Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗНАХОДЖЕННЯ КЛЮЧОВИХ КАДРІВ У ВІДЕОПОТОЦІ (KEYFRAMES DETECTION IN A VIDEO STREAM)

Текстова частина до дипломної роботи

за спеціальністю «Комп'ютерні науки» 122

Керівник курсової роботи

ст.в. Бучко О. А.

(підпис)

Виконав студент 4 курсу

Пожаров Д. С.

«____»_____2024 р.

Київ 2024

Зміст

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри інформатики,

к.ф.-м.н., доц. Гороховський С.С

_____

(підпис)

«___» _____ 2024 р

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на дипломну роботу

студенту 4-го курсу, факультету інформатики Пожарову Дмитру

Тема: Знаходження ключових кадрів у відеопотоці (Keyframes detection in a video stream) .

Вихідні дані:

Зміст ТЧ до дипломної роботи:

Зміст

Вступ

Загальна інформація

Дослідження

Реалізація власного продукту

Висновок

Список використаної літератури

Дата видачі «____»_____ 2024 р. Керівник _____

(підпис)

Завдання отримав:_____

(підпис)

Календарний план виконання курсової роботи

Тема: Знаходження ключових кадрів у відеопотоці (Keyframes detection in a video stream):

№	Назва етапу курсового проєкту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу	19.11.2023	
2.	Аналіз матеріалів за темою дипломної роботи	02.12.2023	
3.	Розробка власного програмного продукту	10.01.2024	
4.	Написання текстової частини дипломної роботи	23.03.2024	
5.	Надання дипломної роботи на перевірку керівником	14.05.2024	
6.	Коригування на основі результатів перевірки	16.05.2024	
7.	Остаточне оформлення роботи та презентації	17.05.2024	
8.	Захист дипломної роботи	31.05.2024	

Пожаров Д. С. _____

Бучко О. А. _____

«____» _____ _____ р.

Анотація

У даній роботі було розроблено продукт для знаходження ключових кадрів, який містить декілька алгоритмів, а саме алгоритм на основі схожості кадрів (SSIM), алгоритм на основі інструменту «FFMpeg», а також алгоритм, який використовує хмарний сервіс «Azure Video Indexer». Розроблене програмне забезпечення є повністю готове для комерційного використання та може бути розповсюджене на ринку. Продукт розроблений для операційної системи Windows. Програмне забезпечення написано мовою програмування C# з використанням платформи .NET та середовища розробки Visual Studio. Це дозволило забезпечити високу продуктивність, масштабованість та легкість у підтримці коду. Програмне забезпечення також містить зручний користувацький інтерфейс, що дозволяє його легко використовувати новим користувачам. Було проведено ретельне тестування для визначення умов використання кожного алгоритму, а також детальне порівняння їх продуктивності та якості результатів.

Вступ

У сучасному світі з кожним днем стрімко зростає кількість відео в мережі. Вони використовуються у безлічі сфер. Ефективний аналіз і структурування великих обсягів відеоконтенту є дуже важливим і складним завданням через об’єм відеоданих. Згідно зі статистикою, кожну хвилину завантажується біля 500 годин відео на платформу «YouTube» [1]. У зв'язку з цим, виявлення ключових кадрів стає вкрай важливим інструментом.

Алгоритм виявлення ключових кадрів може значно спростити процес пошуку і аналізу відео, дозволяючи швидше знаходити потрібний файл або важливі моменти без необхідності перегляду всього відео. Також алгоритм може бути інтегрований у різноманітні системи рекомендацій, що особливо корисно в індустрії розваг, адже користувачу буде простіше зрозуміти вміст матеріалу. Це не тільки економить час глядачів, але й робить перегляд більш зручним, оскільки вони можуть легко перейти до найцікавіших частин.

Таким чином, розвиток і впровадження алгоритмів для виявлення ключових кадрів у відеопотоках відіграє ключову роль у способі класифікації відеоконтенту, відкриваючи нові можливості для його використання в різноманітних галузях і сприяючи більш ефективному підходу до аналітики.

Для реалізації таких алгоритмів існують різні підходи, наприклад, машинне навчання, штучний інтелект, математична подібність кадрів.

Ця дипломна робота спрямована на дослідження та розробку методів знаходження ключових кадрів у відеопотоці. Метою роботи є аналіз існуючих алгоритмів, розробка власного алгоритму та оцінка його ефективності на практиці. Об’єктом розробки дипломної роботи є готовий продукт, в якому користувач може обирати один з трьох різних алгоритмів для виявлення ключових кадрів. Зокрема, в продукт включено один алгоритм, розроблений власноруч, і два сторонніх алгоритми, що дозволяють порівняти ефективність і

час різних підходів. Кінцевий продукт надає користувачам інтуїтивно зрозумілий інтерфейс, який дозволяє легко обрати бажаний алгоритм і швидко отримати результати, що є особливо корисним для комерційного використання.

1. Теоретичні аспекти ключового кадру

1.1 Визначення ключового кадру

Ключовий кадр у відеопотоці — це кадр, який найточніше представляє зміст фрагменту або всього відео. Кадр має бути чітким, містити об'єкти, людей або події, що відбуваються в цій частині, а також бути орієнтиром для розуміння його контексту. Визначення таких кадрів може суттєво спростити процес аналізу відео, оскільки дозволяє зосередитись на ключових моментах без необхідності перегляду всього матеріалу.

Можна виділити такі критерії для визначення ключового кадру:

- **Інформативність:** ключовий кадр має містити достатньо інформації для представлення контексту або важливої події.
- **Якість:** кадр має бути чітким, достатньо світлим, без розмитості або інших візуальних дефектів, що можуть ускладнити розуміння контексту.
- **Унікальність:** кадр повинен вирізнятися від інших кадрів відеопотоку, містити більше корисної інформації.

Слід зазначити, що оцінка ключового кадра є досить неточною і суб’єктивною задачею. Оцінка того, наскільки кадр є ключовим, може суттєво відрізнятися в залежності від відео, контексту, а також людини чи алгоритму, що проводить оцінювання.

1.2 Переваги використання алгоритму

Алгоритм пошуку ключових кадрів може бути корисним у багатьох сферах. Ось декілька ключових застосувань:

- Покращення досвіду користувача:

Алгоритм може використовуватися для автоматичного створення іконки для відео («прев'ю»), яке допомагає користувачеві швидко оцінити вміст контенту і вибрати цікаве для нього відео. Це особливо корисно на великих платформах, де користувачі переглядають велику кількість відео, обираючи те, що їх цікавить. Також алгоритм допомагає знайти потрібні фрагменти у відео, що робить процес пошуку конкретної інформації більш ефективним і зручним.

- Аналіз відео:

  Сучасні методи аналізу відео змушені оброблювати кожен кадр відео в процесі роботи. Опрацювання кожного кадру потребує значних обчислювальних ресурсів, велику кількість часу, що знижує ефективність аналізу. Замість цього можна опрацьовувати тільки ключові кадри. Таким чином буде збережений контекст відео і підвищена ефективність аналізу, або інших алгоритмів, пов'язаних з обробкою відео.

1.3 Приклади практичного застосування

Алгоритми пошуку ключових кадрів відіграють важливу роль у сфері розваг та обробки відео, покращуючи взаємодію користувача з контентом.

Ось декілька прикладів їх застосування у різних галузях:

- YouTube: використовує алгоритм пошуку ключових кадрів для створення іконок відео, які відображають найцікавіші або найважливіші моменти відео. Це допомагає користувачам легко обрати відео, що їх цікавить.[2]
- Adobe Premiere Pro: використовує алгоритм пошуку ключових кадрів для автоматичного створення маркерів кадрів, які можна використовувати для

швидкого переходу до різних частин відео. Це може заощадити час користувачам [3].

- Системи охорони: використовують алгоритм пошуку ключових кадрів для виявлення подій, які можуть бути важливими. Це допомагає службі охорони швидше перевірити всі потрібні моменти у відеозаписі [4].
- Стискання відео: один із підходів до стискання відео полягає у виборі ключового кадра, який потім стає основою для наступних кадрів [5]. Тобто, замість того, щоб зберігати кожен кадр окремо, зберігається лише ключовий кадр і зміни, які відбуваються у наступних кадрах відносно ключового.

2. Аналіз існуючих рішень

2.1 Аналіз існуючих алгоритмів

Аналіз існуючих алгоритмів для визначення ключових кадрів у відео є важливою складовою дослідження, оскільки він дозволяє оцінити різноманіття методів та їх ефективність у різних умовах. Як основні методи пошуку ключових кадрів можна розглядати наступні чотири категорії[6]:

1. На основі фрагментів

   В цьому методі спочатку знаходяться окремі відеофрагменти, і вже з них за допомогою різних методів отримують ключові кадри. Окремий відеофрагмент можна визначити як безперервну зйомку камери, де контекст відео є однаковим. Такий метод використовує штучний інтелект Azure [7].
2. На основі кластерів

   У алгоритмах, заснованих на кластеризації, відеокадри групуються за допомогою різних метрик. Однією з таких метрик є гістограма за HSV-значенням кадрів [8]. Після виконання кластеризації ключові кадри вибираються шляхом вилучення кадрів, найближчих до центру кластера. У роботі «Video Key-Frame Extraction using Unsupervised Clustering and Mutual Comparison» [9] описується та впроваджується метод вилучення ключових кадрів, який використовує підхід кластеризації.
3. На основі контенту

   У цих алгоритмах враховується візуальна інформація кадра. Наприклад, на рішення, чи є кадр ключовим, може вплинути присутність конкретного об’єкта у кадрі. Також у цьому методі може враховуватися кількість інформації, що змінилася між кадрами, наприклад, колір, форми об’єктів тощо.
4. Аналіз руху

У цьому методі ключові кадри визначаються на основі зміни положення предметів між кадрами. Найбільш статичні кадри є чіткими, а отже можуть бути використані як ключові. Наприклад, коли камера сфокусована на тварині, яка стоїть нерухомо. Нерухомі кадри можна знайти, наприклад, за допомогою перевірки послідовних кадрів на схожість (SSIM).

Кожен з цих методів має свої переваги та недоліки та може бути оптимальним в різних випадках. Важливо зазначити, що часто ефективність пошуку ключових кадрів залежить від конкретного відео. Поєднання методів дозволяє досягнути максимальної точності та якості кадрів, що є вважливим для успішного використання у комерційних продуктах.

2.2 Аналіз доступних інструментів

Аналіз доступних інструментів для вилучення ключових кадрів з відео є необхідним для розуміння доступних можливостей. Важливо враховувати як бібліотеки які вже містять в собі готовий алгоритм пошуку ключових кадрів, так і більш загальні, які дозволять розробити власний алгоритм. Ось декілька основних інструментів, які можна використовувати для вилучення ключових кадрів:

- **OpenCV (Open Source Computer Vision Library)**

  Є одною з найбільш відомих і популярних бібліотек комп’ютерного зору. Бібліотека не має вбудованого алгоритму для пошуку ключових кадрів, проте містить велику кількість методів, які допоможуть розробити власний алгоритм.

- **FFmpeg**

  Потужний інструмент командного рядка, призначений для обробки відео та аудіо. FFmpeg та його бібліотеки використовують багато сервісів та

плеєрів, наприклад, VLC [10], Google Chrome, Mozilla Firefox [11]. Також, FFmpeg має вбудований алгоритм для знаходження ключових кадрів.

- **Azure AI Video Indexer**

  Потужний інструмент для аналізу відео, що використовує технології машинного навчання і штучного інтелекту. Дозволяє отримати багато інформації з відео, наприклад, ключові кадри, субтитри, класифіковані сцени, особи, об’єкти.

2.3 Аналіз комерційних продуктів і готових рішень

Комерційних продуктів, які спеціалізуються на вилученні ключових кадрів із відео, доволі мало. Це можна пояснити тим, що потреба у цій функції як в окремому продукті зустрічається вкрай рідко. Зазвичай, алгоритми пошуку ключових кадрів є однією з допоміжних функцій у рамках більших програмних рішень. Наприклад, автоматичний вибір іконки для відео у соціальних мережах. Однак, все ж є декілька комерційних продуктів, які дозволяють здобути кадри користувачеві.

- **Kive AI.**

  Інструмент, призначений для впорядкування графічних елементів і зображень у візуальні бібліотеки та створення тематичних мудбордів (рис 3.3.1). Платформа на основі штучного інтелекту спрощує організацію, пошук і поширення творчого контенту. Має функціонал пошуку ключових кадрів і зберігання їх на окрему дошку.

Рисунок 3.3.1 - Приклад використання сервісу Kive AI.

- **Azure AI Video Indexer**

  Інструмент відеоаналітики, що використовує штучний інтелект для вилучення корисної інформації зі збережених відео (рис 3.3.2). Фрагментує відео по локаціям, кожну локацію розділяє на сцени, а на кожну сцени генерує декілька ключових кадрів.

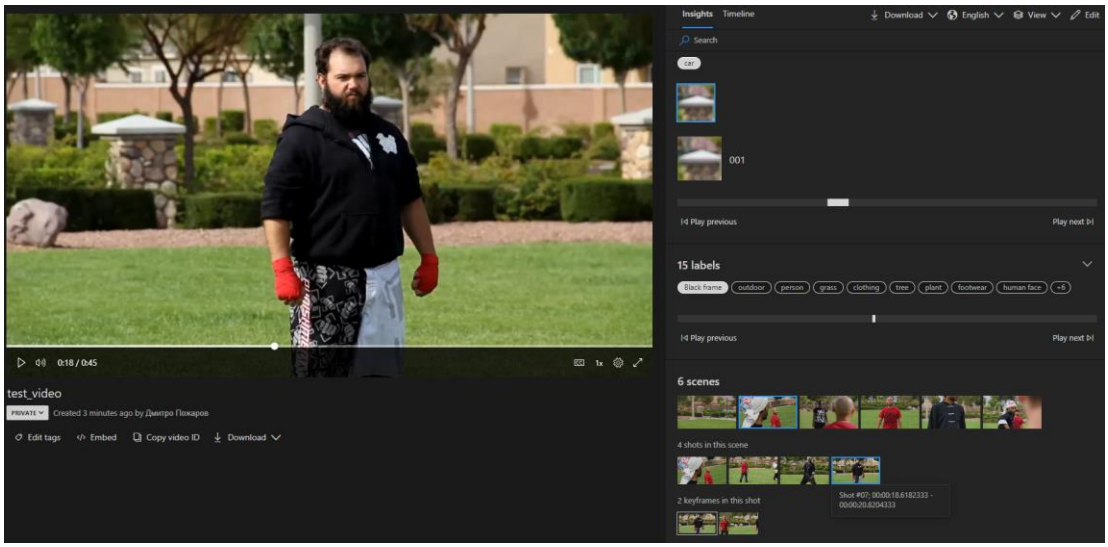


Рисунок 3.3.2 - Приклад використання сервісу Azure AI Video Indexer.

Обидва ці продукти дуже якісно вибирають ключові кадри, однак кількість таких кадрів виявляється обмеженою. Алгоритми цих продуктів не здатні генерувати велику кількість ключових кадрів за потребою. Крім того, ці сервіси не призначені безпосередньо для пошуку ключових кадрів, тому процес їх отримання може бути недостатньо інтуїтивно зрозумілим для користувача.

Не вдалося знайти продукт, який би спеціалізувався конкретно на пошуці ключових кадрів. Здебільшого такі алгоритми інтегровані як частина більш

широких систем. Розробка власного продукту, який би спеціалізувався на вилученні ключових кадрів, може заповнити наявну порожнечу на ринку та надати користувачам більш спеціалізоване рішення.

3. Реалізація власного продукту

3.1 Опис майбутнього продукту

Опис майбутнього продукту: Програмне забезпечення, що дозволяє користувачу обрати власний відеофайл з файлової системи. Після того, як користувач вибере відеофайл, йому автоматично будуть показані згенерована іконка, назва, шлях до файлу та тривалість відео для спрощення роботи користувача з програмним забезпеченням. Після цього користувач може обрати один з трьох алгоритмів знаходження ключових кадрів у відеопотоці і натиснути кнопку, яка почне процес обробки відео за допомогою обраного алгоритма. Після завершення роботи алгоритма, користувачу буде показане повідомлення про успішне виконання роботи і буде відкрита тека з видобутими ключовими кадрами. Користувач може повернутися на головну сторінку для обробки іншого відео, або завершити роботу з програмою.

3.2 Вибір засобів для розробки

Як основний інструмент розробки продукту був вибраний WPF .NET (Windows Presentation Foundation). WPF пропонує широкі можливості для створення настільних застосунків для операційної системи Windows. Також, WPF підтримує можливості для інтеграції з різноманітними бібліотеками .NET, що буде корисним в контексті проєкту.

Для розробки продукту на базі WPF було вирішено використовувати мову програмання C# та Visual Studio як основний редактор коду та інтегроване середовище розробки. VS відомий своїми потужними можливостями для розробки додатків на .NET та підтримкою WPF.

Для системи контролю версій було обрано Git разом із платформою GitHub. Цей вибір є стандартом у сучасній розробці програмного забезпечення, він

дозволяє ефективно керувати версіями коду та зберігати весь код у хмарному сховищі.

3.3 Архітектура проєкту

Для розробки на базі WPF був обраний патерн MVVM (Model-View-ViewModel). MVVM є стандартним вибором для проєктів на WPF, оскільки він орієнтований на роботу з WPF.

Оскільки проєкт передбачає вибір серед трьох різних алгоритмів, патерн «Стратегія» ідеально підходить для цієї задачі. Кожен алгоритм написаний у своєму класі з загальним інтерфейсом, що дозволяє легко перемикатися між алгоритмами.

Така архітектура не тільки сприяє чистоті коду та легкості його тестування, але й відкриває можливості для масштабування проєкту в майбутньому.

Як видно на рисунку 4.3.1 проєкт містить наступні теки:

- **Assets** — містить всі необхідні ресурси, які використовуються у проєкті, такі як зображення та іконки.
- **Config —** містить конфігураційні файли.
- **MVVM —** містить всі компоненти, які входять в патерн Model-View-ViewModel:
    - **Views:**
        - **MainWindow** — основне вікно програми, яке служить як контейнер для інших відображень (Views)
        - **UploadView** — view, який призначений для завантаження відеофайлів користувачем (рис 4.3.2). Містить кнопку для вибору файл з файлової системи.
        - **VideoControlView** — view, який призначений для вибору алгоритму і теки для зберігання ключових кадрів (рис 4.3.3).

Містить кнопку повернення до UploadView, інформацію про відео, кнопки вибору алгоритму та кнопку запуску програми.

- **ViewModel:**
  - **UploadViewModel —** ViewModel для UploadView. Містить логіку вибору відеофайлу.
  - **VideoControlModel —** ViewModel для VideoControlModel. Містить обробку інформації про відео, а також вибір і запуск алгоритму.

- **Services —** містить служби, які використовуються в процесі роботи програми. А саме сервіс навігації і конвертор типів даних.
- **Strategies —** містить реалізації різних алгоритмів вилучення ключових кадрів і інтерфейс необхідний для патерну стратегія.

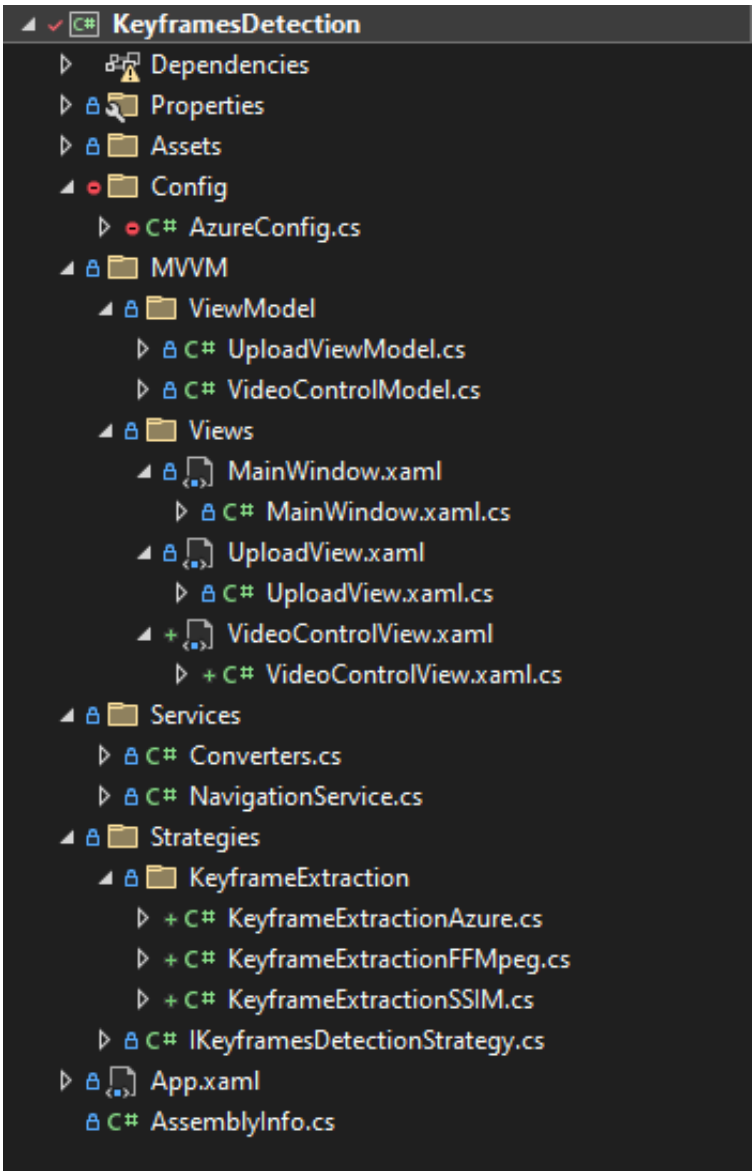


Рисунок 4.3.1 – Архітектура проєкту

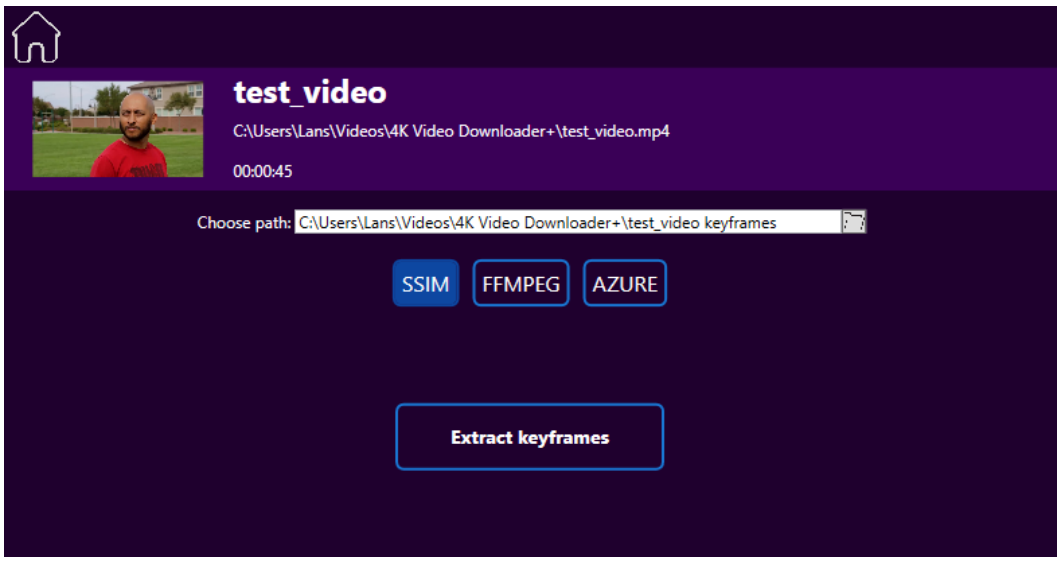


Рисунок 4.3.2 – Відображення «UploadView»

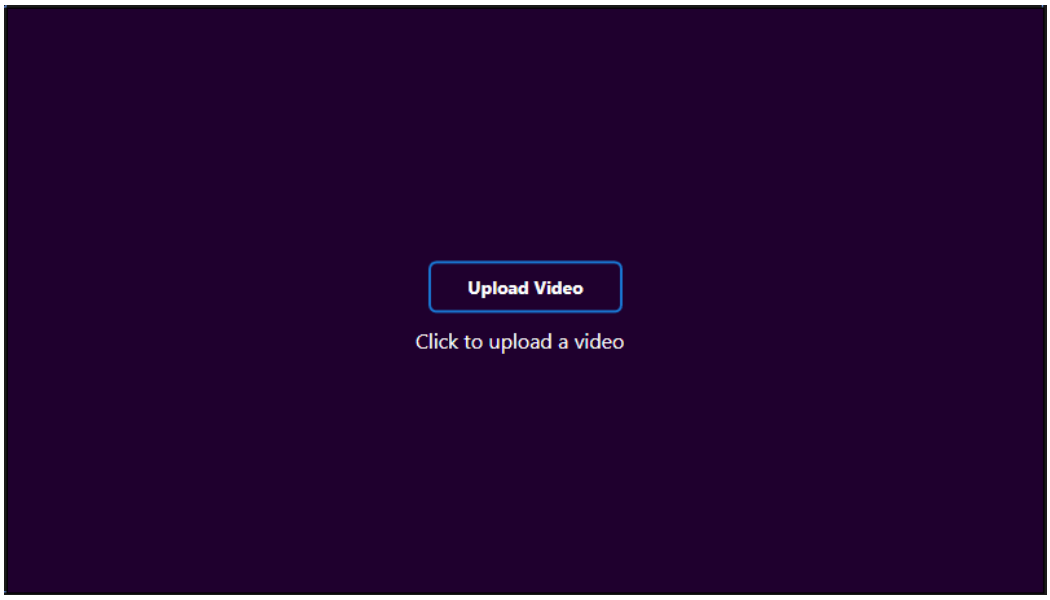


Рисунок 4.3.3 – Відображення «VideoControlView»

3.4 Реалізація NavigationService

NavigationService (рис 4.4.1) є головним компонентом для навігації між різними відображеннями (views) в архітектурі MVVM. Цей сервіс полегшує зміну відображень і сприяє легкому розширенню програми в майбутньому.

NavigationService використовує шаблон «Singleton», забезпечуючи унікальність екземпляру класу у всій програмі. Це дозволяє легко дістатися до сервісу з будь-якого місця у програмі без необхідності постійного створення екземплярів класу.

Використання інтерфейсу «INotifyPropertyChanged» дозволяє повідомляти про зміни поточної ViewModel, що сприяє автоматичному оновленню відображень.

Клас має властивість CurrentViewModel, що зберігає поточну ViewModel. Зміна цієї властивості автоматично оновлює відображення, завдяки механізму зв'язування даних.

NavigationService містить два методи:

- «NavigateToUpload()» — метод, який переводить користувача до вікна завантаження файлів (Рис. 4.3.2), змінюючи властивість «CurrentViewModel» на новий екземпляр класу «UploadViewModel».
- «NavigateToVideoControl(string videoPath)» — метод для навігації до вікна вибору алгоритму пошуку ключових кадрів (Рис. 4.3.3), передаючи шлях до відеофайлу як параметр. Змінює властивість «CurrentViewModel» на новий екземпляр класу.

ViewModel може викликати навігаційні методи без необхідності знання про конкретні відображення, на які вони впливають.

```
public class NavigationService : INotifyPropertyChanged
{
    private static NavigationService _instance;

    2 references
    public static NavigationService Instance
    {
        get
        {
            if (_instance == null)
            {
                _instance = new NavigationService();
            }
            return _instance;
        }
    }

    1 reference
    public NavigationService()
    {
        _instance = this;
        CurrentViewModel = new UploadViewModel();
    }
    private object _currentViewModel;

    public event PropertyChangedEventHandler? PropertyChanged;

    4 references
    public object CurrentViewModel
    {
        get { return _currentViewModel; }
        set { _currentViewModel = value; OnPropertyChanged(nameof(CurrentViewModel));}
    }

    1 reference
    public void NavigateToUpload() => CurrentViewModel = new UploadViewModel();

    1 reference
    public void NavigateToVideoControl(string videoPath) => CurrentViewModel = new VideoControlModel(videoPath);

    1 reference
    protected virtual void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

Рисунок 4.4.1 – Клас «NavigationService»

3.5 Реалізація патерну «MVVM»

Патерн «MVVM» (Model-View-ViewModel) — це один з ключових патернів проєкта, який дозволяє ефективно відокремити розробку графічного інтерфейсу від розробки логіки. Патерн ділиться на 3 частини:

- Model відповідає за логіку роботи з даними та управління даними, необхідних для роботи програми. Це може включати роботу з базами даних, мережеві запити, обробку даних тощо.
- View — це графічний інтерфейс користувача. Він відображає дані з моделі і обробляє взаємодію з користувачем. View відповідає виключно за візуалізацію даних та надсилання команд від користувача до ViewModel.
- ViewModel обробляє всю логіку інтерфейсу, приймає вхідні дані від користувача через View та перетворює їх для Model, а також обробляє дані з Model для демонстрації у View. ViewModel не знає про конкретні елементи View, тому дизайн може бути легко змінений без редагування ViewModel.

В проєкті використовується три view:

- **MainWindow** (рис 4.5.1) є головним вікном програми, що є основним контейнером, до якого встановлюються інші частини користувацького інтерфейсу в залежності від поточного стану програми. Встановлення вмісту контейнера відбувається за допомогою зв'язування даних з властивістю NavigationService «CurrentViewModel».
- **UploadView** (рис 4.5.2-4.5.3) є компонентом користувацького інтерфейсу (UserControl). Це відображення відповідає за візуальне представлення функціоналу завантаження відео. Воно дозволяє користувачам завантажити відеофайл за допомогою кнопки для подальшого аналізу на ключові кадри. UploadView використовує зв’язування даних для зв'язування команд з ViewModel до UploadViewModel.

- **VideoControl** (рис 4.5.4-4.5.9) є компонентом користувацького інтерфейсу (UserControl). Цей view призначений для вибору і запуску алгоритму вилучення ключових кадрів.

  Елементи відображення «VideoControl» :

  - Кнопка повернення до «UploadView»,
  - Інформація про обране відео (іконка, назва, повний шлях, тривалість відео)
  - Вибір теки, до якої будуть збережені ключові кадри
  - Кнопки для вибору алгоритму (SSIM, FFMpeg і Azure)
  - Кнопка запуску обраного алгоритма вилучення ключових кадрів
  - Прогрес-бар, який відображає прогрес алгоритма.

```
<Window x:Class="KeyframesDetection.MVVM.Views.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:KeyframesDetection.Services"
        mc:Ignorable="d"
        Title="Keyframe detection" Height="450" Width="800" ResizeMode="NoResize"
        Icon="/Assets/keyframeIcon.png">

    <Window.DataContext>
        <local:NavigationService />
    </Window.DataContext>

    <Frame x:Name="Frame1" NavigationUIVisibility="Hidden" Content="{Binding CurrentView}" Grid.ColumnSpan="3" />
</Window>
```

Рисунок 4.5.1 – Відображення «MainWindow»

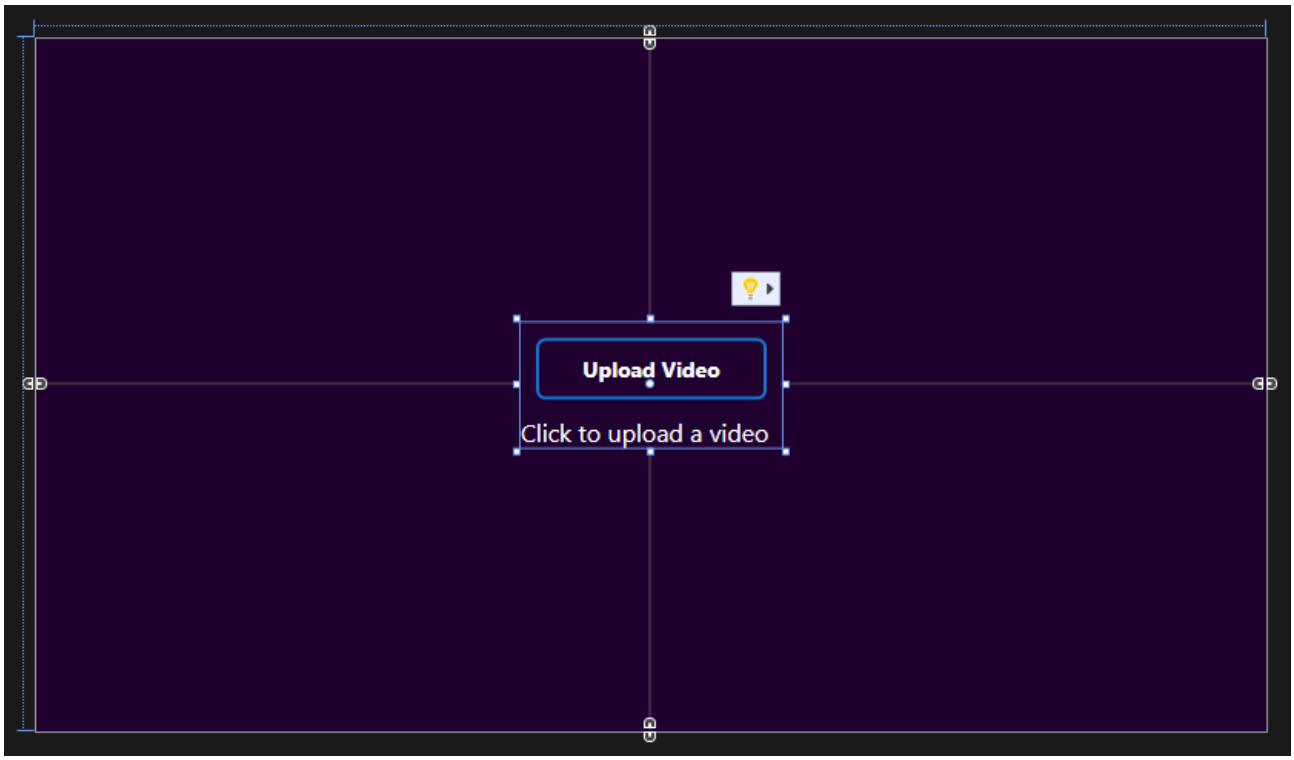


Рисунок 4.5.2 – Відображення «UploadView»

```
<UserControl x:Class="KeyframesDetection.MVVM.Views.UploadView"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:viewmodel="clr-namespace:KeyframesDetection.MVVM.ViewModel" d:DataContext="{d:DesignInstance Type=viewmodel:UploadViewModel}"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">

    <UserControl.Resources>
        <Style x:Key="ButtonStyle" TargetType="Button">
            <Setter Property="Width" Value="150"/>
            <Setter Property="Height" Value="40"/>
            <Setter Property="Background" Value="#2196F3"/>
            <Setter Property="Foreground" Value="White"/>
            <Setter Property="BorderThickness" Value="0"/>
            <Setter Property="FontSize" Value="14"/>
            <Setter Property="FontWeight" Value="Bold"/>
            <Setter Property="Margin" Value="10"/>
            <Setter Property="Cursor" Value="Hand"/>
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="Button">
                        <Border Name="border"
                                BorderThickness="2"
                                CornerRadius="5"
                                BorderBrush="#1976D2">
                            <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
                        </Border>
                        <ControlTemplate.Triggers>
                            <Trigger Property="IsMouseOver" Value="True">
                                <Setter TargetName="border" Property="Background" Value="#1976D2"/>
                            </Trigger>
                            <Trigger Property="IsPressed" Value="True">
                                <Setter TargetName="border" Property="Background" Value="#0D47A1"/>
                            </Trigger>
                        </ControlTemplate.Triggers>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
    </UserControl.Resources>

    <Grid Background="#1f002e">
        <StackPanel VerticalAlignment="Center" HorizontalAlignment="Center">
            <Button Content="Upload Video"
                    Command="{Binding UploadCommand}"
                    Style="{StaticResource ButtonStyle}"/>
            <TextBlock Text="Click to upload a video" FontSize="16" Foreground="White"/>
        </StackPanel>
    </Grid>
</UserControl>
```

Рисунок 4.5.3 – XAML код відображення «UploadView»

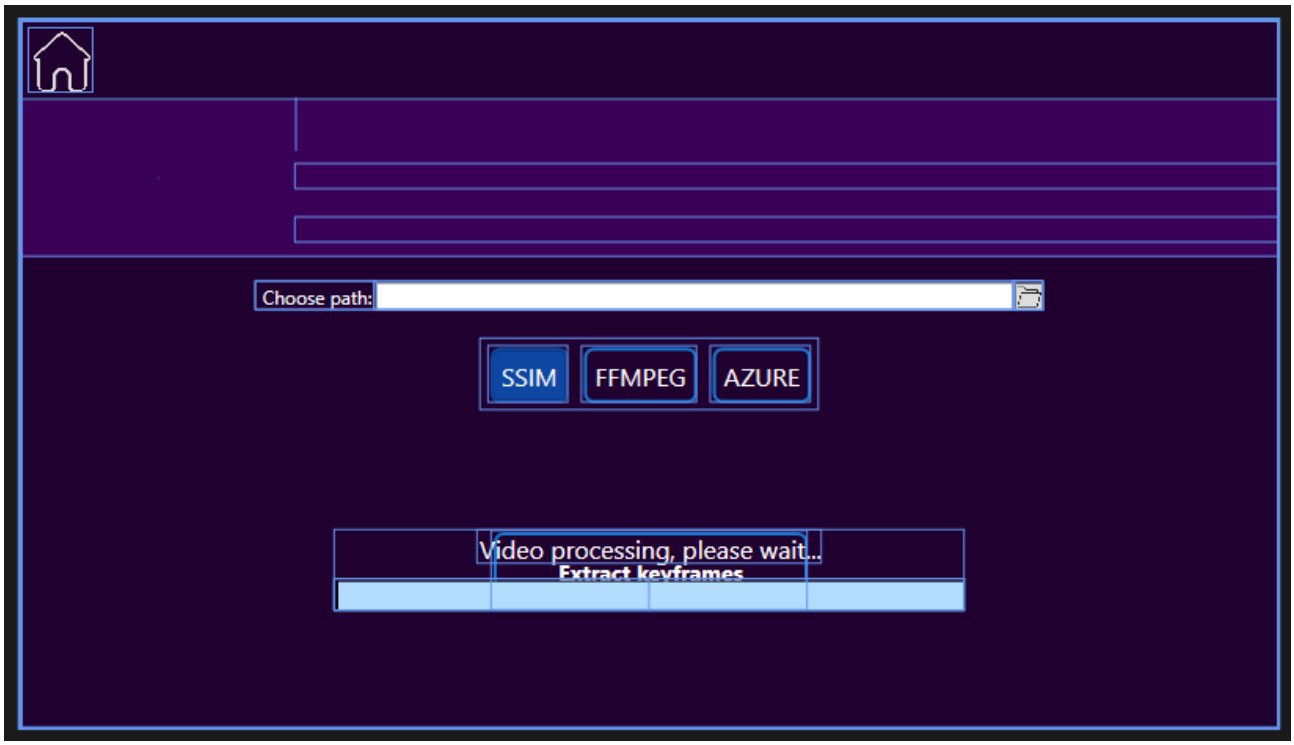


Рисунок 4.5.4 – Відображення «VideoControlView»

```
<UserControl x:Class="KeyframesDetection.MVVM.Views.VideoControl"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:viewmodel="clr-namespace:KeyframesDetection.MVVM.ViewModel"
             xmlns:local="clr-namespace:KeyframesDetection.Services"
             d:DataContext="{d:DesignInstance Type=viewmodel:VideoControlModel}"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
    <UserControl.Resources>
        <local:BooleanToVisibilityConverter x:Key="BooleanToVisibilityConverter"/>
        <local:InverseBooleanToVisibilityConverter x:Key="InverseBooleanToVisibilityConverter"/>

        <Style x:Key="RadioButtonStyle" TargetType="RadioButton">
            <Setter Property="Margin" Value="5"/>
            <Setter Property="Foreground" Value="White"/>
            <Setter Property="FontSize" Value="16"/>
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="RadioButton">
                        <Border x:Name="border"
                            Background="Transparent"
                            BorderThickness="2"
                            CornerRadius="5"
                            BorderBrush="#1976D2">
                            <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center" Margin="5"/>
                        </Border>
                        <ControlTemplate.Triggers>
                            <Trigger Property="IsMouseOver" Value="True">
                                <Setter TargetName="border" Property="Background" Value="#1976D2"/>
                            </Trigger>
                            <Trigger Property="IsChecked" Value="True">
                                <Setter TargetName="border" Property="Background" Value="#0D47A1"/>
                                <Setter TargetName="border" Property="BorderBrush" Value="#0D47A1"/>
                            </Trigger>
                        </ControlTemplate.Triggers>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
```

Рисунок 4.5.5 – XAML код відображення «VideoControlView». Частина 1

```
    <Style x:Key="ButtonStyle" TargetType="Button">
        <Setter Property="Width" Value="150"/>
        <Setter Property="Height" Value="40"/>
        <Setter Property="Background" Value="#2196F3"/>
        <Setter Property="Foreground" Value="White"/>
        <Setter Property="BorderThickness" Value="0"/>
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="FontWeight" Value="Bold"/>
        <Setter Property="Margin" Value="10"/>
        <Setter Property="Cursor" Value="Hand"/>
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="Button">
                    <Border Name="border"
                    BorderThickness="2"
                    CornerRadius="5"
                    BorderBrush="#1976D2">
                        <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
                    </Border>
                    <ControlTemplate.Triggers>
                        <Trigger Property="IsMouseOver" Value="True">
                            <Setter TargetName="border" Property="Background" Value="#1976D2"/>
                        </Trigger>
                        <Trigger Property="IsPressed" Value="True">
                            <Setter TargetName="border" Property="Background" Value="#0D47A1"/>
                        </Trigger>
                    </ControlTemplate.Triggers>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
</UserControl.Resources>
```

Рисунок 4.5.6 – XAML код відображення «VideoControlView». Частина 2

```
<Grid Background="#1f002e">

    <Grid.RowDefinitions>
        <RowDefinition Height="50*"/>
        <RowDefinition Height="100*"/>
        <RowDefinition Height="50*"/>
        <RowDefinition Height="50*"/>
        <RowDefinition Height="200*"/>
    </Grid.RowDefinitions>

    <Button HorizontalAlignment="Left" VerticalAlignment="Center" Width="40" Height="40"
            Margin="5,0,0,0" BorderBrush="#00707070" Foreground="#00707070" Command="{Binding HomeCommand}" >
        <Button.OpacityMask>
            <ImageBrush ImageSource="/Assets/home.png"/>
        </Button.OpacityMask>
        <Button.Background>
            <ImageBrush ImageSource="/Assets/home.png" Stretch="Fill" TileMode="None"/>
        </Button.Background>
    </Button>

    <Grid Grid.Row="1" Background="#3a0057">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="7*"/>
            <ColumnDefinition Width="25*"/>
        </Grid.ColumnDefinitions>

        <Image x:Name="VideoIcon" Grid.RowSpan="3" Margin="10,10,10,10" Source="{Binding VideoIcon}"
               HorizontalAlignment="Center" VerticalAlignment="Center"/>
        <TextBlock x:Name="VideoName" Foreground="White" Grid.Column="1" Grid.Row="0" Text="{Binding VideoName}"
                   HorizontalAlignment="Left" VerticalAlignment="Center" FontWeight="Bold" FontSize="24" />
        <TextBlock x:Name="VideoPath" Foreground="White" Grid.Column="1" Grid.Row="1" Text="{Binding VideoPath}"
                   VerticalAlignment="Center" />
        <TextBlock x:Name="VideoTime" Foreground="White" Grid.Column="1" Grid.Row="2" Text="{Binding VideoTime}"
                   VerticalAlignment="Center" />
    </Grid>
```

Рисунок 4.5.7 – XAML код відображення «VideoControlView». Частина 3

```
<StackPanel Grid.Row="2" HorizontalAlignment="Center" VerticalAlignment="Center" Orientation="Horizontal">
    <TextBox Grid.Row="2" Text="Choose path:" HorizontalAlignment="Center" VerticalAlignment="Center"
             Background="#00707070" BorderBrush="#00707070" Foreground="White" />
    <TextBox Grid.Row="2" Text="{Binding PathToSave}"
             HorizontalAlignment="Center" VerticalAlignment="Center" Width="407"/>
    <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center" Background="#FFDDDDDD" >
        <Button Width="17.96" Height="17.96"  Command="{Binding ChooseFolderCommand}">
            <Button.Background>
                <ImageBrush ImageSource="/Assets/folder.png"/>
            </Button.Background>
        </Button>
    </StackPanel>
</StackPanel>

<Grid Grid.Row="3" HorizontalAlignment="Center" VerticalAlignment="Center">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="Auto"/>
    </Grid.ColumnDefinitions>

    <RadioButton Grid.Column="0" Content="SSIM" Style="{StaticResource RadioButtonStyle}"
                 Command="{Binding SSIMRadioButtonCommand}" IsChecked="True"/>
    <RadioButton Grid.Column="1" Content="FFMPEG" Style="{StaticResource RadioButtonStyle}"
                 Command="{Binding FFMPEGRadioButtonCommand}"/>
    <RadioButton Grid.Column="2" Content="AZURE" Style="{StaticResource RadioButtonStyle}"
                 Command="{Binding AZURERadioButtonCommand}"/>
</Grid>
<Button HorizontalAlignment="Center" VerticalAlignment="Center"
        Visibility="{Binding IsLoading, Converter={StaticResource InverseBooleanToVisibilityConverter}}"
        Width="200" Height="50" Grid.Row="5" Content="Extract keyframes"
        Style="{StaticResource ButtonStyle}" Command="{Binding GenerateKeyframesCommand}"/>
```

Рисунок 4.5.8 – XAML код відображення «VideoControlView». Частина 4

```
    <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center" Grid.Row="5"
                Visibility="{Binding IsLoading, Converter={StaticResource BooleanToVisibilityConverter}}">
        <TextBlock Text="Video processing, please wait..." HorizontalAlignment="Center"
                   VerticalAlignment="Center" FontSize="16" Foreground="White" Margin="0,0,0,10"  />
        <Grid>
            <ProgressBar x:Name="progressBar" Background="#FFB3DDFF"
                         Value="{Binding TaskProgress}" Minimum="0" Maximum="100"
                HorizontalAlignment="Center" VerticalAlignment="Center" Width="400" Height="20"
                         BorderBrush="Black" Foreground="#FF2196F3"/>
            <TextBlock x:Name="percentageText" Text="{Binding TaskProgress, StringFormat={}{0}%}"
                       HorizontalAlignment="Center" VerticalAlignment="Center" FontSize="14"/>
        </Grid>
    </StackPanel>

</Grid>
</UserControl>
```

Рисунок 4.5.9 – XAML код відображення «VideoControlView». Частина 5

У проєкті також існує дві ViewModel:

- UploadViewModel (рис 4.5.10) відповідає за управління процесом завантаження відеофайлів. Ця ViewModel містить логіку виводу діалогового вікна для вибору файлів та подальшого переходу до відображення, де користувач може керувати завантаженим відео. ViewModel містить команду, яка прив'язана до кнопки «Upload» у «UploadView» (рис 4.5.2). Ця команда активує метод UploadVideo, що ініціює «OpenFileDialog», дозволяючи користувачу вибрати відеофайл для завантаження. Після вибору файлу використовується «NavigationService»

для переходу до наступного відображення з шляхом вибраного відеофайлу. UploadViewModel реалізує інтерфейс «INotifyPropertyChanged», що дозволяє повідомляти про зміни поточної ViewModel, що сприяє автоматичному оновленню відображень. Це важливо для випадків, коли стан відповідних властивостей змінюється, і виникає необхідність оповістити View про ці зміни.

- VideoControlModel (рис 4.5.11 – 4.5.19) відповідає за логіку, пов'язану з вибором методу аналізу відео, визначенням місця збереження результатів та запуском процесу пошуку ключових кадрів.

  VideoControlModel має параметризований конструктор (рис 4.5.12), який приймає шлях до відеофайлу («videoPath»). Конструкор виконує наступні кроки:

  - Ініціалізація команд (HomeCommand, ChooseFolderCommand, GenerateKeyframesCommand, SSIMRadioButtonCommand, FFMPEGRadioButtonCommand, AZURERadioButtonCommand), які відповідають за взаємодію з відображенням «VideoControlView»
  - Отримання об’єкта класу «NavigationService» для управління навігацією всередині програми
  - Зберігання шляху до відеофайлу («videoPath») в приватну змінну для подальшого використання
  - Виклик метода «LoadVideoInfo»

  Метод «LoadVideoInfo» (рис 4.5.15) відповідає за завантаження та обробку інформації про відео, яке було обрано користувачем для аналізу. Метод зберігає в приватні змінні назву, іконку і тривалість відео.

  VideoControlModel містить такі команди:

  - **HomeCommand** відповідає за повернення до UploadViewModel, що дає змогу користувачу почати процес з початку, наприклад, щоб обрати інше відео. Викликається про натисканні кнопки «Home» та викликає метод «GoHome».

    Метод «GoHome» (рис 4.5.16) виводить діалогове вікно з питанням

про підтвердження наміру повернення на головний екран, щоб упевнитись, що користувач натиснув кнопку невипадково. У разі підтвердження дії, викликається метод «NavigateToUpload» у «NavigationService», який відповідає за відкриття відображення вибору відеофайлу.

- **ChooseFolderCommand** викликається при натисканні на кнопку вибору директорії. Викликає метод «ChooseFolder». Метод «ChooseFolder» (рис 4.5.17) відповідає за вибір користувачем директорії для збереження ключових кадрів. Він виводить вікно вибору теки «FolderBrowserDialog». Коли користувач підтверджує вибір, метод отримує шлях до обраної теки, формує шлях з новою папкою для збереження ключових кадрів і зберігає її у властивість PathToSave.
- Команди для вибору алгоритму обробки (SSIMRadioButtonCommand, FFMPEGRadioButtonCommand та AZURERadioButtonCommand) викликаються при натисканні відповідних кнопок для вибору алгоритму. Викликають відповідні методи (рис. 4.5.19), які зберігають вибраний алгоритм до змінної.
- **GenerateKeyframesCommand** викликається при натисканні кнопки «Extract keyframes». Викликає асинхронний метод «GenerateKeyframesAsync».

  Метод «GenerateKeyframesAsync» (рис 4.5.18) відповідає за генерацію ключових кадрів за обраним алгоритмом. Метод перевіряє значення властивості «AlgoritmToUse», в якій збережено обраний алгоритм. Після цього метод ініціює відповідний підклас інтерфейсу «IKeyframesDetectionStrategy» і асинхронно запускає алгоритм пошуку кадрів. Після ініціалізації потрібної стратегії, метод встановлює властивість «IsLoading» на true, що використовується для індикації на відображені. Також метод підписується на подію «ProgressChanged» із стратегії для оновлення прогресу в

користувацькому інтерфейсі. Після завершення роботи алгоритму метод встановлює властивість «IsLoading» на false і відкриває теку з ключовими кадрами.

```
public class UploadViewModel : INotifyPropertyChanged
{
    1 reference
    public ICommand UploadCommand { get; }
    private NavigationService navigationService;

    2 references
    public UploadViewModel()
    {
        UploadCommand = new RelayCommand(UploadVideo);
        navigationService = NavigationService.Instance;
    }

    1 reference
    private void UploadVideo()
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();
        openFileDialog.Filter = "Video files (*.mp4)|*.mp4;|All files (*.*)|*.*";
        openFileDialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.MyVideos);

        if (openFileDialog.ShowDialog() == true)
        {
            string videoFilePath = openFileDialog.FileName;
            navigationService.NavigateToVideoControl(videoFilePath);

        }
    }

    public event PropertyChangedEventHandler? PropertyChanged;


}
```

Рисунок 4.5.10 – Клас «UploadViewModel»

```
public class VideoControlModel : INotifyPropertyChanged
{
    1 reference
    public ICommand HomeCommand { get; }
    1 reference
    public ICommand ChooseFolderCommand { get; }
    1 reference
    public ICommand GenerateKeyframesCommand { get; }

    1 reference
    public ICommand SSIMRadioButtonCommand { get; }
    1 reference
    public ICommand FFMPEGRadioButtonCommand { get; }
    1 reference
    public ICommand AZURERadioButtonCommand { get; }

    private BitmapImage _videoIcon;
    private string _videoName;
    private string _videoPath;
    private string _videoTime;

    private string _algoritm = "SSIM";

    private int _progress = 0;

    private bool _isLoading = false;

    private string _pathToSave;

    private NavigationService navigationService;
```

Рисунок 4.5.11 – Клас «VideoControlModel». Частина 1

```
public VideoControlModel(string videoPath)
{
    HomeCommand = new RelayCommand(GoHome);
    ChooseFolderCommand = new RelayCommand(ChooseFolder);
    GenerateKeyframesCommand = new RelayCommand(async () => await GenerateKeyframesAsync());

    SSIMRadioButtonCommand = new RelayCommand(SSIMRadioButton);
    FFMPEGRadioButtonCommand = new RelayCommand(FFMPEGRadioButton);
    AZURERadioButtonCommand = new RelayCommand(AZURERadioButton);

    navigationService = NavigationService.Instance;

    VideoPath = videoPath;
    LoadVideoInfo();
    PathToSave = $"{Path.GetDirectoryName(VideoPath)}\\{VideoName} keyframes";
}
```

Рисунок 4.5.12 – Клас «VideoControlModel». Частина 2. Конструктор

```
public BitmapImage VideoIcon
{
    get { return _videoIcon; }
    set
    {
        _videoIcon = value;
        OnPropertyChanged();
    }
}

4 references
public string VideoName
{
    get { return _videoName; }
    set
    {
        _videoName = value;
        OnPropertyChanged();
    }
}

7 references
public string VideoPath
{
    get { return _videoPath; }
    set
    {
        _videoPath = value;
        OnPropertyChanged();
    }
}

1 reference
public string VideoTime
{
    get { return _videoTime; }
    set
    {
        _videoTime = value;
        OnPropertyChanged();
    }
}
```

Рисунок 4.5.13 – Клас «VideoControlModel». Частина 3. Data binding

```
public string PathToSave
{
    get { return _pathToSave; }
    set
    {
        _pathToSave = value;
        OnPropertyChanged();
    }
}
4 references
public string AlgoritmToUse
{
    get { return _algoritm; }
    set
    {
        _algoritm = value;
        OnPropertyChanged();
    }
}
2 references
public int TaskProgress
{
    get { return _progress; }
    set
    {
        Console.WriteLine($"Progress: {value}");
        _progress = value;
        OnPropertyChanged();
    }
}
2 references
public bool IsLoading
{
    get { return _isLoading; }
    set
    {
        _isLoading = value;
        OnPropertyChanged();
    }
}
```

Рисунок 4.5.14 – Клас «VideoControlModel». Частина 4. Data binding

```
private void LoadVideoInfo()
{
    VideoName = Path.GetFileNameWithoutExtension(VideoPath);

    Console.WriteLine($"Name = {VideoName}. Path = {VideoPath}");

    VideoCapture capture = new VideoCapture(VideoPath);
    int frameCount = (int)capture.Get(VideoCaptureProperties.FrameCount);
    double fps = capture.Get(VideoCaptureProperties.Fps);

    var ffMpeg = new NReco.VideoConverter.FFMpegConverter();

    Stream outputJpegStream = new MemoryStream();

    ffMpeg.GetVideoThumbnail(VideoPath, outputJpegStream, 5);

    outputJpegStream.Seek(0, SeekOrigin.Begin);

    BitmapImage bitmapImage = new BitmapImage();
    bitmapImage.BeginInit();
    bitmapImage.CacheOption = BitmapCacheOption.None;
    bitmapImage.StreamSource = outputJpegStream;
    bitmapImage.EndInit();
    bitmapImage.Freeze();

    VideoIcon = bitmapImage;

    TimeSpan duration = TimeSpan.FromSeconds(frameCount / fps);
    VideoTime = duration.ToString(@"hh\:mm\:ss");
}
```

Рисунок 4.5.15 – Метод «LoadVideoInfo» класу «VideoControlModel»

```
private void GoHome()
{
    MessageBoxResult result =
        System.Windows.MessageBox.Show
        (
            "Are you sure you want to return to the main menu?",
            "Confirmation",
            MessageBoxButton.YesNo,
            MessageBoxImage.Question
        );

    if (result == MessageBoxResult.Yes)
    {
        navigationService.NavigateToUpload();
    }
}
```

Рисунок 4.5.16 – Метод «LoadVideoInfo» класу «GoHome»

```
private void ChooseFolder()
{
    var folderBrowserDialog = new FolderBrowserDialog();
    DialogResult result = folderBrowserDialog.ShowDialog();

    if (result == DialogResult.OK)
    {
        string folderPath = folderBrowserDialog.SelectedPath;
        string newFolderName = $"{VideoName} keyframes";
        if (!folderPath.Contains(newFolderName))
        {
            folderPath = $"{folderPath}\\{newFolderName}";
        }

        PathToSave = folderPath;
    }
}
```

Рисунок 4.5.17 – Метод «ChooseFolder» класу «VideoControlModel»

```
private async Task GenerateKeyframesAsync()
{
    IKeyframesDetectionStrategy strategy;
    switch (AlgoritmToUse)
    {
        case "SSIM":
            strategy = new KeyframeExtractionSSIM();
            break;
        case "FFMPEG":
            strategy = new KeyframeExtractionFFMpeg();
            break;
        case "AZURE":
            strategy = new KeyframeExtractionAzure();
            break;
        default:
            throw new ArgumentException("Unknown strategy number");
    }
    strategy.ProgressChanged += Strategy_ProgressChanged;

    IsLoading = true;

    await Task.Run(() => strategy.ExtractKeyFrames(VideoPath, PathToSave));

    IsLoading = false;


    if (Directory.Exists(PathToSave))
    {
        ProcessStartInfo startInfo = new ProcessStartInfo
        {
            Arguments = PathToSave,
            FileName = "explorer.exe"
        };

        Process.Start(startInfo);
    }
}
```

Рисунок 4.5.18 – Метод «GenerateKeyframesAsync» класу «VideoControlModel»

```
private void Strategy_ProgressChanged(object sender, int progress)
{
    TaskProgress = progress;
    Console.WriteLine($"Progress: {TaskProgress}");
}
1 reference
private void SSIMRadioButton() => AlgoritmToUse = "SSIM";

1 reference
private void FFMPEGRadioButton() => AlgoritmToUse = "FFMPEG";

1 reference
private void AZURERadioButton() => AlgoritmToUse = "AZURE";


public event PropertyChangedEventHandler? PropertyChanged;

8 references
protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
```

Рисунок 4.5.19 – Клас «VideoControlModel». Частина 5

3.6 Реалізація патерну «Стратегія»

Патерн «Стратегія» — це ще один з ключових патернів проєкта, який дозволяє вибирати потрібний алгоритм для знаходження кадрів.

Основний компонент стратегії — інтерфейс IKeyframesDetectionStrategy (рис. 4.6.1), який визначає єдиний інтерфейс для всіх стратегій. Цей інтерфейс включає метод «ExtractKeyFrames», який необхідно реалізувати в кожній конкретній стратегії та подію «ProgressChanged», яка дозволяє демонструвати користувачу відсоток виконання задачі.

```
public interface IKeyframesDetectionStrategy
{
    4 references
    async Task ExtractKeyFrames(string videoPath, string folderPath) { }

    event EventHandler<int> ProgressChanged;
}
```

Рисуток 4.6.1 – Інтерфейс «IKeyframesDetectionStrategy»

3.7 Розробка алгоритму на основі SSIM

Один із способів знаходження ключових кадрів у відео — це використання алгоритму SSIM (Structural Similarity Index Measure). SSIM використовується для аналізу візуальної подібності між послідовними кадрами відео. Цей підхід дозволяє визначити зміни між кадрами, які можуть вказувати на наявність ключового кадру. Важливо, щоб відсоток схожості був не занадто великим, бо це може вказувати на мінімальні або відсутні зміни. Також не варто, щоб схожість була занадто низькою, адже це може означати, що зміни в кадрі занадто суттєві, а значить кадр може бути розмитим.

Реалізація алгоритму включає:

- **Встановлення початкових налаштувань:** спочатку перевіряється існування директорії для збереження кадрів і, за потреби, створюється нова.
- **Відкриття відеофайлу за допомогою VideoCapture:** перевірка на можливість відкриття відео для обробки.
- **Пошук активної області в кадрі:** визначення області в кадрі, яка має найбільшу різницю у середньоквадратичному відхиленні (MSE) між послідовними кадрами, що може вказувати на важливі візуальні зміни. MSE між кадрами X та Y вираховується за формулою:

$$(4.1) \qquad MSE(X,Y) = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}[Y(i,j) - X(i,j)]^2$$

де m і n — кількість пікселів за шириною і висотою у області, що нас цікавить, відповідно.

- **Обробка кадрів і визначення SSIM:** порівняння кадрів в обраному регіоні для визначення SSIM. Кадри, що мають значення SSIM у певному

діапазоні, можуть бути класифіковані як ключові. SSIM між кадрами X та Y вираховується за формулою [12]:

$$SSIM(X,Y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{XY} + C_2)}{(\mu_X^2 + \mu_Y^2 + C_1)(\sigma_X^2 + \sigma_Y^2 + C_2)} \quad (4.2)$$

де μ_X і μ_Y — середнє значення пікселів у кадрах X та Y, відповідно. σ_X і σ_Y — дисперсія пікселів у кадрах X та Y, відповідно. σ_{XY} — коваріація значень пікселів у кадрах X та Y, відповідно. $C_1 = (k_1 L)^2$, $C_2 = ((k_2 L)^2)$, де L —діапазон значень пікселів, а $k_1 = 0{,}01$ і $k_2 = 0{,}03$ — константи.

Для цілей виявлення ключових кадрів було обрано діапазон значень SSIM від 0.65 до 0.90. Цей діапазон є оптимальним для визначення кадрів, які містять суттєві, але не радикальні зміни.

- **Запис ключових кадрів у файл:** ключові кадри зберігаються у вказаній директорії.

Клас «KeyframeExtractionSSIM» наслідується від інтерфейсу «IKeyframesDetectionStrategy». Для імплементації інтерфейсу необхідно реалізувати метод «ExtractKeyFrames», який відповідає за знаходження ключових кадрів на основі подібності кадрів (рис. 4.7.1).

```
public async Task ExtractKeyFrames(string videoPath, string folderPath)
{
    UpdateProgress(0);

    if (!Directory.Exists(folderPath))
    {
        Directory.CreateDirectory(folderPath);
    }
    else
    {
        Directory.Delete(folderPath, true);
        Directory.CreateDirectory(folderPath);

    }

    using var capture = new VideoCapture(videoPath);
    if (!capture.IsOpened())
    {
        throw new Exception("Error: Unable to open video file.");
    }

    Rect actionTemplateRegion = new Rect();

    if (!findActiveRegion(capture, ref actionTemplateRegion))
    {
        throw new Exception("Error: Unable to find Active Region.");
    }

    await findFrames(capture, actionTemplateRegion, folderPath);
}
```

Рисунок 4.7.1 – Метод «ExtractKeyFrames» в класі «KeyframeExtractionSSIM»

Метод «ExtractKeyFrames» перевіряє доступність теки. Якщо тека відсутня, створюється нова. Також важливо перевірити правильність шляху до відео. Наступний важливий етап — знайти активну область в кадрі за допомогою порівняння MSE (Формула 4.1) (Рис. 4.7.2 – 4.7.3).

```
private bool findActiveRegion(VideoCapture capture, ref Rect region)
{
    Mat frame1 = new Mat();
    Mat frame2 = new Mat();

    capture.Read(frame1);
    capture.Read(frame2);

    if (frame1.Empty() || frame2.Empty())
    {
        Console.WriteLine("Error: Unable to capture frames.");
        return false;
    }

    Mat greyFrame1 = new Mat();
    Mat greyFrame2 = new Mat();

    Cv2.CvtColor(frame1, greyFrame1, ColorConversionCodes.BGR2GRAY);
    Cv2.CvtColor(frame2, greyFrame2, ColorConversionCodes.BGR2GRAY);

    int width = greyFrame1.Width;
    int height = greyFrame1.Height;
    int regionWidth = width / 3;
    Rect region1 = new Rect(0, 0, regionWidth, height);
    Rect region2 = new Rect(regionWidth, 0, regionWidth, height);
    Rect region3 = new Rect(2 * regionWidth, 0, regionWidth, height);

    double mse1 = CalculateMSE(greyFrame1[region1], greyFrame2[region1]);
    double mse2 = CalculateMSE(greyFrame1[region2], greyFrame2[region2]);
    double mse3 = CalculateMSE(greyFrame1[region3], greyFrame2[region3]);

    double maxMSE = Math.Max(mse1, Math.Max(mse2, mse3));
    Rect actionTemplateRegion = maxMSE switch
    {
        _ when maxMSE == mse1 => region1,
        _ when maxMSE == mse2 => region2,
        _ when maxMSE == mse3 => region3,
        _ => throw new InvalidOperationException("Invalid MSE values.")
    };

    region = actionTemplateRegion;

    return true;
}
```

Рисунок 4.7.2 – Пошук активної області

```
private double CalculateMSE(Mat frame1, Mat frame2)
{
    Mat diff = new Mat();
    Cv2.Absdiff(frame1, frame2, diff);
    diff = diff.Mul(diff);
    Scalar mse = Cv2.Mean(diff);

    return mse.Val0;
}
```

Рисунок 4.7.3 – Пошук MSE між кадрами (Формула 4.1)

Після цього, запускається метод «findFrames», який приймає відео, активну область і шлях для зберігання ключових кадрів (Рис. 4.7.4).

```
async private Task findFrames(VideoCapture capture, Rect region, string folderPath)
{
    Mat frame1 = new Mat();
    Mat frame2 = new Mat();

    int totalFrames = (int)capture.Get(VideoCaptureProperties.FrameCount);

    int frameIndex = 0;
    while (true)
    {
        frameIndex+=1;
        if (!frame2.Empty())
        {
            frame1 = frame2.Clone();
            capture.Read(frame2);
        }
        else
        {
            capture.Read(frame1);
            capture.Read(frame2);
        }

        if (frame1.Empty())
        {
            break;
        }


        if (frame2.Empty())
        {
            break;
        }

        double ssim = CalculateSSIM(frame1[region], frame2[region]);
        Console.WriteLine(ssim);

        if (ssim > 0.65 && ssim < 0.90)
        {
            string imagePath = Path.Combine(folderPath, $"keyframe_{++i}.jpg");
            frame1.ImWrite(imagePath);
        }

        UpdateProgress((int)((float)frameIndex / (float)totalFrames * 100));
    }

    UpdateProgress(100);
}
```

Рисунок 4.7.4 – Метод «findFrames»

Метод «findFrames» для кожних двох послідовних кадрів вираховує SSIM за допомогою метода «CalculateSSIM» (Формула 4.2) (Рис. 4.7.5).

```
1 reference
private double CalculateSSIM(Mat frame1, Mat frame2)
{

    const double k1 = 0.01;
    const double k2 = 0.03;

    double muX = Cv2.Mean(frame1).Val0;
    double muY = Cv2.Mean(frame2).Val0;

    double sigmaX_2 = CalculateVariance(frame1);
    double sigmaY_2 = CalculateVariance(frame2);

    double sigmaXY = CalculateCovariance(frame1, frame2);

    double L = 255;

    double C1 = Math.Pow((k1 * L), 2);
    double C2 = Math.Pow((k2 * L), 2);

    double ssim = ((2 * muX * muY + C1) * (2 * sigmaXY + C2)) / ((Math.Pow(muX, 2) + Math.Pow(muY, 2) + C1) * (sigmaX_2 + sigmaY_2 + C2));

    System.GC.Collect();

    return ssim;
}
```

Рисунок 4.7.5 – Метод «CalculateSSIM» (Формула 4.1)

Якщо SSIM знаходиться у вибраному діапазоні, кадр вважається ключовим і зберігається у визначену директорію. Після обробки кожної пари кадрів запускається метод «UpdateProgress» (рис 4.7.6), щоб оновити прогрес на екрані користувача за допомогою імплементованої з інтерфейсу події «ProgressChanged».

```
private void UpdateProgress(int percent)
{
    ProgressChanged.Invoke(null, percent);
}
```

Рисунок 4.7.6 – Метод «UpdateProgress»

Після обробки кожної пари послідовних кадрів і збереження ключових у теку алгоритм завершує свою роботу. Програма готова до роботи з іншими відеофайлами.

3.8 Інтеграція сторонньої бібліотеки FFMPEG

Клас «KeyframeExtractionFFMpeg» наслідується від інтерфейсу «IKeyframesDetectionStrategy». Для імплементації інтерфейсу необхідно реалізувати метод «ExtractKeyFrames», який відповідає за знаходження ключових кадрів за допомогою інструменту FFmpeg (рис. 4.8.1).

```
public async Task ExtractKeyFrames(string videoPath, string folderPath)
{
    ProgressChanged.Invoke(null, 50);

    var ffMpeg = new FFMpegConverter();

    if (!Directory.Exists(folderPath))
    {
        Directory.CreateDirectory(folderPath);
    }
    else
    {
        Directory.Delete(folderPath, true);
        Directory.CreateDirectory(folderPath);
    }

    string filterComplex = "[0:v]select='eq(pict_type,PICT_TYPE_I)'[keyframes]";

    ConvertSettings convertSettings = new ConvertSettings();

    string ffmpegArgs = $"-i \"{videoPath}\" -vf \"{filterComplex}\" -r 1 \"{folderPath}\\keyframe_%d.jpg\"";

    ffMpeg.Invoke(ffmpegArgs);

    ProgressChanged.Invoke(null, 100);

}
```

Рисунок 4.8.1 – Метод «ExtractKeyFrames» » в класі «KeyframeExtractionFFMpeg»

Метод «ExtractKeyFrames» перевіряє доступність теки. Якщо тека відсутня, створюються нова. Для вилучення ключових кадрів використовується спеціальна конфігурація FFmpeg. В командному рядку FFmpeg встановлюється фільтр select='eq(pict_type,PICT_TYPE_I)', який вибирає тільки I-frames (ключові кадри) з відеопотоку.
Команда FFmpeg виконується через інтерфейс FFMpegConverter. Команда містить шлях до відеофайлу та параметри, які визначають, що результатом виконання буде серія зображень (кожен ключовий кадр зберігається як окреме зображення у вказаній директорії).

Після завершення виконання команди FFmpeg подія ProgressChanged встановлює прогрес на 100%, сигналізуючи про завершення процесу.

3.9 Інтеграція штучного інтелекту Azure

Клас «KeyframeExtractionAzure» наслідується від інтерфейсу «IKeyframesDetectionStrategy». Для імплементації інтерфейсу необхідно

реалізувати метод «ExtractKeyFrames», який відповідає за знаходження ключових кадрів за допомогою штучного інтелекту Azure AI Video Indexer (Рис. 4.9.1).

```
public async Task ExtractKeyFrames(string videoPath, string folderPath)
{
    UpdateProgress(0);

    string apiKey = AzureConfig.ApiKey;
    string accountId = AzureConfig.AccountId;
    string permission = AzureConfig.Permission;
    string location = AzureConfig.Location;

    string name = Path.GetFileNameWithoutExtension(videoPath);

    string accessToken = await GetAccessToken(apiKey, accountId, permission, location);

    if (accessToken == "")
        throw new Exception("Auth failed");

    UpdateProgress(5);

    string video_id = await UploadVideo(apiKey, accountId, accessToken, location, name, videoPath);

    UpdateProgress(10);


    await WaitIndexing(apiKey, location, accountId, video_id, accessToken);

    await GetandSaveThumbnails(apiKey, location, accountId, video_id, accessToken, folderPath);

    UpdateProgress(95);

    await DeleteVideo(apiKey, location, accountId, video_id, accessToken);

    UpdateProgress(100);
}
```

Рисунок 4.9.1 – Метод «ExtractKeyFrames» в класі «KeyframeExtractionAzure»

Процес починається з оновлення індикатора прогресу до нуля. Після цього здійснюється авторизація за допомогою метода «GetAccessToken». Api-ключ, ідентифікатор акаунта, рівень дозволу і регіон Azure знаходяться у файлі конфігурації. Ці дані передаються в метод «GetAccessToken» (Рис. 4.9.2).

```
async Task<string> GetAccessToken(string apiKey, string accountId, string permission, string location)
{
    string accessToken = "";

    string apiAuthVideoUrl =
        $"https://api.videoindexer.ai/Auth/{location}/Accounts/{accountId}/AccessTokenWithPermission?permission={permission}";

    using (var client = new HttpClient())
    {
        client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", apiKey);

        var response = await client.GetAsync(apiAuthVideoUrl);

        if (response.IsSuccessStatusCode)
        {
            string responseBody = await response.Content.ReadAsStringAsync();
            accessToken = JsonConvert.DeserializeObject(responseBody).ToString();
            Console.WriteLine("Access token retrieved successfully.");
            Console.WriteLine("Access Token: " + accessToken);
        }
        else
        {
            Console.WriteLine("Failed to retrieve access token. Status code: " + response.StatusCode + response.ToString());
            return null;
        }
    }

    return accessToken;
}
```

Рисунок 4.9.2 – Метод «GetAccessToken»

Метод «GetAccessToken» відповідає за авторизацію і отримання ключу доступа, який необхідний для запитів до Azure Video Indexer API.

Метод складається з таких кроків:

- Формування URL для запиту, використовуючи параметри «location», «accountID» та «permission» (регіон Azure, ідентифікатор акаунта і рівень дозволу, відповідно).
- Надсилання GET запиту на сформовану URL адресу за допомогою бібліотеки «HttpClient». У запиті вказується api-ключ («apiKey») у заголовках запиту, що дозволяє авторизувати запит.
- Після відправки запиту, метод чекає на відповідь від сервера. Якщо відповідь успішна (код статусу 200), метод читає тіло відповіді, яке містить JSON об'єкт з ключем доступу. Ключ доступу десеріалізується і зберігається як рядок.

Після успішної авторизації, запускається метод «UploadVideo» (рис 4.9.3).

```
async Task<string> UploadVideo(string apiKey, string accountId, string accessToken, string location, string name, string videoPath)
{
    string apiUploadVideoUrl =
        $"https://api.videoindexer.ai/{location}/Accounts/{accountId}/Videos?name={name}&accessToken={accessToken}";

    using (var client = new HttpClient())
    {
        byte[] videoBytes = File.ReadAllBytes(videoPath);

        var content = new MultipartFormDataContent();
        content.Headers.Add("Ocp-Apim-Subscription-Key", apiKey);
        content.Add(new ByteArrayContent(videoBytes), "file", Path.GetFileName(videoPath));

        var response = await client.PostAsync(apiUploadVideoUrl, content);

        if (response.IsSuccessStatusCode)
        {
            string resp = await response.Content.ReadAsStringAsync();

            dynamic data = JObject.Parse(resp);
            string video_id = data.id;

            Console.WriteLine("Video uploaded successfully.");
            Console.WriteLine("video_id: " + video_id);
            return video_id;
        }
        else
        {
            Console.WriteLine("Failed to upload video. Status code: " + response.StatusCode + response.ToString());
            return "";
        }
    }
}
```

Рисунок 4.9.3 – Метод «UploadVideo»

Метод «UploadVideo» відповідає за відправку відео на сервер Azure.

Метод складається з таких кроків:

- Формується URL для запиту, використовуючи параметри «location», «accountID», «name» та «accessToken» (регіон Azure, ідентифікатор акаунта, ім’я відео та авторизаційний токен, відповідно).
- Відеофайл перетворюється на масив байтів, щоб його можна було передати запитом.
- Надсилається POST запит на сформовану URL-адресу за допомогою бібліотеки «HttpClient». Створюється MultipartFormDataContent, до якого додається відеофайл як байтовий масив. У запиті вказується api-ключ («apiKey») у заголовках запиту, що дозволяє авторизувати запит.
- Після відправки запиту, метод чекає на відповідь від сервера. Якщо відповідь успішна (код статусу 200), метод читає тіло відповіді, яке містить JSON об'єкт з ідентифікаційним кодом відео. Ідентифікаційний код відео зберігається як рядок.

Після успішної відправки відео, необхідно зчекати, поки відео індексується на сервері. Для цього запускається асинхронний метод «WaitIndexing» (рис 4.9.4).

```
async Task WaitIndexing(string apiKey, string location, string accountId, string videoId, string accessToken)
{
    while (true)
    {
        int percent = await GetVideoState(apiKey, location, accountId, videoId, accessToken);

        if (percent == -1)
        {
            Console.WriteLine("Error: Video cannot be indexed.");
            break;
        }
        else if (percent == 101)
        {
            Console.WriteLine("Video indexed.");
            UpdateProgress(90);
            break;
        }
        else
        {
            UpdateProgress((int)(10 + percent*0.8));
        }
        await Task.Delay(5000);
    }
}
```

Рисунок 4.9.4 – Метод «WaitIndexing»

Метод WaitIndexing використовує цикл, що працює до того моменту, поки відео не буде повністю оброблено або поки не виникне помилка. У кожній ітерації циклу за допомогою метода «GetVideoState» (рис 4.9.5) робиться запит до API Video Indexer, щоб отримати поточний статус відео. Метод «GetVideoState» повертає відсоток прогресу індексації відео. За допомогою події «ProgressChanged», метод «WaitIndexing» відправляє інформацію про прогрес у головний потік програми, що дозволяє користувачеві бачити зміни у відсотках завершення обробки.

```
async Task<int> GetVideoState(string apiKey, string location, string accountId, string videoId, string accessToken)
{
    dynamic data = await GetVideoIndex(apiKey, location, accountId, videoId, accessToken);

    if (data != null)
    {
        dynamic data_videos = data.videos;
        string state = data.state;

        switch (state)
        {
            case "Processed":
                return 101;

            case "Processing":
                try
                {
                    string percent = "";
                    percent = data_videos[0].processingProgress;

                    if (percent.Equals(""))
                        return 0;
                    string percentWithoutSymbol = percent.Replace("%", "");
                    int percentInt = int.Parse(percentWithoutSymbol);
                    return percentInt;
                }
                catch (Exception)
                {
                    Console.WriteLine("Error: Processing progress is not in the correct format.");
                    return -1;
                }

            default:
                return -1;
        }
    }
    else
    {
        return -1;
    }
}
```

Рисунок 4.9.5 – Метод «GetVideoState»

Метод «GetVideoState» відповідає за моніторинг відсотку індексації відео, яке завантажено на Azure Video Indexer. Метод виконує наступні кроки:

- Викликає метод GetVideoIndex (рис 4.9.6), для отримання детальної інформації про відео, включаючи його поточний стан (наприклад, "Processed" або "Processing").
- Після отримання відповіді від API, JSON відповідь перевіряється для визначення поточного стану відео. Стани відео можуть бути наступними:
    - Processed — відео повністю оброблено, метод повертає «101», який вказує на це;
    - Processing — відео досі обробляється, метод обчислює і повертає відсоток прогресу індексації;

- Інші стани: Якщо статус відео невідомий або вказує на помилку, метод повертає -1, чим вказує на помилку.

```
2 references
async Task<dynamic> GetVideoIndex(string apiKey, string location, string accountId, string videoId, string accessToken)
{
    string apiGetVideoIndexUrl = $"https://api.videoindexer.ai/{location}/Accounts/{accountId}/Videos/{videoId}/Index?accessToken={accessToken}";

    using (var client = new HttpClient())
    {
        client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", apiKey);

        var response = await client.GetAsync(apiGetVideoIndexUrl);

        if (response.IsSuccessStatusCode)
        {
            string resp = await response.Content.ReadAsStringAsync();
            dynamic data = JObject.Parse(resp);
            return data;
        }
        else
        {
            Console.WriteLine("Failed to retrieve video index. Status code: " + response.StatusCode + response.ToString());
            return null;
        }
    }
}
```

Рисунок 4.9.6 – Метод «GetVideoIndex»

Метод GetVideoIndex у класі KeyframeExtractionAzure виконує запит до Azure Video Indexer API для отримання повного індексу відео. Цей індекс містить детальну інформацію про вміст відео, включаючи метадані, структуру, ключові моменти та інші дані. Метод реалізує наступні кроки:

- Формується URL для запиту до API, використовуючи параметри «location», «accountID», «videoId» та «accessToken» (регіон Azure, ідентифікатор акаунта, ідентифікатор відео та авторизаційний токен, відповідно).
- Надсилається GET запит на сформовану URL адресу за допомогою бібліотеки «HttpClient». У запиті вказується api-ключ («apiKey») у заголовках запиту, що дозволяє авторизувати запит.
- Після відправки запиту, метод чекає на відповідь від сервера. Якщо відповідь успішна (код статусу 200), метод читає тіло відповіді, яке містить JSON об'єкт з індексом відео. JSON десеріалізується для подальшого аналізу та використання в інших частинах програми.

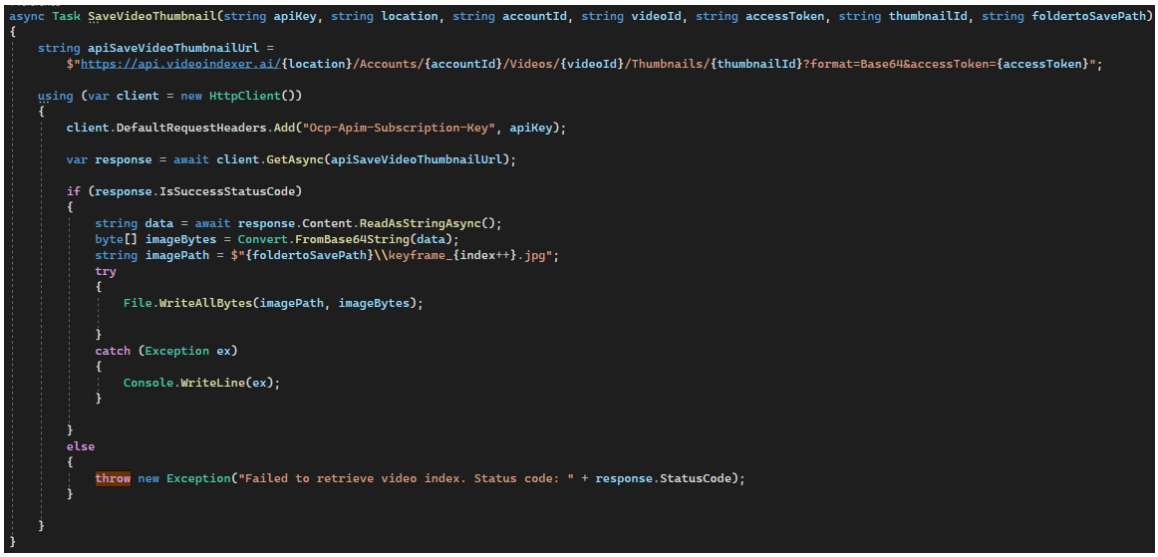
Після завершення індексації відео метод «GetandSaveThumbnails» (Рис. 4.9.7) зберігає ключові кадри в задану теку.

```
async Task GetandSaveThumbnails(string apiKey, string location, string accountId, string videoId, string accessToken, string foldertoSavePath)
{

    if (!Directory.Exists(foldertoSavePath))
    {
        Directory.CreateDirectory(foldertoSavePath);
    }
    else
    {
        Directory.Delete(foldertoSavePath, true);
        Directory.CreateDirectory(foldertoSavePath);
    }

    dynamic data = await GetVideoIndex(apiKey, location, accountId, videoId, accessToken);
    dynamic data_videos = data.videos;
    dynamic data_video = data_videos[0];
    dynamic data_insights = data_video.insights;
    dynamic data_shots = data_insights.shots;

    foreach (dynamic shot in data_shots)
    {
        dynamic keyFrames = shot.keyFrames;

        foreach (dynamic keyFrame in keyFrames)
        {
            dynamic instance = keyFrame.instances[0];
            string thumbnailId = instance.thumbnailId;

            await SaveVideoThumbnail(apiKey, location, accountId, videoId, accessToken, thumbnailId, foldertoSavePath);
        }
    }
}
```

Рисунок 4.9.7 – Метод «GetandSaveThumbnails»

Метод «GetandSaveThumbnails» відповідає за збір і збереження ключових кадрів, які були ідентифіковані штучним інтелектом Azure Video Indexer. Цей метод викликається після успішної індексації відео і має наступні кроки:

- Перевірка доступності директорії. Якщо тека відсутня, створюються нова.
- Отримання даних індексації відео. Метод викликає функцію GetVideoIndex (Рис. 4.9.6), яка повертає інформацію про відео, оброблену штучним інтелектом.
- Отримання ідентифікаторів ключових кадрів. В отриманих даних індексу з Azure Video Indexer ключові кадри зберігаються у вигляді ідентифікаторів, які знаходяться у різних сценах («Shot»).
- Збереження ключових кадрів у директорію. Для кожного ідентифікатора ключового кадру виконується метод «SaveVideoThumbnail» (рис 4.9.8) для збереження кадру у теку.



```
async Task SaveVideoThumbnail(string apiKey, string location, string accountId, string videoId, string accessToken, string thumbnailId, string foldertoSavePath)
{
    string apiSaveVideoThumbnailUrl =
        $"https://api.videoindexer.ai/{location}/Accounts/{accountId}/Videos/{videoId}/Thumbnails/{thumbnailId}?format=Base64&accessToken={accessToken}";

    using (var client = new HttpClient())
    {
        client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", apiKey);

        var response = await client.GetAsync(apiSaveVideoThumbnailUrl);

        if (response.IsSuccessStatusCode)
        {
            string data = await response.Content.ReadAsStringAsync();
            byte[] imageBytes = Convert.FromBase64String(data);
            string imagePath = $"{foldertoSavePath}\\keyframe_{index++}.jpg";
            try
            {
                File.WriteAllBytes(imagePath, imageBytes);

            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
            }

        }
        else
        {
            throw new Exception("Failed to retrieve video index. Status code: " + response.StatusCode);
        }

    }
}
```

Рисунок 4.9.8 – Метод «SaveVideoThumbnail»

Метод «SaveVideoThumbnail» відповідає за збереження ключових на основі їх унікальних ідентифікаторів у задану директорію. Метод виконує наступні кроки:

- Формується URL для запиту, використовуючи параметри «location», «accountID», «videoId», «thumbnailId» та «accessToken» (регіон Azure, ідентифікатор акаунта, ідентифікатор відео, ідентифікатор кадру та авторизаційний токен, відповідно).
- Надсилається GET запит на сформовану URL адресу за допомогою бібліотеки «HttpClient». У запиті вказується api-ключ («apiKey») у заголовках запиту, що дозволяє авторизувати запит.
- Після відправки запиту, метод чекає на відповідь від сервера. Якщо відповідь успішна (код статусу 200), зміст відповіді, який містить мініатюру у форматі Base64, конвертується у байти і зберігається в директорію.

Після того, як всі кадри збережені в задану директорію, проіндексований файл видаляється з сервісу за допомогою метода «DeleteVideo» (рис 4.9.9), звільняючи ресурси та гарантуючи конфіденційність даних.

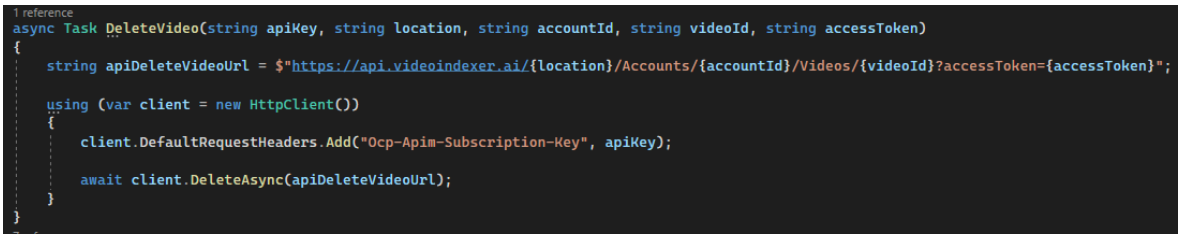


Рисунок 4.9.9 – Метод «DeleteVideo»

Метод «DeleteVideo» відповідає за видалення відео з сервісу Azure Video Indexer. Цей метод викликається після успішного збереження ключових кадрів і має виконує наступні кроки:

- Формується URL для запиту, використовуючи параметри «location», «accountID», «videoId» та «accessToken» (регіон Azure, ідентифікатор акаунта, ідентифікатор відео та авторизаційний токен, відповідно).
- Надсилається DELETE запит на сформовану URL адресу за допомогою бібліотеки «HttpClient». У запиті вказується api-ключ («apiKey») у заголовках запиту, що дозволяє авторизувати запит.
- Після відправки запиту, метод чекає на відповідь від сервера і перевіряє статус відповіді. Успішний запит на видалення повертає статус 204 («No Content»), що означає, що відео було успішно видалено.

Всі ключові кадри були успішно збережені локально в дану директорію. Процес від початку завантаження до завершення видалення відео з хмари є повністю завершеним. Програма готова до роботи з іншими відео.

3.10 Огляд продукту

- **Відкриття застосунку.** Після запуску програми, користувач бачить перед собою меню вибору відеофайлу (рис 4.10.1). По центру екрану знаходиться велика кнопка «Upload video». Це меню створено таким чином, щоб бути

максимально інтуїтивно зрозумілим і не перевантаженим зайвими деталями, що сприяє легкості використання для нових користувачів.

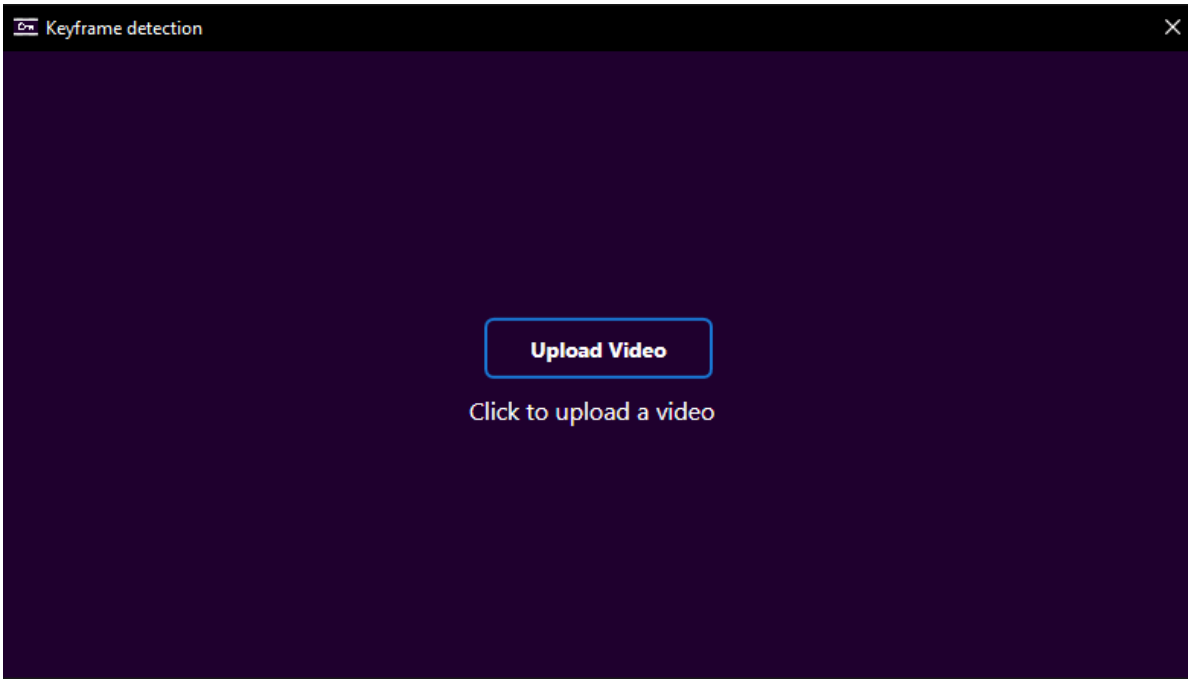


Рисунок 4.10.1 – Меню вибору відео

- **Натискання кнопки «Upload video».** Після натискання користувачем кнопки «Upload video», відкривається вікно вибору відеофайлу (рис 4.10.2). Це дозволяє швидко переглянути доступні файли на пристрої та вибрати бажаний відеофайл для завантаження. Процес вибору файлу має бути зрозумілим, тому використається стандартне діалогове вікно операційної системи, яке є знайомим для більшості користувачів.

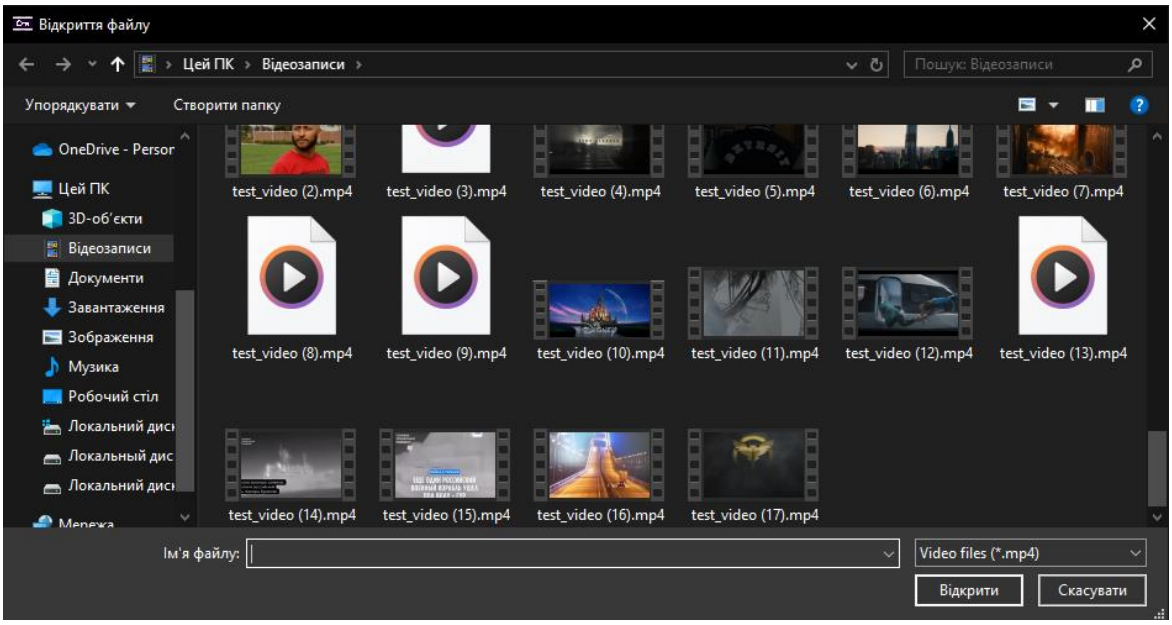


Рисунок 4.10.2 – Діалогове вікно вибору відео

- **Відкриття меню вибору алгоритму.** Після вибору відеофайлу користувачем автоматично відкриється меню вибору алгоритма (рис 4.10.3). У меню є такі елементи:
    - **Кнопка «Home».** За допомогою цієї кнопки користувач може повернутися до меню вибору відеофайлу. Це корисно, коли користувач хоче почати роботу з іншим відео.
    - **Загальна інформація про відео.** Ця частина інтерфейсу демонструє користувачу іконку, назву, шлях і тривалість відео. Це важливо для користувача, адже він впевнений, що не помилився з вибором відео та може переглянути всю важливу інформацію перед тим, як розпочати роботу
    - **Вибір шляху для збереження ключових кадрів.** За замовчуванням, шлях для збереження ключових кадрів автоматично встановлюється на нову теку у папці, де розташоване обране відео. Користувач має можливість ввести власний шлях вручну або вибрати іншу директорію через стандартне діалогове вікно, натиснувши на відповідну кнопку.
    - **Вибір алгоритму знаходження ключових кадрів.** Користувачу надано три кнопки для вибору алгоритму обробки відео: SSIM,

FFmpeg, і Azure. Обрати можна лише один алгоритм одночасно. За замовчуванням встановлена кнопка SSIM, яка є рекомендованим вибором, забезпечуючи баланс між швидкістю та якістю ключових кадрів.

- **Кнопка «Extract keyframes».** Натискання кнопки запускає алгоритм, який вибрав користувач, Ключові кадри зберігатимуться в попередньо заданій директорії.

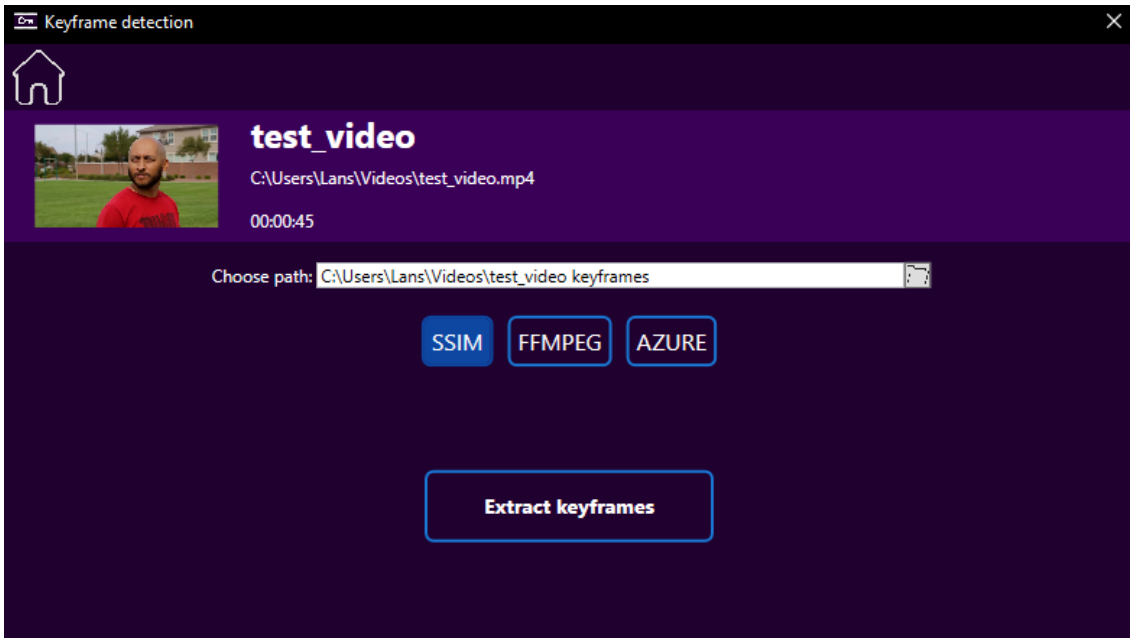


Рисунок 4.10.3 – Меню вибору алгоритму

- **Процес обробки відео.** Після натискання кнопки «Extract keyframes», алгоритм починає свою роботу і виводить на екран приблизний відсоток прогресу (рис 4.10.4). Це дає користувачам можливість відстежувати стан обробки в реальному часі. Показник прогресу забезпечує візуальну індикацію того, яка частина роботи вже виконана, і допомагає користувачам оцінити, скільки часу може зайняти завершення процесу. Це особливо зручно при обробці великих відеофайлів.

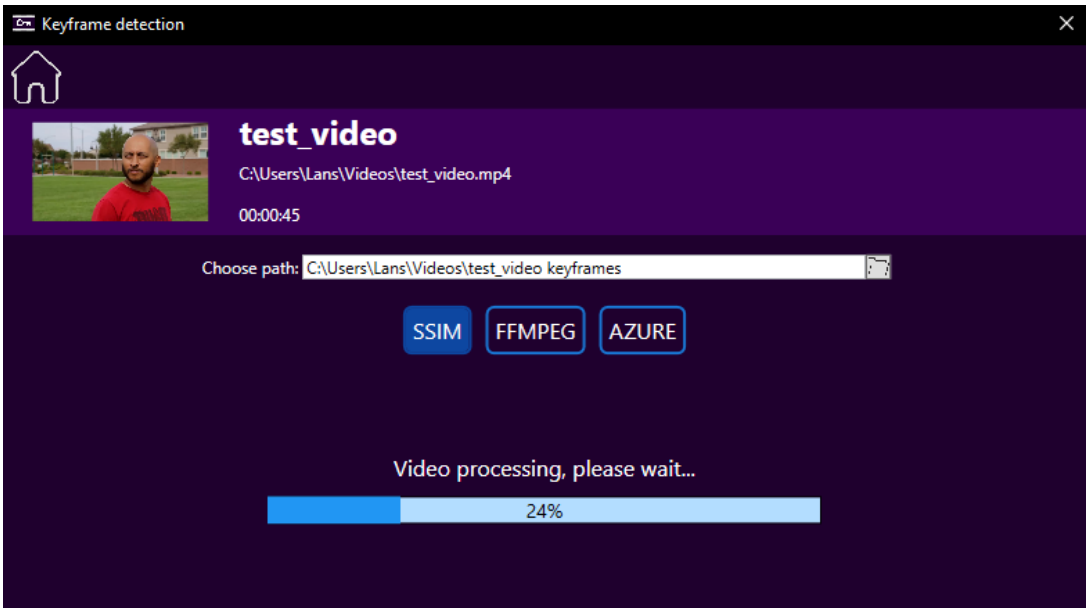


Рисунок 4.10.4 – Меню вибору алгоритму з індикацією прогресу

- **Завершення обробки відео.** Коли відео успішно оброблено і ключові кадри збережено у вказаній директорії, програма автоматично відкриває цю теку за допомогою файлового менеджера системи (рис 4.10.5). Цей крок надає користувачам зручний та швидкий доступ до результатів роботи алгоритма, дозволяючи їм одразу переглянути згенеровані ключові кадри.

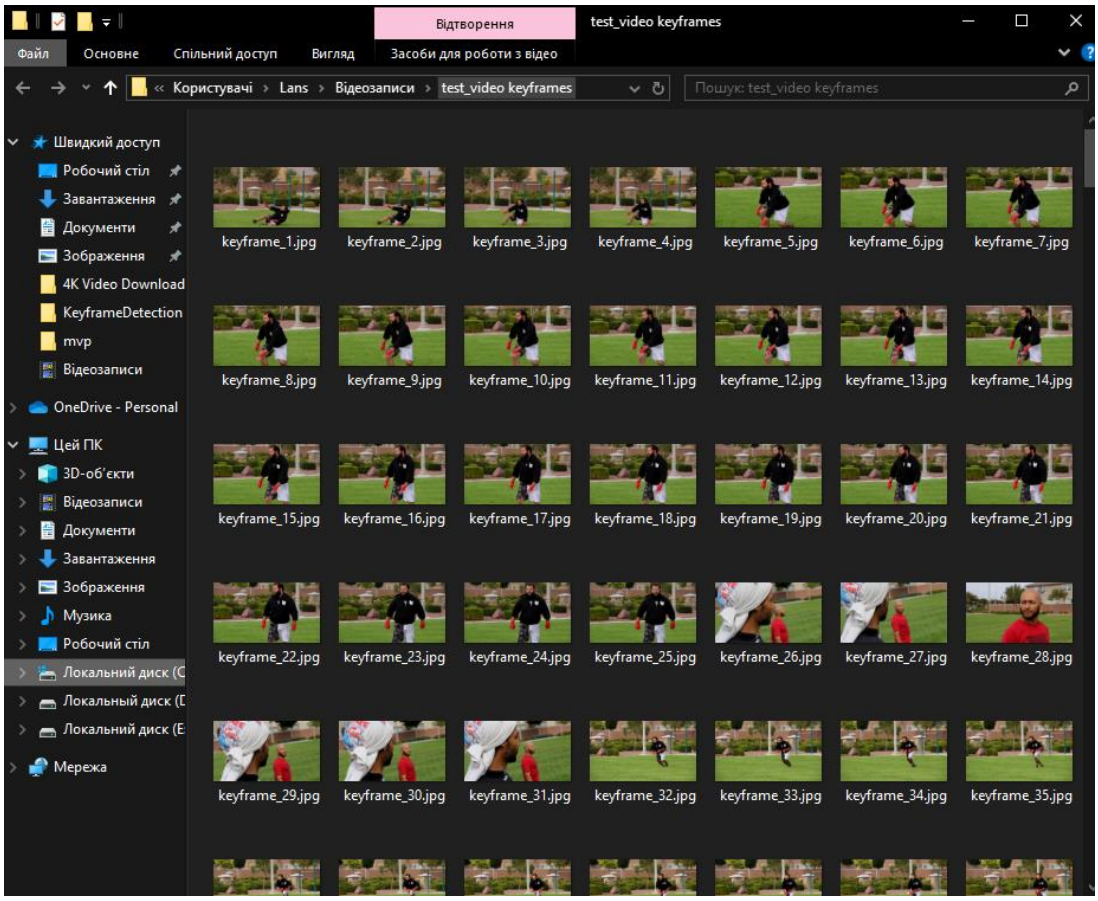


Рисунок 4.10.5 – Тека з ключовими кадрами.

- **Наступні дії користувача.** Користувач може закрити програму, вибрати інший алгоритм для роботи або повернутися до головного меню за допомогою кнопки «Home», щоб завантажити нове відео і продовжити роботу з програмою.

3.11 Тестування продукту

Для тестування було вибрано відео з динамічними сценами тривалістю 45 секунд. Відео має роздільну здатність 1920x1080 пікселів (Full HD). Обсяг файлу становить 20 мегабайт.

Програма успішно зібрала всю необхідну інформацію про відео (рис 4.11.1), включно з іконкою, яка точно відповідає зображенню, яке було обране файловою системою для цього відеофайлу.

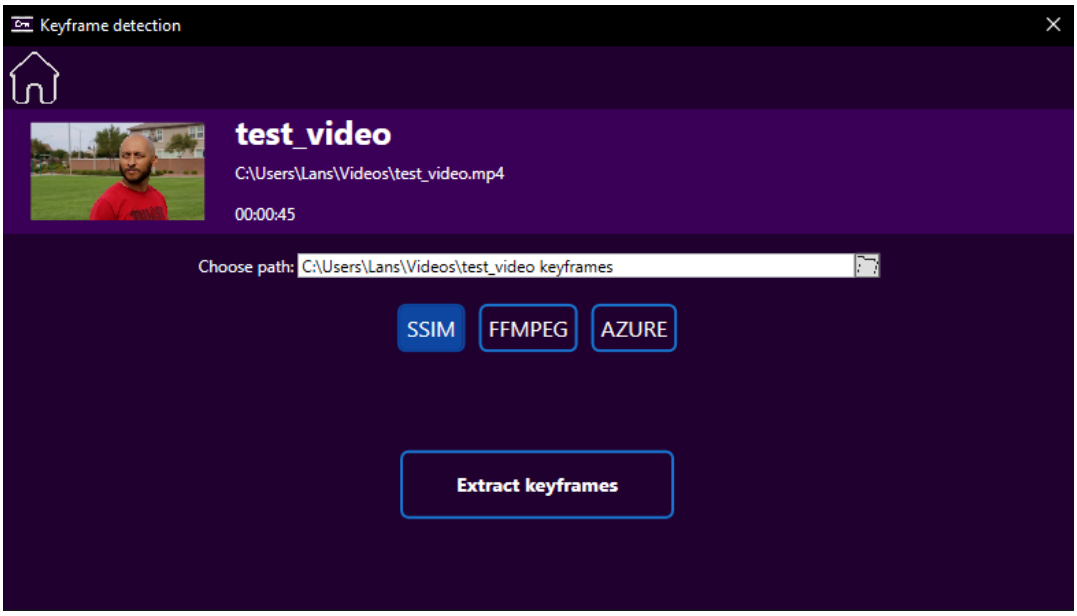


Рисунок 4.11.1 – Інформація про відео

Відео було протестоване на всіх трьох алгоритмах. Були отримані такі результати:

- **Алгоритм на основі SSIM** здійснив обробку відео за 34 секунди, за цей час вдалося вилучити 290 кадрів. Отримані кадри хорошої якості і змістовні, хоча робота зайняла певний час.
- **Алгоритм на основі FFmpeg.** показав значну швидкість, обробивши відео всього за 2 секунди і вилучивши 45 кадрів. Цей підхід демонструє велику швидкість обробки, проте кількість вилучених кадрів і їх якість поступається алгоритму на основі SSIM.
- **Алгоритм на основі Azure Video Indexe** працював набагато довше — 3 хвилини, в результаті чого було отримано 29 кадрів. Незважаючи на довгий час обробки, якість кадрів є найвищою, що робить цей метод оптимальним для ситуацій, коли якість важливіша часу.

3.12 Порівняння алгоритмів

У застосунку користувач може вибирати з трьох алгоритмів для знаходження ключових кадрів:

- **Алгоритм на основі SSIM**. iSSIM (Structural Similarity Index Measure) — це метрика, що використувається для оцінки якості кадру порівнюючи їх схожість.
- **Алгоритм на основі FFmpeg** — потужний інструмент для обробки мультимедіа, що включає можливості для пошуку ключових кадрів.
- **Алгоритм на основі Azure Video Indexer:** Це хмарне рішення від Microsoft, яке використовує алгоритми штучного інтелекту для аналізу відеоконтенту. Воно може виявляти ключові кадри на основі різних критеріїв, включаючи зміни в русі, кольорах чи навіть змістовні події.

Всі алгоритми були ретельно протестовані на різноманітних відео, тому можуть бути оцінені за декількома критеріями наступним чином::

Якість кадрів:

- **Алгоритм на основі SSIM** генерує ключові кадри непоганої якості. Іноді кадри можуть бути трохи розмитими, занадто темними або світлими. Проте кадр зазвичай передає багато інформації, за якою можна зрозуміти контекст відео.
- **Алгоритм на основі FFmpeg** знаходить ключові кадри середньої якості, знайдені кадри часто хронологічно далекі один від одного, а отже зрозуміти контекст відео в цілому складно. Кадри поступаються в якості оригінальному відеофайлу, а отже використовувати їх як іконку до відео не бажано. Знайдені кадри часто повторюють один одного. Іноді кадри можуть бути чорними, або дуже темними, наприклад, коли у відео використовується затемнення між сценами.
- **Алгоритм на основі Azure Video Indexer** знаходить ключові кадри дуже високої якості. Кадри завжди змістовні і чіткі. Це відбувається тому, що алгоритм працює на основі штучного інтелекту і чітко знає, що відбувається у відео. Проте кадри часто хронологічно далекі один від одного, що може ускладнити розуміння контексту відео.

Швидкість виконання:

- **Алгоритм на основі SSIM** оброблює ключові кадри з помірною швидкістю. Наприклад, для відео тривалістю одну хвилину, цей алгоритм потребує приблизно п'ятнадцять секунд на обробку. Зі збільшенням тривалості відео, час, необхідний для обробки, зростає пропорційно, що може призвести до значних затримок при роботі з тривалими відео.
- **Алгоритм на основі FFmpeg** вирізняється високою швидкістю обробки відео. Він здатний обробити 30 хвилин відео високої якості всього за дві хвилини. Ця швидкість обробки дозволяє ефективно та швидко працювати з великими відеофайлами.
- **Алгоритм на основі Azure Video Indexer** вимагає великої кількості часу порівняно з іншими алгоритмами. Відео спочатку завантажується на сервер, потім індексується за допомогою штучного інтелекту, а потім ключові кадри завантажуються з серверу назад до користувача. Тільки індексація відео тривалістю одну хвилину може зайняти від 5 до 10 хвилин. Працювати з великими відеофайлами дуже складно, адже це потребує значної кількості часу.

Затрати ресурсів комп’ютера:

- **Алгоритм на основі SSIM** вимагає великих обчислень, особливо при аналізі відео високої роздільної здатності. Алгоритм порівнює структурну подібність між всіма послідовними кадрами, що може призвести до великих витрат пам’яті і значного навантаження на процесор.
- **Алгоритм на основі FFmpeg**, як інструмент для роботи з медіа дуже оптимізований. Хоча він може сильно навантажувати процесор та використовувати багато пам’яті, загалом він є менш вимогливим до ресурсів у порівнянні з алгоритмом на основі SSIM.
- **Алгоритм на основі Azure Video Indexer** використовує хмарне рішення, всі обчислення відбуваються на серверах Azure, отже алгоритм зовсім не вимагає ресурсів на локальному комп’ютері. Але передача відеофайлу на

сервер через інтернет, може значно навантажувати мережу, особливо при роботі з великим відеофайлом.

Вартість:

- **Алгоритм на основі SSIM** має нульову вартість, був розроблений самостійно і базується лише на математичних формулах для оцінки схожості кадрів. Це робить алгоритм доступним рішенням для обробки відео, не вимагаючи витрат на програмне забезпечення.
- **Алгоритм на основі FFmpeg** також є безкоштовним, оскільки бібліотека FFmpeg ліцензована під GNU Lesser General Public License (LGPL) [13], що дозволяє використовувати його в межах програмного продукту, навіть в комерційних цілях.
- **Алгоритм на основі Azure Video Indexer** є платним. Вартість залежить від обсягу використання і вибраного тарифного плану. Azure пропонує декілька тарифних планів, які включають базовий безкоштовний обмежений рівень, який не підходить для комерційного використання, адже в ньому значно обмежена кількість годин контенту на місяць. Для комерційного програмного забезпечення підходять платні плани, які дозволяють завантажити більшу кількість відеоконтенту. Отже, вартість використання Azure Video Indexer залежить від обсягу відеоданих, які потребують обробки.

Аналізуючи порівняння, можна зробити наступні висновки:

- **Алгоритм на основі SSIM** займає середній час для генерації ключових кадрів, доброї якості, хоча й не ідеальні. Цей метод може суттєво навантажувати систему, проте є абсолютно безкоштовним. Завдяки цьому, алгоритм на основі SSIM є своєрідною «золотою серединою» між різними алгоритмами пошуку ключових кадрів, забезпечуючи збалансований вибір між швидкістю, якістю та вартістю.
- **Алгоритм на основі FFmpeg** вирізняється своєю швидкістю роботи, і також є безкоштовним для використання. Однак, ключові кадри, які він

генерує, часто мають не дуже високу якість. Це може бути важливим чинником, коли якість зображення є критичною. Отже, алгоритм на основі FFmpeg добре підходить для задач, коли швидкість обробки є більш важливою за детальність і точність кадрів, або якщо потрібно швидко обробити великі обсяги відеоданих.

- **Алгоритм на основі Azure Video Indexer** вирізняється дуже довгим часом роботи, але забезпечує високу якість ключових кадрів, яка є найкращою порівняно з іншими методами. Алгоритм не вимагає ресурсів комп'ютера, однак для роботи необхідне стабільне інтернет-з'єднання. Azure Video Indexer не є безкоштовним, і це може стати важливим фактором при виборі алгоритму. Алгоритм на основі Azure Video Indexer є хорошим вибором для задач, де необхідна висока точність визначення ключових кадрів. Проте для його використання необхідні значні часові та фінансові ресурси.

Підсумовуючи, кожен з алгоритмів має свої сильні та слабкі сторони, які визначають їхню придатність для різних задач. Вибір оптимального алгоритму залежить від вимог проєкту, доступних ресурсів і пріоритетів, таких як швидкість, вартість, та якість кадрів.

Висновок

У цій роботі було розглянуто визначення ключових кадрів, важливість алгоритмів їх виявлення, в також надані приклади реального використання цих алгоритмів. Також було здійснено дослідження наявних бібліотек, готових рішень та інструментів, які можуть бути використані для реалізації алгоритму.

На основі проведеного дослідження був розроблений власний продукт, який містить декілька алгоритмів, а саме алгоритм на основі схожості кадрів (SSIM), алгоритм на основі інструменту «FFMpeg», а також алгоритм, який використовує хмарний сервіс «Azure Video Indexer» для вилучення ключових кадрів. Розроблене програмне забезпечення є повністю готове для комерційного використання та може бути розповсюджене на ринку.

У роботі представлений детальний опис коду, який включає в себе ретельний аналіз кожного класу, методу та інтерфейсу. Огляд зосереджений на архітектурі програми, де кожен компонент розглянутий з метою забезпечення чіткого розуміння їх функціоналу та взаємодії з іншими компонентами. Також описані графічні відображення, опис взаємодії користувача з програмою через графічний інтерфейс, а також логіка навігації між різними частинами програми.

Програмний продукт був розроблений на мові програмування «C#», використовуючи платформу «.Net» та підсистему «WPF». В процесі розробки були застосовані патерни програмування «MVVM» та «Стратегія», що сприяло підвищенню модульності та гнучкості коду.

У роботі наведений детальний огляд розробленого програмного продукту з точки зору користувача, зроблений після завершення всіх етапів розробки. Також було проведено ретельне тестування для визначення умов використання кожного алгоритму, а також детальне порівняння їх продуктивності та якості результатів. Це дозволило розробити рекомендації для практичного використання кожного алгоритма в залежності від наявної задачі та конкретного відеофайлу.

В майбутньому до програмного забезпечення можна зручно додати більшу кількість алгоритмів, завдяки патерну «Стратегія». Також можна оновити графічний інтерфейс на більш сучасний. Програмне забезпечення можливо інтегрувати до інших платформ, такі як IOS та Linux, що зробить його доступним для більш широкого кола користувачів.

Список використаної літератури

1. [Електронний ресурс] Hours of video uploaded to YouTube every minute https://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/
   (Перевірка на доступ до ресурсу — 10.04.2024)

2. [Електронний ресурс] YouTube thumbnails. https://developers.google.com/youtube/v3/docs/thumbnails?hl=en
   (Перевірка на доступ до ресурсу — 10.04.2024)

3. Adobe Premiere Pro poster frames https://helpx.adobe.com/premiere-pro/using/create-multi-camera-source-sequence.html
   (Перевірка на доступ до ресурсу — 10.04.2024)

4. An overview of video analytics in security https://www.ifsecglobal.com/video-surveillance/overview-video-analytics-security/
   (Перевірка на доступ до ресурсу — 10.04.2024)

5. A Detailed Overview Of Popular Video Compression Techniques https://imagekit.io/blog/video-compression-techniques/#:~:text=Video%20compression%20algorithms%20use%20both,redundancies%20within%20a%20single%20frame.
   (Перевірка на доступ до ресурсу — 10.04.2024)

6. [Електронний ресурс] A Comparison Between KeyFrame Extraction Methods for Clothing Recognition

https://uu.diva-portal.org/smash/get/diva2:1779690/FULLTEXT01.pdf
(Перевірка на доступ до ресурсу — 10.04.2024)

7. [Електронний ресурс] Azure AI Video Indexer
https://azure.microsoft.com/en-us/products/ai-video-indexer
(Перевірка на доступ до ресурсу — 10.04.2024)

8. [Електронний ресурс] Key-Frame Extraction Based on HSV Histogram and Adaptive Clustering
https://www.hindawi.com/journals/mpe/2019/5217961/
(Перевірка на доступ до ресурсу — 10.04.2024)

9. [Електронний ресурс] Video Key-Frame Extraction using Unsupervised Clustering and Mutual Comparison
https://www.semanticscholar.org/paper/Video-Key-Frame-Extraction-using-Unsupervised-and-Janwe-Bhoyar/961054bb6da0cab77906962e2e35d8a56cfaa417
(Перевірка на доступ до ресурсу — 10.04.2024)

10. [Електронний ресурс] VLC media player
https://www.videolan.org/developers/vlc.html
(Перевірка на доступ до ресурсу — 10.04.2024)

11. [Електронний ресурс] mozilla-central files
https://hg.mozilla.org/mozilla-central/file/tip/media/ffvpx/libavcodec
(Перевірка на доступ до ресурсу — 10.04.2024)

12. [Електронний ресурс] Wang Z, Bovik AC, Sheikh HR, Simoncelli EP et al (2004) Image quality assessment: from error visibility to structural similarity
https://ieeexplore.ieee.org/document/1284395

(Перевірка на доступ до ресурсу — 10.04.2024)

13. [Електронний ресурс] FFmpeg License https://ffmpeg.org/legal.html
(Перевірка на доступ до ресурсу — 10.04.2024)

14. [Електронний ресурс] A novel keyframe extraction method for video classification using deep neural networks
https://d-nb.info/1244593923/34
(Перевірка на доступ до ресурсу — 10.04.2024)