МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ЗАПОБІГАННЯ БОТ-АТАК З ВИКОРИСТАННЯМ САРТСНА

Студент групи ІН.м-13 Єлизавета ШОЛОМІЙ

Науковий керівник Сергій ПЕТРОВ
доцент, к.т.н.

В.о. завідувача кафедри Ігор Шелехов
доцент, к.т.н.

Суми 2022

Сумський державний університет

Факультет *ЕлІт* Кафедра *Комп'ютерних наук*
Спеціальність *«122 – Комп'ютерні науки»*

Затверджую_____
В.о. зав. кафедри_____
"_____" __._____2022 р.

ЗАВДАННЯ
до кваліфікаційної роботи магістра

Шоломій Єлизаветі Сергіївні

1. Тема: Інформаційна технологія запобігання бот-атак з використанням CAPTCHA.
Затверджена наказом по СумДУ №_____
від «___»_____2022 р.

2. Термін здачі здобувачем закінченої роботи _____

3. Вхідні дані до роботи *монографії, інтернет-ресурси, підручники, періодичні видання*_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
*аналіз проблеми, методика дослідження, практична реалізація*_____

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) *приклади текстового CAPTCHA, фотографічні CAPTCHA, математичні CAPTCHA, загальна схема роботи бота, Класичне представлення афінного перетворення*

6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання "_____"_____ 2022 р.
Керівник випускної роботи_____ Петров С.О.
Завдання прийняв до виконання_____ Шоломій Є.С.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проєкту (роботи)	Термін виконання проєкту (роботи)	Приміт ка
1.	Аналіз літературних джерел щодо проблематики дослідження		
2.	Постановка задачі та формування завдань дослідження.		
3.	Огляд підходів до розв’язку поставленої задачі		
4.	Визначення методів розв’язання поставленої задачі		
5.	Проєктування інформаційної технології		
6.	Виконання програмної реалізації системи		
7.	Проведення експериментальних запусків розробленого комплексу		
8.	Оформлення результатів дослідження		

Студент – дипломник _____
(підпис)

Керівник проєкту _____
(підпис)

ЗМІСТ

РЕФЕРАТ

Записка: 77 стр., 9 рис., 1 додаток, 20 літературних джерел.

Об'єкт дослідження — Механізми і методи автоматичного подолання формалізованого тесту Тьюрінга для вирішення CAPTCHA

Мета роботи — розробка вдосконаленої технології CAPTCHA для запобігання її автоматичного вирішення з використанням бота.

Результати — здійснено грунтовний огляд існуючих підходів до автоматичного подолання CAPTCHA, запропоновано удосконалення процесу побудови CAPTCHA що не ускладнює її вирішення для людини, при цьому знижує можливість її автоматичного її подолання ботом до рівня не більше 40% та збільшує час на її розвязання.

ІНФОРМАЦІЙНА СИСТЕМА, CAPTCHA, OCR, TESSERACT, БОТ-АТАКА

ВСТУП

Починаючи з 2000х років все більше послуг та сервісів перейшли в онлайн. Зі збільшенням кількості безкоштовних послуг в Інтернеті ми виявляємо гостру потребу захистити ці служби від зловживань. Автоматизовані програми (часто їх називають ботами) розроблено для атаки або компроментування різноманітних служб. Наприклад, поширені атаки на постачальників безкоштовної електронної пошти з метою отримання облікових записів, сервіси відправки смс, сервіси онлайн-замовлень та багато інших. Злочинні боти використовують ці облікові записи, щоб розсилати спам, розміщувати спам і рекламу на форумах, а також спотворювати результати онлайн-опитувань.

Щоб запобігти автоматизованим атакам, необхідно розпізнати хто саме користується сервісом, чи то є людина чи бот, тому служби часто просять користувачів вирішити головоломку, перш ніж отримати доступ до служби. Ці головоломки, вперше представлені von Ahn et al. у 2003 році [2] були CAPTCHA: повністю автоматизований публічний тест Тьюрінга для розрізнення комп'ютерів і людей. Тест Тьюрінга було запропоновано вченим, Аланом Тьюрінгом в

1950 году задля виявлення здібності комп'ютера виявляти інтелектуальні дії, далі саме цей тест було запропоновано для того щоб відрізнити людину від машини, оскільки вченим стверджувалось що лише людина може пройти тест.

CAPTCHA — це програма, яка захищає веб-сайти від ботів, генеруючи та оцінюючи тести, які можуть пройти люди, але не можуть пройти поточні комп'ютерні програми. «CAPTCHA» — це абревіатура від «Повністю автоматизований публічний тест Тьюрінга для розрізнення комп'ютерів і людей». CAPTCHA розроблено як прості проблеми, які можуть швидко вирішити люди, але комп'ютерам їх важко вирішити. Використовуючи CAPTCHA, служби можуть відрізняти законних користувачів від комп'ютерних ботів, вимагаючи від користувача мінімальних зусиль.

Ми представляємо нову CAPTCHA, яка вимагає від користувачів налаштування вертикальної орієнтації випадково повернутих зображень. Попередні дослідження показали, що люди можуть досягти рівня точності понад 90% для обертання зображень високої роздільної здатності до їхньої вертикальної орієнтації та можуть досягти рівня успішності приблизно 84% для ескізів зображень [27]. Однак поворот зображень у вертикальну орієнтацію є

складним завданням для комп'ютерів і може бути успішно виконано лише для підмножини зображень [15][19].

CAPTCHA використовується будь-яким веб-сайтом, який хоче обмежити використання ботами. Конкретне використання включає:

- Підтримка точності опитування — CAPTCHA може запобігти викривленню результатів опитування, гарантуючи, що кожен голос вводиться людиною. Хоча це не обмежує загальну кількість голосів, які можна зробити, це збільшує час, необхідний для кожного голосування, перешкоджаючи повторному голосуванню.
- Обмеження реєстрації для служб — служби можуть використовувати CAPTCHA, щоб запобігти розповсюдженню ботами спаму в системи реєстрації для створення підроблених облікових записів. Обмеження створення облікових записів запобігає марнотратству ресурсів служби та зменшує можливості для шахрайства.
- Запобігання помилкових/масових реєстрацій на безкоштовні події.

- Запобігання фальшивим коментарям. CAPTCHA може завадити роботам розповсюджувати спам на дошках оголошень, контактних формах або сайтах із відгуками.

Понад 6,3 мільйона веб-сайтів використовують CAPTCHA для захисту своїх веб-сайтів від зловмисних ботів.

Такий широкий спектр застосування CAPTCHA свідчить про високу актуальність розвитку даного напрямку, оскільки то як розвивається CAPTCHA з іншого боку стимулює розвиток різноманітних технологій і засобів її обходу.

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМИ

1.1 Види CAPTCHA та тест Тьюрінга

Сучасні CAPTCHA поділяються на три основні категорії: на основі тексту, на основі зображень та аудіо.

Текстові CAPTCHA – це оригінальний спосіб перевірки людей. Ці CAPTCHA можуть використовувати відомі слова чи фрази або випадкові комбінації цифр і літер записаних випадковим шрифтом якій складний для розпізнавання. Деякі текстові CAPTCHA також включають варіанти використання великих і малих літер.

CAPTCHA представляє ці символи у відчужений спосіб, який вимагає інтерпретації. Відчуження може передбачати масштабування, обертання та спотворення символів. Це також може включати накладання символів із графічними елементами, такими як колір, фоновий шум, лінії, дуги або точки. Це відчуження забезпечує захист від ботів з недостатніми алгоритмами розпізнавання тексту, але також може бути складним для тлумачення людьми.

Методи створення текстових CAPTCHA включають:

- Gimpy — вибирає довільну кількість слів зі словника з 850 слів і надає їх у спотвореному вигляді.

- EZ-Gimpy — це варіант Gimpy, який використовує лише одне слово.
- Gimpy-r — вибирає випадкові літери, потім спотворює та додає фоновий шум до символів.
- HIP — вибирає випадкові літери та цифри, а потім спотворює символи дугами та кольорами.

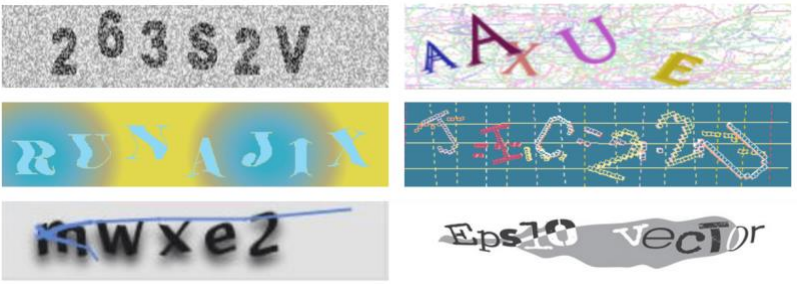

Рис. 1.1.1 - приклади текстового CAPTCHA

Фото-зображення CAPTCHA

CAPTCHA на основі зображень було розроблено, щоб замінити текстові. Ці CAPTCHA використовують впізнавані графічні елементи, такі як фотографії тварин, фігур або сцен. Зазвичай CAPTCHA на основі зображень вимагає від користувачів вибору зображень, які відповідають темі, або визначення зображень, які не підходять.

CAPTCHA на основі зображень, як правило, легше інтерпретувати людям, ніж на основі тексту. Однак ці інструменти створюють певні проблеми з доступністю для користувачів із вадами зору. Для ботів CAPTCHA на основі зображень складніше інтерпретувати, ніж текст, оскільки ці інструменти вимагають як розпізнавання зображень, так і семантичної класифікації.

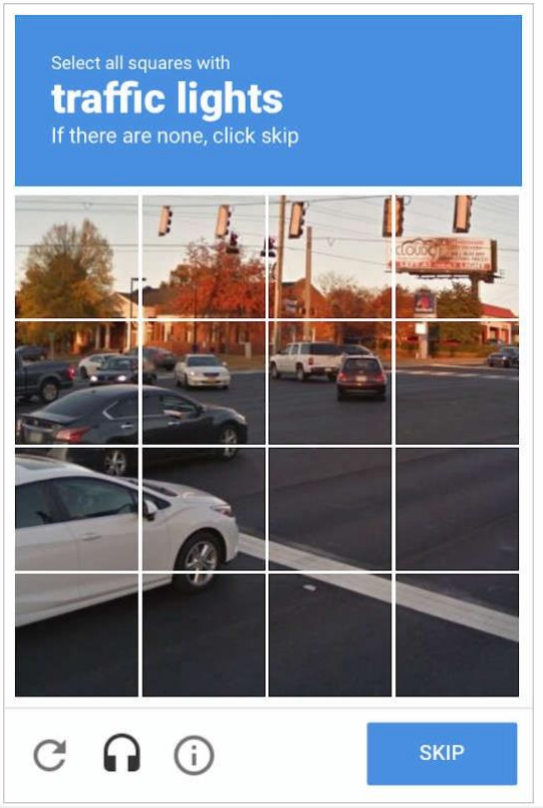

Рис. 1.1.2 - Фотографічні CAPTCHA

Аудіо CAPTCHA було розроблено як альтернативу, яка надає доступ користувачам із вадами зору. Ці CAPTCHA часто використовуються в поєднанні з текстовими або графічними CAPTCHA. Аудіо CAPTCHA представляє аудіозапис серії літер або цифр, які потім вводить користувач.

Ці CAPTCHA покладаються на те, що боти не можуть відрізнити відповідні символи від фонового шуму. Як і текстові CAPTCHA, ці інструменти можуть бути складними для інтерпретації як людьми, так і роботами.

Математичні або тестові завдання

Деякі механізми CAPTCHA пропонують користувачам вирішити просту математичну задачу, наприклад «3+4» або «18-3». Припускається, що боту буде важко ідентифікувати запитання та придумати відповідь. Іншим варіантом є текстова проблема, яка просить користувача ввести пропущене слово в реченні або завершити послідовність кількох пов’язаних термінів. Ці типи проблем доступні користувачам із вадами зору, але в той же час поганим роботам їх легше вирішити.

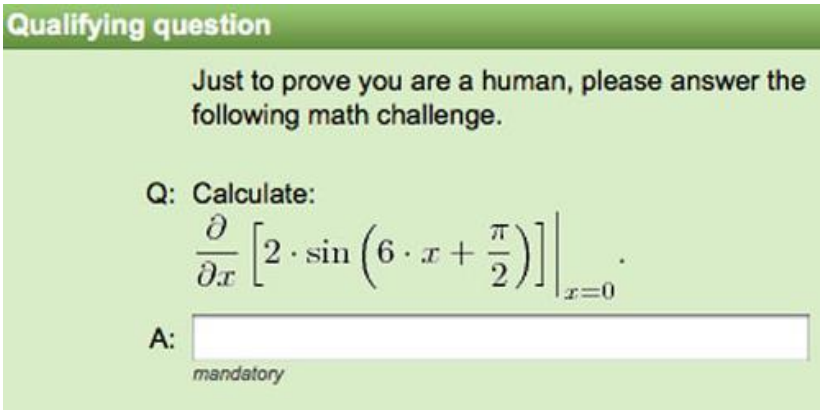

Рис. 1.1.3 - Математичні CAPTCHA

Тест Тьюрінга

Буква «T» у CAPTCHA розшифровується як тест Тьюрінга, концепція, створена Аланом Тьюрінгом у 1950 році. Тест Тьюрінга включає людину, яка спостерігає за двома учасниками розмови та визначає, хто з них — людина, а хто — штучний інтелект. CAPTCHA використовує цю концепцію та, як випливає з назви, автоматизує її.

Концепція CAPTCHA була винайдена в 1997 році командою людей, включаючи Мануэля Блума, Ніколаса Дж. Хоппера, Джона Лэнгфорда, Джілі Раанана, Ерана Решефа, Ейлона Солана та Луїса фон Ан. Вони написали про ідею та використання CAPTCHA у травневій публікації 2003 року під назвою «CAPTCHA: використання жорстких проблем

штучного інтелекту для безпеки», і того ж року ввели термін «CAPTCHA».

Одним із найперших застосувань CAPTCHA був тест Гаузебека-Левчина, засіб безпеки, створений Максом Левчіном і використаний PayPal у 2001 році.

1.2 Аналітичний огляд методів обходу CAPTCHA

Існує низка автоматизованих технологій, включаючи API, плагіни браузера та розширення, які дозволяють зловмисникам обходити або вирішувати виклики CAPTCHA. Ось кілька прикладів:

Група дослідників з Ланкастерського, Північно-західного та Пекінського університетів використала концепцію генеративної змагальної мережі (GAN), щоб створити надзвичайно швидкий і точний розв’язувач CAPTCHA.

Існує кілька безкоштовних онлайн-сервісів і бібліотек розв’язання CAPTCHA, які використовують технології глибокого навчання, зокрема GRIS, Alchemy, Clarifai та NeuralTalk. Академічні дослідження показують, що підходи на основі глибокого навчання дуже точні у вирішенні завдань CAPTCHA.

DeCaptcher є прикладом однієї зі служб вирішення проблем, доступних через API, що полегшує інтеграцію в програми. Базуючись на системі оптичного розпізнавання символів, сервіс розв'язує завдання та надає для завантаження файл із детальною інформацією про час, зображення завдання та текст, який використовується для вирішення завдання.

Інструменти з відкритим кодом і розширення для браузера, зокрема Buster і UnCaptcha, використовують розпізнавання аудіо, призначене для допомоги користувачам із вадами зору, і зловживають ним, щоб автоматично обходити механізми CAPTCHA.

1.3 Постановка задачі

Останнім часом багато CAPTCHA для розпізнавання символів було розшифровано за допомогою автоматизованих методів комп'ютерного зору. Ці методи були спеціально розроблені для видалення шуму та сегментації зображень, щоб зробити символи придатними для оптичного розпізнавання символів [3][4][5]. Через великі прагматичні та економічні стимули для спамерів подолати CAPTCHA, методи, запроваджені в наукових колах для подолання CAPTCHA, незабаром, ймовірно,

будуть широко використовуватися спамерами. Щоб звести до мінімуму успіх цих автоматизованих методів, системи збільшують шум і викривлення, які використовуються в цих CAPTCHA. На жаль, це не тільки ускладнює розв'язання комп'ютерами, але й ускладнює розв'язання людьми, що призводить до вищого рівня помилок [8][9] і пов'язаного з цим рівня розчарування.

Щоб вирішити цю проблему, було запропоновано численні альтернативні CAPTCHA (включно з зображеннями) [1][6][7][8]. Розробляючи нову CAPTCHA, слід пам'ятати основні принципи створення CAPTCHA (з [10]):

1. Легко для більшості людей
2. Важко вирішити автоматизованим роботам
3. Легко генерувати та оцінювати

Створити систему, яка відповідає першим двом вимогам, досить просто. Перша вимога свідчить про необхідність оцінювання зручності використання, що гарантує, що люди можуть розв'язати CAPTCHA за прийнятний проміжок часу та з прийнятним рівнем успіху. Друга вимога передбачає перевірку найсучасніших автоматизованих методів на відповідність CAPTCHA. У

запропонованій тут CAPTCHA ми гарантуємо, що автоматизовані методи не можуть бути використані для подолання нашої CAPTCHA, використовуючи їх для фільтрації зображень, які можна автоматично розпізнавати та орієнтувати.

Третю вимогу важче виконати; саме ця вимога є найбільшою проблемою для систем CAPTCHA на основі зображень. Першому успіху текстових CAPTCHA сприяла легкість їх створення – можна було вибирати випадкові послідовності літер, спотворювати їх і додавати відволікаючі пікселі, шуми, кольори тощо. Пізніше були запропоновані CAPTCHA на основі зображень, які вимагали від користувачів ідентифікувати зображення за допомогою міток.

Складність цих систем полягає в тому, що вони вимагають апріорного знання міток зображень. Надійні мітки недоступні для більшості зображень в Інтернеті, тому для отримання використовуються загальні методи включені етикетки:

(1) використання мітки, присвоєної зображенню пошуковою системою,

(2) використання контексту сторінки для визначення мітки,

(3) використання зображень, які були позначені, коли вони зустрічалися в іншому завданні, або
(4) використання ігор для отримання міток від користувачів (наприклад гра ESP [11]).

На жаль, багато разів ярлики, отримані першими двома методи часто галасливі та ненадійні на практиці, оскільки люди необхідні для ручної перевірки міток. Останні два підходи забезпечують менш шумні мітки. Однак навіть у тих випадках, коли мітки можна отримати, необхідно бути обережним, як вони використовуються. Може бути складно попросити користувача придумати мітку, якщо кожному зображенню не присвоєно багато міток. Крім того, якщо не введено точні збіги, відстані подібності між наданими та очікуваними відповідями може бути досить складним для обчислення (наприклад, можна використовувати ряд вимірювань: відстань редагування, семантичну відстань спеціального призначення, відстань тезауруса, відстань word-net, тощо).

2. МЕТОДИКА ДОСЛІДЖЕННЯ

2.1 Проведення тест-навчання системи

При побудові і вибору методу формування CAPTCHA необхідно враховувати фактори UI/UX оскільки існує велика кількість негативних відгуків від користувачів, оскільки кожного разу користувачу потрібно буде подавати CAPTCHA.

Домінуючим підходом стосовно застосування CAPTCHA є пазли, оскільки існує твердження що лише людина здатна розв'язувати пазли.

Існує два основні підходи до реалізації автоматичного розв’язувача CAPTCHA:

1. Використання сторонніх сервісів аналізу CAPTCHA.
2. Створення бота, який використовує оптичне розпізнавання символів (OCR), щоб спробувати розгадати символи CAPTCHA.

На даний момент існує кілька постачальників сторонніх послуг розв’язання CAPTCHA, наприклад: Death by CAPTCHA (http://deathbyCAPTCHA.com), de-captcher (http://www.de-captcher.com/) і decaptcher2 (http://decaptcher2.com/) [7]-[9]. Більшість цих служб працює за допомогою «людської автоматизації», тобто вони

використовують людську автоматизацію для розпізнавання символів CAPTCHA та надсилання вам результату. Типовим є приклад коли, такі системи інтегруються до сайтів забороненого контенту, де глядачам періодично доводиться вводити CAPTCHA для продовження перегляду, зрозуміло, що такі користувачі не будуть скаржитись.

Розглянемо переваги та недоліки таких підходів. Плюси: вірогідність точності вища, ніж використання підходу OCR, оскільки автоматизація людини за своєю суттю краще розпізнає CAPTCHA, ніж машини, а постачальники послуг зазвичай надають вам простий у використанні API для взаємодії з їх службою розпізнавання CAPTCHA через мережу.

Мінуси: вартість великої кількості запитів CAPTCHA є надзвичайно високою, оскільки вона швидко зростає з часом і виникає проблема затримки. Якщо швидкість розпізнавання CAPTCHA не відповідає вашій вимозі щодо «тайм-ауту» розв’язання — в останньому випадку CAPTCHA розгадується правильно, але це займає надто багато часу, тому що сеанс для сторінки розв’язання CAPTCHA закінчився, що вимагає повторення вводу.

За результатами проведеного аналізу,, служби розв’язання CAPTCHA, як правило, краще розв’язують CAPTCHA — відносно підходу OCR — але мають вищезгадану проблему затримки.

Другий підхід набагато складніший за перший, ніж використання сторонніх служб розв’язання CAPTCHA. Однак йому не вистачає точності порівняно з першим підходом. Крім того, другий підхід не міг вирішити складні CAPTCHA у багатьох ситуаціях. Однак для досить тривіальних CAPTCHA другий підхід є набагато економічнішим і більш-менш зручним. Ви можете бути здивовані тим, що на практиці тривіальні CAPTCHA все ще широко використовуються, особливо для веб-сайтів із дуже специфічними послугами, як-от оператори мобільного зв’язку (зазвичай передплачені, де абоненти можуть поповнювати свій рахунок через Інтернет, іншим прикладом є продаж квитків онлайн). для заходів і так далі. Ці постачальники послуг не мають багато звернень, тому що лише ті, хто хоче скористатися їхніми послугами, відвідуватимуть їхні веб-сайти. Можливо, це причина, чому вони не використовують складні CAPTCHA, або, можливо, тривіального CAPTCHA їм достатньо.

Надалі ми плануємо створювати бот який буде будуватись на базі OCR обробки CAPTCHA.

2.2 Обробка текстових CAPTCHA на основі OCR

Текстові CAPTCHA використовуються найчастіше, але через свою просту структуру вони також є найбільш вразливими. Цей тип капчі зазвичай вимагає розпізнавання геометрично спотвореної послідовності літер і цифр.

Щоб подолати текстову CAPTCHA, зазвичай потрібні три кроки: попередня обробка за допомогою видалення шуму, сегментація для отримання окремих символів і розпізнавання щоб ідентифікувати кожний символ. Ці три кроки розглядаються як однаково важливі і в принципі можуть бути представлені як кожна окрема задача.

Що стосується етапу попередньої обробки, оскільки на продуктивність може впливати багато шуму, ми повинні зменшити ефект, щоб отримати чіткі зображення для кращої продуктивності. В даний час для досягнення мети пропонується безліч методів, як і деякі способи обробки зображень і алгоритми машинного навчання, такі як медіанний фільтр [4], фільтр сусідства, вейвлет-поріг, алгоритм K-найближчих сусідів, опорна векторна машина

тощо. Однак тільки тоді, коли це доречно вибрано метод шумозаглушення, ми можемо отримати найчіткіший результат [5].

Після попередньої обробки ми повинні розділити зображення на окремі символи. Це пояснюється тим, що якщо зображення можна ідеально розділити, це буде корисно для наступного кроку щодо точності розпізнавання. Як ми вже згадували раніше, що зображення CAPTCHA не завжди однакові, тому результати подалання CAPTCHA сильно залежать від чіткості деталей зображення, інформаційного контексту, тощо.. Після знешумлення найбільш розповсюдженим підходом до сегментації є побудова гістограми інтенсивності зображення та кластеризації кольорів [6].

Розпізнавання є останнім кроком для отримання результату подолання CAPTCHA. Завдяки численним наявним дослідження дослідженням ми дійшли висновку про три швидкі та надійні підходи до розпізнавання:

1. способами, це оптичне розпізнавання символів (OCR),
2. зіставлення шаблонів (TM)
3. згорточна нейронна мережа (CNN).

Перш за все, OCR використовується для конвертації символів-слова на зображеннях до формального друкованого тексту [7]. Насправді OCR застосовано вже зараз широко застосовується у широкому спектрі прикладних задач з розпізнавання текстів у таких документах, як контракти, паспорти, квитанції, кредитні картки. Крім того, було багато комерційних застосувань розроблений для завдання ідентифікації машинно набраного тексту і навіть таблиць. При тому, поки OCR не вдалося подалати CAPTCHA через різні причин:

1. деякі комерційні OCR методи можуть добре працювати лише на чорно-білих зображеннях;
2. частина з них працює на не зашумленому зображені.

При цьому більшість CAPTCHA які нам доводилось зустрічати мають кольоровий текст, та навмисно зашумлені випадковим чином, та навіть повернуті [8] [1]. Крім того, є приклади наших наборів даних з застосуванням різних шрифтів, напрямків і навіть інших спотворень. Так що поєднання із цих існуючих проблем не можна легко вирішити, використовуючи лише OCR.

Для підвищення безпеки використовуються різні механізми захисту, які можна розділити на антисегментацію та антирозпізнавання. Перша група

механізмів спрямована на ускладнення процесу поділу символів, а друга група — на розпізнавання самих символів. На таблиці 2.1 показані приклади різних підходів до захисту від CAPTCHA.

Таблиця 2.1 - Види захисту текстових CAPTCHA

Механізм захисту		Приклади
Проти сегментації	Зміна символів та їх контурів	
	Спільне розташування тексту CCT	
	Зашумлення фону	
	Багато рівні зображення	
Проти розпізна-вання	Різні шрифти, зсуви, повороти	
	Різні мови тексту	

Як видно з таблиці 2.1 зазначені захисти значно ускладнюють алгоритмам OCR можливість автоматичного розпізнавання з іншого боку це створює додатковий дискомфорт і користувачам які навіть власними очима не можуть чітко побачити що саме їм потрібно ввести і інколи потрібно повторювати спробу заповлення CAPTCHA декілька разів.

Розуміння того як сучасни технологію намагаються обходити CAPTCHA ми зможемо побудувати зручну в використанні та надійну в сенсі OCR систему.

Таким чином, схема роботи бота-розпінавача CAPTCHA будується наступним чином рис 2.2.1

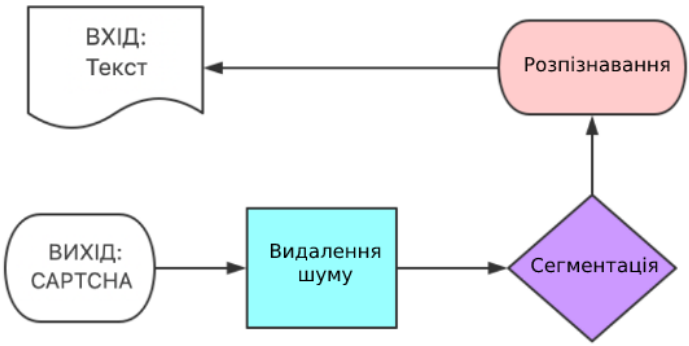

Рис. 2.2.1 - Загальна схема роботи бота

Почнемо з першого кроку - видалення шуму. Існує кілька можливих методів зменшення шуму на зображеннях.

Огляд існуючих рішень дає розуміння, що медіанний фільтр ефективний для нелінійних методів фільтрації шуму. Також важливу роль відіграє афінне перетворення у попередній обробці зображення. Насправді афінне перетворення не обов'язково зберігає кути між лініями або відстані між точками, хоча воно зберігає відношення відстаней між точками, що лежать на прямій [17]. Після афінного перетворення, набори паралельних прямих все ще можуть залишатися паралельними. Під час видалення шуму найбільша проблема це визначення розміру порогового значення. Основна складність тут – знайти оптимальну інтенсивність всього зображення та різницю між реальним зображенням і шумом, оскільки там немає прямого звязку між пікселями. Ніхто не може гарантувати, що пікселі між собою, визначені процесом порогового значення, є безперервними. Ми можемо легко включити пікселі які не відносяться до зображення і визначити їх як частина потрібного регіону реального зображення при тому якщо вони є шумом і навпаки видалити частину реального зображення визначавши його як шум. Ці ефекти отримують дедалі гірше, оскільки тип шуму стає дедалі складнішим, просто тому, що більш імовірно, що інтенсивність пікселя не може представляти вільний шум інтенсивність

зображення в регіоні [18]. Коли ми використовуємо пороговий метод, ми зазвичай маємо балансувати з компромісом у плані втрати занадто великої частини інформації з визначеного регіону. Ще тіні що є на зображенні можуть бути прийняте за шум.

Медіанний фільтр ефективний для видалення нелінійного шуму з цифрового зображення, а також збереження країв під час видалення шуму. Зниження шуму є типовим етапом попередньої обробки для покращення кінцевих результатів подальшої обробки (наприклад, виявлення країв на зображенні). Основна ідея медіанного фільтра - пробігти по зображенню, замінюючи кожен піксель медіаною сусідніх пикселів.

Наведемо класичний медіанний алгоритм: нехай дано зображення X розміру n*m з радіусом ядра t та гістограмою розподілу H. На виході необхідно отримати нове зображення Y розміру також n*m за наступною процедурою:

```
for i = 1 to m do
        for j = 1 to n do
                for k = -τ to τ do
                        Видалити Xi+k,j−t−1 з H
                        Додати Xi+k,j+t до H
```

end for

Y i,j ← median(H)

end for

end for

Афінне перетворення нам відомо з геометрії, де афінне перетворення або афінне відображення або спорідненість - це функція між афінними просторами, яка зберігає точки прямої прямих і площин. Афінне перетворення може не обов'язково зберігати кути між лініями або відстані між точками, хоча він зберігає їх співвідношення та відстані між точками, що лежать на прямій [17].

Афінна карта складається з двох функцій: трансляції та лінійної карти.

Звичайна векторна алгебра представляє лінійні відображення множенням матриць, іі представляє переклади векторним додаванням. Якщо лінійна карта представлена як множення на матрицю A і переклад як додавання вектора b, афінне відображення f, що діє на вектор x, можна представити як:

$$y = f(x) = Ax + b$$

Рисунок 2.2.2 наочно демонструє суть афінного перетворення.

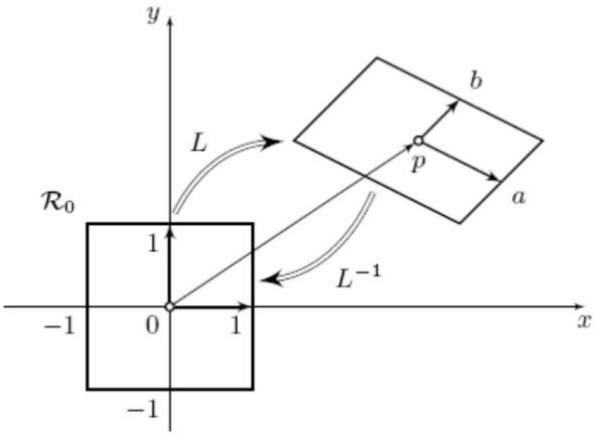

Рис 2.2.2 - Класичне представлення афінного перетворення

Сегментація є досить важливою, її результати можуть сильно вплинути на кінцевий результат. Найпоширенішим способом аналізу сегментації є розділення CAPTCHA на зображення з кількох окремих частин. Сегментація використовується для виявлення прикладів меж CAPTCHA, таких як лінії або криві, з урахуванням обертання, шуму і навіть перекручення символів.

Кластеризація на основі гістограми та K-середніх є двома ефективними методами сегментації. На основі гістограми підраховується кількість пікселів у кожному

рядку або стовпці у інтенсивності сірого. Базовий алгоритм буде проілюстровано як y0, y1, ... yn, де yi кількість пікселів у зображенні з рівнем сірого i, а n – максимальний досягнутий рівень сірого. Уявіть, що відстань на гістограмі між кожним символом дуже велика то його було б легко відокремити.

Тоді як K-середніх визначає кластеризацію і це інший метод сегментації. Насправді сегментація дуже залежна від структури CAPTCHA і може значно відрізнятися. Поки що немає ефективних алгоритми для сегментації скручених, зігнутих. Існує кілька поколінь методів K-середніх. Нижче наведено базовий псевдокод алгоритму.

Крок 1. Виберемо K точок в центроїді

Крок 2. Для K кластерів знайдемо всі відповідні точки до найблищого центроїду

Крок 3. Переобчислити центроїди для всіх кластерів

Крок 4. Якщо центроїди змінились перйти на **Крок 2**.

Останнім кроком є автоматичне розпізнавання символів та отримання друкованого повідомлення для завершення всього процесу розв'язання CAPTCHA. Це можна зробити одним з трьох способів: оптичне розпізнавання символів (OCR), шаблонне зіставлення і

згорточна нейронна мережа (CNN). Для OCR ми вибираємо програмне забезпечення з відкритим кодом під назвою Tesseract від Google. Якщо формат виглядає як сітка, точність розпізнавання буде кращою.

Поки не всі CAPTCHA подібні, у деяких нестандартних випадках застосування шаблонного зіставлення є не прийнятним оскільки пошук схожих кандидатів може бути достатньо довгим. Ідея полягає в тому, щоб перемістити шаблонне зображення поверх вхідного зображення, а потім отримати подібність матриці один одного, щоб отримати найкращого кандидата.

CNN є більш складною технікою штучного інтелекту порівняно з OCR і TM. Оскільки для навчання потрібно набагато більше попередньо визначених даних і класифікатор для системи з точки зору більш високого рівня точності та меншої складності за часом. Тому, що чим більше даних ми можемо використовувати, тим кращий класифікатор ми отримаємо. Обмеження CNN також очевидні з точки зору розміру наборів даних, оскільки ми потрібно спочатку вручну зняти шум із зображення, а потім сегментувати їх на окремі символи, тому робоча процедура є порівняно складнішою, ніж у інші.

2.3 Вибір між OpenCV та Tesseract

При створенні різних застосунків що використовують OCR значна кількість дослідників вагається яку саме бібліотеку вибрати для використання. Слід зазначити, що чітка та однозначна відповідь на це питання не існує. Кожна з цих бібліотек має свої переваги та недоліки і питання вибору є в більшій мірі суб'єктивним.

Для того, щоб визначитись з бібліотекою в нашій роботі було враховане наступне:

Tesseract — це двигун OCR. Він використовується, розробляється та фінансується Google спеціально для читання тексту із зображень, виконання базової сегментації документів і роботи з певними вихідними зображеннями (окреме слово, рядок, абзац, сторінка, обмежені словники тощо).

OpenCV, з іншого боку, — це бібліотека комп’ютерного зору, яка містить функції, які дозволяють виконувати визначення характеристик даних і їх класифікацію. З допомогою OpenCV можна створити простий сегментатор літер і класифікатор, який виконує базове оптичне розпізнавання символів, але це не дуже хороший механізм оптичного розпізнавання символів.

Розробники описують OpenCV як «бібліотеку комп'ютерного зору з відкритим кодом». OpenCV було розроблено для ефективності обчислень і з сильним акцентом на програмах та алгоритмах реального часу. Бібліотека, написана на оптимізованим C/C++, може скористатися перевагами багатоядерної розподіленої обробки. Завдяки підтримці OpenCL він може використовувати переваги апаратного прискорення основної гетерогенної обчислювальної платформи. З іншого боку, Tesseract OCR позиціонується як «Tesseract Open Source OCR Engine». Tesseract був спочатку розроблений у Hewlett-Packard Laboratories Bristol та Hewlett-Packard Co, Greeley Colorado між 1985 і 1994 роками, з деякими змінами, внесеними в 1996 році для перенесення на Windows, і деякими C++ оптимізацію у 1998 році. У 2005 Tesseract став двигуном з відкритим кодом. З 2006 року його розробляє і підтримує Google.

OpenCV належить до категорії «Обробка та керування зображеннями» з поглибленням в аспекти технічного стеку, тоді як Tesseract OCR можна позиціонувати в основному як «API аналізу зображень».

Виходячи з цих міркувань де нам потрібно саме готове рішення з OCR для розпізнавання зображень ми і вибираємо Tesseract.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Види науково-дослідних робіт студентів

В якості базового модулю системи будемо використовувати відкриту бібліотеку оптичного розпізнавання символів, щоб створити наш тестовий бот-розв’язувач CAPTCHA. Деталі інструментів для отримання зображень CAPTCHA тут не пояснюватимуться. Основна увага зосереджена лише на створенні невеликої програми для вирішення легкодоступного зображення CAPTCHA. Незважаючи на це, у цій статті пояснюється загальна архітектура повного рішення CAPTCHA.

Загальна архітектура рішення CAPTCHA показана на рис 3.1.1. Існує два основні компоненти рішення CAPTCHA: веб-парсер і сам інструмент CAPTCHA, як показано на 3.1.

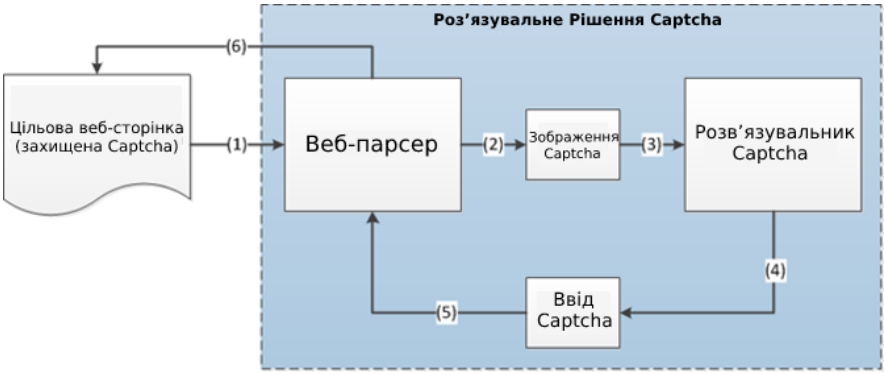

Рис. 3.1.1 - загальна структура CAPTCHA бота

Метою веб-парсера є сканування цільової веб-сторінки, тобто «перегляд» цільової веб-сторінки так, ніби людина переглядає веб-сторінку, отримання даних, необхідних для обробки сторінки та надсилання «автоматичного» зворотного зв'язку на цільову веб-сторінку. Тобто на даному етапі для підготовки розпізнавання нам потрібно імітувати звернення на ресурс імітуючи людину, в тому числі зі збереженням cookie та інших сесійних даних.

Наприклад, якщо веб-форма знаходиться на цільовій веб-сторінці https://sumdu.edu.ua/administrator/, веб-парсер витягує структуру форми <form ../> з веб-сторінки, потім веб-парсер передає керування програмі яка розпізнає CAPTCHA і у відповідь заповнює необхідні дані для записів

форми та надсилає цю «відповідь» на цільову веб-сторінку. — ніби людина вводить необхідні дані, а потім імітує натискання кнопку «Надіслати» на цільовій веб-сторінці. На більш складний цільовій веб-сторінці процес введення даних захищено CAPTCHA або двократним проходженням CAPTCHA. Таким чином, веб-парсер повинен викликати та ореалізувати розв’язувач CAPTCHA, щоб виконати вимогу перевірки CAPTCHA.

Давайте розглянемо рішення більш детально. Це кроки, виконані:

1. Веб-парсер отримує вміст цільової веб-сторінки.
2. Веб-парсер витягує зображення CAPTCHA з цільової веб-сторінки.
3. Зображення CAPTCHA надсилається до програми CAPTCHA.
4. Програма CAPTCHA розв’язує CAPTCHA та видає рядок CAPTCHA як результат.
5. Рядок CAPTCHA надсилається назад до веб-парсер.
6. Веб-парсер надсилає відповідь, включно з рядком CAPTCHA, на цільову URL-адресу веб-сторінки.

В даній роботі ми фокусуємося на створенні компоненту розв'язування CAPTCHA та аналізу

захищеності від розв'язання. Що стосується веб-парсера, то це інша тема, яка залежить від специфіки веб-сайту, який опрацьовується.

3.2 Опис генератора зображення

Для генерування зображень CAPTCHA ми будемо використовувати мову програмування Python оскільки вона має готовий набір бібліотек для роботи з зображеннями і дозволяє генерувати початкове зображення та змінювати його. Відповідно автоматично ми маємо механізми для масштабування, повороту, спотворення символів. Це також може включати накладання символів із графічними елементами, такими як колір, фоновий шум, лінії, дуги або точки. Це зашумлення забезпечує захист від ботів із недостатніми алгоритмами розпізнавання тексту, але при зловживання цими методами також може бути складним для тлумачення людьми.

Для встановлення бібліотеки роботи з CAPTCHA на python необхідно виконати наступну команду:

```
1. pip install captcha
```

Після цього імпортуємо модуль та створюємо представника ImageCaptcha()

```
1. image = ImageCaptcha(width = 280, height = 90)
```

Визначаючи розміри результуючого зображення. На наступному кроці необхідно підготувати сам текст який буде згенеровано на CAPTCHA зображенні. Враховуючи результати наведені в другому розділі ми будемо використовувати цифрові послідовності які необхідно людині розташувати у правильному порядку і після цього ввести текст який зображено на CAPTCHA.

Для створення зображення у вигляді пазла необхідно розділити початкове зображення на певну кількість блоків та перемішати їх. Для цього будемо використовувати бібліотеку що вміє це робити, а саме бібліотеку split-image. Для її встановлення виконаємо команду:

```
1. pip install split-image
```

Дана бібліотека має консольний інтерфейс та може бути застосована з наступним синтаксисом:

```
1. split-image [-h] [-s] image_path rows cols
```

У випадку її інтегрування в діючий python доданок вона використовується у наступному вигляді:

```
1. from split_image import split_image
2.
3. split_image(image_path, rows, cols, should_square,
   should_cleanup, [output_dir])
4. # наприклад split_image("CAPTCHA.jpg", 2, 2, True,
   False)
```

В результаті роботи цього метода з суцільного зображення отримуємо декілька файлів, приклад роботи функції показано на рис 3.2.1.

В поточній директорії ми отримуємо список файлів з фрагментами зображення, наприклад

- CAPTCHA_0.jpg
- CAPTCHA_1.jpg
- CAPTCHA_2.jpg
- CAPTCHA_3.jpg

Дана нумерація нам дає розуміння про правильний порядок складання пазлу та дає змогу виконати

перемішування зображень використовуючи їх індекси в іменах файлів.

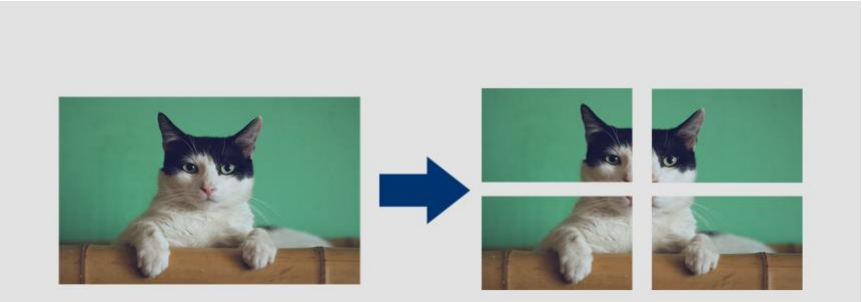

Рис 3.2.1 - Приклад роботи функції *split_image*

При тому що, кількість розбиттів ми можемо регулювати параметрами rows та cols, що дозволяє створювати більш масові розбиття показані рис 3.2.1.

Перемішування зображень відбувається з використанням наступного фрагменту

```
1. from random import shuffle
2.
3. images = [[i] for i in range(image.size())]
4. shuffle(images )
5. print(images)
```

Рис. 3.2.2 - Розбиття зображення з використанням параметрів 3 на 4

Підставляємо згенерований контекст в метод для генерування результуючого зображення.

```
1. data = image.generate(obj)
```

Та зберігаємо результат у вигляді одного графічного файла з наперед заданим ім'ям наприклад CAPTCHA.png.

```
1. image.write(obj, 'CAPTCHA.png')
```

Таким чином згенероване зображення може бути вставлене в будь-який web-доданок у вигляді фрейма чи спливаючого вікна, що дозволяє його широко використовувати для будь-якої платформи, навіть на мобільних пристроях.

3.3 Структура модуля розпізнавання

Одним із способів автоматичного захисту від CAPTCHA є використання бібліотеки OCR для розпізнавання рядка в CAPTCHA. Всупереч тому, що ви могли б подумати, бібліотека OCR розпізнає рядок не лише за допомогою спроб розпізнати окремі літери (та цифри), але й за допомогою контекстної інформації. Наприклад, якщо ви знаєте, що рядок, який ви намагаєтеся розпізнати, містить лише літери, ви можете передати цю інформацію в бібліотеку OCR, щоб підвищити точність розпізнавання. Так само, якщо цільовий рядок містить лише цифри без алфавіту, ви можете вказати бібліотеці розпізнавати лише цифри, а не літери. Іншим можливим контекстом є мова рядка, який ви намагаєтеся розпізнати. Окремі бібліотеки можуть отримувати контексту інформацію стосовно можливого типу зображення що розпізнається, наприклад квіт, транспорт, обличчя, тощо.

Для реалізації модуля розв'язання CAPTCHA будемо використовувати бібліотеку Tesseract OCR [15] з відкритим кодом. Бібліотека доступна за адресою https://code.google.com/p/tesseract-ocr/. Tesseract написаний мовою C++. Таким чином, найприродніший спосіб його використання – це написати свій розв’язувач CAPTCHA на

C++ або C. Однак необхідно врахувати, що C++ використовує простори імен - namespaces, тобто ім'я функції у вихідному коді не збігається з ім'ям у вихідному коді. Скомпільований об'єктний файл, dll або виконуваний файл, створений компілятором.

Для використання Tesseract в своїй програмі, необхідно задовольнити його залежить від Leptonica, іншої бібліотеки з відкритим кодом, яка обробляє різні формати файлів зображень. Таким чином, необхідно встановити та поєднати потрібно зв'язатися з Leptonica та Tesseract у вашій програмі, щоб використовувати Tesseract OCR для вирішення CAPTCHA.

Наведена тут реалізація є специфічною для Windows. Ви можете завантажити код Visual Studio 2008 для Tesseract за цим посиланням: https://code.google.com/p/tesseract-ocr/downloads/detail?name=tesseract-ocr-3.02-vs2008.zip&can=2&q=.

Задовільтини залежність від Leptonica v1.68 тут: https://code.google.com/p/leptonica/downloads/list. Заради переносимості між різними мовами тут реалізовано у формі «простої» бібліотеки Windows DLL, яка взаємодіє з DLL Tesseract — і опосередковано з DLL Leptonica, оскільки

Tesseract залежить від Leptonica. Нижче наведено приклад інтеграції бібліотек рис 3.3.1

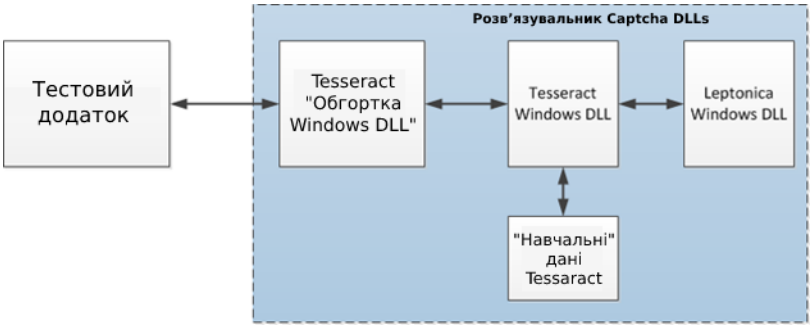

Рис. 3.3.1 - Схема використання відкритих бібліотек

На рисунку 3.3.1 показано як створюються дві речі: по-перше, це код оболонки DLL Windows, а по-друге, це програма-розпізнавач, що безпосередньо використовує щоб розв'язати капчу. Код оболонки DLL Windows складається з двох файлів: CAPTCHA_solver_dll.h і CAPTCHA_solver_dll.cpp.

Перед використанням бібліотеки, необхідно провести її навчання Після установки Tesseract на своїй машині, ці дані розміщуються в каталозі tessdata в каталозі встановлення Tesseract. Наявність навчальних даних є обов'язковою, оскільки при зміні структури вхідного зображення потрібно також оновлювати навчальні дані.

Однак вам потрібно мати «навчальні» дані Tesseract — каталог tessdata та його вміст — десь на машині, яка запускатиме вашу програму, і ви повинні встановити змінну середовища TESSDATA_PREFIX на абсолютний шлях до каталога, що містить каталог tessdata, а не шлях до каталога tessdata.

Зміну даного шляху можна зробити через через Панель керування|Система|Додаткові параметри системи|Змінні середовища|Системні змінні.

Після цього настійно радимо вийти з системи та знову ввійти або перезавантажити машину, оскільки іноді нова змінна середовища не оновлюється, як ми хотіли, якщо ви цього не зробите. Налаштування змінної середовища TESSDATA_PREFIX потрібне, оскільки Tesseract вимагає цієї змінної середовища під час виконання запиту до даних «навчання».

3.4 Опис застосування Tesseract

Тепер перейдемо до деталей використання Tesseract у файлі CAPTCHA_solver_dll.cpp. Користуватися Tesseract досить просто. Наступні кроки застосовуються для розпізнавання зображення CAPTCHA за допомогою Tesseract:

1. Ініціалізувати об’єкт API tesseract, який буде використовуватися.
2. Перевірка, чи підтримується формат вхідного файлу.
3. Обробити вхідний файл зображення, щоб отримати рядок CAPTCHA.
4. Скопіюйте рядок результату до вихідного буфера. Це потрібно, тому що Tesseract використовує внутрішнє представлення рядка, яке не гарантовано сумісне з потрібним нам форматом рядка — звичайний рядок C/C++, тобто рядок із закінченням \0.

Нижче показано код на C++ який реалізує на алгоритм описаний вище.

```
1. #include “stdafx.h”
2. #include “CAPTCHA_solver_dll.h”
```

```
3.
4. …
5.
6. // Рядок-результат обробки CAPTCHA
7. static char g_CAPTCHA_string[MAX_CAPTCHA_STRING_LENGTH + 1];
8.
9. …
10.
11. // Головна функція.
12. ///
13. /// Дана функція викликає бібліотеку передаючи їй зображення image
14. /// як вхідний image_file_path параметр.
15. ///
16. /// Шлях до файла.
17. /// В якості результата отримуємо покажчик на рядок розпізнавання
18. ///
19. CAPTCHA_SOLVER_DLL_API char* solve_CAPTCHA( const char* image_file_path )
20. {
```

```
21. //
22. // Крок 1: Ініціалізація об'єктів бібліотеки
23. //
24. const char* lang = “eng”;
25. char* config_file_path = “digits”; /* Hardcode the config
    file to be used to “$TESSDATA_PREFIX/configs/digits”
26. NOTE: As long as $TESSDATA_PREFIX has been
27. exported to as Windows environment variable,
28. using only the word “digits” here should work. */
29. tesseract::TessBaseAPI api;
30.
31. api.Init(image_file_path /* шлях до файлу*/,
32. lang /* мова */,
33. tesseract::OEM_DEFAULT /* режим OCR */,
34. &config_file_path /* char **configs */,
35. 1 /* configs_size — файл конфігурації config_file_path
    */,
36. NULL /* додаткові налаштування бібліотеки const
    GenericVector *vars_vec */,
37. NULL ,
38. false /* включення режиму відлагодження
    set_only_non_debug_params */);
39.
```

```
40. tesseract::PageSegMode pagesegmode = tesseract::PSM_AUTO;
41.
42. if (api.GetPageSegMode() == tesseract::PSM_SINGLE_BLOCK)
43. api.SetPageSegMode(pagesegmode);
44.
45. //
46. // Крок 2: Перевірка відповідності файла формату що підтримується
47. //
48. FILE* fin = fopen(image_file_path, “rb”);
49. if (fin == NULL) {
50. return NULL;
51. }
52. fclose(fin);
53.
54. PIX *pixs;
55. if ((pixs = pixRead(image_file_path)) == NULL) {
56. return NULL;
57. }
58. pixDestroy(&pixs);
59.
```

```
60. //
61. // Крок 3: Обробка зображення.
62. //Результат - це об'єкт STRING який передається в змінну by text_out.
63. //
64. STRING text_out;
65. if (!api.ProcessPages(image_file_path, NULL, 0, &text_out)) {
66. return NULL;
67. }
68.
69. //
70. // Крок 4: Копіювання результатів в буфер виводу
71. // a. Використовуємо text_out.strdup() для отримання покажчика на об'єкт копіювання результати розпізнавання CAPTCHA.
72. // b. Очищаэмо память що була використана під час копіювання результатів роботи бібліотеки.
73. //
74. memset(g_CAPTCHA_string, '\0', sizeof(g_CAPTCHA_string));
75. char* result = text_out.strdup();
```

```
76. strncpy(g_CAPTCHA_string,                result,
    sizeof(g_CAPTCHA_string));
77. free(result);
78.
79. return g_CAPTCHA_string;
80. }
```

Як показано в функції solve_CAPTCHA(), яка викликає Tesseract для «вирішення» розпізнавання рядка CAPTCHA, переданого у вхідному зображенні CAPTCHA, переданому функції через вхідний параметр image_file_path. Це єдина функція, яка потрібна програмі для використання Tesseract через нашу обгортку DLL для Windows. Вхідний параметр image_file_path у функції solve_CAPTCHA() містить шлях до зображення CAPTCHA, яке потрібно виконати. В даному випадку це не весь код, а лише та частина яка міститься у CAPTCHA_solver_dll.cpp, лише той, який важливий для реалізації дуже тонкої оболонки для Tesseract.

Об’єкт PIX у коді наведеному вище є об’єктом Leptonica. Об’єкт PIX обробляє вхідне зображення, яке буде передано Tesseract. Більшу частину обробки зображень у Tesseract виконує Leptonica.

Ідентифікатор CAPTCHA_SOLVER_DLL_API в є макросом для визначення типу зв'язку функції. Можна побачити деталі цього ідентифікатора нижче в (CAPTCHA_solver_dll.h).

Ідентифікатор CAPTCHA_SOLVER_DLL_API в зіставляється з __declspec(dllexport), оскільки константа CAPTCHA_SOLVER_DLL_EXPORTS визначена в налаштуваннях препроцесора проєкту Visual Studio, що містить файл CAPTCHA_solver_dll.cpp. Як можна побачити, якщо визначено константу CAPTCHA_SOLVER_DLL_EXPORTS, ідентифікатор CAPTCHA_SOLVER_DLL_API перетворюється на __declspec(dllexport). Це, дає «контекстну» підказку — або ж евристику — для Tesseract у формі налаштувань мови та налаштувань файлу конфігурації. Мова встановлена на англійську, а файл конфігурації налаштований лише на цифри, тобто Tesseract має інтерпретувати вхідні дані лише як цифри. Це робиться на кроці 1 під час загального конфігурування. Якщо контест змінюється конфігурування потрібно зробити заново.

Це можна зробити, оскільки припускається, що ми виконали попередню оцінку цільової CAPTCHA, і в

результаті вхідна CAPTCHA завжди складається з цифр, якщо це не так конфігурацію потрібно міняти оскільки якість отриманих зображень суттєво падає.

```
1. #define __CAPTCHA_SOLVER_DLL_H__
2.
3. // Наступни блок ifdef це стандартний підхід до сткорення макросів яки використовуються для спрощення екпорту методів з DLL
4. // Всі файли даної DLL скомпільовані зі значенням константи
5. // CAPTCHA_SOLVER_DLL_EXPORTS визначеної в командному рядку.
6.
7. #ifdef CAPTCHA_SOLVER_DLL_EXPORTS
8. #define CAPTCHA_SOLVER_DLL_API __declspec(dllexport)
9. #else
10. #define CAPTCHA_SOLVER_DLL_API __declspec(dllimport)
11. #endif
12.
13. #ifndef MAX_CAPTCHA_STRING_LENGTH
14. #define MAX_CAPTCHA_STRING_LENGTH 256
```

```
15. #endif
16.
17. #ifdef __cplusplus
18. extern “C” {
19. #endif
20.
21. CAPTCHA_SOLVER_DLL_API char*
    solve_CAPTCHA( const char* image_file_path );
22.
23. #ifdef __cplusplus
24. }
25. #endif
26.
27. #endif // __CAPTCHA_SOLVER_DLL_H__
28. [/c]
```

Після завершення обгортки Windows DLL ми можемо перейти до вихідного коду основної програми. Дана програма показує вихідний код тестової програми для нашої бібліотеки обгортки Tesseract. Важливе уточнення, що дана обгортка зроблена під операційну систему Windows, у випадку коли дана система буде запускатись на інших операційних системах то потрібно в вихідний код обгортки вносити зміни.

Для компіляції проєкту в Visual Studio, установіть набір символів у налаштуваннях проєкту на Багатобайтовий набір символів (MBCS)—через «Властивості проєкту»|Властивості конфігурації|За замовчуванням|Набір символів. Цей параметр наказує Visual Studio скомпілювати проєкт у режимі MBCS, тобто ANSI C-сумісному режимі. Таким чином, обробка рядка в коді буде встановлена на «режим» рядка C ANSI. Це важливо зробити, тому що за замовчуванням Visual Studio встановлює набір символів у Unicode, що не сумісно з виводом бібліотеки обгортки Tesseract, яку ми створили раніше.

```
1. // CAPTCHA_solver_dll_main_app.cpp : Точка входу консольного доданку.
2. //
3.
4. #include “stdafx.h”
5. #include “CAPTCHA_solver_dll.h”
6.
7. int _tmain(int argc, _TCHAR* argv[])
8. {
9. char CAPTCHA_string[MAX_CAPTCHA_STRING_LENGTH];
```

```
10.
11. /// Синтаксис виклику: main_app [image_file_path]
12. if (argc != 2) {
13. printf(“Error! Wrong input parametersn”);
14. printf(“Usage: %s [image_file_path]n”, argv[0]);
15. return 0;
16. }
17.
18. /// Крок 1: розпізнавання CAPTCHA
19. memset(CAPTCHA_string, ‘\0’,
    sizeof(CAPTCHA_string));
20. strncpy_s(CAPTCHA_string, sizeof(CAPTCHA_string),
    solve_CAPTCHA(argv[1]), _TRUNCATE);
21.
22. /// Крок 2: результат розпізнавання CAPTCHA
23. printf(“CAPTCHA string = %sn”, CAPTCHA_string);
24.
25. return 0;
26. }
```

Результат і є вихідним кодом C для Windows, оскільки функція string є специфічною для Windows — це безпечна версія стандартної функції C string. Рядок який

викликає функцію solve_CAPTCHA() у DLL-обгортці, яку ми створили раніше, виглядає так:

```
1. strncpy_s(CAPTCHA_string, sizeof(CAPTCHA_string),
   solve_CAPTCHA(argv[1]), _TRUNCATE);
```

Як видно, використання DLL-обгортки передбачає лише один виклик функції в коді, який використовує бібліотеку. Основна перевага даного підходу дозволяє легко зв'язатися з бібліотекою оболонки у проєкті Visual Studio або в іншому типі IDE, який використовується. Даний підхід створює простий і зрозумілий програмний інтерфейс для використання бібліотеки Tesseract для розпізнавання CAPTCHA.

3.5 Тестові приклади роботи

На цьому етапі, налаштування програмного забезпечення розпізнавання CAPTCHA завершено, тому настав час його перевірити. Для тестової перевірки було згенеровано способом реалізованим вище наступні вхідні зображення у кількості 10 шт (табл 3.1).

Таблиця 3.1 - Вхідні зображення для тестування алгоритму.

Приклад зображення	Характеристика
	Зображення зберігається у файлі 0.jpg Навмисно зашумлене та розташовано по центру.
	Зображення зберігається у файлі 1.jpg Навмисно зашумлене та розташовано зі зсувом вправо.
	Зображення зберігається у файлі 2.jpg Навмисно зашумлене та розташовано з суттєвим зсувом вправо та збільшені літери.
	Зображення зберігається у файлі 3.jpg Навмисно зашумлене та цифри зменшуються в розмірі.
	Зображення зберігається у файлі 4.jpg Навмисно зашумлене та цифри збільшуються в розмірі.
	Зображення зберігається у файлі 5.jpg Навмисно зашумлене та цифри однакового розміру.
	Зображення зберігається у файлі 6.jpg Навмисно зашумлене та цифри написані по діагоналі.

Продовження таблиці 3.1

	Зображення зберігається у файлі 7.jpg Суттєво навмисно зашумлене та цифри одного розміру розташовані по центру.
	Зображення зберігається у файлі 8.jpg Суттєво навмисно зашумлене та цифри одного різного розміру шрифта.
	Зображення зберігається у файлі 9.jpg Суттєво навмисно зашумлене.

На таблиці 3.1 показано CAPTCHA, які я використовувала для тестування рішення CAPTCHA, описаного в попередніх розділах.

Як згадувалося в раніше, DLL-обгортка надає бібліотеці Tesseract евристичні дані про те, що вхідні дані складаються з цифр, і їх слід розглядати як англійські за своєю природою, а не інші набори символів, такі як римські цифри або ішні.

Нище наведено результати виклику нашої тестової програми з вищезазначеними вхідними файлами (CAPTCHA) табл. 3.2.

Таблиця 3.2. - Результат роботи бота для обходження захисту CAPTCHA.

Імя файлу	Рідок на зображенні	Результат розпізнавання
0.jpg	159769	159759
1.jpg	816675	816675
2.jpg	671684	671584
3.jpg	321338	321335
4.jpg	670834	5193311
5.jpg	682209	5822179
6.jpg	223143	223143
7.jpg	805928	7805928
8.jpg	970825	970825
9.jpg	686608	686608

В таблиці 3.2 червоним кольором визначено повністю помилкове розпізнавання, жовтим - частково

вірне (допускається помилка в одній цифрі), зеленим - повністю вірна робота алгоритму.

Таким чином, загальна достовірність автоматичного подолання такої CAPTCHA складає лише 40% при тому, що для людини її вирішення лишається інтуїтивно простим, навіть якщо людина має певні обмеження по зору.

ВИСНОВКИ

В ході виконання наукового дослідження за даною темою, нами було встановлення, що сучасні web-ресурси та web-сервіси потерпають від різного роду бот-атак на них, що спричиняє часткову відмову в обслуговуванні, порушення цілісності даних та в окремих випадках навіть вихід з ладу. Тому використання технології CAPTCHA є актуальним і обгрунтованим. З іншого боку наявна ситуація конкуренції між захистом та технологіями взлому мета яких оминання CAPTCHA та таким чином нівелювання наявного захисту що вона створює, частіше за все технологіями графічного розпізнавання зображень OCR такими потужними бібліотеками як Tesseract та OpenCV.

В роботі зазначено, що має існувати баланс між складною CAPTCHA та прийнятним інтерфейсом для користувача з метою запобігання створення надскладних сесійних CAPTCHA які розв'язати людині складно, а інколи навіть неможливо.

В якості компромісного варіанту враховуючи досвід користувачів розглянутих в першому та другому розділах було обрано модель яка включає в себе генерування певного числового рядка у вигляді зашумленого зображення, розрізання цього зображення на певну сітку розміру NxM та

запит до користувача відтворити вхідний числовий рядок. Даний підхід дозволив не створюючи додаткової складності для людини та зменшити можливість автоматичного розпізнавання даної CAPTCHA до рівня в 40% що є досить прийнятним значення з точки зору web-сервісів..

Потрібно також врахувати, що є кілька можливих способів підвищити точність розв'язання CAPTCHA. По-перше, ми можемо виконати попередню обробку, щоб зробити зображення CAPTCHA чіткішим, а по-друге, ми можемо додати ще один «контекст» як евристичний розв’язок розв’язування CAPTCHA, наприклад, дати підказку Tesseract, що введення завжди складається з шести символів, це має враховувати і генератор зображень.

Для контролю параметру автоматичного розпізнавання в заданих межах достатньо оперувати довжиною числового рядка L та значеннями параметрів розбиття N та M які зараз визначені на рівні L = 6, N = 3 та M = 2 значення яких встановлено експериментально.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Baker B.S. On finding duplication and near-duplication in large software systems // Reverse Eng. - Work. Conf. Proc. IEEE, 1995. P. 86–95.
2. Roy C.K., Cordy J.R. An empirical study of function clones in open source software // Proc. - Work. Conf. Reverse Eng. WCRE. 2018. P. 81–90.
3. Mariani L., Micucci D. AuDeNTES // ACM Trans. Comput. Educ. ACM PUB27, New York, NY, USA, 2012. Vol. 12, № 1
4. Noury, Zahra & Rezaei, Mahdi. (2020). Deep-CAPTCHA: a deep learning based CAPTCHA solver for vulnerability assessment. 10.31219/osf.io/km35b.
5. Wang, Ye & Lu, Mi. (2018). An Optimized System to Solve Text-Based Captcha. International Journal of Artificial Intelligence & Applications. 9. 19-36. 10.5121/ijaia.2018.9302.
6. Generative Adversarial Learning Framework for Breaking Text-Based CAPTCHA in the Dark Web2020 IEEE International Conference On Intelligence And Security Informatics (ISI)2020 Ning ZhangMohammadreza EbrahimiWeifeng LiHsinchun Chen

7. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human level performance on imagenet classification. 2015 IEEE International Conference on Computer Vision (ICCV), Dec 2015.
8. Saumya Solanki, Gautam Krishnan, Varshini Sampath, and Jason Polakis. In (cyber)space bots can hear you speak: Breaking audio captchas using ots speech recognition. pages 69–80, 11 2017
9. Bengio, Y., Ducharme, R., Vincent, P., Janvin, C.: A neural probabilistic language model. The Journal of Machine Learning Research 3, 1137–1155 (2003)
10. Simard, P.Y., Steinkraus, D., Platt, J.C.: Best practices for convolutional neural networks applied to visual document analysis. In: 2013 12th International Conference on Document Analysis and Recognition. vol. 2, pp. 958–958. IEEE Computer Society (2003)
11. Gao, H., Wang, W., Qi, J., Wang, X., Liu, X., & Yan, J. (2013, November). The robustness of hollow CAPTCHAs. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (pp. 1075-1086). ACM.

12. Hussain, R., Gao, H., & Shaikh, R. A. (2017). Segmentation of connected characters in text-based CAPTCHAs for intelligent character recognition. Multimedia Tools and Applications
13. Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., & Shet, V. Multitdigit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks.
14. Gao, H., Tang, M., Liu, Y., Zhang, P., & Liu, X. (2017). Research on the Security of Microsoft's Two-Layer Captcha. IEEE Transactions on Information Forensics and Security, 12(7), 1671-1685.
15. Golle, P. (2008, October). Machine learning attacks against the Asirra CAPTCHA. In Proceedings of the 15th ACM conference on Computer and communications security (pp. 535-542). ACM.
16. The robustness of face-based CAPTCHAs. Haichang Gao, Lei Lei, Xin Zhou, Jiawei Li, Xiyang Liu. Institute of Software Engineering, Xidian University.
17. Chow, R., Golle, P., Jakobsson, M., Wang, L., & Wang, X. (2008, February). Making captchas clickable
18. The SoundsRight CAPTCHA: an improved approach to audio human interaction proofs for blind users. Jonathan

Lazar, Jinjuan Heidi Feng, Tim Brooks, Genna Melamed, Jon Holman, Abiodun Olalere, Nnanna Ekedebe, and Brian Wentz

19. Gao, H., Liu, H., Yao, D., Liu, X., & Aickelin, U. (2010, July). An audio CAPTCHA to distinguish humans from computers.
20. G. Huang, Z. Liu, L. V. D Maaten, et al., Densely connected convolutional networks, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, (2017), 2261–2269.

ДОДАТОК А.

Лістінг програм

CAPTCHA_solver_dll_main_app.cpp

```
1. // CAPTCHA_solver_dll_main_app.cpp : Точка входу
   консольного доданку.
2. //
3.
4. #include “stdafx.h”
5. #include “CAPTCHA_solver_dll.h”
6.
7. int _tmain(int argc, _TCHAR* argv[])
8. {
9. char
   CAPTCHA_string[MAX_CAPTCHA_STRING_LENG
   TH];
10.
11. /// Синтаксис виклику: main_app [image_file_path]
12. if (argc != 2) {
13. printf(“Error! Wrong input parametersn”);
14. printf(“Usage: %s [image_file_path]n”, argv[0]);
15. return 0;
16. }
17.
18. /// Крок 1: розпізнавання CAPTCHA
19. memset(CAPTCHA_string, ‘\0’,
   sizeof(CAPTCHA_string));
20. strncpy_s(CAPTCHA_string,
   sizeof(CAPTCHA_string), solve_CAPTCHA(argv[1]),
   _TRUNCATE);
21.
22. /// Крок 2: результат розпізнавання CAPTCHA
23. printf(“CAPTCHA string = %sn”, CAPTCHA_string);
```

```
24.
25. return 0;
26. }
```

CAPTCHA_SOLVER_DLL.h

```
1.  #define __CAPTCHA_SOLVER_DLL_H__
2.
3.  // Наступни блок ifdef це стандартний підхід до
    стскорення макросів яки використовуються для
    спрощення екпорту методів з DLL
4.  // Всі файли даної DLL скомпільовані зі значенням
    константи
5.  // CAPTCHA_SOLVER_DLL_EXPORTS визначеної
    в командному рядку.
6.
7.  #ifdef CAPTCHA_SOLVER_DLL_EXPORTS
8.  #define CAPTCHA_SOLVER_DLL_API
    __declspec(dllexport)
9.  #else
10. #define CAPTCHA_SOLVER_DLL_API
    __declspec(dllimport)
11. #endif
12.
13. #ifndef MAX_CAPTCHA_STRING_LENGTH
14. #define MAX_CAPTCHA_STRING_LENGTH 256
15. #endif
16.
17. #ifdef __cplusplus
18. extern “C” {
19. #endif
20.
21. CAPTCHA_SOLVER_DLL_API char*
    solve_CAPTCHA( const char* image_file_path );
22.
```

```
#ifdef __cplusplus
}
#endif

#endif // __CAPTCHA_SOLVER_DLL_H__
```

Main.cpp

```
#include “stdafx.h”
#include “CAPTCHA_solver_dll.h”

…

// Рядок-результат обробки CAPTCHA
static char g_CAPTCHA_string[MAX_CAPTCHA_STRING_LENGTH + 1];

…

// Головна функція.
///
/// Дана функція викликає бібліотеку передаючи їй зображення image
/// як вхідний image_file_path параметр.
///
/// Шлях до файла.
/// В якості результата отримуємо покажчик на рядок розпізнавання
///
CAPTCHA_SOLVER_DLL_API char* solve_CAPTCHA( const char* image_file_path )
{
//
// Крок 1: Ініціалізація об'єктів бібліотеки
```

```
23. //
24. const char* lang = “eng”;
25. char* config_file_path = “digits”; /* Hardcode the
    config file to be used to
    “$TESSDATA  PREFIX/configs/digits”
26. NOTE: As long as $TESSDATA_PREFIX has been
27. exported to as Windows environment variable,
28. using only the word “digits” here should work. */
29. tesseract::TessBaseAPI api;
30.
31. api.Init(image_file_path /* шлях до файлу*/,
32. lang /* мова */,
33. tesseract::OEM_DEFAULT /* режим OCR */,
34. &config_file_path /* char **configs */,
35. 1 /* configs_size — файл конфігурації config_file_path
    */,
36. NULL /* додаткові налаштування бібліотеки const
    GenericVector *vars_vec */,
37. NULL ,
38. false /* включення режиму відлагодження
    set_only_non_debug_params */);
39.
40. tesseract::PageSegMode pagesegmode =
    tesseract::PSM_AUTO;
41.
42. if (api.GetPageSegMode() ==
    tesseract::PSM_SINGLE_BLOCK)
43. api.SetPageSegMode(pagesegmode);
44.
45. //
46. // Крок 2: Перевірка відповідності файла формату що
    підтримується
47. //
48. FILE* fin = fopen(image_file_path, “rb”);
```

```
49. if (fin == NULL) {
50. return NULL;
51. }
52. fclose(fin);
53.
54. PIX *pixs;
55. if ((pixs = pixRead(image_file_path)) == NULL) {
56. return NULL;
57. }
58. pixDestroy(&pixs);
59.
60. //
61. // Крок 3: Обробка зображення.
62. //Результат - це об'єкт STRING який передається в змінну by text_out.
63. //
64. STRING text_out;
65. if (!api.ProcessPages(image_file_path, NULL, 0, &text_out)) {
66. return NULL;
67. }
68.
69. //
70. // Крок 4: Копіювання результатів в буфер виводу
71. // a. Використовуємо text_out.strdup() для отримання покажчика на об'єкт копіювання результати розпізнавання CAPTCHA.
72. // b. Очищаэмо память що була використана під час копіювання результатів роботи бібліотеки.
73. //
74. memset(g_CAPTCHA_string, '\0', sizeof(g_CAPTCHA_string));
75. char* result = text_out.strdup();
```

```
76. strncpy(g_CAPTCHA_string, result,
    sizeof(g_CAPTCHA_string));
77. free(result);
78.
79. return g_CAPTCHA_string;
80. }
```

Generator.py

```
1.  from split_image import split_image
2.  from random import shuffle
3.
4.  import random
5.  import string
6.
7.  def get_random_string(length):
8.
9.      letters = string.numbers
10.     result_str = ''.join(random.choice(numbers) for i in
    range(length))
11.
12.
13.
14. image = ImageCaptcha(width = 280, height = 90)
15. obj = get_random_string(6)
16. data = image.generate(obj)
17. image.write(obj, 'CAPTCHA.png')
18.
19. image = split_image("CAPTCHA.png", 2, 2, true, false,
    "..")
20. images = [[i] for i in range(image.size())]
21. shuffle(images);
22.
23. img = image.compose(images)
```