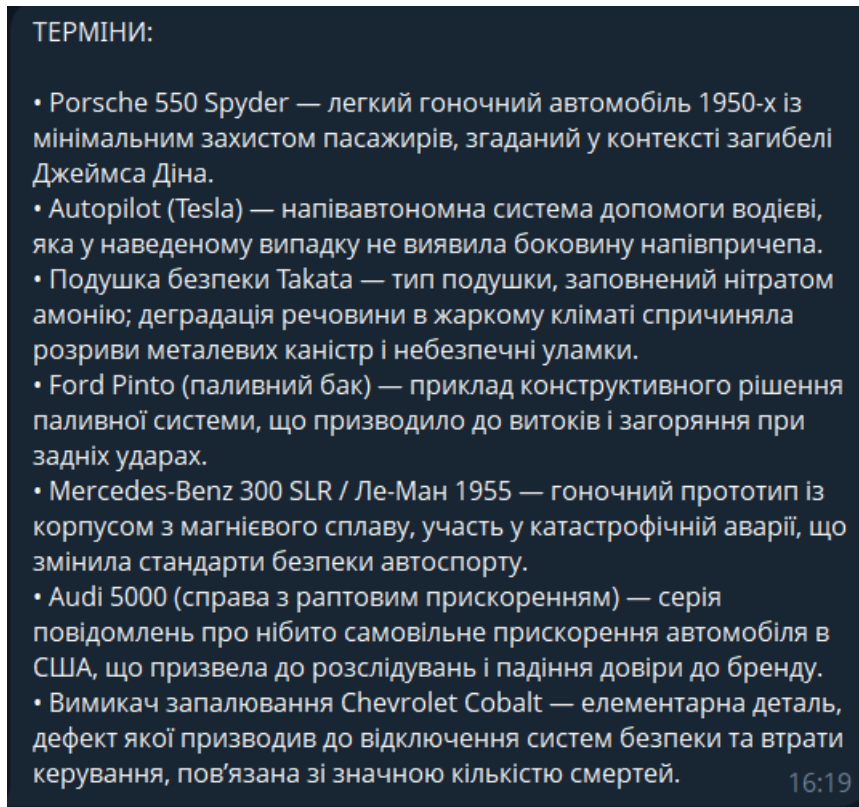
## Б.7. Приклад отримання списку термінів

Повідомлення користувача:



Терміни 16:19 ✓✓

Відповідь системи:



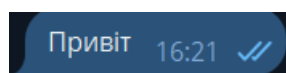
ТЕРМІНИ:

- Porsche 550 Spyder — легкий гоночний автомобіль 1950-х із мінімальним захистом пасажирів, згаданий у контексті загибелі Джеймса Діна.
- Autopilot (Tesla) — напіваавтономна система допомоги водієві, яка у наведеному випадку не виявила боковину напівпричепа.
- Подушка безпеки Takata — тип подушки, заповнений нітратом амонію; деградація речовини в жаркому кліматі спричиняла розриви металевих каністр і небезпечні уламки.
- Ford Pinto (паливний бак) — приклад конструктивного рішення паливної системи, що призводило до витоків і загоряння при задніх ударах.
- Mercedes-Benz 300 SLR / Ле-Ман 1955 — гоночний прототип із корпусом з магнієвого сплаву, участь у катастрофічній аварії, що змінила стандарти безпеки автоспорту.
- Audi 5000 (справа з раптовим прискоренням) — серія повідомлень про нібито самовільне прискорення автомобіля в США, що призвела до розслідувань і падіння довіри до бренду.
- Вимикач запалювання Chevrolet Cobalt — елементарна деталь, дефект якої призводив до відключення систем безпеки та втрати керування, пов'язана зі значною кількістю смертей.

16:19

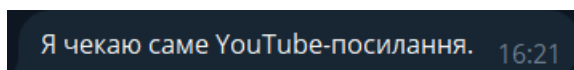
## Б.8. Приклад повідомлення про помилку

Повідомлення користувача:



Привіт 16:21 ✓✓

Відповідь системи:



Я чекаю саме YouTube-посилання. 16:21

У додатку наведені скріншоти діалогу з ботом, приклади перекладу субтитрів, сформованого конспекту та списку термінів.