



POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Marcin Jagielak

Nr albumu s21298

Wykorzystanie modeli językowych do wspomagania procesu edycji wideo

Praca magisterska

Promotor:

**dr. inż. Mariusz Trzaska,
prof. PJATK**

Warszawa, wrzesień 2026

29.08.2026, Warszawa

(data, miejscowość)

OŚWIADCZENIE STUDENTA DOTYCZĄCE WYKORZYSTANIA SZTUCZNEJ INTELIGENCJI (AI)

Ja, niżej podpisany/a: Marcin Jagielak

student/ka kierunku: Informatyka, II stopnia

numer albumu studenta: s21298

Oświadczam, że zapoznałem/am się z treścią Zarządzenia Rektora Polsko-Japońskiej Akademii Technik Komputerowych z dnia 1 kwietnia 2026 r. oraz z „Wytycznymi dotyczącymi wykorzystywania sztucznej inteligencji w procesie dydaktycznym PJATK”, stanowiącymi załącznik do ww. zarządzenia.

Oświadczam, że ewentualne wykorzystanie narzędzi sztucznej inteligencji (AI) w niniejszej pracy miało charakter wyłącznie pomocniczy i nie stanowiło zastępstwa dla mojej samodzielnej pracy.

Oświadczam, że treść pracy stanowi moje własne, samodzielne opracowanie oraz ponoszę pełną odpowiedzialność za jej zawartość, niezależnie od wykorzystania narzędzi AI.

Oświadczam, że nie wykorzystywałem/am narzędzi AI do generowania treści stanowiących zasadniczą część pracy, w szczególności analiz, wniosków, interpretacji ani części podlegających ocenie jako przejaw twórczości własnej.

W przypadku korzystania z narzędzi AI sporządziłem/am Raport użycia AI zgodnie z obowiązującymi Wytycznymi i dołączyłem/am go do pracy.

Marcin Jagielak

(czytelny podpis studenta/studentki)

Streszczenie

Niniejsza praca przedstawia projekt i implementację prototypu wykorzystującego duże modele językowe do wspomagania procesu edycji wideo w DaVinci Resolve 21 Free. Omówiono proces jego powstania: od analizy możliwości automatyzacji montażu i istniejących rozwiązań, przez opracowanie architektury oraz implementację poszczególnych komponentów, po integrację z edytorem i ocenę gotowego rozwiązania. Celem było zmniejszenie bariery związanej z obsługą profesjonalnego środowiska montażowego, przy zachowaniu kontroli użytkownika nad zmianami w projekcie. System ma postać lokalnej aplikacji desktopowej współpracującej z edytorem i pełniącej funkcję wtyczki. Składa się z interfejsu użytkownika, warstwy serwerowej oraz mostu wykonawczego działającego wewnątrz DaVinci Resolve. Aplikacja analizuje materiały multimedialne, przygotowuje kontekst dla modeli, rozpoznaje zamiar użytkownika i opracowuje koncepcję montażu. Wyniki działania modeli nie są przekazywane bezpośrednio do edytora. Deterministyczne moduły warstwy serwerowej przekształcają je w typowany i walidowany plan edycji, zawierający wyłącznie operacje obsługiwane przez system. Plan jest przedstawiany użytkownikowi i realizowany dopiero po jego zaakceptowaniu. Prototyp zweryfikowano za pomocą serii prób obejmujących pełny przepływ działania, z użyciem różnych poleceń i materiałów źródłowych. W badanych przypadkach umożliwił on tworzenie oraz modyfikowanie krótkich filmów przy zachowaniu możliwości dalszej edycji projektu. Główne ograniczenia wynikają z niedeterministycznego działania modeli językowych oraz niepełnego zakresu operacji udostępnianych przez interfejsy programistyczne bezpłatnej wersji DaVinci Resolve.

Słowa kluczowe: duże modele językowe, edycja wideo, automatyzacja montażu, DaVinci Resolve, analiza multimediiów, język naturalny.

Spis treści

1.	WSTĘP	6
1.1.	Motywacja	6
1.2.	Cel pracy.....	6
1.3.	Rozwiązanie przyjęte w pracy.....	7
1.4.	Rezultaty pracy	7
1.5.	Organizacja pracy	8
2.	WSPOMAGANIE I AUTOMATYZACJA PROCESU EDYCJI WIDEO	9
2.1.	Cel, zakres i metodyka analizy.....	9
2.2.	Proces edycji wideo i zakres jego automatyzacji	10
2.3.	Metody analizy materiałów multimedialnych	11
2.3.1	<i>Odczyt i przygotowanie materiału multimedialnego</i>	<i>11</i>
2.3.2	<i>Wykrywanie granic ujęć.....</i>	<i>12</i>
2.3.3	<i>Wybór i ocena klatek reprezentatywnych.....</i>	<i>13</i>
2.3.4	<i>Rozpoznawanie tekstu w obrazie</i>	<i>13</i>
2.3.5	<i>Analiza sygnału dźwiękowego</i>	<i>14</i>
2.3.6	<i>Reprezentacja wyników analizy</i>	<i>15</i>
2.4.	Modele językowe we wspomaganiu montażu wideo	16
2.4.1	<i>Podstawy działania dużych modeli językowych.....</i>	<i>16</i>
2.4.2	<i>Interpretacja poleceń użytkownika.....</i>	<i>18</i>
2.4.3	<i>Ustrukturyzowane odpowiedzi i wywoływanie narzędzi</i>	<i>19</i>
2.4.4	<i>Planowanie wieloetapowe i systemy agentowe</i>	<i>20</i>
2.4.5	<i>Ograniczenia modeli językowych i mechanizmy kontroli.....</i>	<i>21</i>
2.5.	Istniejące rozwiązania wspomagające edycję wideo.....	22
2.5.1	<i>Profesjonalne edytory wideo z funkcjami automatyzacji</i>	<i>22</i>
2.5.2	<i>Wtyczki rozszerzające możliwości programów do edycji wideo.....</i>	<i>25</i>
2.5.3	<i>Aplikacje wspomagające edycję wideo.....</i>	<i>26</i>
2.5.4	<i>Prototypy i rozwiązania badawcze</i>	<i>27</i>
2.6.	Analiza i wnioski.....	28
3.	PROJEKT SYSTEMU CUTTIE-AI	31
3.1.	Założenia projektu	31
3.2.	Architektura rozwiązania	32
3.3.	Analiza i reprezentacja materiałów multimedialnych.....	33
3.4.	Interpretacja polecenia i orkiestracja procesu planowania.....	35
3.5.	Budowanie i walidacja planu edycji.....	37
3.6.	Kontrolowane wykonanie planu w DaVinci Resolve.....	38
4.	NARZĘDZIA ZASTOSOWANE W SYSTEMIE WSPOMAGANIA MONTAŻU WIDEO	40
4.1.	Kryteria doboru narzędzi i technologii.....	40
4.2.	Technologie warstwy serwerowej	41
4.2.1	<i>Python i FastAPI</i>	<i>41</i>
4.2.2	<i>Walidacja i modele danych</i>	<i>42</i>
4.2.3	<i>Przechowywanie danych i ochrona informacji</i>	<i>42</i>
4.3.	Narzędzia integracji modeli językowych.....	43
4.3.1	<i>DSPy.....</i>	<i>43</i>
4.3.2	<i>LiteLLM.....</i>	<i>45</i>
4.4.	Narzędzia analizy materiałów multimedialnych	46
4.4.1	<i>Przetwarzanie plików multimedialnych.....</i>	<i>46</i>
4.4.2	<i>Analiza obrazu i detekcja scen.....</i>	<i>47</i>
4.4.3	<i>Analiza mowy i dźwięku.....</i>	<i>47</i>
4.5.	Technologie aplikacji desktopowej i interfejsu użytkownika.....	48

4.5.1	<i>Electron jako środowisko aplikacji desktopowej</i>	48
4.5.2	<i>React i warstwa interfejsu użytkownika</i>	48
4.6.	Środowisko DaVinci Resolve	49
4.6.1	<i>DaVinci Resolve i Fusion API</i>	49
4.6.2	<i>Lua i mechanizmy skryptowe wewnątrz Resolve</i>	50
4.7.	Wybór dostawcy modeli językowych	51
4.8.	Narzędzia testowania rozwiązania	51
5.	IMPLEMENTACJA PROTOTYPU SYSTEMU CUTTIE-AI	53
5.1.	Zakres implementacji prototypu	53
5.2.	Organizacja kodu źródłowego i zakres opisu implementacji	54
5.3.	Implementacja analizy materiałów i przygotowania kontekstu	55
5.4.	Implementacja interpretacji polecenia i sterowania rolami	58
5.5.	Implementacja typowanego planu edycji	61
5.6.	Implementacja zatwierdzania i wykonania planu	64
6.	WERYFIKACJA ZAŁOŻEŃ PROJEKTOWYCH I ANALIZA WYNIKÓW	67
6.1.	Najważniejsze problemy implementacyjne	67
6.1.1	<i>Dobór sposobu wykorzystania modeli</i>	67
6.1.2	<i>Niedeterministyczność modeli i kontrola wykonania</i>	68
6.1.3	<i>Ograniczenia integracji z DaVinci Resolve 21 Free</i>	69
6.1.4	<i>Trudności związane z testowaniem</i>	71
6.2.	Wyniki weryfikacji	71
6.3.	Ocena realizacji założeń projektowych	73
6.4.	Potencjalne zastosowania	73
7.	PODSUMOWANIE	75
	PRACE CYTOWANE	76
	DODATKI	81
	Skróty	81
	Terminologia	83
	Spis rysunków	85
	Spis tabel	86
	Spis listingów	87

1. Wstęp

Rozwój mediów społecznościowych oraz powszechna dostępność urządzeń do rejestrowania obrazu sprawiły, że tworzenie i publikowanie treści audiowizualnych przestało być domeną profesjonalnych twórców. Samo posiadanie materiału i pomysłu nie wystarcza jednak do urzeczywistnienia zamierzonej wizji. Użytkownik musi dodatkowo potrafić przełożyć oczekiwany rezultat na konkretne operacje wykonywane w programie do edycji wideo. Niniejsza praca koncentruje się na opracowaniu rozwiązania ograniczającego tę barierę poprzez wykorzystanie modeli językowych do interpretowania poleceń użytkownika i przekształcania ich w kontrolowany plan operacji montażowych.

1.1. Motywacja

W mediach społecznościowych szczególne znaczenie zyskały krótkie formy wideo. Kilkudziesięciosekundowy materiał może łączyć obraz, mowę, muzykę i tekst, dzięki czemu pozwala przekazać treść w bardziej skondensowanej formie niż kilkudziesięciowy opis. Przygotowanie takiego materiału nadal wymaga jednak wyboru odpowiednich fragmentów nagrań, ustalenia ich kolejności oraz dopasowania warstwy wizualnej i dźwiękowej.

Użytkownik często posiada określoną koncepcję montażu, ale nie zawsze potrafi przełożyć ją na konkretne funkcje programu. Koncepcja ta jest zazwyczaj formułowana na poziomie oczekiwanego efektu końcowego. Program do montażu wymaga natomiast wybrania odpowiednich narzędzi, określenia ich parametrów oraz zastosowania ich we właściwej kolejności. Realizacja nawet stosunkowo prostego zadania montażowego może zatem wymagać znajomości wielu funkcji oraz zależności zachodzących pomiędzy wykonywanymi operacjami. Problem ten nie dotyczy jednak wyłącznie początkujących użytkowników. Również doświadczeni montażyści poświęcają czas na wielokrotne wykonywanie podobnych operacji i konfigurowanie ich parametrów. Możliwość opisania zamierzonego efektu w języku naturalnym mogłaby uprościć ten proces i przyspieszyć realizację powtarzalnych zadań. Stanowi to uzasadnienie dla opracowania rozwiązania, które przekształca intencję użytkownika w kontrolowalny i możliwy do zweryfikowania plan działań montażowych.

1.2. Cel pracy

Przedmiotem pracy jest zaprojektowanie i wykonanie systemu umożliwiającego obsługę wybranych funkcji programu DaVinci Resolve za pomocą poleceń w języku naturalnym. Rozwiązanie ma ułatwić korzystanie z edytora osobom o różnym poziomie doświadczenia w montażu. Osobom początkującym system ma pozwolić przygotować prosty materiał lub edytowalny szkic projektu bez znajomości interfejsu, terminologii i sposobu wykonania poszczególnych operacji. Profesjonalnym montażyście ma natomiast pomóc szybciej wykonywać powtarzalne czynności. Polecenia użytkownika powinny być przekształcane w plan działań, który można sprawdzić i zatwierdzić przed wprowadzeniem zmian na osi czasu.

1.3. Rozwiązanie przyjęte w pracy

Przyjęte rozwiązanie jest lokalną aplikacją desktopową działającą jako wtyczka do programu DaVinci Resolve 21 Free. Aplikacja analizuje materiały multimedialne i na podstawie poleceń użytkownika sformułowanych w języku naturalnym przygotowuje plan montażu. DaVinci Resolve musi być uruchomiony podczas wykonywania zaplanowanych operacji.

System składa się z interfejsu użytkownika, lokalnej warstwy serwerowej odpowiedzialnej za logikę aplikacji oraz mostu wykonawczego działającego wewnątrz edytora. Podział ten wynika z ograniczeń integracji z bezpłatną wersją DaVinci Resolve: dostęp do projektu musi odbywać się z poziomu programu, natomiast pozostałe elementy aplikacji są uruchamiane jako osobne procesy. Do implementacji poszczególnych części wykorzystano:

- **Electron, React i TypeScript** – interfejs aplikacji stacjonarnej;
- **Python i FastAPI** – lokalna warstwa serwerowa;
- **Lua** i plikowa komunikacja międzyprocesowa – połączenie z edytorem oraz wykonanie operacji;
- **Modele językowe i multimodalne** – analizę materiałów i planowanie montażu;
- **EditPlan** – typowany kontrakt między planowaniem a wykonaniem.

Analiza materiałów poprzedza planowanie i obejmuje treść obrazu, mowę oraz cechy sygnału dźwiękowego. Uzyskane informacje opisują zawartość nagrań i tworzą kontekst potrzebny do interpretacji polecenia, wyboru fragmentów oraz przygotowania koncepcji montażu.

Przyjęta metoda oddziela zadania wymagające interpretacji od operacji modyfikujących projekt. Modele multimodalne wspomagają analizę materiałów, a modele językowe interpretują polecenie użytkownika, uczestniczą w wyborze fragmentów i przygotowują koncepcję montażu. Zadania te rozdzielono pomiędzy wyspecjalizowane role agentowe. Moduły deterministyczne budują końcowy plan operacji i sprawdzają jego poprawność.

Wynikiem planowania jest EditPlan, czyli model danych zawierający wyłącznie operacje ze zdefiniowanego zbioru poleceń. Lokalna warstwa serwerowa waliduje plan, po czym interfejs przedstawia go użytkownikowi do zatwierdzenia. Po akceptacji most wykonawczy realizuje zawarte w nim operacje w DaVinci Resolve.

1.4. Rezultaty pracy

Założony cel pracy oraz zakres funkcjonalny zrealizowano w stopniu przewidzianym dla prototypu. Powstała aplikacja obejmuje proces od analizy materiałów i interpretacji polecenia użytkownika do zatwierdzenia oraz wykonania planu edycji w DaVinci Resolve 21 Free.

Jednym z założeń rozwiązania było zapewnienie użytkownikowi możliwie dużej swobody w doborze materiałów i formułowaniu poleceń. Z tego względu, a także z uwagi na niedeterministyczny charakter modeli językowych, nie jest możliwa wyczerpująca weryfikacja wszystkich przypadków użycia. Nie oznacza to jednak, że system nie podlega testowaniu. Jego deterministyczne elementy można weryfikować w sposób powtarzalny, natomiast odpowiedzi modeli oceniać na podstawie reprezentatywnych scenariuszy oraz wizualnej kontroli rezultatów uzyskanych w edytorze wideo.

Praktycznym rezultatem pracy jest możliwość sterowania obsługiwanymi operacjami montażowymi za pomocą języka naturalnego bez przekazywania modelowi bezpośredniej kontroli nad projektem. Przyjęta konstrukcja pozwala również rozszerzać aplikację o kolejne operacje, metody analizy materiałów i modele.

1.5. Organizacja pracy

Struktura pracy obejmuje siedem rozdziałów. Rozdział 1 przedstawia kontekst podjętego problemu, przyjęte rozwiązanie i uzyskane rezultaty.

Rozdział 2 przedstawia stan sztuki dotyczący zagadnień związanych z automatyzacją montażu wideo. Uwzględniono metody analizy materiałów multimedialnych, możliwości i ograniczenia modeli językowych oraz rozwiązania obecnie dostępne na rynku.

Rozdział 3 przedstawia założenia i architekturę systemu oraz sposób analizy materiałów, interpretacji poleceń, planowania operacji i ich kontrolowanego wykonywania w DaVinci Resolve.

Rozdział 4 omawia technologie i narzędzia wykorzystane do budowy lokalnej warstwy serwerowej aplikacji desktopowej, integracji modeli językowych, analizy materiałów multimedialnych oraz testowania rozwiązania.

Rozdział 5 przedstawia organizację kodu źródłowego oraz najważniejsze rozwiązania implementacyjne.

Rozdział 6 omawia problemy napotkane podczas realizacji, sposób weryfikacji, uzyskane wyniki i ograniczenia prototypu.

Rozdział 7 podsumowuje realizację celu pracy oraz wskazuje możliwe kierunki dalszego rozwoju.

2. Wspomaganie i automatyzacja procesu edycji wideo

Jednym z podstawowych problemów wspomaganie pracy z oprogramowaniem specjalistycznym jest przekształcenie nieformalnego opisu oczekiwanego rezultatu w zestaw konkretnych działań. Zadanie to tradycyjnie wykonuje człowiek posiadający wiedzę dziedzinową, który interpretuje ogólnie sformułowany cel i dobiera środki potrzebne do jego realizacji. W przypadku edycji wideo rolę tę pełni przede wszystkim montażysta. Na podstawie scenariusza, wytycznych, uwag pozostałych uczestników procesu produkcyjnego oraz analizy dostępnych materiałów podejmuje on decyzje dotyczące wyboru ujęć, ich kolejności, punktów cięć, rytmu, relacji obrazu z dźwiękiem i zastosowanych środków montażowych.

Opis oczekiwanego rezultatu i jego realizacja w środowisku montażowym należą do dwóch różnych poziomów pracy. Użytkownik może posługiwać się określeniami odnoszącymi do treści, narracji, tempa, stylu lub przeznaczenia filmu, nie wskazując przy tym dokładnej sekwencji czynności koniecznych do przygotowania materiału. Program do edycji wymaga natomiast jednoznacznego określenia materiałów źródłowych, zakresów czasowych, kolejności klipów, ich rozmieszczenia na ścieżkach oraz parametrów wykonywanych operacji. Przejście od ogólnego opisu do gotowego materiału wymaga zatem nie tylko rozpoznania celu użytkownika, lecz także odniesienia go do zawartości nagrań i możliwości oferowanych przez środowisko edycyjne.

Na potrzeby tego rozdziału **duże modele językowe** (ang. *large language models*, **LLM**) rozumie się jako modele generatywne, które przetwarzają polecenia sformułowane w języku naturalnym wraz z przekazanym kontekstem i na tej podstawie tworzą odpowiedzi. Rozwój modeli językowych umożliwił częściowe przenoszenie tak rozumianej interpretacji do systemów informatycznych, w których język naturalny może pełnić funkcje warstwy pośredniej pomiędzy opisem celu a planem działań wykonywanych przez specjalistyczne oprogramowanie. Model językowy może wspomagać rozpoznawanie intencji, porządkowanie wymagań i proponowanie czynności odpowiadających poleceniu użytkownika. Samo przetworzenie tekstu nie jest jednak wystarczające do przeprowadzenia montażu. System musi również dysponować informacjami o zawartości obrazu, mowy i dźwięku, uwzględniać aktualny stan projektu oraz przekształcać wynik interpretacji w działania możliwe do jednoznacznego sprawdzenia i wykonania.

Zakres możliwości automatyzacji różni się jednak w zależności od rodzaju czynności montażowej. Część z nich ma charakter powtarzalny i może zostać opisana za pomocą jednoznacznych reguł, podczas gdy inne wymagają oceny znaczenia materiałów, zamierzonej narracji, rytmu oraz preferencji twórczych. Wspomaganie edycji wideo może zatem obejmować zarówno przygotowanie propozycji i planów działania, jak i wykonanie wybranych operacji po ich wcześniejszym sprawdzeniu i zatwierdzeniu. Istotne staje się przy tym określenie, które zadania można przekazać systemowi, a które decyzje powinny pozostać pod kontrolą użytkownika.

2.1. Cel, zakres i metodyka analizy

Analiza dotyczy sposobów wykorzystania technik informatycznych do wspomaganie i automatyzacji montażu wideo. Uwzględniono zarówno narzędzia wykonujące operacje na osi czasu, jak i techniki służące do pozyskiwania z materiałów źródłowych informacji potrzebnych do zaplanowania tych operacji. Pozwala to porównać rozwiązania automatyzujące pojedyncze czynności z systemami wspierającymi większą część procesu montażowego.

Zakres przeglądu obejmuje profesjonalne systemy **nieliniowej edycji wideo** (ang. *non-linear editing*, **NLE**), czyli programy umożliwiające swobodny dostęp do fragmentów materiału,

zmianę ich kolejności i parametrów na osi czasu bez modyfikowania plików źródłowych. Uwzględniono również wtyczki automatyzujące wybrane czynności, uproszczone edytory nastawione na szybkie przygotowanie treści, prototypy badawcze oraz środowiska programistyczne, w których sposób tworzenia filmu opisuje się za pomocą kodu. Do przeglądu włączono publicznie opisane rozwiązania dotyczące edycji istniejących materiałów, analizy ich zawartości, wykorzystania języka naturalnego w sterowaniu edycją lub planowania wieloetapowego. Wyłączono systemy służące wyłącznie do generowania gotowego wideo z tekstu. Wybrane rozwiązania porównano pod względem zakresu automatyzacji, sposobu interpretowania poleceń użytkownika i zawartości materiałów, środowiska działania, postaci uzyskiwanego wyniku oraz możliwości kontroli, sprawdzania poprawności, obsługi błędów i cofania zmian. Uwzględniono również model dostępu, potencjalny koszt korzystania z rozwiązania oraz stopień, w jakim pozwala ono użytkownikowi bez specjalistycznej wiedzy opisać oczekiwany rezultat.

W odniesieniu do systemów wykorzystujących LLM rozpatrzono, czy informacje pozyskane z obrazu i ścieżki dźwiękowej mogą stanowić wystarczający kontekst do zinterpretowania polecenia użytkownika i przygotowania zgodnego z nim planu operacji montażowych.

Podstawę przeglądu stanowiły publikacje naukowe, oficjalne dokumentacje techniczne oraz materiały udostępniane przez producentów analizowanych narzędzi. Ze względu na szybkie tempo rozwoju sztucznej inteligencji, zwłaszcza dużych modeli językowych, przegląd przedstawia stan publicznie udokumentowanych rozwiązań na dzień 28 sierpnia 2026 r. Późniejsze zmiany w możliwościach modeli oraz funkcjach analizowanych narzędzi mogą wpłynąć na aktualność przedstawionych ustaleń.

2.2. Proces edycji wideo i zakres jego automatyzacji

Proces montażowy rozpoczyna się od importu i uporządkowania nagrań, tak aby można było je sprawnie przeglądać i odnajdywać potrzebne ujęcia. Po selekcji materiału powstaje montaż wstępny, w którym ustala się kolejność scen i ocenia ogólny przebieg filmu. Dopiero na kolejnym etapie dopracowuje się długość ujęć, punkty cięć, rytm oraz relację obrazu z dźwiękiem [1], [2]. Gdy struktura filmu jest już ustalona, można przejść do wykończenia, obejmującego korektę obrazu i dźwięku, dodanie napisów, przejść, animacji, plansz tytułowych oraz elementów identyfikacji wizualnej. Własne szablony, logotypy i znaki wodne pomagają zachować rozpoznawalny styl oraz spójność pomiędzy kolejnymi publikacjami [2], [3].

Ten sam materiał źródłowy może posłużyć do przygotowania kilku wersji filmu dostosowanych do różnych form publikacji. Poszczególne warianty mogą różnić się długością, proporcjami obrazu oraz sposobem prowadzenia narracji. Dłuższy materiał może na przykład posłużyć do stworzenia krótkich klipów w formacie pionowym. Nie wystarczy jednak wyciąć dowolnego fragmentu i zmienić proporcji obrazu. Krótka wersja powinna zachować zrozumiały kontekst i stanowić samodzielną całość, dlatego może wymagać innego początku, szybszego tempa, ponownego kadrowania, dodania napisów oraz wyróżnienia najważniejszych informacji [4]. Ponownego opracowania może wymagać także ścieżka dźwiękowa, aby nie urwać wypowiedzi ani nie zaburzyć ciągłości muzycznej. Elementy tekstowe i graficzne należy ponadto rozmieścić tak, aby nie zostały zasłonięte przez interfejs platformy [5]. Przygotowanie takiego wariantu stanowi więc osobne zadanie montażowe, a nie jedynie kolejny eksport tego samego projektu.

Możliwości automatyzacji pojawiają się na różnych etapach procesu montażowego. Na początku mogą obejmować porządkowanie materiałów, wykrywanie zmian ujęć, transkrypcję mowy, oczyszczanie dźwięku i przygotowanie napisów [6]. Podczas budowania montażu

wstępnego transkrypcja pozwala szybciej odnajdywać potrzebne wypowiedzi, a jej edycja może powodować automatyczne przycinanie i rozmieszczanie powiązanych klipów na osi czasu [7]. Na etapie wykończenia automatyzacji mogą podlegać powtarzalne operacje związane ze stosowaniem szablonów, przygotowywaniem elementów graficznych oraz eksportowaniem materiału w kilku wariantach [6]. Oprogramowanie może więc wspierać montaż od pierwszego przeglądu nagrań aż do przygotowania wersji przeznaczonych do publikacji.

Zakres automatyzacji powinien wynikać z charakteru zadania, a nie jedynie z technicznej możliwości jego wykonania. To, że system potrafi utworzyć cięcie, nie oznacza jeszcze, że powinien samodzielnie decydować o jego miejscu. Pełniejsza automatyzacja dobrze odpowiada czynnościom powtarzalnym i łatwym do oceny, natomiast decyzje zależne od narracji, rytmu i preferencji twórczych lepiej wspierać przez przedstawianie propozycji możliwych do zmiany [8]. Zachowanie możliwości korekty, odrzucenia wyniku lub powrotu do pracy ręcznej pozwala użytkownikowi przejąć kontrolę, gdy rezultat nie odpowiada jego zamierzeniom [9]. Rezultatem wspomagania może być zarówno edytowalny montaż wstępny przeznaczony do dalszego opracowania, jak i gotowy materiał realizujący prosty, wcześniej określony scenariusz publikacji. Przygotowanie takich propozycji wymaga jednak wcześniejszego pozyskania wiarygodnych informacji z materiałów multimedialnych.

2.3. Metody analizy materiałów multimedialnych

Materiały wideo można traktować jako uporządkowaną w czasie sekwencję obrazów połączoną ze ścieżką dźwiękową. Algorytmiczne metody przetwarzania obrazu i dźwięku pozwalają pozyskać informacje o materiale, które mogą następnie stanowić kontekst dla dużych modeli językowych. Za ich pomocą można między innymi wykrywać granice ujęć, wybierać klatki reprezentatywne, oceniać ich ostrość, rozpoznawać tekst widoczny w obrazie oraz wyznaczać ciszę, energię i rytm ścieżki dźwiękowej. Powiązanie wyników ze znacznikami czasu umożliwia wskazanie konkretnych fragmentów materiału i wykorzystanie ich przez kolejne elementy systemu. Generowanie opisów zawartości przez modele multimodalne oraz interpretowanie poleceń użytkownika przez duże modele językowe omówiono w dalszej części pracy.

2.3.1 Odczyt i przygotowanie materiału multimedialnego

Z perspektywy użytkownika materiał wideo jest pojedynczym plikiem, na przykład plikiem MP4. Wewnątrz może on jednak zawierać kilka oddzielnych strumieni danych: obraz, dźwięk, napisy oraz **metadane**, czyli informacje techniczne potrzebne do ich odtworzenia. Strumienie te są połączone wspólną osią czasu, dzięki czemu odpowiednie klatki obrazu, fragmenty dźwięku i napisy mogą zostać odtworzone we właściwym momencie. Kontener można więc traktować jako opakowanie, które przechowuje zakodowane dane i opisuje ich rozmieszczenie. Samo rozszerzenie pliku nie określa przy tym sposobu kompresji obrazu lub dźwięku. Określa go **kodek**, czyli mechanizm służący do kodowania i odkodowywania danych. Dlatego dwa pliki MP4 mogą zawierać dane zakodowane w różny sposób i wymagać innych dekodatorów. Kontener MP4 przechowuje dane w hierarchicznie uporządkowanych blokach nazywanych **atomami**. Atom **moov** zawiera opis strumieni obrazu i dźwięku oraz informacje potrzebne do odnalezienia ich zawartości w pliku. Zakodowane dane tych strumieni są natomiast przechowywane w atomie typu **mdat** (ang. *media data*) [10].

Odczyt materiału rozpoczyna się od rozpoznania kontenera i znajdujących się w nim strumieni. Program ustala między innymi rodzaj kodeka, rozdzielczość obrazu, liczbę klatek na sekundę, liczbę kanałów dźwięku i częstotliwość próbkowania. Jeżeli plik zawiera kilka ścieżek dźwiękowych lub napisów, należy również wybrać te, które mają zostać poddane analizie. Oddzielenie poszczególnych strumieni nazywa się **demultipleksacją**. Następnie dekodator

przekształca skompresowany strumień obrazu w kolejne klatki, a strumień dźwięku w ciąg próbek opisujących zmiany sygnału w czasie. Dopiero taka postać danych może zostać bezpośrednio wykorzystana przez algorytmy analizujące obraz i dźwięk.

Przed rozpoczęciem analizy dane mogą wymagać ujednolicenia. Klatki można przeskalować do ustalonej rozdzielczości oraz przekształcić do oczekiwanej przestrzeni barw i formatu pikseli, czyli przyjętego sposobu opisu kolorów i zapisu składowych pojedynczego piksela. W przypadku dźwięku można ujednolicić liczbę kanałów, częstotliwość próbkowania i sposób zapisu próbek. Operacje te nie muszą zmieniać pliku źródłowego, ponieważ przygotowana reprezentacja może istnieć tylko na czas analizy. Dla każdej klatki i każdego fragmentu dźwięku należy zachować znacznik czasu określający ich położenie w nagraniu. Pozwala to powiązać wykryte granice ujęć, wybrane klatki reprezentatywne i wyniki analizy dźwięku z odpowiednimi fragmentami materiału oraz połączyć uzyskane informacje we wspólnym opisie. Taki opis może później stanowić kontekst dla modułu interpretującego polecenie użytkownika i przygotowującego plan montażu.

2.3.2 Wykrywanie granic ujęć

Podział materiału na ujęcia jest kolejnym etapem analizy po odkodowaniu i przygotowaniu strumienia obrazu. Wyznaczenie początku i końca każdego ujęcia umożliwia przypisanie do niego informacji o zawartości oraz późniejsze wykorzystanie odpowiadającego mu przedziału czasowego podczas wyboru fragmentów do montażu. W niniejszej pracy **technikami deterministycznymi** nazywa się metody przetwarzania danych oparte na ustalonych regułach, które dla tych samych danych wejściowych i parametrów zwracają ten sam wynik. W przypadku wykrywania granic ujęć techniki te porównują cechy kolejnych klatek i wskazują granicę po spełnieniu określonego warunku, najczęściej po przekroczeniu ustalonej wartości progowej. Do takich technik należą:

- **Porównywanie wartości pikseli** – polega na zestawieniu wartości odpowiadających sobie pikseli dwóch kolejnych klatek. Dla każdego piksela obliczana jest bezwzględna różnica jasności lub składowych koloru, a otrzymane wartości są łączone w jeden wynik, na przykład przez obliczenie ich średniej. Jeżeli wynik przekroczy ustalony próg, klatka może zostać uznana za początek nowego ujęcia. Metoda jest prosta obliczeniowo, ale może reagować również na szybki ruch kamery, nagłą zmianę oświetlenia lub ruch dużego obiektu w obrębie tego samego ujęcia;
- **Porównywanie składowych przestrzeni HSV** – przestrzeń barw HSV (ang. *hue, saturation, value*) opisuje kolor za pomocą odcienia, nasycenia i wartości odpowiadającej jego jasności. Po przekształceniu klatek do tej reprezentacji różnice poszczególnych składowych oblicza się oddzielnie. Mogą one otrzymać określone wagi, a następnie zostać połączone w jeden wynik zmiany obrazu;
- **Porównywanie histogramów** – histogram obrazu jest liczbowym opisem rozkładu barw lub poziomów jasności w klatce. W celu jego utworzenia zakres możliwych wartości dzieli się na przedziały, a następnie zlicza się piksele należące do każdego z nich. Przed porównaniem wartości histogramów są normalizowane, czyli przeskalowywane tak, aby na wynik nie wpływała całkowita liczba pikseli. Biblioteka OpenCV umożliwia porównanie histogramów między innymi za pomocą korelacji oraz odległości Bhattacharyyi [11]. W pierwszym przypadku spadek wyniku oznacza mniejsze podobieństwo rozkładów, natomiast w drugim o większej różnicy świadczy wzrost wyniku. Histogram przechowuje informacje o liczbie pikseli o określonych wartościach, ale nie o ich położeniu. Z tego powodu klatki przedstawiające inne obiekty lub kompozycje mogą zostać uznane za podobne, jeżeli mają zbliżoną kolorystykę;

- **Podobieństwo strukturalne SSIM** – wskaźnik podobieństwa strukturalnego **SSIM** (ang. *structural similarity index measure*) służy do porównywania obrazów z uwzględnieniem ich jasności, kontrastu i lokalnej struktury. Dla kolejnych klatek obliczana jest wartość podobieństwa, a jej spadek poniżej ustalonego progu może wskazywać granicę ujęcia. Funkcja `structural_similarity` biblioteki `scikit-image` oblicza średnią wartość SSIM dla dwóch obrazów o jednakowych wymiarach [12]. Nie jest ona gotowym detektorem cięć, dlatego sposób przygotowania klatek i wartość progową należy określić osobno;
- **Haszowanie percepcyjne** – polega na przekształceniu każdej klatki w krótki, najczęściej binarny skrót opisujący jej ogólną strukturę wizualną. W odróżnieniu od skrótu kryptograficznego hasz percepcyjny powinien przyjmować podobne wartości dla obrazów wyglądających podobnie. Skróty kolejnych klatek można porównać za pomocą odległości Hamminga, czyli liczby pozycji, na których zawierają one różne wartości. Jeżeli odległość przekroczy ustalony próg, klatki mogą zostać przypisane do różnych ujęć. Uproszczona reprezentacja ogranicza koszt porównania, ale może pomijać mniej wyraźne zmiany obrazu.

Opisane techniki wykorzystują różne sposoby reprezentowania zmiany zachodzącej pomiędzy klatkami, dlatego nie są jednakowo skuteczne dla każdego materiału. Dobór metody oraz wartości progowej powinien uwzględniać między innymi dynamikę nagrania, zmiany oświetlenia i rodzaj występujących przejść. Metody porównujące sąsiednie klatki są przeznaczone przede wszystkim do wykrywania nagłych cięć, natomiast przejścia stopniowe wymagają analizy zmian zachodzących w dłuższej sekwencji klatek.

2.3.3 Wybór i ocena klatek reprezentatywnych

Materiał wideo składa się z kolejnych klatek, z których wiele w obrębie jednego ujęcia przedstawia zbliżoną zawartość. W pracy przez klatkę reprezentatywną rozumie się pojedynczy obraz wybrany jako wizualna reprezentacja ujęcia lub jego części. Klatka taka może zostać wykorzystana do wstępnego określenia zawartości materiału bez konieczności analizowania wszystkich obrazów tworzących nagranie. Nie oznacza to, że każde ujęcie musi być opisane przez jeden obraz. Długie lub zmienne fragmenty mogą wymagać wyboru kilku klatek reprezentatywnych.

Wybór takich klatek za pomocą metod deterministycznych opiera się na ich mierzalnych właściwościach, takich jak położenie na osi czasu, ostrość lub różnica względem innych obrazów. Metody te nie rozpoznają samodzielnie znaczenia przedstawionej treści.

Kandydaci mogą być wybierani w stałych odstępach czasu, ze środka ujęcia albo w miejscach, w których występuje większa zmiana obrazu. Następnie można odrzucić klatki niespełniające wymagań jakościowych. Wariancja odpowiedzi operatora Laplace’a pozwala oszacować ostrość na podstawie obecności wyraźnych krawędzi [13]. Z kolei porównanie pozostałych klatek umożliwia ograniczenie liczby obrazów o zbliżonej zawartości.

Ostateczny zestaw powinien stanowić kompromis między ilością zachowanych informacji a kosztem dalszego przetwarzania. Dla krótkiego i statycznego ujęcia może wystarczyć jedna klatka, natomiast fragment zawierający ruch albo zmiany prezentowanej treści może wymagać kilku próbek. Ocena techniczna wspomaga zatem wybór ale nie powinna być utożsamiana z oceną znaczenia klatki.

2.3.4 Rozpoznawanie tekstu w obrazie

Optyczne rozpoznawanie znaków (ang. *optical character recognition*, **OCR**) jest procesem przekształcania tekstu widocznego w obrazie do postaci znaków możliwych do zapisania i dalszego przetwarzania przez program. Technika ta jest powszechnie wykorzystywana podczas

digitalizacji dokumentów papierowych. Po zeskanowaniu stron umożliwia utworzenie tekstu, który można przeszukiwać, kopiować, indeksować i poddawać dalszej analizie. W materiałach wideo OCR może służyć do odczytywania plansz tytułowych, prezentacji, tablic, elementów interfejsu oraz napisów trwale zapisanych w obrazie.

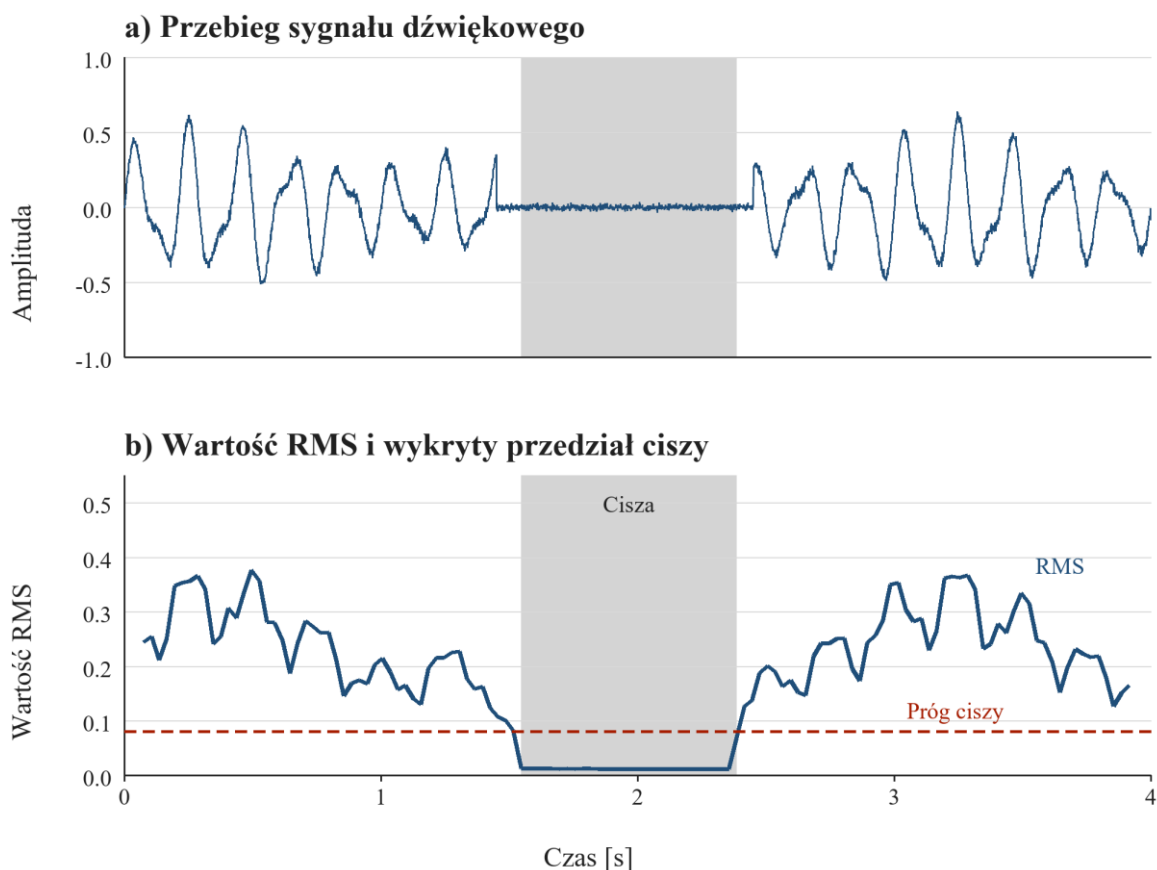
W przypadku materiału wideo OCR nie musi być wykonywany dla każdej klatki. Analizie można poddać klatki reprezentatywne wybrane dla poszczególnych ujęć, ograniczając w ten sposób ilość przetwarzanych danych. Takie rozwiązanie może być wystarczające dla tekstu pozostającego na ekranie dłuższy czas. Krótkotrwały albo poruszający się napis może jednak wystąpić pomiędzy wybranymi klatkami. Jego wykrycie wymaga wówczas częstszego próbkowania obrazu lub przeanalizowania większej liczby klatek należących do danego ujęcia.

Proces rozpoznawania obejmuje wykrycie obszarów tekstowych i odczytanie znajdujących się w nich znaków. Obraz może wcześniej zostać przeskalowany, przekształcony do skali szarości oraz poddany korekcji kontrastu, usuwaniu szumu, binaryzacji oddzielającej znaki od tła i korekcji perspektywy. Wynikiem jest tekst powiązany z położeniem w klatce, czasem wystąpienia i pewnością rozpoznania. Powtarzające się wyniki z kolejnych klatek można połączyć w jeden przedział czasowy. Na jakość rozpoznania wpływają przede wszystkim rozmycie, słaby kontrast, nietypowy krój pisma i częściowe zasłonięcie znaków. Metoda ta odczytuje tekst ale nie określa jego znaczenia ani przydatności dla montażu.

2.3.5 Analiza sygnału dźwiękowego

Sygnał dźwiękowy może być analizowany na podstawie jego mierzalnych właściwości, takich jak amplituda, poziom energii oraz rozmieszczenie zmian zachodzących w czasie. Po odkodowaniu ścieżki dźwiękowej dane mają postać ciągu próbek, których wartości opisują chwilową amplitudę sygnału. Ponieważ liczba próbek jest duża, łączy się je w krótkie okna czasowe. Dla każdego okna można następnie obliczyć wartości opisujące dany fragment nagrania. Krótsze okna pozwalają dokładniej umiejscowić zmiany w czasie, natomiast dłuższe ograniczają wpływ pojedynczych i krótkotrwałych wahań sygnału.

Jedną z podstawowych miar jest wartość skuteczna **RMS** (ang. *root mean square*), wyznaczana jako pierwiastek ze średniej kwadratów wartości próbek [14]. Kolejne wyniki tworzą krzywą przedstawiającą zmiany poziomu sygnału w czasie. RMS nie opisuje bezpośrednio głośności odbieranej przez człowieka, ale umożliwia porównywanie poziomu kolejnych fragmentów tego samego nagrania. Jeżeli wartość pozostaje poniżej ustalonego progu przez określony czas, dany przedział może zostać oznaczony jako cisza. Cisza nie oznacza przy tym całkowitego braku sygnału, ponieważ nagranie może zawierać szum tła. Z tego powodu zarówno próg, jak i minimalny czas trwania ciszy powinny być dobierane do właściwości materiału. Rysunek 1 przedstawia przykładowy przebieg sygnału, odpowiadającą mu krzywą RMS oraz przedział zaklasyfikowany jako cisza po spadku wartości poniżej przyjętego progu.



Rysunek 1. Wykrywanie przedziału ciszy na podstawie wartości RMS. Źródło: opracowanie własne na podstawie [14].

Zmiany poziomu lub częstotliwości sygnału mogą wskazywać momenty rozpoczęcia nowych zdarzeń dźwiękowych. Początkiem może być między innymi uderzenie instrumentu, wejście nowego dźwięku lub nagły wzrost intensywności nagrania. W materiale muzycznym rozmieszczenie takich zdarzeń pozwala oszacować tempo wyrażone w uderzeniach na minutę, określanych skrótem **BPM** (ang. *beats per minute*), oraz wskazać położenie kolejnych uderzeń rytmicznych. Przykłady obliczania wartości RMS oraz analizy tempa i rytmu przedstawiono w artykule poświęconym ekstrakcji i interpretacji cech sygnału dźwiękowego [14].

Uzyskane przedziały ciszy, krzywa poziomu sygnału oraz znaczniki rytmiczne mogą wspierać wybór fragmentów i miejsc wykonania cięć. Gwałtowny wzrost poziomu może zostać oznaczony jako kandydat na akcent muzyczny, ale nie przesądza o jego znaczeniu dla montażu. Metody deterministyczne pozwalają zatem określić, kiedy w sygnale występuje określona zmiana, lecz nie wyjaśniają jej znaczenia. Rozpoznanie treści wypowiedzi wymaga osobnego etapu transkrypcji.

2.3.6 Reprezentacja wyników analizy

Samo zastosowanie opisanych technik deterministycznych nie wystarcza do pełnej automatyzacji procesu edycji wideo. Pozwalają one wykrywać mierzalne właściwości materiału, ale nie interpretują celu użytkownika ani nie podejmują decyzji dotyczących znaczenia i kolejności poszczególnych fragmentów. Mogą jednak zautomatyzować wstępną analizę, uporządkować materiał i ograniczyć zakres danych wymagających dalszego przetwarzania.

Bezpośrednie przekazanie całego materiału wideo do dalszej analizy wiąże się z przetwarzaniem dużej liczby klatek i próbek dźwięku. Podział nagrania na ujęcia, wybór klatek reprezentatywnych, odczytanie widocznego tekstu oraz wskazanie zmian sygnału dźwiękowego pozwalają przekształcić surowy materiał w mniejszy i uporządkowany zbiór informacji. Wynikiem nie jest gotowy opis znaczenia nagrania, lecz dane dotyczące jego struktury, zawartości i zmian występujących na osi czasu.

Dane przeznaczone dla dalszej analizy mogą zawierać identyfikatory materiałów, przedziały czasowe ujęć, odwołania do wybranych klatek, rozpoznany tekst, przedziały czasowe ciszy, tempo i znaczniki rytmiczne. Powiązanie tych informacji ze wspólną osią czasu pozwala przypisać je do konkretnych fragmentów nagrania. Tak przygotowany zbiór danych może następnie stanowić kontekst dla modułów interpretujących polecenie użytkownika i przygotowujących plan operacji montażowych.

Uzyskanych miar nie należy traktować jako jednoznacznych ocen treści. Granica ujęcia oznacza zmianę obrazu, ale nie musi wyznaczać nowej sceny narracyjnej, natomiast wysoka wartość RMS wskazuje na wyższy poziom sygnału, lecz nie przesądza o znaczeniu danego fragmentu. Zastosowanie metod deterministycznych może jednak poprawić wydajność całego procesu, ponieważ ogranicza zakres dalszej analizy i ilość danych przekazywanych do kolejnych modułów. Podstawowe właściwości obrazu i dźwięku zostają dzięki temu wyznaczone przed etapem interpretacji, którego metody opisano w rozdziale 2.4.

2.4. Modele językowe we wspomaganiu montażu wideo

Automatyzację montażu można rozpatrywać jako połączenie trzech etapów: analizy materiału, interpretacji celu użytkownika oraz wykonania operacji edycji. Pierwszy etap dostarcza informacji o obrazie i dźwięku, natomiast ostatni odpowiada za wprowadzenie zmian w projekcie montażowym. Pomiedzy nimi potrzebny jest mechanizm, który powiąże polecenie użytkownika z dostępnymi danymi i określi działania prowadzące do oczekiwanego rezultatu.

Polecenie sformułowane w języku naturalnym zazwyczaj opisuje oczekiwany efekt, ale nie musi zawierać nazw funkcji programu, dokładnych przedziałów czasowych ani parametrów poszczególnych operacji. Jego wykonanie wymaga zestawienia intencji użytkownika z informacjami o zawartości materiałów, ich położeniu na osi czasu oraz funkcjach udostępnianych przez środowisko edycyjne. Wynikiem tego etapu powinien być jednoznaczny plan, który można sprawdzić przed wprowadzeniem zmian w projekcie.

Rolę mechanizmu pośredniczącego mogą pełnić duże modele językowe. Nie zastępują one programu do edycji wideo, lecz mogą wspomagać interpretowanie poleceń, dobieranie dostępnych funkcji i ustalanie kolejności ich użycia. Przygotowany przez model wynik może następnie zostać przekazany do narzędzi odpowiedzialnych za sprawdzenie parametrów i wykonanie operacji. Takie rozdzielenie pozwala wykorzystać elastyczność języka naturalnego, zachowując kontrolę nad zmianami wprowadzanymi w materiale. Ocena przydatności takiego rozwiązania wymaga jednak uwzględnienia sposobu działania modeli językowych oraz ograniczeń wpływających na przygotowywane przez nie wyniki.

2.4.1 Podstawy działania dużych modeli językowych

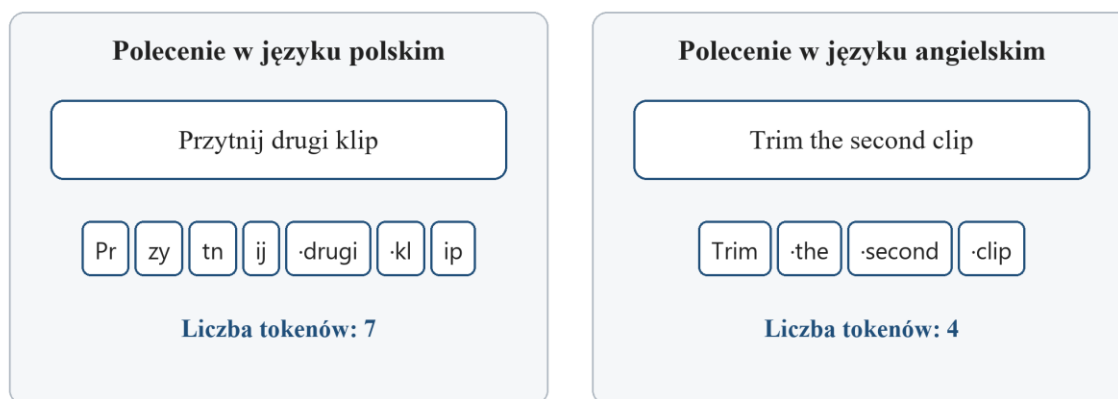
Modele językowe generują odpowiedź etapami, przewidując kolejny token na podstawie polecenia, przekazanego kontekstu oraz wcześniej utworzonej części wypowiedzi. **Token** jest podstawową jednostką tekstu przetwarzaną przez model i może odpowiadać całemu wyrazowi, jego fragmentowi, liczbie albo znakowi interpunkcyjnemu. Na każdym etapie model określa prawdopodobieństwo wystąpienia dostępnych tokenów, a następnie wybiera jeden z nich. Proces

ten jest powtarzany aż do zakończenia odpowiedzi. Model nie przygotowuje więc całego wyniku z góry, lecz rozwija go krok po kroku [15].

Z takim sposobem generowania wiąże się **niedeterminizm**. System deterministyczny dla tych samych danych wejściowych i tego samego stanu powinien za każdym razem zwrócić identyczny wynik. W przypadku generatywnego użycia modeli językowych warunek ten zazwyczaj nie jest spełniony. To samo polecenie może prowadzić do odpowiedzi różniących się doбором słów, strukturą lub proponowanym sposobem wykonania zadania. OpenAI opisuje tę właściwość wprost słowami: „*model behavior is inherently non-deterministic*”, czyli zachowanie modelu jest z natury niedeterministyczne [16].

Niedeterminizm nie oznacza jednak, że odpowiedź jest całkowicie przypadkowa. Polecenie, kontekst oraz parametry generowania wpływają na prawdopodobieństwo wyboru poszczególnych tokenów. Jednym z takich parametrów jest temperatura. Jej niższa wartość zwiększa przewidywalność odpowiedzi, natomiast wyższa sprzyja uzyskiwaniu bardziej zróżnicowanych wyników. Dokumentacja Google Cloud wskazuje, że nawet po zastosowaniu ustawień ograniczających losowość pełna powtarzalność odpowiedzi nie jest gwarantowana [17].

Przed rozpoczęciem przetwarzania tekst jest dzielony na tokeny. Każdemu tokenowi przypisywany jest identyfikator liczbowy, który następnie zostaje przekształcony w reprezentację możliwą do przetworzenia przez model. Sposób podziału zależy od użytego tokenizera oraz języka tekstu, dlatego liczba tokenów nie musi odpowiadać liczbie słów ani znaków. Tokeny tworzą również jednostkę, w której określana jest wielkość okna kontekstowego modelu. W jego granicach muszą zmieścić się polecenie, dodatkowe informacje oraz generowana odpowiedź [15]. Rysunek 2 przedstawia wynik tokenizacji równoważnych poleceń w języku polskim i angielskim przy użyciu tokenizera o200k_base.



Tokenizer: o200k_base. Symbol „-” oznacza spację poprzedzającą token.

Rysunek 2. Porównanie tokenizacji polecenia w języku polskim i angielskim. Źródło: opracowanie własne.

Podstawą większości współczesnych modeli językowych jest architektura Transformer. Jej istotnym elementem jest mechanizm uwagi, który pozwala określać zależności pomiędzy tokenami znajdującymi się w różnych miejscach przetwarzanej sekwencji. Dzięki temu znaczenie danego tokenu może być rozpatrywane z uwzględnieniem otaczającego go tekstu. Informacja o kolejności elementów jest zachowywana za pomocą reprezentacji pozycyjnych [15].

Model nie otrzymuje gotowych reguł opisujących znaczenie słów ani sposób budowania zdań. Przed rozpoczęciem uczenia tekst jest dzielony na tokeny, którym przypisywane są identyfikatory liczbowe. Następnie model wielokrotnie próbuje przewidzieć dalszy ciąg tekstu. Jeżeli otrzyma na przykład fragment „*materiał został podzielony na*”, powinien przypisać wysokie prawdopodobieństwo tokenom takim jak „*ujęcia*” lub „*fragmenty*”, a niskie tokenom niezwiązanym z tym kontekstem. Podczas uczenia przewidywanie modelu jest porównywane z rzeczywistym tokenem występującym w materiale treningowym. Każdy błąd prowadzi do niewielkiej zmiany parametrów modelu. Po przeanalizowaniu ogromnej liczby przykładów parametry zaczynają odwzorowywać zależności pomiędzy tokenami, ich znaczeniem oraz kontekstem, w którym występują. Gotowy model wykorzystuje te zależności do generowania odpowiedzi na nowe polecenia [15].

Rozwinięciem modeli pracujących wyłącznie na tekście są modele multimodalne. Mogą one przetwarzać i łączyć informacje pochodzące z różnych rodzajów danych, takich jak tekst, obraz, dźwięk i wideo. Każdy rodzaj danych wymaga odpowiedniej reprezentacji, która umożliwia zestawienie informacji pochodzących z różnych źródeł. W przypadku materiału filmowego znaczenie mają również zmiany zachodzące w czasie oraz zależności pomiędzy obrazem i ścieżką dźwiękową [18].

Niedeterminizm modeli językowych może być zarówno ich ograniczeniem, jak i źródłem możliwości niedostępnych dla programów opartych wyłącznie na stałych regułach. Microsoft wskazuje, że rozwiązania wykorzystujące sztuczną inteligencję wprowadzają zachowanie niedeterministyczne w zadaniach, dla których ustalenie jednego stałego wyniku jest niepraktyczne [19]. Montaż materiału filmowego należy do tej grupy problemów, ponieważ dla tego samego materiału i celu może istnieć wiele poprawnych sposobów wyboru ujęć, określenia ich kolejności oraz zbudowania narracji. W niniejszej pracy niedeterminizm jest zatem cechą pożądaną podczas interpretowania polecenia i przygotowywania koncepcji montażu. Oczekiwana jest zmienność tworzonych propozycji, a nie nieprzewidywalność późniejszego wykonywania operacji.

2.4.2 Interpretacja poleceń użytkownika

Interpretacja polecenia polega na przekształceniu swobodnej wypowiedzi w opis jej znaczenia. Obejmuje to rozpoznanie celu, wskazanych obiektów, ograniczeń oraz preferencji. Trudność tego procesu wynika z tego, że język naturalny nie wymaga stosowania jednoznacznej składni. Ten sam zamiar może zostać wyrażony na wiele sposobów, a część informacji może wynikać z kontekstu zamiast z samego polecenia.

Wymagania powinny zostać rozdzielone na warunki obowiązkowe oraz określenia dopuszczające kilka poprawnych interpretacji. Pierwsza grupa obejmuje informacje możliwe do jednoznacznego sprawdzenia, natomiast druga może opisywać styl, charakter lub ogólny kierunek oczekiwanego rezultatu. Model powinien również rozpoznawać informacje brakujące i oceniać, czy mogą zostać wywnioskowane z kontekstu, czy wymagają doprecyzowania.

Na interpretację wpływa także język polecenia. Rozbudowana fleksja języka polskiego pozwala zawierać część informacji gramatycznych w formach wyrazów, dzięki czemu niektóre relacje mogą zostać wyrażone bez dodatkowych objaśnień. W badaniu OneRuler [20] polski osiągnął najwyższą średnią skuteczność wśród 26 języków w zadaniach wyszukiwania informacji w kontekstach liczących 64 i 128 tysięcy tokenów. Uzyskany wynik wyniósł 88%, wobec 83,9% dla języka angielskiego. Rezultat ten pokazuje przydatność języka polskiego w wybranych zadaniach wykorzystujących bardzo długi kontekst, ale nie dowodzi jego ogólnej przewagi ani nie wyjaśnia wpływu składni na otrzymane wyniki.

Angielski pozostaje zazwyczaj bardziej ekonomiczny pod względem liczby tokenów. OpenAI wskazuje, że teksty nieangielskie często mają wyższy stosunek liczby tokenów do liczby

znaków, co może zwiększać koszt pojedynczego wywołania modelu [21]. Całkowity koszt rozwiązania zależy jednak również od trafności odpowiedzi oraz liczby wymaganych poprawek. Język zapewniający dokładniejszą interpretację w danym zadaniu może więc teoretycznie ograniczyć koszt kolejnych wywołań, nawet jeżeli pojedyncze polecenie wymaga większej liczby tokenów.

Zdolność do rozpoznawania intencji, kontekstu i określeń dopuszczających różne interpretacje odpowiada charakterowi montażu wideo. Oczekiwany rezultat może być opisany za pomocą pojęć semantycznych i stylistycznych, bez wskazywania technicznego sposobu wykonania poszczególnych operacji.

2.4.3 Ustrukturyzowane odpowiedzi i wywoływanie narzędzi

Rezultat interpretacji polecenia, omówionej w podrozdziale 2.4.2, musi zostać zapisany w formie możliwej do automatycznego przetworzenia. Potrzebna jest zatem reprezentacja pośrednia stanowiąca kontrakt pomiędzy modelem językowym a kodem odpowiedzialnym za dalszą realizację zadania. Powinna ona jednoznacznie określać rodzaj planowanego działania, jego parametry oraz zależności względem pozostałych operacji.

Mechanizm odpowiedzi ustrukturyzowanych (ang. *Structured Outputs*) [16] umożliwia zdefiniowanie oczekiwanej postaci odpowiedzi za pomocą schematu JSON (ang. *JavaScript Object Notation*). Schemat określa wymagane pola, ich typy, dopuszczalne wartości oraz możliwe warianty danych. Pozwala to ograniczyć wynik modelu do struktury, którą pozostałe elementy systemu mogą odczytać bez analizowania swobodnego tekstu.

Zgodność ze schematem potwierdza jednak wyłącznie poprawność formalną odpowiedzi. Należy odróżnić ją od poprawności dziedzinowej oraz zgodności z aktualnym stanem systemu. Prawidłowo zbudowany obiekt może nadal zawierać odwołanie do nieistniejącego zasobu, wartość wykraczającą poza dostępny zakres albo operację, której nie można wykonać w danym stanie. Z tego względu odpowiedź modelu musi zostać poddana dodatkowej walidacji przed jej wykorzystaniem.

Uzupełnieniem odpowiedzi ustrukturyzowanych jest mechanizm wywoływania narzędzi. Model otrzymuje zamknięty katalog dostępnych funkcji wraz z opisem ich argumentów, a następnie wskazuje wybraną funkcję i przygotowuje wartości jej parametrów. Nie wykonuje jednak operacji samodzielnie. Zwraca jedynie propozycję wywołania, którą kod może sprawdzić, zaakceptować albo odrzucić. Kolejne wywołania mogą zostać połączone w typowany plan i przedstawione użytkownikowi przed przekazaniem do warstwy wykonawczej. Rysunek 3 przedstawia uproszczony przebieg prowadzący od polecenia użytkownika do odpowiedzi ustrukturyzowanej opisującej wywołanie narzędzia.

System wieloagentowy składa się z kilku agentów realizujących osobne zadania. Poszczególni agenci mogą wykorzystywać ten sam model bazowy, lecz otrzymywać odmienne instrukcje, narzędzia i zakres kontekstu. Najczęściej stosowany jest układ koordynator–wykonawcy. Agent nadrzędny dzieli zadanie, przekazuje jego części wyspecjalizowanym agentom, oczekuje na wyniki i przygotowuje wspólną odpowiedź. Zadania niezależne mogą być wykonywane równolegle, natomiast zadania zależne są realizowane sekwencyjnie. Alternatywnym rozwiązaniem jest przekazanie całego zadania innemu agentowi, którego specjalizacja lepiej odpowiada rozpoznaniu problemowi.

Szczegóły infrastruktury dużych dostawców pozostają częściowo niejawne, ale publiczne interfejsy wskazują na podobny model działania. Przykładem jest mechanizm wieloagentowy (ang. *multi-agent*) udostępniany przez OpenAI, w którym agent główny może tworzyć agentów podrzędnych, przekazywać im zadania i wiadomości, oczekiwać na rezultaty, a następnie łączyć otrzymane odpowiedzi. Każdy agent utrzymuje własny kontekst, natomiast komunikacja i synchronizacja są obsługiwane przez warstwę orkiestracji [23]. System wieloagentowy nie oznacza zatem podziału jednego modelu na kilka części, lecz wykonanie wielu wywołań modelu połączonych mechanizmem wymiany wyników.

W montażu wideo agent koordynujący może rozpoznać zakres zadania i przekazać jego części wyspecjalizowanym agentom. Poszczególne role mogą odpowiadać między innymi za interpretację celu użytkownika, ocenę zawartości materiałów, wybór odpowiednich fragmentów, ustalenie ich kolejności lub przygotowanie koncepcji narracyjnej. Koordynator nie musi uruchamiać wszystkich agentów dla każdego polecenia, lecz może dobierać ich zależnie od charakteru zadania, przekazywać im potrzebny kontekst, a następnie zestawiać otrzymane wyniki. Specjalizacja wynika z odmiennych instrukcji, dostępnych danych i narzędzi, dlatego nie wymaga stosowania osobnego modelu dla każdej roli. Przygotowany w ten sposób wspólny plan może zostać zweryfikowany i wykonany przez deterministyczne moduły, bez przekazywania agentom bezpośredniej kontroli nad środowiskiem edycyjnym.

2.4.5 Ograniczenia modeli językowych i mechanizmy kontroli

Jednym z ograniczeń modeli językowych jest skończona długość kontekstu. Obejmuje on nie tylko polecenie użytkownika, lecz także instrukcje systemowe, opisy narzędzi, historię interakcji oraz wyniki wcześniejszych etapów. W przypadku obszernych materiałów transkrypcje, opisy obrazu i pozostałe informacje mogą szybko wypełnić dostępne okno kontekstowe. Przekroczenie limitu wymaga pominięcia części danych, ich podziału albo streszczenia. Sama możliwość przesłania długiego kontekstu nie gwarantuje też jego skutecznego wykorzystania. Badania wykazały, że trafność odpowiedzi może zależeć od położenia istotnych informacji, a dane umieszczone w środkowej części długiego kontekstu bywają wykorzystywane mniej skutecznie [24].

Rozmiar okna kontekstowego zależy od konkretnego modelu i jego wersji. Według stanu na 28 sierpnia 2026 r. okno kontekstowe modelu GPT-5.6 Sol [25] w wersji `gpt-5.6-sol` wynosi 1 050 000 tokenów. Gemini 3.7 Flash [26] w wersji `gemini-3.7-flash` przyjmuje do 1 048 576 tokenów wejściowych, natomiast Claude Fable 5 [27] w wersji `claude-fable-5` udostępnia okno kontekstowe obejmujące 1 000 000 tokenów.

Drugim ograniczeniem jest koszt korzystania z modeli udostępnianych za pośrednictwem zewnętrznych usług. Rozliczeniu podlegają przede wszystkim tokeny wejściowe i wyjściowe, natomiast niektóre narzędzia mogą generować dodatkowe opłaty [28]. W systemach wieloetapowych koszt zwiększają kolejne wywołania, ponowienia oraz przekazywanie kontekstu pomiędzy agentami. Zwiększanie liczby etapów lub ilości danych nie musi przy tym prowadzić do proporcjonalnej poprawy rezultatu. Automatyzacja przenosi więc część nakładu z pracy człowieka na koszt wywołań modeli, czas oczekiwania oraz kontrolę otrzymanych wyników. Jej

opłacalność zależy od wielkości materiału, liczby wywołań, zastosowanego modelu i wartości czasu zaoszczędzonego użytkownikowi.

Koszty i wykorzystanie kontekstu można ograniczać przez ustalenie budżetu tokenów, maksymalnej liczby iteracji i warunków zakończenia zadania. Znaczenie ma również przekazywanie tylko danych potrzebnych w danym etapie, używanie mniejszych modeli do prostszych zadań oraz ponowne wykorzystywanie niezmiennych fragmentów kontekstu. Mechanizmy te nie usuwają ograniczeń modeli, ale pozwalają kontrolować zakres i cenę ich zastosowania.

Poza długością kontekstu i kosztem istotnym ograniczeniem jest brak gwarancji poprawności merytorycznej odpowiedzi modelu. Wynik może być zgodny z wymaganym schematem danych, a mimo to wskazywać nieistniejący materiał, przedział czasowy wykraczający poza długość nagrania lub operację nieobsługiwaną przez program. Generowanie ma ponadto charakter niedeterministyczny, dlatego ponowne przetworzenie tego samego polecenia może prowadzić do odmiennej propozycji. Jeżeli model jest udostępniany jako usługa zewnętrzna, jego użycie może wiązać się z przekazaniem poza lokalne środowisko części danych wejściowych, na przykład polecenia użytkownika, opisów obrazu lub fragmentów transkrypcji. Zakres przesyłanych informacji powinien być zatem ograniczony do danych potrzebnych na danym etapie.

W przypadku wspomagania montażu wideo ograniczenia te uzasadniają połączenie modeli językowych z metodami deterministycznymi. Algorytmy lokalne mogą odczytywać metadane, wykrywać granice ujęć, mierzyć czas trwania i cechy sygnału oraz wstępnie ograniczać zbiór analizowanych fragmentów bez przysyłania danych do zewnętrznego interfejsu modelu. Do modelu powinny trafiać dopiero wybrane i ustrukturyzowane informacje oraz zadania wymagające interpretacji znaczenia, oceny zgodności z intencją użytkownika lub przygotowania koncepcji montażu. Takie rozdzielenie pozwala ograniczyć użycie modelu do zadań, w których metody oparte na jednoznacznych regułach nie zapewniają oczekiwanego rezultatu. Odpowiedź modelu nie powinna być jednak przekazywana bezpośrednio do warstwy wykonawczej. Należy odwzorować ją na zamknięty zbiór obsługiwanych typów poleceń, deterministycznie sprawdzić ich parametry, wskazane materiały i zależności, a następnie przedstawić cały plan użytkownikowi do zatwierdzenia. Dopiero zatwierdzony plan może zostać wykonany w projekcie montażowym.

2.5. Istniejące rozwiązania wspomagające edycję wideo

Po omówieniu procesu montażowego, metod analizy materiałów oraz możliwości modeli językowych należy sprawdzić, w jaki sposób mechanizmy te są wykorzystywane w dostępnych systemach. Istniejące rozwiązania różnią się zarówno zakresem automatyzacji, jak i miejscem działania. Niektóre wspierają pojedyncze czynności, inne przygotowują montaż wstępny lub pozwalają sterować większą częścią procesu za pomocą języka naturalnego.

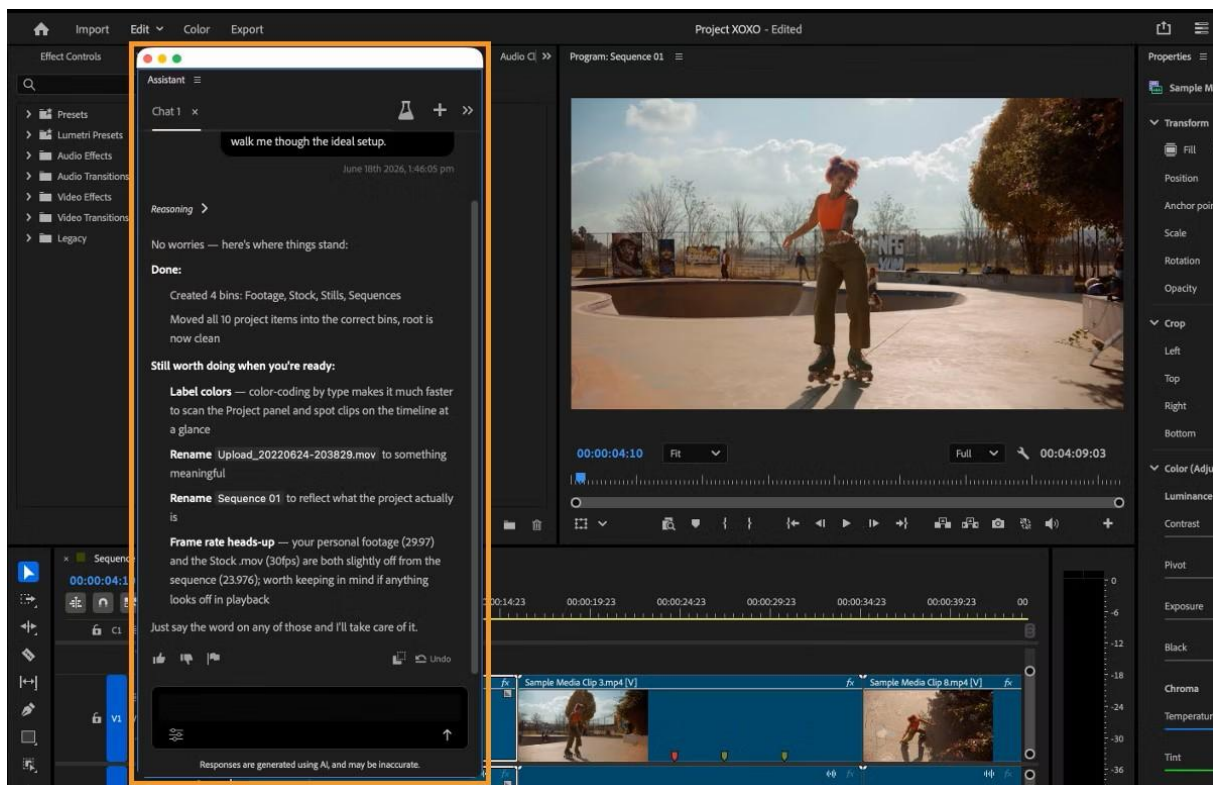
O przydatności rozwiązania decyduje sposób, w jaki przechodzi ono od polecenia użytkownika i informacji o materiałach do zmian w projekcie montażowym. System może jedynie wskazywać odpowiednie fragmenty, przygotowywać propozycję montażu albo wykonywać ciąg powiązanych operacji. Znaczenie ma również zakres decyzji powierzonych modelowi oraz moment, w którym użytkownik może sprawdzić, zmienić lub odrzucić przygotowany wynik. Zestawienie tych cech pozwala ocenić, które sposoby automatyzacji mogą objąć większą część procesu montażowego bez odbierania użytkownikowi kontroli nad projektem.

2.5.1 Profesjonalne edytory wideo z funkcjami automatyzacji

Profesjonalny edytor wideo umożliwia przeprowadzenie pełnego procesu postprodukcji w jednym środowisku. Zapewnia wielościeżkową oś czasu, precyzyjną kontrolę nad klipami, obrazem i dźwiękiem oraz możliwość dalszego modyfikowania wszystkich elementów projektu. Systemy tej klasy są przystosowane do obsługi rozbudowanych projektów, dużej liczby materiałów i wieloosobowych procesów produkcyjnych, dlatego znajdują zastosowanie również w produkcjach telewizyjnych i kinowych.

Jednym z przykładów takiego środowiska jest Adobe Premiere [29]. Wersja stacjonarna programu jest rozwiązaniem komercyjnym, udostępnianym w ramach miesięcznej lub rocznej subskrypcji Creative Cloud. Program udostępnia kilka funkcji reprezentujących różne sposoby wspomagania montażu. *Text-Based Editing* pozwala przygotować montaż wstępny przez zaznaczanie, usuwanie i zmianę kolejności fragmentów transkrypcji. Operacje wykonywane na tekście powodują odpowiednie przycięcie i rozmieszczenie klipów na osi czasu. *Media Intelligence* umożliwia wyszukiwanie materiałów na podstawie opisu ich zawartości, natomiast *Generative Extend* wykorzystuje generatywną sztuczną inteligencję do tworzenia dodatkowych klatek obrazu lub fragmentów dźwięku. Są to wyspecjalizowane funkcje realizujące określone zadania, a nie mechanizmy planujące cały proces montażowy.

Szerszy zakres działania ma *Premiere AI Assistant* [30], udostępniony jako wczesna publiczna beta. Jest to konwersacyjny system agentowy, który interpretuje polecenie użytkownika, wybiera narzędzia programu i wykonuje wieloetapowe operacje. Może porządkować materiały w folderach projektu, zmieniać nazwy i oznaczenia klipów, przygotowywać transkrypcje, wykrywać klapsy filmowe, dodawać markery oraz budować ciąg ujęć lub pierwszą wersję montażu. Panel asystenta przedstawia sposób interpretacji polecenia, zaplanowane kroki oraz wykorzystane narzędzia. Użytkownik może wymagać zatwierdzenia działań albo zezwolić na ich automatyczne wykonanie, a wprowadzone zmiany są zapisywane w mechanizmach *Undo* i *History*.



Rysunek 4. Panel Premiere AI Assistant przedstawiający wykonane czynności i propozycje kolejnych działań. Źródło: Adobe [30]

Adobe nie wskazuje jednego modelu językowego stanowiącego podstawę całego asystenta. Dokumentacja opisuje wykorzystanie różnych narzędzi oraz modeli własnych i partnerskich, dlatego rozwiązanie należy traktować jako system agentowy, a nie pojedynczy LLM. Obecna wersja jest zoptymalizowana pod język angielski, nie pozwala podłączyć własnego modelu, a część narzędzi wymaga przetwarzania materiałów w chmurze. Producent zaznacza również, że rozwiązanie pozostaje eksperymentalne i nie zaleca jeszcze jego stosowania w pracy z materiałami klienckimi.

DaVinci Resolve [31] wykorzystuje odmienny model dystrybucji. Podstawowa wersja programu jest bezpłatna i obejmuje narzędzia montażu, korekcji barwnej, efektów wizualnych oraz postprodukcji dźwięku. Płatna wersja DaVinci Resolve Studio rozszerza ten zestaw między innymi o *DaVinci AI Neural Engine*, montaż oparty na tekście i dodatkowe funkcje wykorzystujące sztuczną inteligencję. Do wersji Studio należą najważniejsze mechanizmy automatyzacji omawiane w tym podrozdziale, w tym *IntelliSearch* oraz *IntelliScript*.

IntelliSearch analizuje materiały i umożliwia wyszukiwanie osób, obiektów, wypowiedzi oraz metadanych. *IntelliScript* wykorzystuje natomiast transkrypcję do dopasowania nagrań do scenariusza i utworzenia osi czasu. Wybrany dubel umieszcza na pierwszej ścieżce, a alternatywne wersje na kolejnych. Producent wskazuje jednak, że wybór opiera się wyłącznie na zgodności dialogu ze scenariuszem. Funkcja nie analizuje obrazu ani metadanych określających jakość ujęcia.

Wersja Studio zawiera również narzędzia realizujące inne zadania montażowe. *Multicam SmartSwitch* automatycznie dobiera kąty kamery na podstawie aktywnego mówcy, *IntelliCut* wykrywa i usuwa ciszę, *Audio Assistant* przygotowuje wstępny miks ścieżek, natomiast *Music Editor* dopasowuje długość utworu muzycznego do montażu. W przeciwieństwie do *Premiere AI Assistant* funkcje te nie tworzą jednego systemu konwersacyjnego planującego kolejne działania.

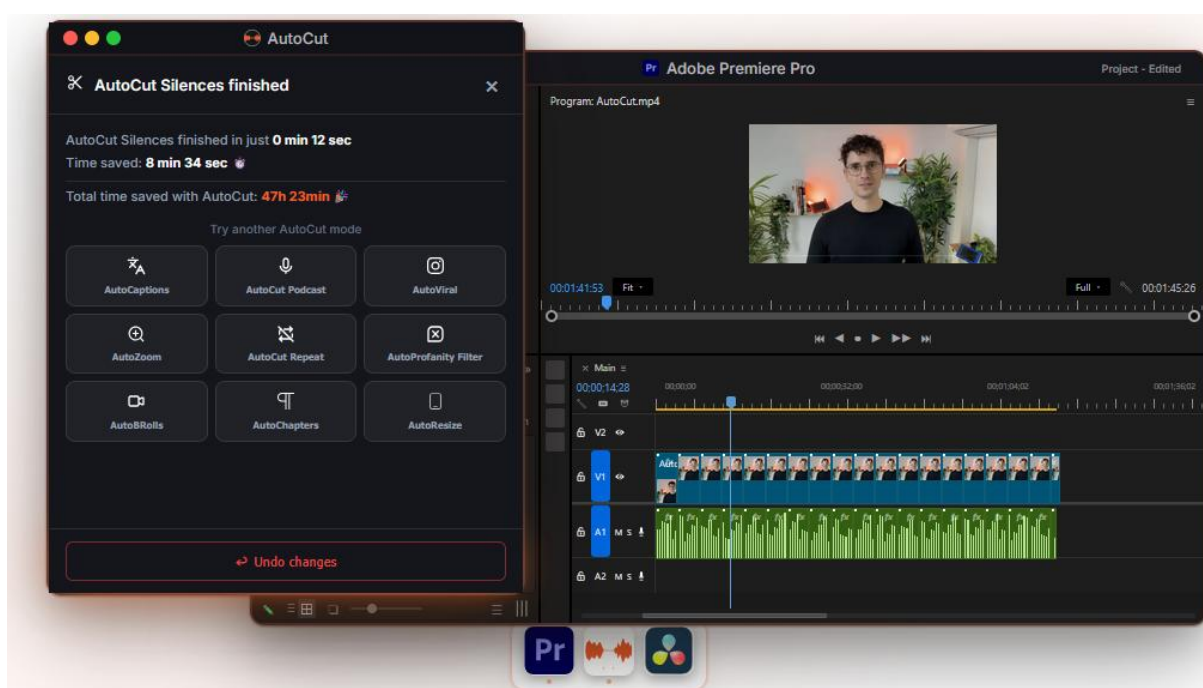
Każda z nich realizuje wcześniej określone zadanie, a otrzymany rezultat może być dalej modyfikowany na osi czasu.

Przegląd profesjonalnych edytorów wskazuje, że rozwój automatyzacji nie oznacza jej powszechnej dostępności. Adobe udostępnia swoje środowisko w modelu subskrypcyjnym, a Blackmagic Design ogranicza większość zaawansowanych funkcji sztucznej inteligencji do płatnej wersji Studio. Powstaje w ten sposób wyraźna różnica między możliwościami samego edytora a zakresem automatyzacji dostępnym dla użytkownika bez dodatkowej licencji.

2.5.2 Wtyczki rozszerzające możliwości programów do edycji wideo

Profesjonalne edytory mogą być rozszerzane za pomocą wtyczek, paneli oraz skryptów korzystających z interfejsów udostępnionych przez producenta. Rozwiązania te dodają funkcje automatyzacji bez konieczności zastępowania całego środowiska montażowego. Część działa bezpośrednio na osi czasu, natomiast inne pełnią funkcję mostu pomiędzy edytorem a osobną aplikacją odpowiedzialną za analizę materiałów.

- **AutoCut** [32] – wtyczka działająca bezpośrednio w Adobe Premiere oraz DaVinci Resolve. Umożliwia między innymi usuwanie ciszy, tworzenie napisów, automatyczne przełączanie kamer, wykrywanie powtórzonych ujęć, dodawanie zbliżeń oraz dobieranie materiałów uzupełniających. Jej zaletą jest wykonywanie operacji na edytowalnej osi czasu bez eksportowania projektu do innego programu. Wszystkie funkcje wtyczki mogą być używane zarówno w bezpłatnej, jak i płatnej wersji DaVinci Resolve. Ograniczeniem jest zamknięty katalog operacji: użytkownik konfiguruje poszczególne narzędzia, ale nie może zlecić dowolnego celu montażowego ani uzyskać wieloetapowego planu jego realizacji. Po zakończeniu wersji próbnej dostęp do funkcji wymaga płatnego planu AutoCut;



Rysunek 5. Panel wtyczki AutoCut po zakończeniu automatycznego usuwania ciszy w programie Adobe Premiere Pro. Źródło: AutoCut [32]

- **Eddie AI** [33] – rozwiązanie wykorzystujące język naturalny do wyszukiwania wypowiedzi, wyboru fragmentów nagrań i przygotowywania montażu wstępnego. Panel programu

Premiere oraz skrypt dla DaVinci Resolve jedynie pośredniczą w komunikacji z osobną aplikacją Eddie, która importuje materiały, analizuje je, a następnie przekazuje przygotowaną oś czasu z powrotem do edytora. Rozwiązanie dobrze odpowiada pracy z wywiadami, vlogami i nagraniami zawierającymi wyraźny dialog. Jego ograniczeniem jest konieczność przechodzenia między aplikacjami oraz silne uzależnienie montażu od transkrypcji. Materiały bez wypowiedzi mogą zostać wykorzystane jako przebitki, ale nie stanowią podstawy narracji budowanej przez system. Integracja jest dostępna dla DaVinci Resolve Free i Studio, natomiast analiza oraz kolejne operacje wykorzystują płatne kredyty usługi Eddie;

- **Jumper** [34] – wtyczka przeznaczona do lokalnej analizy i semantycznego wyszukiwania obrazu, wypowiedzi oraz występujących w materiale osób. Za pomocą protokołu *Model Context Protocol* (MCP) może współpracować z agentami takimi jak Codex lub Claude, które wyszukują fragmenty, przygotowują zestawy ujęć i eksportują je do dalszego montażu. Zaletą rozwiązania jest lokalne przetwarzanie materiałów i brak konieczności przesyłania nagrań do modelu językowego. Jumper koncentruje się jednak przede wszystkim na wyszukiwaniu i wyborze materiałów, a działanie agentowe wymaga osobnego agenta zgodnego z MCP. Integracja z Adobe Premiere jest dostępna od wersji 24, natomiast w przypadku DaVinci Resolve wymagana jest płatna wersja Studio.

Przykłady te potwierdzają, że Adobe Premiere posiada rozwinięty ekosystem wtyczek związanych z automatyzacją montażu. W przypadku DaVinci Resolve dostępność rozszerzeń zależy od sposobu ich integracji: AutoCut i Eddie AI współpracują z wersją Free oraz Studio, natomiast Jumper wymaga wersji Studio. Wśród przeanalizowanych rozwiązań nie występuje jednak połączenie sterowania różnymi operacjami za pomocą języka naturalnego, przedstawiania użytkownikowi planu działań przed jego wykonaniem oraz bezpośredniej pracy na edytowalnym projekcie profesjonalnego środowiska NLE.

2.5.3 Aplikacje wspomagające edycję wideo

Przygotowanie krótkiego materiału do publikacji nie zawsze wymaga korzystania z pełnego środowiska postprodukcyjnego. W odpowiedzi na potrzeby użytkowników tworzących filmy do mediów społecznościowych, podcasty i materiały informacyjne powstały aplikacje, które prowadzą przez wybrane etapy montażu i automatyzują znaczną część wykonywanych czynności.

- **CapCut** [35] – jest dostępny jako aplikacja stacjonarna, mobilna i internetowa. Udostępnia uproszczoną oś czasu, gotowe szablony oraz funkcje automatycznego tworzenia napisów, usuwania słów wypełniających, inteligentnego wyszukiwania, zmiany proporcji obrazu i generowania filmu na podstawie scenariusza. Rozwiązanie jest ukierunkowane przede wszystkim na szybkie przygotowywanie materiałów publikowanych w mediach społecznościowych. Chociaż umożliwia również ręczną pracę na osi czasu, nie zapewnia kontroli nad rozbudowanym projektem, obrazem i dźwiękiem w zakresie porównywalnym z profesjonalnymi środowiskami postprodukcyjnymi. Z tego względu sprawdza się głównie w realizacji krótkich i powtarzalnych formatów, ale nie zastępuje profesjonalnego edytora przy bardziej złożonych produkcjach;
- **Descript** [36] – organizuje montaż wokół transkrypcji. Usuwanie lub przenoszenie fragmentów tekstu powoduje odpowiednią zmianę nagrania, a agent Underlord może wykonywać polecenia dotyczące skracania materiału, wyboru wypowiedzi, usuwania powtórzeń oraz przygotowywania krótkich wersji filmu. Takie podejście dobrze sprawdza się w podcastach, wywiadach i prezentacjach, w których głównym elementem materiału jest mowa. Staje się jednak mniej użyteczne, gdy montaż wymaga precyzyjnej pracy z wieloma ścieżkami, kompozycją obrazu, korekcją barwną lub rozbudowanymi efektami.

Descript upraszcza więc montaż materiałów opartych na wypowiedziach, ale nie zastępuje pełnego środowiska NLE;

- **OpusClip** [37] – automatycznie przekształca dłuższe nagrania w krótkie materiały przeznaczone do publikacji w serwisach społecznościowych. Analizuje obraz, dźwięk i wypowiedzi, wybiera fragmenty, zmienia proporcje kadru oraz dodaje napisy i materiały uzupełniające. Użytkownik może poprawić rezultat w prostym edytorze lub wyeksportować go w formacie rozszerzalnego języka znaczników (ang. *Extensible Markup Language*, **XML**) do Adobe Premiere albo DaVinci Resolve. Zakres działania aplikacji pozostaje jednak podporządkowany jednemu scenariuszowi: ponownemu wykorzystaniu długiego nagrania w postaci krótkich klipów. Nie umożliwia ona swobodnego zaprojektowania rozbudowanego montażu od podstaw, a możliwość eksportu do profesjonalnego edytora wskazuje, że dalsza obróbka może wymagać osobnego środowiska NLE.

Aplikacje tej grupy dobrze realizują zadania, dla których zostały zaprojektowane, ale zwykle nie pozwalają dowolnie rozszerzać wykonywanego procesu. Użytkownik korzysta z określonych wcześniej mechanizmów automatyzacji, a zakres jego kontroli zależy od funkcji udostępnionych przez producenta. Ogranicza to zastosowanie tych rozwiązań w złożonej postprodukcji i uzasadnia poszukiwanie podejścia łączącego prostotę wydawania poleceń z edytowalnością projektu w profesjonalnym środowisku.

2.5.4 Prototypy i rozwiązania badawcze

W badaniach nad automatyzacją montażu pojawiają się prototypy, które wykorzystują modele językowe do interpretowania poleceń, wybierania materiałów i budowania osi czasu. Rozwiązania te służą przede wszystkim sprawdzeniu nowych sposobów współpracy człowieka z systemem oraz ocenie, czy modele mogą podejmować wieloetapowe decyzje montażowe.

Jednym z najbardziej zbliżonych przykładów jest EditDuet [38]. System wykorzystuje dwóch agentów: *Editor* modyfikuje oś czasu, natomiast *Critic* ocenia rezultat i przekazuje instrukcje dotyczące poprawek. Badanie ograniczono do przygotowywania warstwy *B-roll* dla wcześniej ustalonego materiału *A-roll*, a operacje wykonywano we własnym środowisku eksperymentalnym. Rozwiązanie nie współpracuje bezpośrednio z komercyjnym edytorem i obsługuje ograniczony zbiór działań. Pomimo zastosowania strukturyzowanych wywołań funkcji 8,2% prób nie zakończyło się poprawnym wyrenderowaniem filmu. Przyczynami były między innymi odwołania do nieistniejących funkcji lub plików, błędne indeksy oraz znaczniki czasu wykraczające poza długość nagrania.

Nowsze podejście przedstawia prototyp Crayotter [39], w którym proces podzielono na analizę materiałów, przygotowanie planu oraz wykonanie operacji na osi czasu. Raporty z analizy, plany, wywołania narzędzi i pośrednie rezultaty są zapisywane jako artefakty dostępne do kontroli, dzięki czemu można śledzić przebieg pracy, diagnozować błędy i ponawiać wybrane etapy. Crayotter pozostaje jednak samodzielnym środowiskiem, które buduje własną oś czasu i eksportuje gotowy film. Nie przekazuje planu do wykonania wewnątrz istniejącego profesjonalnego edytora ani nie modyfikuje jego projektu.

Przedstawione prototypy potwierdzają, że modele językowe mogą analizować materiały, planować montaż i wykonywać operacje za pomocą wyspecjalizowanych narzędzi. Mają one jednak charakter eksperymentalny albo koncentrują się na ograniczonych scenariuszach montażowych. Wśród przeanalizowanych prac nie zidentyfikowano systemu, który łączyłby agentowe sterowanie językiem naturalnym z bezpośrednią pracą na edytowalnym projekcie w istniejącym profesjonalnym środowisku NLE. Wskazuje to na możliwość przeniesienia mechanizmów rozwijanych w prototypach badawczych do narzędzia współpracującego z profesjonalnym edytorem wideo.

2.6. Analiza i wnioski

Przedstawione rozwiązania różnią się zakresem automatyzacji, sposobem sterowania, miejscem wykonywania operacji oraz poziomem kontroli pozostawionym użytkownikowi. Część z nich automatyzuje pojedyncze, dobrze określone czynności, takie jak usuwanie ciszy, generowanie napisów lub wybór ujęć. Inne pozwalają opisać oczekiwany rezultat w języku naturalnym i powierzają modelowi przygotowanie wieloetapowej koncepcji montażu. Istotną różnicę stanowi również postać wyniku. W profesjonalnym procesie montażowym korzystniejsze jest utworzenie edytowalnej osi czasu niż wygenerowanie wyłącznie końcowego pliku wideo, ponieważ montażysta zachowuje możliwość sprawdzenia i poprawienia rezultatu.

Na podstawie kryteriów określonych w podrozdziale 2.1 oraz wniosków dotyczących automatyzacji montażu, analizy materiałów i zastosowania modeli językowych przedstawionych w podrozdziałach 2.2–2.4, w Tabeli 1 zestawiono rozwiązania omówione w podrozdziale 2.5. Porównanie obejmuje zakres automatyzacji, sposób kontrolowania wykonania oraz dostępność poszczególnych rozwiązań, ze szczególnym uwzględnieniem ich modelu kosztowego. Zestawienie opracowano na podstawie publicznie dostępnej dokumentacji producentów i publikacji opisujących prototypy badawcze. Brak opisu danego mechanizmu nie przesądza o jego nieobecności, lecz oznacza, że nie można było jej potwierdzić w ramach przeprowadzonego przeglądu.

Tabela 1. Porównanie analizowanych rozwiązań wspomagających montaż wideo

Rozwiązanie	Zakres automatyzacji	Kontrola wykonania	Dostępność i koszt
Premiere AI Assistant [30]	Interpretacja poleceń w języku naturalnym i planowanie wieloetapowe	Edytowalny wynik, zatwierdzanie działań i możliwość ich cofnięcia, brak opisu typowanego planu walidowanego przed wykonaniem	Płatna subskrypcja, funkcja pozostaje wersją beta
Funkcje AI DaVinci Resolve [31]	Wyspecjalizowane funkcje realizujące określone zadania	Przewidywalne operacje wykonywane na projekcie, lecz bez wspólnego planu konwersacyjnego	Większość zaawansowanych funkcji wymaga płatnej wersji Studio
AutoCut [32]	Automatyzacja wcześniej zdefiniowanych czynności montażowych	Zamknięty i przewidywalny zestaw operacji, bez swobodnej interpretacji celu	Płatna subskrypcja po okresie próbnym
Eddie AI [33]	Interpretacja poleceń, analiza materiałów i przygotowanie montażu wstępnego	Wynik może być dalej edytowany, dokumentacja nie opisuje typowanego planu sprawdzanego przed wykonaniem	Rozliczenie wykorzystujące płatne kredyty

Rozwiązanie	Zakres automatyzacji	Kontrola wykonania	Dostępność i koszt
Jumper [34]	Lokalna analiza i semantyczne wyszukiwanie materiałów	Wspomaga wyszukiwanie i wybór materiałów, ale nie wykonuje kompletnego planu montażu	Płatna licencja
EditDuet i Crayotter [38], [39]	Wieloetapowe planowanie i podział zadań pomiędzy agentów	Mechanizmy oceny i śledzenia działań, lecz w środowiskach eksperymentalnych	Prototypy badawcze, brak gotowego produktu użytkowego i możliwość występowania kosztów modeli

Źródło: opracowanie własne na podstawie [30] – [39].

Przeprowadzone porównanie nie wskazuje na brak narzędzi automatyzujących montaż, lecz na rozproszenie potrzebnych możliwości pomiędzy różnymi rozwiązaniami. Uproszczone aplikacje pozwalają szybko przygotować materiał do publikacji, ale ograniczają zakres dalszej edycji. Profesjonalne edytory zachowują pełną kontrolę nad projektem, jednak ich zaawansowane funkcje automatyzacji są często dostępne odpłatnie albo realizują jedynie określone wcześniej zadania. Systemy sterowane językiem naturalnym zapewniają większą swobodę, lecz nie zawsze opisują sposób walidowania przygotowanych działań. Prototypy badawcze potwierdzają natomiast możliwość wieloetapowego planowania, ale nie stanowią jeszcze narzędzi gotowych do zastosowania w typowym środowisku montażowym.

W ramach przyjętego zakresu przeglądu nie zidentyfikowano rozwiązania dla DaVinci Resolve 21 Free, które łączyłoby interpretację poleceń w języku naturalnym i analizę materiałów z utworzeniem jawnego, typowanego i walidowanego planu, którego operacje byłyby wykonywane w edytowalnym projekcie dopiero po zatwierdzeniu przez użytkownika. Polecenie użytkownika powinno zostać zinterpretowane z uwzględnieniem zawartości materiałów, a następnie przekształcone w plan złożony wyłącznie z operacji obsługiwanych przez aplikację. Plan powinien zostać sprawdzony i przedstawiony użytkownikowi przed wykonaniem. Modele językowe mogą dzięki temu odpowiadać za zadania wymagające interpretacji, natomiast zatwierdzone operacje mogą być wykonywane przez przewidywalne moduły oparte na jednoznacznych regułach.

Rozwiązanie powinno być dostępne również dla osoby nieposiadającej specjalistycznej wiedzy z zakresu montażu. Użytkownik nie musiałby znać nazw narzędzi programu ani samodzielnie określać wszystkich operacji na osi czasu, lecz opisywałby oczekiwany rezultat w języku naturalnym. W przypadku prostych zadań wynikiem mógłby być krótki materiał przeznaczony bezpośrednio do publikacji w mediach społecznościowych. Przy bardziej złożonych realizacjach system przygotowywałby edytowalny montaż wstępny, który profesjonalny montażysta mógłby następnie sprawdzić, poprawić i rozwinąć bez konieczności rozpoczynania pracy od początku. Narzędzie nie zastępowałoby zatem montażyisty, lecz ograniczałoby czas potrzebny na selekcję materiałów, przygotowanie pierwszej wersji filmu i wykonanie powtarzalnych czynności.

Wymóg zachowania edytowalnego wyniku uzasadnia osadzenie rozwiązania w profesjonalnym środowisku NLE, natomiast potrzeba ograniczenia kosztu wejścia przemawia za wykorzystaniem programu dostępnego w pełnej wersji montażowej bez opłaty licencyjnej. Na tej

podstawie przyjęto stworzenie prostej z perspektywy użytkownika wtyczki do DaVinci Resolve Free. Jej zakres został celowo ograniczony do zamkniętego zbioru możliwych do zweryfikowania operacji, wykonywanych dopiero po zatwierdzeniu planu. Chociaż projekt nie jest systemem współpracy zespołowej, przedstawienie intencji użytkownika w postaci jawnego planu i edytowalnej osi czasu może również usprawnić komunikację pomiędzy zleceniodawcą, osobą przygotowującą pierwszą wersję materiału i profesjonalnym montażystą. Architekturę rozwiązania realizującego te założenia przedstawiono w rozdziale 3.

3. Projekt systemu Cuttie-ai

Analiza przedstawiona w rozdziale 2 wskazała potrzebę połączenia sterowania w języku naturalnym z możliwością pracy na edytowalnym projekcie profesjonalnego programu do montażu. Odpowiedzią na te wymagania jest projekt lokalnego systemu współpracującego z DaVinci Resolve 21 Free. W rozdziale przedstawiono jego założenia, architekturę oraz przepływ prowadzący od analizy materiałów i interpretacji polecenia do zatwierdzenia i wykonania planu edycji.

3.1. Założenia projektu

Wnioski z analizy istniejących rozwiązań wskazały potrzebę połączenia swobodnego sposobu opisywania oczekiwanego rezultatu z możliwością pracy na edytowalnym projekcie profesjonalnego programu NLE. Na tej podstawie przyjęto, że projektowany system będzie interpretował polecenia w języku naturalnym, wykorzystywał informacje pozyskane z materiałów oraz przedstawiał przygotowane działania użytkownikowi przed ich wykonaniem.

Rozwiązanie zaprojektowano jako lokalną aplikację działającą w systemie Windows oraz współpracującą z DaVinci Resolve 21 Free. System może analizować pliki wskazane przez użytkownika oraz materiały znajdujące się w *Media Pool*, czyli bibliotece materiałów otwartego projektu DaVinci Resolve. Przygotowana reprezentacja obrazu, mowy i dźwięku stanowi kontekst wykorzystywany przy wyborze fragmentów i planowaniu kolejnych operacji. Na tej podstawie powstaje plan, którego wykonanie może prowadzić do utworzenia nowej osi czasu albo wprowadzenia zmian w istniejącym montażu.

Lokalny charakter rozwiązania odnosi się do sposobu uruchomienia jego głównych komponentów oraz bezpośredniego dostępu do projektu i materiałów znajdujących się na komputerze użytkownika. Wybrane zadania związane z analizą i planowaniem mogą korzystać z zewnętrznych usług modeli. Bez uruchomionego edytora system może analizować materiały znajdujące się na dysku i rozpocząć interpretację polecenia. Dostęp do *Media Pool*, odczyt stanu projektu, przygotowanie końcowego planu edycji oraz jego wykonanie wymagają jednak uruchomionego DaVinci Resolve i otwartego projektu.

Zachowanie edytowalnego rezultatu oznacza, że aplikacja nie ogranicza wyniku działania do końcowego pliku wideo. Operacje są wykonywane bezpośrednio w projekcie DaVinci Resolve, który pozostaje źródłem informacji o bieżącym stanie osi czasu. Przygotowany montaż może być następnie samodzielnie poprawiany i rozwijany przez użytkownika.

Kontrolę nad zmianami zapewnia rozdzielenie planowania od wykonania. Modele uczestniczą w zadaniach wymagających interpretacji, natomiast kod aplikacji buduje i waliduje plan złożony z obsługiwanych operacji. Do warstwy wykonawczej trafia wyłącznie zweryfikowany plan, a nie swobodna odpowiedź modelu ani wygenerowany przez niego kod. W ramach bieżącej sesji system utrzymuje jeden wspólny stan roboczy i jeden plan oczekujący na zatwierdzenie.

Zakres projektu nie obejmuje wykonywania zmian bez zgody użytkownika, współpracy wielu użytkowników ani zastępowania pełnego edytora wideo. System nie dokonuje również automatycznej oceny artystycznej końcowego rezultatu. Ocena przygotowanego montażu oraz decyzja o jego dalszym wykorzystaniu pozostają po stronie użytkownika.

3.2. Architektura rozwiązania

Architektura logiczna przedstawia podział systemu na główne komponenty, zakres ich odpowiedzialności oraz sposób wymiany danych. W projekcie podział ten podporządkowano przepływowi prowadzącemu od działania użytkownika do kontrolowanej zmiany projektu montażowego. Rozwiązanie składa się z trzech głównych komponentów: aplikacji desktopowej, lokalnej warstwy serwerowej oraz mostu wykonawczego działającego wewnątrz DaVinci Resolve. Dodatkowym elementem są zewnętrzne usługi modeli wykorzystywane przez wybrane funkcje warstwy serwerowej.

Aplikacja stacjonarna odpowiada za obsługę użytkownika i prezentowanie stanu systemu. Udostępnia widoki rozmowy, materiałów, projektu DaVinci Resolve oraz ustawień. Wewnętrznie składa się z warstwy prezentacji, ograniczonej warstwy pośredniczącej oraz procesu głównego. Warstwa prezentacji nie łączy się bezpośrednio z warstwą serwerową ani z systemem plików. Przekazuje żądania do procesu głównego, który obsługuje wybór plików, kontroluje dostępne operacje i komunikuje się z warstwą serwerową przez lokalny interfejs wykorzystujący protokół HTTP.

Warstwa serwerowa skupia logikę systemu. Odpowiada za analizę materiałów, komunikację z usługami modeli, interpretację poleceń oraz organizację wyspecjalizowanych ról uczestniczących w planowaniu montażu. Deterministyczna część warstwy serwerowej buduje i waliduje plany edycji. W warstwie serwerowej znajduje się również moduł integracji z DaVinci Resolve, wykorzystywany do odczytywania *Media Pool*, bieżącego stanu projektu i zawartości osi czasu.

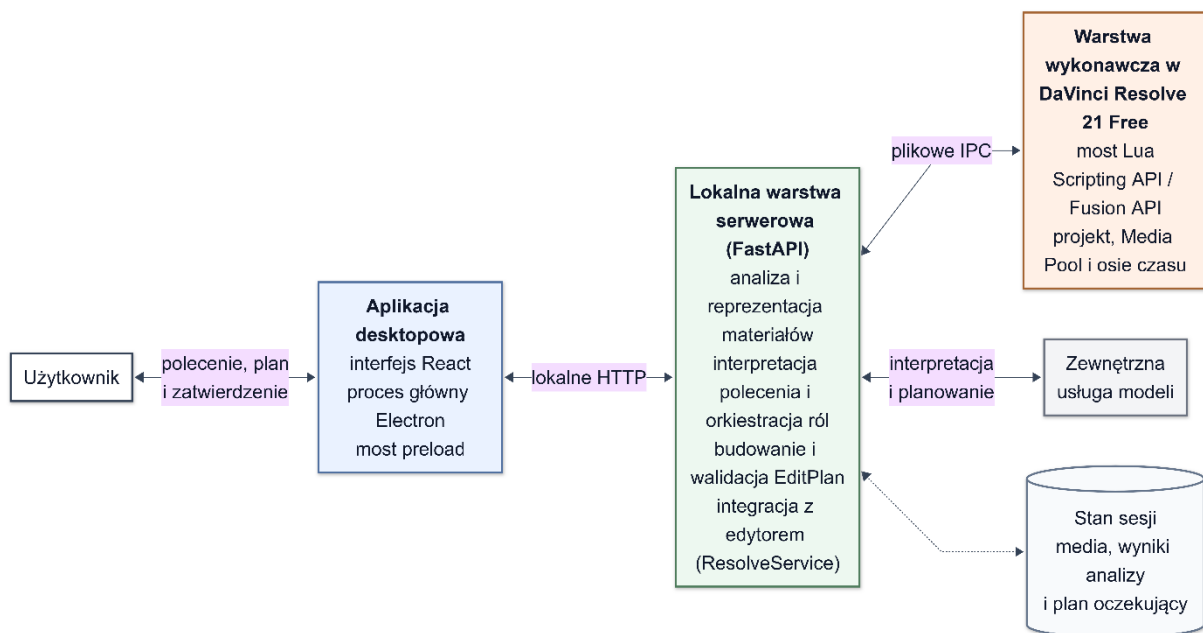
Proces rozpoczyna się od wskazania plików znajdujących się na dysku albo wybrania materiałów z otwartego projektu. Warstwa serwerowa przygotowuje ich reprezentację, a następnie łączy ją z poleceniem użytkownika, historią rozmowy i aktualnym stanem projektu. Na tej podstawie powstaje typowany plan edycji zawierający wyłącznie operacje obsługiwane przez system. Przygotowanie planu nie powoduje jeszcze zmiany projektu. Plan jest przechowywany w warstwie serwerowej i przedstawiany użytkownikowi, który może go zatwierdzić albo odrzucić.

Po zatwierdzeniu warstwa serwerowa ponownie sprawdza, czy plan nadal odnosi się do właściwego projektu, osi czasu i materiałów źródłowych. Następnie przekazuje go przez plikowy kanał komunikacji międzyprocesowej (ang. *inter-process communication*, **IPC**) do mostu uruchomionego w DaVinci Resolve. Most odwzorowuje polecenia na operacje udostępniane przez interfejsy programistyczne edytora: Scripting API, służące do pracy z projektem i osią czasu, oraz Fusion API, wykorzystywane do obsługi kompozycji i efektów. Nie podejmuje przy tym decyzji kreatywnych ani nie otrzymuje bezpośrednio odpowiedzi modelu. Wynik wykonania wraca przez warstwę serwerową do aplikacji desktopowej.

DaVinci Resolve pozostaje nadrzędnym źródłem informacji o projekcie i osiach czasu. Warstwa serwerowa przechowuje stan bieżącej sesji, obejmujący rejestr mediów, wyniki ich analizy oraz plan oczekujący na zatwierdzenie. Ustawienia i historia rozmów są zapisywane w trwałym magazynie danych, natomiast klucze dostępowe trafiają do odseparowanego magazynu poświadczeń systemu. Pliki IPC przechowują jedynie techniczny stan komunikacji z mostem. Rozmieszczenie danych pomiędzy komponentami przedstawiono w Tabeli 2. Architekturę logiczną systemu oraz przepływ danych pomiędzy jego komponentami przedstawiono na Rysunek 6.

Tabela 2. Rozmieszczenie stanu systemu

Dane	Właściciel	Trwałość
Projekt, osie czasu i klipy	DaVinci Resolve	Cykl życia projektu
Media, klasyfikacje i oczekujący plan	Warstwa serwerowa	Bieżąca sesja
Ustawienia i rozmowy	Lokalny magazyn danych	Trwałe
Klucze dostępowe	Magazyn poświadczeń	Trwałe i odseparowane
Komenda, wynik i stan mostu	Kanał IPC	Stan techniczny



Rysunek 6. Architektura logiczna systemu i przepływ danych pomiędzy jego komponentami. Źródło: opracowanie własne

3.3. Analiza i reprezentacja materiałów multimedialnych

Materiały wejściowe różnią się rodzajem oraz zakresem informacji, które mogą być wykorzystane podczas montażu. Nagranie wideo zawiera obraz, dźwięk i przebieg czasowy, obraz statyczny wymaga opisu pojedynczej zawartości, natomiast plik tekstowy może dostarczać dodatkowego kontekstu. Z tego względu system rozdziela rejestrację materiału od wyniku jego analizy.

Zarejestrowany materiał jest reprezentowany przez podstawowe informacje identyfikacyjne, takie jak ścieżka, nazwa, rodzaj i źródło pochodzenia. Może również posiadać podpowiedź określającą jego przewidywaną rolę, na przykład materiał główny, muzykę lub nagranie lektorskie. Informacje te nie opisują jeszcze treści pliku, lecz pozwalają jednoznacznie wskazać go podczas kolejnych etapów przetwarzania.

Rezultat analizy jest przechowywany w strukturze `MediaClassification`. Stanowi ona wspólny kontrakt dla wszystkich obsługiwanych rodzajów materiałów. W przypadku wideo może zawierać sceny, opisy wybranych klatek, transkrypcję i cechy ścieżki dźwiękowej. Dla audio przechowuje transkrypcję oraz informacje o przebiegu dźwięku. Obraz statyczny jest reprezentowany przez opis jego zawartości, natomiast plik tekstowy przez odczytaną treść i krótkie streszczenie. Pola, które nie dotyczą danego rodzaju materiału, pozostają niewykorzystane.

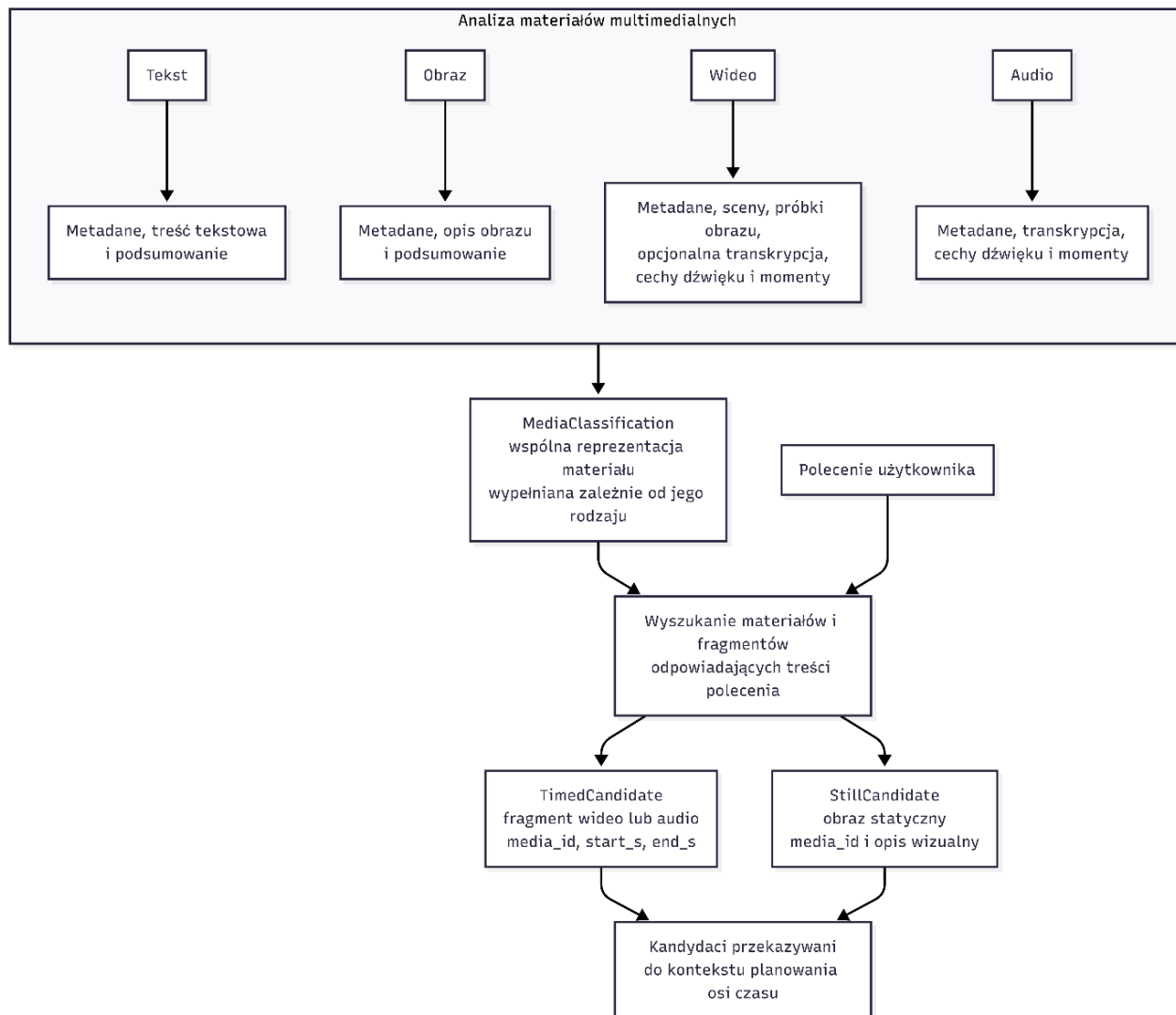
Ważną częścią reprezentacji wideo i audio są momenty. Każdy moment odpowiada krótkiemu przedziałowi czasowemu i skupia dane potrzebne do oceny danego fragmentu. Może zawierać wypowiedź rozpoznaną w tym przedziale, opis widocznej zawartości, oznaczenia tematyczne oraz lokalne informacje o dźwięku. Taki podział pozwala odwoływać się do konkretnych fragmentów nagrania bez konieczności ponownego przetwarzania całego pliku.

Dokładność reprezentacji wizualnej zależy od wybranego trybu analizy. Tryb podstawowy tworzy ogólny opis przy użyciu mniejszej liczby próbek, a tryb szczegółowy uwzględnia większą liczbę fragmentów obrazu. Tryb analizy jest zapisywany wraz z wynikiem, dzięki czemu wiadomo, w jaki sposób powstała dana reprezentacja. Użytkownik może w razie potrzeby ponownie przeanalizować materiał w innym trybie.

Kandydaci do wykorzystania w montażu nie są stałą częścią klasyfikacji. Powstają dopiero po otrzymaniu polecenia użytkownika, ponieważ ten sam materiał może być przydatny w różnych zadaniach. System zestawia słowa występujące w poleceniu z nazwami plików, transkrypcjami, opisami obrazu i oznaczeniami momentów. Wybrane fragmenty oraz skrócone opisy materiałów tworzą kontekst przekazywany do procesu planowania.

Wyniki klasyfikacji są przechowywane jako stan bieżącej sesji. Użytkownik może skorygować ich tekstową część, na przykład transkrypcję, podsumowanie albo opis momentu. Dzięki temu proces planowania korzysta ze wspólnej, ale możliwej do zweryfikowania reprezentacji materiałów.

Rysunek 7 przedstawia sposób łączenia wyników analizy we wspólną reprezentację materiału oraz wybór kandydatów przekazywanych do planowania.



Rysunek 7. Model reprezentacji materiałów i wykorzystanie wyników analizy podczas planowania. Źródło: opracowanie własne.

3.4. Interpretacja polecenia i orkiestracja procesu planowania

Przed interpretacją polecenia system zbiera informacje potrzebne w bieżącej interakcji. Oprócz wiadomości użytkownika i ograniczonej historii rozmowy obejmują one podstawowe dane o materiałach, takie jak nazwa, rodzaj, czas trwania i krótkie podsumowanie zawartości. Na podstawie wyników analizy opisanej w podrozdziale 3.3 system dołącza również fragmenty nagrań i obrazy, które mogą odpowiadać poleceniu użytkownika, a także pliki dźwiękowe możliwe do wykorzystania jako muzyka. Jeżeli wcześniej przygotowano plan oczekujący na zatwierdzenie, jego podsumowanie także zostaje uwzględnione. System odczytuje ponadto format aktywnej osi czasu i informacje o znajdujących się na niej klipach. Gdy edytor jest niedostępny, system może nadal analizować materiały znajdujące się na dysku i rozpocząć interpretację polecenia. Zbudowanie końcowego planu edycji, również dla nowego montażu, wymaga jednak dostępu do uruchomionego DaVinci Resolve i otwartego projektu.

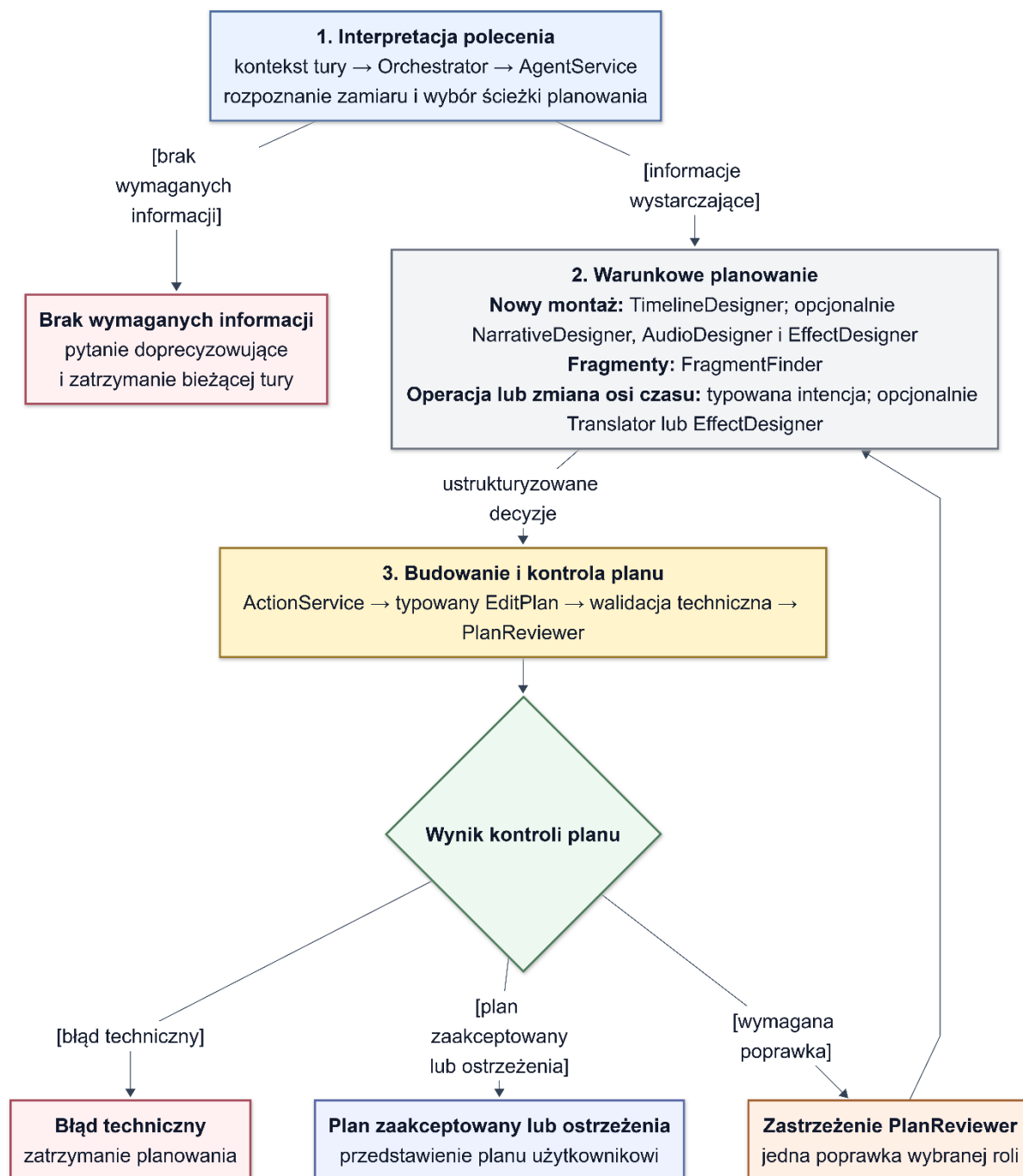
Kontekst trafia najpierw do roli **Orchestrator**. Rola ta rozpoznaje wymagania zawarte w poleceniu, określa, które z nich są obowiązkowe, i wskazuje elementy wykraczające poza zakres aplikacji. Zwraca typowaną decyzję delegacji zawierającą intencje dalszych działań oraz opisy zadań dla potrzebnych ról. Końcowy plan edycji powstaje dopiero w następnych etapach. Jeżeli dostępne informacje nie wystarczają do wyboru dalszego działania, **Orchestrator** formułuje jedno pytanie doprecyzowujące, a system wstrzymuje planowanie do czasu uzyskania odpowiedzi.

Kod aplikacji odczytuje decyzję roli **Orchestrator** i wybiera odpowiednią ścieżkę. Polecenie może dotyczyć utworzenia nowego montażu, modyfikacji aktywnej osi czasu, wyszukania fragmentów, renderowania materiału albo wykonania pojedynczej operacji obsługiwanej przez system. Jednoznaczne czynności są zapisywane jako typowane intencje, czyli struktury określające rodzaj operacji i jej parametry, a następnie trafiają do deterministycznej części warstwy serwerowej. Jeżeli zadanie wymaga interpretacji treści, wyboru materiałów lub opracowania koncepcji montażu, **AgentService** uruchamia jednego lub kilku wyspecjalizowanych agentów wykorzystujących modele językowe.

Podczas przygotowywania nowego montażu **AgentService**, na podstawie decyzji **Orchestrator**, może najpierw uruchomić agenta **NarrativeDesigner**. Agent ten opracowuje motyw, otwarcie i przebieg narracji na podstawie celu użytkownika, podsumowań materiałów oraz wybranych fragmentów transkrypcji. Jeżeli montaż wymaga muzyki, uruchamiany jest agent **AudioDesigner**, który wybiera utwór, jego zakres oraz sposób połączenia z dźwiękiem źródłowym. Oba wywołania są opcjonalne. Następnie agent **TimelineDesigner** wybiera materiały i fragmenty nagrań, ustala ich kolejność oraz przygotowuje strukturę osi czasu w zadanym formacie. Po wskazaniu docelowego segmentu może zostać uruchomiony agent **EffectDesigner**, ale tylko wtedy, gdy żądanego efektu nie obsługuje gotowy mechanizm aplikacji. Każdy agent zwraca wynik o określonej strukturze, który **AgentService** przekazuje do następnego etapu. Agenci nie uruchamiają się wzajemnie, a odpowiedź niezgodna z wymaganym typem danych zatrzymuje bieżącą turę.

Deterministyczna część warstwy serwerowej łączy wyniki agentów, buduje plan edycji i przeprowadza jego obowiązkową walidację techniczną. Dopiero plan poprawny pod względem struktury, parametrów i zależności jest przekazywany do **PlanReviewer**. Agent ten nie powtarza walidacji technicznej, lecz ocenia zgodność koncepcji montażu z poleceniem i wymaganiami użytkownika oraz spójność decyzji dotyczących narracji, muzyki, osi czasu i efektów. Może zaakceptować plan, zgłosić ostrzeżenia albo wskazać problem i rolę odpowiedzialną za jego usunięcie. **PlanReviewer** sam nie zmienia planu ani wyników pozostałych agentów.

W przypadku nowego montażu blokujący problem może spowodować najwyżej jedną automatyczną poprawkę, wyłącznie gdy **PlanReviewer** przypisze go do roli **NarrativeDesigner**, **AudioDesigner**, **TimelineDesigner** albo **EffectDesigner**. **AgentService** uruchamia wskazaną rolę jeszcze raz. Jeżeli zmiana dotyczy narracji lub dźwięku, wywołuje również **TimelineDesigner**, ponieważ jego wynik zależy od tych danych. Warstwa serwerowa buduje i waliduje nową wersję planu, a następnie przekazuje ją do kolejnej recenzji. Jeżeli zastrzeżenia pozostają, system nie wykonuje kolejnej automatycznej poprawki, lecz przedstawia plan użytkownikowi wraz z uwagami. Użytkownik może przekazać instrukcję dalszej zmiany albo zatwierdzić plan mimo zastrzeżeń. Przebieg interpretacji polecenia i przygotowania planu edycji przedstawiono na Rysunek 8.



Rysunek 8. Diagram aktywności przedstawiający przebieg interpretacji polecenia i przygotowania planu edycji. Źródło: opracowanie własne.

3.5. Budowanie i walidacja planu edycji

Wyniki etapu planowania trafiają do deterministycznej części warstwy serwerowej realizowanej przez ActionService. Mogą mieć postać ustrukturyzowanego opisu pojedynczej operacji albo wyników przygotowanych przez wyspecjalizowanych agentów. Dane te opisują

oczekiwany rezultat montażu, natomiast **ActionService** przekształca je w operacje obsługiwane przez aplikację.

Sposób budowania planu zależy od ścieżki wybranej wcześniej przez rolę **Orchestrator**. W przypadku nowego montażu wykorzystywane są informacje o wybranych materiałach, ich kolejności, zakresach czasowych, formacie osi czasu, muzyce i efektach. Dla pojedynczych działań, takich jak dodanie napisów lub zmiana istniejącej osi czasu, podstawą jest opis operacji wraz z jej parametrami. Obie ścieżki prowadzą do utworzenia struktury **EditPlan**.

EditPlan wskazuje docelowy projekt oraz, jeżeli operacja dotyczy już istniejącego montażu, odpowiednią oś czasu. Zawiera również podsumowanie, ostrzeżenia i uporządkowaną listę poleceń. Lista może składać się wyłącznie z operacji wcześniej zdefiniowanych w aplikacji, takich jak utworzenie osi czasu, dodanie klipu, muzyki lub napisów, zastosowanie efektu i renderowanie.

Podczas budowania planu **ActionService** oblicza parametry techniczne i ustala kolejność poleceń. Tworzy również tymczasowe referencje, które pozwalają kolejnym operacjom wskazywać oś czasu lub klip powstający dopiero podczas wykonania planu. Dzięki temu można wcześniej opisać zależności pomiędzy operacjami bez znajomości identyfikatorów, które zostaną nadane przez **DaVinci Resolve**.

Każdy utworzony **EditPlan** obowiązkowo podlega walidacji technicznej. Kontrola obejmuje typy i wartości parametrów, unikalność referencji oraz kolejność tworzenia i wykorzystywania elementów. Nie można na przykład odwołać się do klipu przed jego dodaniem ani umieścić renderowania przed innymi poleceniami. Ocena **PlanReviewer** jest odrębna od tej walidacji i dotyczy zgodności koncepcji montażu z poleceniem użytkownika, a nie technicznej poprawności **EditPlan**. Po ewentualnej poprawce **ActionService** ponownie buduje i waliduje plan.

Po zakończeniu kontroli plan zostaje zapisany jako oczekujący na decyzję użytkownika. Jego przygotowanie nie zmienia jeszcze projektu montażowego.

3.6. Kontrolowane wykonanie planu w **DaVinci Resolve**

Plan zapisany jako oczekujący jest przedstawiany użytkownikowi wraz z podsumowaniem, listą poleceń i ostrzeżeniami. Ewentualne zastrzeżenia **PlanReviewer** są prezentowane razem z planem, ale nie zastępują decyzji użytkownika. Użytkownik może odrzucić plan albo zatwierdzić go mimo tych zastrzeżeń i rozpocząć wykonanie. Możliwość zatwierdzenia dotyczy wyłącznie planu, który wcześniej przeszedł obowiązkową walidację techniczną. Plan niespełniający jej wymagań nie trafia do etapu zatwierdzania. Odrzucenie usuwa plan bez wprowadzania zmian w projekcie, natomiast zatwierdzenie przekazuje do warstwy serwerowej jego identyfikator i rozpoczyna etap wykonania.

Przed wysłaniem planu warstwy serwerowej ponownie sprawdza warunki, dla których został on przygotowany. Porównuje identyfikator aktualnie otwartego projektu z projektem wskazanym w **EditPlan**. Jeżeli plan dotyczy istniejącej osi czasu, sprawdza również, czy nadal znajduje się ona w projekcie. Weryfikowane są także pliki źródłowe, w tym ich ścieżki, rozmiary i czasy modyfikacji. Wykrycie niezgodności zatrzymuje proces przed rozpoczęciem zmian.

Poprawny plan jest przekazywany w całości przez plikowy kanał IPC do mostu Lua działającego wewnątrz **DaVinci Resolve**. Most ponownie sprawdza typ żądania, kompletność poleceń i tożsamość projektu. Przyjmuje wyłącznie operacje należące do zbioru obsługiwanego przez aplikację. Nie interpretuje polecenia użytkownika i nie korzysta z modeli językowych.

Polecenia są wykonywane kolejno za pomocą Scripting API oraz, w przypadku kompozycji i efektów, Fusion API. Pierwszy napotkany błąd zatrzymuje dalsze wykonanie. Operacje zakończone wcześniej nie są automatycznie cofane, dlatego plan nie ma charakteru transakcyjnego.

Most zwraca wynik zawierający stan wykonania, listę zakończonych poleceń oraz, w przypadku niepowodzenia, wskazanie błędnej operacji i opis problemu. Jeżeli warstwa serwerowa nie otrzyma wiarygodnego wyniku, nie ponawia planu automatycznie, ponieważ część zmian mogła już zostać wykonana. Rezultat jest przedstawiany użytkownikowi, który może sprawdzić stan projektu bezpośrednio w edytorze.

4. Narzędzia zastosowane w systemie wspomagania montażu wideo

Realizacja programu do wspomagania montażu wideo wymagała połączenia technologii przeznaczonych do różnych zadań w ramach jednej aplikacji desktopowej. Dobór narzędzi musiał uwzględniać wymagania poszczególnych komponentów oraz możliwości ich połączenia w jednym lokalnie uruchamianym programie. Lokalna warstwa serwerowa odpowiada za logikę programu i przetwarzanie danych, narzędzia integracji modeli językowych umożliwiają komunikację z wykorzystywanymi modelami, a biblioteki multimedialne pozwalają analizować obraz, mowę i sygnał dźwiękowy. Osobny obszar stanowią technologie wykorzystywane do budowy interfejsu użytkownika oraz mechanizmy pozwalające wykonywać operacje wewnątrz programu DaVinci Resolve.

Poszczególne komponenty działają w różnych środowiskach wykonawczych i korzystają z odmiennych mechanizmów komunikacji. Interfejs aplikacji, lokalna warstwa serwerowa oraz komponent działający wewnątrz DaVinci Resolve wymieniają dane pomimo wykorzystania różnych języków i interfejsów programistycznych. Zastosowany zestaw narzędzi obejmuje więc również mechanizmy umożliwiające komunikację pomiędzy tymi komponentami w ramach lokalnie uruchamianej aplikacji.

W rozdziale przedstawiono najważniejsze narzędzia, biblioteki i środowiska wykorzystane podczas realizacji rozwiązania. Opis koncentruje się na ich właściwościach istotnych dla programu oraz przesłankach wpływających na ich wybór.

4.1. Kryteria doboru narzędzi i technologii

Dobór technologii dla programu do wspomagania montażu wideo wynikał z wymagań przyjętych podczas projektowania jego architektury. Program działa lokalnie, ale składa się z komponentów uruchamianych w różnych środowiskach i realizujących odmienne zadania. Zastosowane narzędzia musiały odpowiadać wymaganiom poszczególnych części programu, a jednocześnie umożliwiać ich połączenie w ramach jednej aplikacji.

Na wybór technologii wpływał przyjęty wcześniej sposób integracji z DaVinci Resolve 21 Free. Ograniczenia wynikające z możliwości udostępnianych przez tę wersję edytora wymagały dopasowania narzędzi wykorzystywanych do wykonywania operacji montażowych oraz komunikacji z pozostałymi komponentami programu. Technologie zastosowane poza środowiskiem Resolve musiały jednocześnie umożliwiać współpracę z komponentem działającym wewnątrz edytora bez konieczności przenoszenia do niego całej logiki aplikacji.

Dostępność bibliotek miała szczególne znaczenie w obszarze analizy materiałów multimedialnych. Techniki deterministyczne przedstawione w rozdziale 2.3 obejmują między innymi odczyt i przygotowanie materiałów, analizę kolejnych klatek, wykrywanie granic ujęć, ocenę wybranych właściwości obrazu oraz analizę sygnału dźwiękowego. Technologia wykorzystywana przez komponent odpowiedzialny za te zadania powinna zapewniać dostęp do bibliotek obsługujących wymagane operacje na obrazie i dźwięku. Ten sam ekosystem powinien również umożliwiać wykorzystanie transkrypcji oraz łączenie wyników metod deterministycznych z informacjami pozyskiwanymi przy użyciu modeli językowych i multimodalnych. Pozwala to korzystać z istniejących implementacji podstawowych metod przetwarzania zamiast przygotowywać je od podstaw na potrzeby programu.

Podział rozwiązania na kilka komponentów wymaga zapewnienia ich wzajemnej komunikacji. Interfejs aplikacji desktopowej, lokalna warstwa serwerowa oraz komponent działający wewnątrz DaVinci Resolve korzystają z różnych środowisk wykonawczych i języków programowania. Muszą jednak wymieniać informacje dotyczące między innymi stanu aplikacji, materiałów, wyników analizy, przygotowanych planów oraz rezultatów wykonywanych operacji. Z tego względu przy doborze technologii należało uwzględnić możliwość tworzenia jednoznacznych interfejsów pomiędzy komponentami i przesyłania ustrukturyzowanych danych w sposób niezależny od języka ich implementacji. Komunikacja może dzięki temu wykorzystywać lokalne mechanizmy, takie jak HTTP pomiędzy częścią aplikacji desktopowej i lokalnej warstwy serwerowej oraz osobny kanał wymiany danych z komponentem działającym wewnątrz DaVinci Resolve.

Lokalny charakter aplikacji wymaga, aby jej podstawowe komponenty mogły być uruchamiane na komputerze użytkownika. Dotyczy to interfejsu, lokalnej warstwy serwerowej, dostępu do materiałów multimedialnych oraz komunikacji z działającym projektem w DaVinci Resolve. Podstawowe funkcje programu nie powinny więc wymagać utrzymywania osobnej infrastruktury serwerowej. Usługi zewnętrzne mogą być wykorzystywane przez wybrane funkcje związane z modelami sztucznej inteligencji, ale nie stanowią środowiska wykonawczego dla głównej logiki aplikacji.

Wykorzystanie zewnętrznych modeli wymaga również uwzględnienia sposobu przetwarzania danych. Modele uruchamiane lokalnie ograniczają konieczność przekazywania informacji poza komputer użytkownika, natomiast usługi chmurowe mogą wymagać przesłania części danych potrzebnych do wykonania zadania. Sam sposób uruchomienia modelu nie może jednak stanowić jedyne kryterium jego wyboru, ponieważ zastosowane rozwiązanie musi również zapewniać możliwości wymagane przez wykonywane zadania. Narzędzia integracyjne powinny zatem umożliwiać kontrolowanie zakresu danych przekazywanych do usług zewnętrznych oraz oddzielanie danych dostępowych od pozostałych informacji przechowywanych przez aplikację.

4.2. Technologie warstwy serwerowej

Warstwa serwerowa programu jest komponentem aplikacji desktopowej uruchamianym na komputerze użytkownika. Odpowiada za logikę programu, przetwarzanie danych oraz udostępnianie lokalnego interfejsu komunikacyjnego pozostałym komponentom. W jej środowisku działają również biblioteki wykorzystywane do analizy materiałów multimedialnych oraz integracji z modelami sztucznej inteligencji.

Warstwa ta opiera się na środowisku uruchomieniowym i frameworku lokalnej usługi HTTP, mechanizmach modelowania i walidacji danych oraz rozwiązaniach służących do przechowywania stanu aplikacji i informacji poufnych.

4.2.1 Python i FastAPI

Jako podstawowy język lokalnej warstwy serwerowej wybrano Python. W programie wykorzystano wersję 3.13. Za wyborem tego języka przemawiał przede wszystkim ekosystem bibliotek związanych z przetwarzaniem danych, analizą materiałów multimedialnych, uczeniem maszynowym oraz integracją modeli językowych i multimodalnych. Pozwala to realizować większość zadań warstwy serwerowej w jednym środowisku wykonawczym, bez konieczności tworzenia dodatkowych warstw integracyjnych pomiędzy narzędziami przeznaczonymi dla różnych języków.

Znaczenie tego ekosystemu wynika bezpośrednio z zakresu metod przedstawionych w rozdziale 2.3. Analiza materiałów wymaga odczytu i dekodowania plików, przetwarzania obrazu i sygnału dźwiękowego, wykrywania granic ujęć, wyboru klatek reprezentatywnych oraz transkrypcji. Dla Pythona dostępne są biblioteki obsługujące te operacje, a także narzędzia przeznaczone do komunikacji z modelami językowymi i multimodalnymi. Dzięki temu wyniki metod deterministycznych mogą być przetwarzane i przekazywane do kolejnych etapów analizy bez zmiany środowiska programistycznego. W przypadku tego programu dostępność wspólnego ekosystemu bibliotek była jednym z głównych argumentów przemawiających za wyborem Pythona.

Do utworzenia lokalnego interfejsu HTTP wykorzystano FastAPI [40]. Jest to framework webowy dla języka Python przeznaczony do budowania interfejsów API, wykorzystujący standardowe adnotacje typów języka do definiowania parametrów i struktur danych. Takie podejście odpowiada organizacji lokalnej warstwy serwerowej przyjętej w programie, w której komunikacja pomiędzy komponentami odbywa się za pomocą określonych struktur żądań i odpowiedzi.

FastAPI współpracuje z biblioteką Pydantic [41], służącą do definiowania modeli oraz walidacji danych na podstawie adnotacji typów Pythona. Pozwala to stosować modele danych zarówno wewnątrz warstwy serwerowej, jak i na granicy lokalnego interfejsu HTTP. Dane otrzymane przez API mogą zostać sprawdzone przed przekazaniem ich do dalszej logiki programu. Sposób wykorzystania modeli i mechanizmów walidacji został szerzej przedstawiony w podrozdziale 4.2.2.

FastAPI wybrano ze względu na jego dopasowanie do typowanej warstwy serwerowej napisanej w Pythonie oraz możliwość utworzenia niewielkiej lokalnej usługi HTTP bez wprowadzania dodatkowego środowiska technologicznego. Warstwa serwerowa działa na komputerze użytkownika i udostępnia interfejs pozostałym komponentom aplikacji desktopowej. Zastosowanie HTTP pozwala zachować niezależność technologiczną pomiędzy interfejsem aplikacji a warstwą serwerową i wymieniać dane przez jednoznacznie określoną granicę komunikacyjną.

4.2.2 Walidacja i modele danych

Do modelowania i walidacji danych w lokalnej warstwie serwerowej wykorzystano bibliotekę Pydantic [41]. Umożliwia ona definiowanie modeli na podstawie adnotacji typów Pythona oraz sprawdzanie zgodności otrzymanych danych z określoną strukturą. Modele wykorzystywane w programie mają zamkniętą strukturę, dzięki czemu dane zawierające nieprzewidziane pola lub wartości niespełniające określonych reguł mogą zostać odrzucone. Te same modele są wykorzystywane na granicy lokalnego interfejsu FastAPI oraz w wewnętrznej logice warstwy serwerowej.

Jednym z zastosowań Pydantic jest model `EditPlan`, zawierający różne rodzaje operacji montażowych reprezentowane za pomocą unii dyskryminowanej. Mechanizm ten wykorzystuje wskazane pole do określenia wariantu modelu, względem którego dane powinny zostać zweryfikowane [42]. Pozwala to zachować wspólny kontrakt planu, a jednocześnie przypisać poszczególnym rodzajom operacji właściwe pola i reguły walidacji.

Modele Pydantic są wykorzystywane również do sprawdzania danych pochodzących z modeli językowych. Pydantic nie odpowiada za samo wygenerowanie ustrukturyzowanej odpowiedzi, lecz definiuje oczekiwaną postać danych i umożliwia zweryfikowanie otrzymanego wyniku przed jego dalszym wykorzystaniem. Dane niespełniające zdefiniowanego kontraktu nie są dzięki temu bezpośrednio przekazywane do deterministycznych elementów programu.

4.2.3 Przechowywanie danych i ochrona informacji

Część danych wykorzystywanych przez program musi być zachowana pomiędzy kolejnymi uruchomieniami aplikacji. Można je zapisywać bezpośrednio w plikach, jednak wraz ze wzrostem liczby powiązanych informacji takie rozwiązanie wymaga samodzielnego zarządzania ich strukturą, aktualizacją i spójnością. Z tego względu do trwałego przechowywania danych zastosowano lokalną bazę danych.

Do trwałego zapisu wykorzystano SQLite [43], czyli relacyjną bazę danych działającą bez osobnego procesu serwera. Umożliwia ona lokalne przechowywanie danych oraz ich transakcyjny zapis. Nie wymaga uruchamiania ani konfigurowania dodatkowej usługi bazodanowej, co odpowiada charakterowi aplikacji desktopowej. Wybrano ją ze względu na możliwość zapewnienia ustrukturyzowanego i trwałego zapisu przy niewielkich wymaganiach infrastrukturalnych.

Informacje poufne wymagają odrębnego sposobu przechowywania. Do ich obsługi wykorzystano bibliotekę keyring [44], która udostępnia interfejs do magazynu poświadczeń zapewnianego przez system operacyjny. W systemie Windows pozwala to korzystać z Windows Credential Manager i przechowywać dane dostępne poza pozostałymi danymi aplikacji. Bibliotekę wybrano, ponieważ umożliwia wykorzystanie systemowego magazynu poświadczeń bez implementowania własnego mechanizmu przechowywania sekretów.

4.3. Narzędzia integracji modeli językowych

Integracja modeli językowych w programie obejmuje dwa odrębne poziomy. Jeden dotyczy definiowania wykonywanych zadań, danych wejściowych, oczekiwanej postaci wyniku oraz organizacji wywołań. Drugi obejmuje techniczną komunikację z usługą udostępniającą model, w tym przekazanie wymaganych parametrów i odebranie odpowiedzi. Taki podział pozwala rozwijać logikę programu niezależnie od szczegółów dostępu do konkretnej usługi.

Dostawcy modeli udostępniają własne interfejsy programistyczne, które różnią się sposobem identyfikowania modeli, formatem żądań, zakresem obsługiwanych parametrów oraz postacią zwracanych danych. Obsługa kilku dostawców bez wspólnej warstwy integracyjnej wymagałaby utrzymywania osobnych mechanizmów komunikacji dla każdego z nich. Ujednolicony interfejs skupia te różnice w wydzielonej części programu i pozwala pozostałym komponentom korzystać z modeli w ten sam sposób.

Obszar modeli językowych i multimodalnych rozwija się szybko, a wraz z nim zmieniają się dostępne modele, ich możliwości oraz sposoby udostępniania. Bezpośrednie powiązanie logiki programu z interfejsem jednego dostawcy utrudniałoby późniejszą zmianę wykorzystywanego modelu lub usługi. Wydzielenie warstwy integracyjnej pozwala zachować osobno logikę zadań wykonywanych przez modele i techniczne mechanizmy potrzebne do ich wywoływania.

4.3.1 DSPy

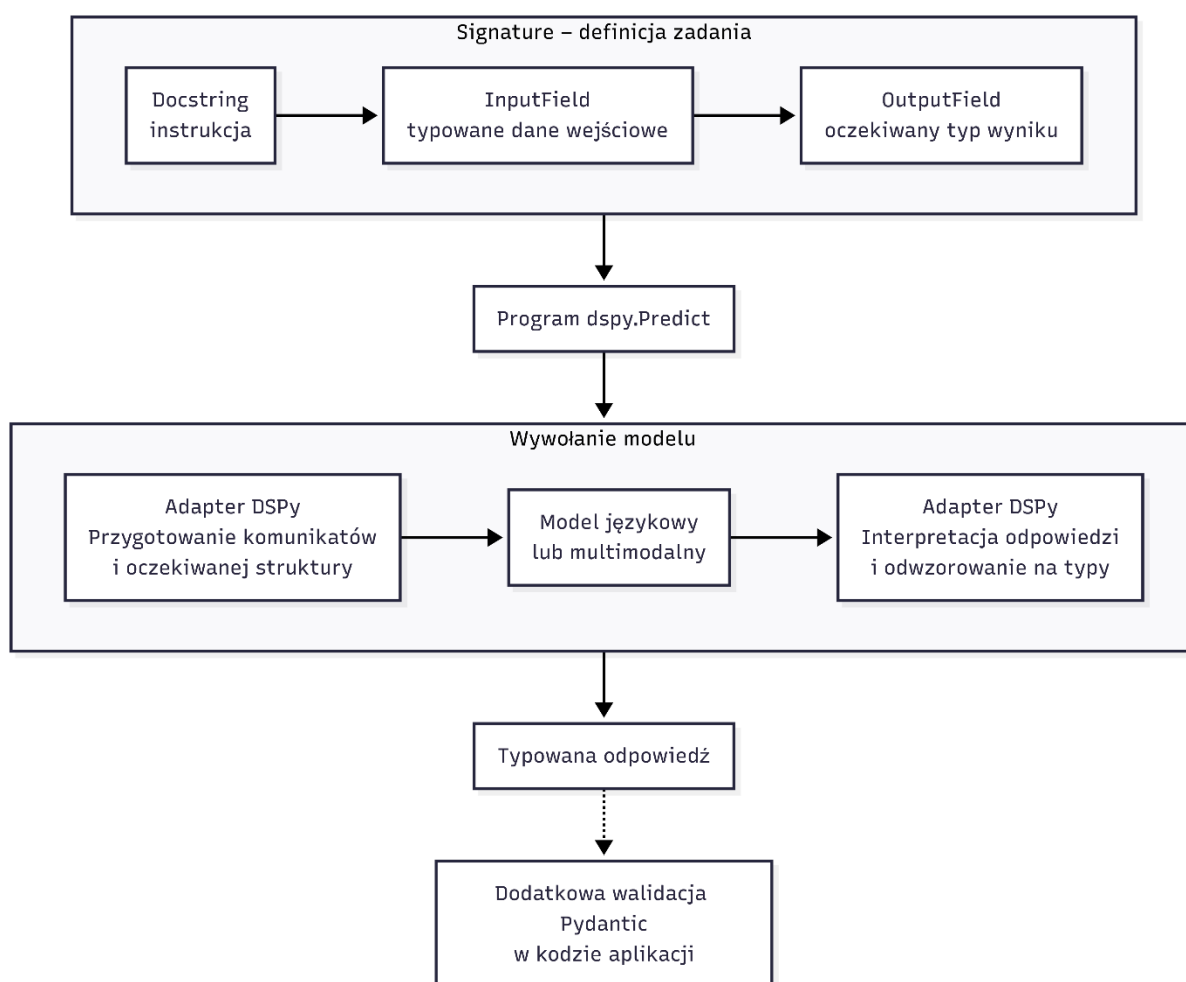
W programie DSPy [45] (ang. *Declarative Self-improving Python*) pełni rolę warstwy służącej do definiowania zadań powierzanych modelom językowym i multimodalnym. Framework opisuje takie zadanie za pomocą instrukcji, danych wejściowych oraz oczekiwanej postaci wyniku. Podstawą tego podejścia są sygnatury *Signature*, które łączą opis zadania ze strukturą danych przekazywanych do modelu i zwracanych w odpowiedzi.

Sygnaturę można zdefiniować jako klasę dziedziczącą po `dspy.Signature`. Pola wejściowe oznacza się za pomocą `InputField`, a pola wyniku za pomocą `OutputField`. Każde pole ma nazwę i typ wynikający z adnotacji typów Pythona, natomiast opcjonalny opis może dostarczyć

modelowi dodatkowego kontekstu. Interfejs zadania określa w ten sposób wymagane dane oraz postać wyniku, który powinien zostać zwrócony [46].

Łańcuch dokumentacyjny (ang. *docstring*) klasy pełni funkcję instrukcji opisującej cel zadania. Podczas przygotowywania komunikatów framework łączy go z informacjami o polach wejściowych i wyjściowych, ich typach oraz dodatkowych opisach. Na tej podstawie przygotowywana jest reprezentacja przekazywana do modelu. Instrukcja pozostaje dzięki temu bezpośrednio związana z typowanym interfejsem zadania, bez konieczności utrzymywania osobnego ciągu tekstowego zawierającego pełny opis zadania [46].

Za komunikację pomiędzy sygnaturą a modelem odpowiada adapter. Przekształca instrukcję, definicje pól i przekazane wartości w komunikaty zgodne ze sposobem wywołania modelu. Po otrzymaniu odpowiedzi interpretuje wynik i przekształca go do wartości odpowiadających typom zadeklarowanym w sygnaturze. Opis zadania pozostaje w ten sposób oddzielony od technicznego formatu komunikacji, a ta sama sygnatura może być używana przy różnych sposobach przygotowywania wywołań [47]. Rysunek 9 przedstawia przepływ danych pomiędzy sygnaturą, adapterem i modelem.



Rysunek 9. Przepływ danych pomiędzy sygnaturą, adapterem i modelem w DSP-y. Źródło: opracowanie własne.

Typ zadeklarowany w `OutputField` może opisywać wynik dokładniej niż zwykła odpowiedź tekstowa. Pole wyjściowe może reprezentować pojedynczą wartość, listę, ograniczony

zbiór wartości lub bardziej złożony model danych. Informacje o typach są wykorzystywane podczas przetwarzania odpowiedzi i przekształcania wyniku modelu do postaci zgodnej z deklaracją sygnatury. Implementacja *Signature* korzysta przy tym z mechanizmów *Pydantic*, między innymi do obsługi typów i schematów danych [46].

Pozwala to uzyskiwać ustrukturyzowane wyniki, których wartości odpowiadają wcześniej zdefiniowanym polom. Sposób uzyskania takiej odpowiedzi zależy od adaptera oraz możliwości konkretnego modelu. Adapter może opisać oczekiwaną strukturę w komunikacie przekazywanym do modelu, a w przypadku obsługi odpowiednich mechanizmów wykorzystać również natywny tryb ustrukturyzowanej odpowiedzi (ang. *structured output*). Otrzymany wynik jest następnie przetwarzany zgodnie z typami określonymi przez sygnaturę [47].

W programie typowane wyniki są dodatkowo sprawdzane przez mechanizmy walidacji opisane w podrozdziale 4.2.2. Sygnatura definiuje oczekiwaną strukturę wyniku, natomiast modele *Pydantic* pozwalają zweryfikować otrzymane dane przed przekazaniem ich do kolejnych elementów programu. Dalsza logika może dzięki temu operować na określonych strukturach danych zamiast samodzielnie interpretować dowolną odpowiedź tekstową.

Sygnatury służą również do definiowania wyspecjalizowanych ról modeli. Poszczególne role mogą otrzymywać różne zestawy danych wejściowych, własne instrukcje oraz oddzielnie określone wyniki. Każde wywołanie ma więc jawnie określone wejście i oczekiwany rezultat, a instrukcja pozostaje związana z zadaniem, którego dotyczy. Sposób współpracy ról i kolejność ich uruchamiania omówiono w podrozdziale 3.4.

Ten sam mechanizm może obejmować dane inne niż tekst. Typ `dspy.Image` pozwala zdefiniować obraz jako pole wejściowe sygnatury i przekazać go do modelu obsługującego dane wizualne [48]. Umożliwia to stosowanie wspólnego sposobu definiowania wejść i wyjść dla zadań językowych oraz multimodalnych. W programie rozwiązanie to jest wykorzystywane podczas analizy wizualnej materiałów, gdzie model otrzymuje dane obrazowe i zwraca informacje zgodne ze strukturą określoną przez sygnaturę.

Framework udostępnia również mechanizmy wykraczające poza opisane zastosowania, między innymi obsługę narzędzi, wieloetapowych strategii działania oraz optymalizacji instrukcji. `dspy.Tool` umożliwia na przykład przedstawienie funkcji Pythona jako narzędzia dostępnego dla modelu oraz utworzenie schematu jej argumentów [49]. Mechanizmy te rozszerzają zakres możliwych sposobów budowania programów wykorzystujących modele, ale nie wszystkie są wykorzystywane w opisywanym rozwiązaniu.

DSP-y wybrano ze względu na typowany sposób definiowania zadań oraz oddzielenie ich interfejsu od technicznego formatu komunikacji z modelem. Sygnatury pozwalają powiązać instrukcję z danymi wejściowymi i oczekiwanym wynikiem, a obsługa ustrukturyzowanych odpowiedzi i danych multimodalnych odpowiada sposobowi dalszego przetwarzania wyników w programie.

4.3.2 LiteLLM

LiteLLM [50] jest odrębną biblioteką wykorzystywaną przez DSP-y jako warstwa komunikacji z usługami udostępniającymi modele. Jej obecność w programie wynika bezpośrednio z integracji modeli za pomocą DSP-y. Biblioteka ujednolica wykonywanie wywołań i obsługuje techniczne różnice pomiędzy interfejsami poszczególnych dostawców.

Dostawcy modeli udostępniają własne interfejsy API, różniące się między innymi formatem żądań, sposobem wskazywania modelu, zakresem obsługiwanych parametrów oraz strukturą odpowiedzi. LiteLLM ujednolica tę komunikację, udostępniając wspólny interfejs dla wielu dostawców i przekształcając żądania do postaci wymaganej przez wybraną usługę. Odpowiedzi są następnie zwracane w ujednoliconym formacie [51].

Model konfiguruje się za pomocą obiektu `dspy.LM`, któremu przekazywany jest identyfikator dostawcy i wybranego modelu. Obsługę komunikacji z poszczególnymi dostawcami realizuje LiteLLM, dzięki czemu kod programu nie musi korzystać bezpośrednio z osobnego klienta API dla każdego z nich [50]. Pozwala to oddzielić definicję zadania od technicznej metody przesłania żądania do konkretnej usługi.

Ta sama warstwa transportowa jest wykorzystywana w programie przy modelach tekstowych i multimodalnych, a także przy usługach rozpoznawania i syntezy mowy. Kod aplikacji wywołuje obsługiwane usługi przez wspólny interfejs, zamiast odwoływać się bezpośrednio do bibliotek klienckich poszczególnych dostawców. Mechanizmy uruchamiane lokalnie, takie jak lokalna transkrypcja, pozostają od tej warstwy niezależne.

LiteLLM ujednolica komunikację z różnymi usługami modelowymi i ogranicza zależność kodu programu od ich indywidualnych interfejsów. Definiowanie zadań, ich wejść i oczekiwanych wyników pozostaje przy tym oddzielone od technicznej obsługi wywołań modeli.

4.4. Narzędzia analizy materiałów multimedialnych

Analiza materiału audiowizualnego nie jest realizowana przez pojedynczą bibliotekę. Obejmuje operacje o różnym charakterze: dostęp do strumieni zapisanych w pliku, przetwarzanie pojedynczych klatek, wykrywanie zmian obrazu, rozpoznawanie mowy oraz analizę numerycznych właściwości sygnału dźwiękowego. Z tego względu w programie wykorzystano zestaw wyspecjalizowanych narzędzi, z których każde odpowiada za określoną grupę operacji.

Techniki analizy materiałów multimedialnych zastosowane w programie zostały omówione w rozdziale 2.3. W tym miejscu przedstawiono narzędzia programistyczne służące do ich realizacji, koncentrując się na funkcjach faktycznie wdrożonych w aplikacji. Biblioteki pogrupowano według trzech obszarów: przygotowania plików multimedialnych, analizy obrazu i detekcji scen oraz analizy mowy i dźwięku. Narzędzia należące do każdego z nich opisano w kolejnych podrozdziałach.

4.4.1 Przetwarzanie plików multimedialnych

Odczyt i przygotowanie pliku wideo stanowią pierwszy etap poprzedzający dalszą analizę materiału, której techniki przedstawiono we wcześniejszych rozdziałach. W proponowanym rozwiązaniu zadanie to realizuje biblioteka PyAV [52], zapewniająca interfejs języka Python do bibliotek projektu FFmpeg [53]. Pozwala ona uzyskać dostęp do informacji o pliku, jego strumieniach obrazu i dźwięku oraz przekształcić zapisane dane do postaci możliwej do dalszego przetwarzania.

FFmpeg dostarcza mechanizmy przetwarzania multimediów wykorzystywane przez PyAV. Umożliwia między innymi odczytywanie, przekształcanie i zapisywanie materiałów audio i wideo w różnych formatach, a także zmianę ich parametrów [54]. Rozwiązania oparte na FFmpeg są wykorzystywane również w dużych systemach przetwarzania multimediów, czego przykładem jest infrastruktura Meta [53]. Zastosowanie PyAV pozwala korzystać z tych możliwości bezpośrednio w środowisku Pythona i przekazywać uzyskane klatki oraz dane dźwiękowe do dalszej analizy.

Pozostałe operacje związane z przygotowaniem danych realizują Pillow i NumPy. Pillow jest wykorzystywane przede wszystkim do obróbki pojedynczych obrazów przed ich dalszym wykorzystaniem. Obejmuje to poprawne ustawienie orientacji obrazu, zmianę jego rozmiaru, kadrowanie, obsługę przezroczystości oraz zapis w wymaganym formacie. NumPy zapewnia natomiast tablicową reprezentację danych numerycznych, która ułatwia wykonywanie dalszych

obliczeń na obrazie i sygnale dźwiękowym oraz przekazywanie danych pomiędzy poszczególnymi bibliotekami.

4.4.2 Analiza obrazu i detekcja scen

Po odkodowaniu materiału kolejne etapy analizy wymagają określenia jego struktury wizualnej oraz wyboru klatek reprezentujących poszczególne fragmenty nagrania. Techniki służące do realizacji tych zadań zostały omówione wcześniej w rozdziale 2.3. W proponowanym rozwiązaniu ich wykonanie opiera się głównie na bibliotekach PySceneDetect i OpenCV.

PySceneDetect jest biblioteką przeznaczoną do automatycznego wykrywania granic ujęć w materiale wideo [55]. Analiza polega na porównywaniu zmian zachodzących pomiędzy kolejnymi klatkami i wskazywaniu miejsc, w których ich wielkość może świadczyć o rozpoczęciu nowego ujęcia. Wykorzystanie gotowego narzędzia pozwala zastosować opisane wcześniej techniki detekcji bez konieczności tworzenia od podstaw całego mechanizmu podziału materiału.

Do bardziej ogólnego przetwarzania obrazu wykorzystano OpenCV [56]. Biblioteka umożliwia wykonywanie operacji na pojedynczych klatkach oraz wyznaczanie ich mierzalnych właściwości. W proponowanym rozwiązaniu wspiera techniki związane z analizą zmian obrazu pomiędzy kolejnymi klatkami oraz oceną technicznej jakości klatek wybieranych do dalszego przetwarzania.

Obie biblioteki pełnią różne role w procesie analizy obrazu. PySceneDetect zapewnia wyspecjalizowany mechanizm analizy czasowej materiału i wykrywania jego podziału na ujęcia, wykorzystując przy tym OpenCV jako jedną z bibliotek obsługujących materiał wideo i operacje na obrazie. OpenCV jest jednocześnie wykorzystywane bezpośrednio do przetwarzania klatek oraz obliczania cech potrzebnych w dalszej logice programu.

4.4.3 Analiza mowy i dźwięku

Ścieżka dźwiękowa dostarcza dwóch różnych rodzajów informacji. Treść wypowiedzi można pozyskać za pomocą automatycznego rozpoznawania mowy, natomiast takie właściwości jak cisza, poziom energii czy rytm wymagają analizy samego sygnału. W proponowanym rozwiązaniu zadania te realizują osobne narzędzia, ponieważ transkrypcja i analiza numeryczna odpowiadają na inne pytania dotyczące tego samego materiału.

Do rozpoznawania mowy zastosowano faster-whisper [57]. Biblioteka umożliwia lokalne wykonywanie transkrypcji przy użyciu modeli Whisper i może działać zarówno na CPU, jak i przy wykorzystaniu CUDA. Oprócz rozpoznanego tekstu może zwracać znaczniki czasowe przypisane do słów. Informacja ta jest szczególnie użyteczna w materiale wideo, ponieważ pozwala wskazać dokładne miejsce wystąpienia określonej wypowiedzi i powiązać ją z osią czasu. Lokalna implementacja stanowi przy tym alternatywę dla usługi transkrypcyjnej wymagającej przesłania danych do zewnętrznego dostawcy.

Sama transkrypcja nie opisuje jednak właściwości akustycznych nagrania. Do ich analizy wykorzystano bibliotekę librosa [58]. Dostarcza ona narzędzia przeznaczone do analizy sygnałów audio i pozwala wyznaczać informacje związane między innymi z ciszą, poziomem energii, tempem oraz położeniem kolejnych uderzeń rytmicznych. Techniki te zostały omówione wcześniej, dlatego w tym miejscu istotna jest przede wszystkim możliwość ich bezpośredniego zastosowania do danych przygotowanych przez warstwę serwerową.

Operacje wymagające własnych obliczeń na próbkach realizowane są z wykorzystaniem NumPy. Tablicowa reprezentacja sygnału pozwala obliczać wartości takie jak RMS oraz stosować własne przekształcenia poziomu dźwięku, w tym regulację wzmocnienia oraz automatyczne ściszenie muzyki pod mowę. Librosa odpowiada zatem za wyspecjalizowane metody analizy

sygnału, a NumPy dostarcza podstawę numeryczną dla operacji realizowanych bezpośrednio przez kod programu.

4.5. Technologie aplikacji desktopowej i interfejsu użytkownika

Interfejs użytkownika stanowi odrębną część programu, odpowiedzialną za prezentowanie stanu aplikacji oraz obsługę działań wykonywanych przez użytkownika. W proponowanym rozwiązaniu został zrealizowany jako aplikacja stacjonarna, ale do jego budowy wykorzystano technologie wywodzące się ze środowiska aplikacji webowych.

Takie podejście pozwala korzystać z dojrzałego ekosystemu narzędzi do tworzenia interfejsów, zachowując jednocześnie możliwość uruchamiania programu jako lokalnej aplikacji. W kolejnych podrozdziałach przedstawiono środowisko Electron oraz technologie wykorzystane do budowy warstwy interfejsu.

4.5.1 Electron jako środowisko aplikacji desktopowej

Do realizacji aplikacji desktopowej wykorzystano Electron [59]. Jest to framework umożliwiający tworzenie programów desktopowych z użyciem technologii webowych, łączący środowisko renderowania oparte na Chromium z możliwościami Node.js. Pozwala to budować interfejs za pomocą HTML, CSS i JavaScript, zachowując jednocześnie charakter lokalnej aplikacji uruchamianej na komputerze użytkownika.

Takie podejście dobrze odpowiada architekturze przyjętej w rozwiązaniu. Warstwa interfejsu może działać jako aplikacja stacjonarna, natomiast logika wymagająca przetwarzania danych pozostaje w lokalnej warstwie serwerowej udostępniając określony interfejs komunikacyjny. Electron nie narzuca konkretnego sposobu realizacji tej komunikacji, ale umożliwia oddzielenie warstwy prezentacji od pozostałych części programu i współpracę z usługami działającymi lokalnie.

Framework jest stosowany również w rozbudowanych aplikacjach użytkowych. Jednym z rozpoznawalnych przykładów jest Visual Studio Code, którego wersję stacjonarną zbudowano w Electron [60]. Pokazuje to, że technologie webowe mogą stanowić podstawę złożonych programów desktopowych.

Na wybór Electron wpłynęło także wcześniejsze doświadczenie autora pracy w tworzeniu aplikacji webowych. Znajomość ekosystemu HTML, CSS i TypeScript pozwoliła wykorzystać posiadane kompetencje przy budowie interfejsu desktopowego, bez wprowadzania odrębnego frameworka dla natywnych aplikacji okienkowych.

4.5.2 React i warstwa interfejsu użytkownika

Warstwę interfejsu użytkownika zrealizowano przy użyciu React [61] i TypeScript. React jest biblioteką przeznaczoną do budowy interfejsów z komponentów, które mogą być niezależnie definiowane, łączone i ponownie wykorzystywane. Taki model odpowiada interfejsowi aplikacji składającemu się z wielu elementów prezentujących stan programu i obsługujących działania użytkownika. TypeScript uzupełnia tę warstwę o statyczne typowanie i pozwala zachować zgodność struktur danych wykorzystywanych przez poszczególne części aplikacji desktopowej.

Środowisko deweloperskie i proces budowania renderer oparto na Vite. Narzędzie odpowiada za uruchamianie aplikacji podczas rozwoju oraz przygotowanie wynikowej wersji interfejsu. Od strony wizualnej wykorzystano Tailwind CSS oraz pomocnicze biblioteki komponentów, co pozwala definiować wygląd interfejsu bez tworzenia odrębnego systemu stylowania dla każdego jego elementu.

React nie jest wymagany przez Electron i może zostać zastąpiony inną technologią zgodną ze środowiskiem webowym. Rozdzielenie interfejsu od funkcji specyficznych dla Electron ułatwia również ponowne wykorzystanie znacznej części warstwy prezentacji w aplikacji uruchamianej bezpośrednio w przeglądarce. Podobny kierunek wykorzystuje Discord, który udostępnia zarówno wersję przeglądarkową, jak i desktopową opartą na Electron, a w warstwie klientów korzysta z React [62]. Nie oznacza to automatycznej przenośności całej aplikacji, ale ogranicza zakres zmian wymaganych w samej warstwie interfejsu.

4.6. Środowisko DaVinci Resolve

DaVinci Resolve stanowi środowisko, w którym ostatecznie wykonywane są operacje przygotowane przez program do wspomagania montażu. Przyjęty sposób integracji opiera się na mechanizmach programistycznych udostępnianych bezpośrednio przez edytor, które umożliwiają wykonywanie operacji na jego wewnętrznych strukturach.

Środowisko Resolve zapewnia dostęp do elementów projektu, osi czasu, materiałów oraz narzędzi związanych z kompozycją i efektami. Udostępniane interfejsy programistyczne tworzą warstwę wykonawczą, za pośrednictwem której przygotowane operacje mogą zostać odwzorowane na zmiany w projekcie montażowym.

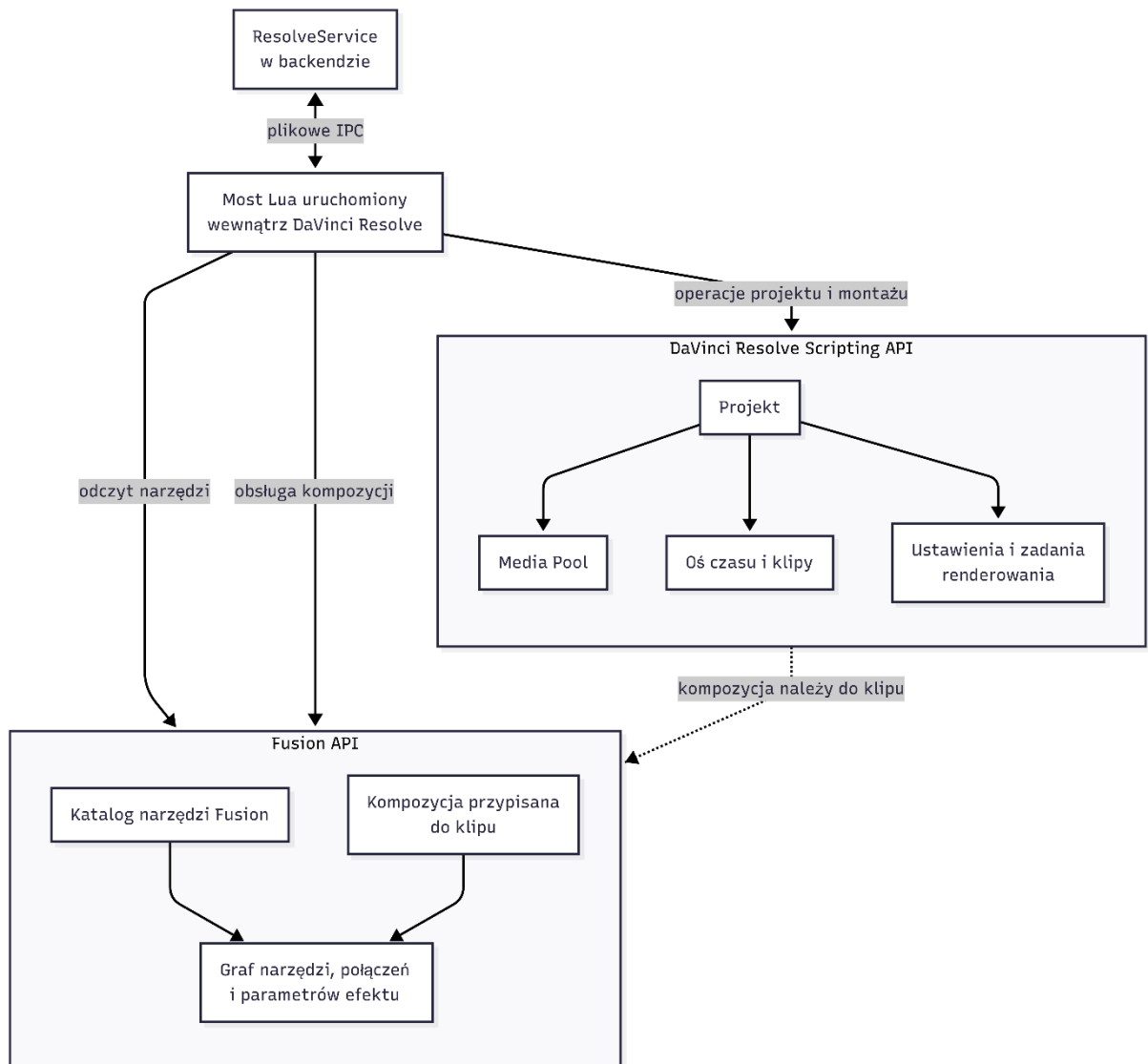
4.6.1 DaVinci Resolve i Fusion API

Programistyczny dostęp do DaVinci Resolve jest realizowany za pomocą Scripting API [31]. Interfejs udostępnia obiekty odpowiadające elementom występującym w środowisku edytora, takim jak projekt, biblioteka materiałów czy oś czasu. Operacje wykonywane na tych obiektach pozwalają zarówno odczytywać stan projektu, jak i wprowadzać wybrane zmiany.

Takie odwzorowanie elementów edytora na obiekty dostępne z poziomu kodu pozwala wykonywać operacje montażowe bez konieczności odwoływania się do interfejsu graficznego. Z punktu widzenia proponowanego rozwiązania API stanowi warstwę, za pomocą której operacja opisana wcześniej w postaci danych może zostać wykonana na konkretnych elementach projektu.

DaVinci Resolve zawiera również środowisko Fusion przeznaczone do kompozycji obrazu i tworzenia efektów wizualnych. Jego model działania różni się od klasycznej osi czasu, ponieważ kolejne operacje obrazu są reprezentowane jako połączone elementy tworzące graf przetwarzania. Fusion API [63] udostępnia programistyczny dostęp do tej struktury i pozwala tworzyć, łączyć oraz konfigurować jej elementy.

Podział pomiędzy oba interfejsy wynika z różnych poziomów pracy w środowisku Resolve. Scripting API zapewnia dostęp do projektu i jego organizacji montażowej, natomiast Fusion API pozwala ingerować w strukturę pojedynczych kompozycji i efektów. Wykorzystanie obu mechanizmów rozszerza zakres operacji możliwych do wykonania programowo poza podstawową edycję osi czasu. Rysunek 10 przedstawia zakres odpowiedzialności obu interfejsów w warstwie wykonawczej.



Rysunek 10. Zakres odpowiedzialności DaVinci Resolve Scripting API i Fusion API w warstwie wykonawczej. Źródło: opracowanie własne.

4.6.2 Lua i mechanizmy skryptowe wewnątrz Resolve

Do wykonywania operacji bezpośrednio w DaVinci Resolve wykorzystano język Lua. Jest on jednym z języków skryptowych obsługiwanych przez środowisko Resolve i pozwala korzystać z udostępnianych przez edytor interfejsów programistycznych [63]. Resolve umożliwia również tworzenie skryptów w Pythonie, jednak korzystanie z jego API z zewnętrznego środowiska wymaga dodatkowej konfiguracji, a w wersji DaVinci Resolve 21 Free dostęp do zewnętrznego skryptowania jest ograniczony. Z tego względu w projekcie przyjęto model, w którym niewielka warstwa skryptowa działa bezpośrednio wewnątrz Resolve, natomiast pozostała część aplikacji jest uruchamiana niezależnie. Lua dobrze odpowiada takiemu sposobowi integracji, ponieważ może być wykonywana bezpośrednio w środowisku skryptowym edytora.

Kod napisany w Lua pełni rolę wykonawcy operacji przygotowanych przez lokalną warstwę serwerową FastAPI. Warstwa serwerowa odpowiada za przetwarzanie danych i przygotowanie poleceń, natomiast warstwa działająca wewnątrz Resolve odwzorowuje otrzymane operacje na wywołania DaVinci Resolve Scripting API lub Fusion API. Lua nie realizuje analizy materiałów ani

planowania montażu, lecz wykonuje wcześniej przygotowane działania na aktualnie otwartym projekcie.

4.7. Wybór dostawcy modeli językowych

Przyjęta warstwa integracji umożliwia korzystanie z usług różnych dostawców modeli. LiteLLM obsługuje wiele platform poprzez wspólny interfejs, dlatego technicznie możliwe jest wykorzystanie między innymi modeli pochodzących od różnych producentów bez bezpośredniego wiązania logiki programu z ich bibliotekami klienckimi [51].

W proponowanym rozwiązaniu jako domyślnego dostawcę wybrano OpenAI. Decyzja wynikała przede wszystkim z szerokiego zakresu API. Ta sama platforma udostępnia modele do generowania i analizy tekstu, przetwarzania obrazów, zamiany mowy na tekst oraz syntezy mowy [64], [65]. Pozwala to ograniczyć liczbę niezależnych integracji potrzebnych do obsługi funkcji wykorzystujących różne typy danych.

Znaczenie miała również postać samego interfejsu programistycznego. OpenAI rozwija mechanizmy przeznaczone do bezpośredniego wykorzystania modeli przez aplikacje, w tym ustrukturyzowane odpowiedzi zgodne z dostarczonym schematem [66]. Jest to zgodne z rozwiązaniem, w którym wynik modelu powinien mieć postać możliwą do jednoznacznego sprawdzenia i dalszego przetwarzania przez kod programu.

Format API OpenAI jest ponadto wykorzystywany przez LiteLLM jako wspólna reprezentacja komunikacji z obsługiwanymi dostawcami [51]. Z punktu widzenia zastosowanego stosu technologicznego oznacza to, że OpenAI odpowiada bezpośrednio modelowi komunikacji, który i tak stanowi podstawę warstwy pośredniczącej.

Wybór ten nie zakłada przewagi modeli OpenAI w każdym możliwym zastosowaniu. W kontekście realizowanego programu dostawca zapewnia jednak wymagany zakres usług, obsługę ustrukturyzowanych odpowiedzi oraz możliwości potrzebne do realizacji funkcji aplikacji. OpenAI spełnia więc wymagania określone dla dostawcy modeli językowych zarówno pod względem funkcjonalnym, jak i technologicznym.

4.8. Narzędzia testowania rozwiązania

Do automatycznego testowania warstwy serwerowej wykorzystano pytest [67]. Framework służy do organizowania testów oraz przygotowywania wspólnych danych i elementów środowiska testowego. Konfiguracja projektu pozwala oddzielić podstawowy zestaw testów od scenariuszy wymagających rzeczywistych usług zewnętrznych.

Interfejs HTTP warstwy serwerowej jest testowany za pomocą FastAPI TestClient [68]. Narzędzie umożliwia wykonywanie żądań do aplikacji FastAPI wewnątrz procesu testowego, bez uruchamiania osobnego procesu Uvicorn ani aplikacji Electron.

Dla warstwy interfejsu użytkownika nie przewidziano automatycznych testów jednostkowych, komponentowych ani testów pełnego przepływu. Kontrolę techniczną tej części aplikacji zapewniają TypeScript działający w trybie ścisłym [69] oraz proces budowania wykorzystujący esbuild i Vite.

Scenariusze wymagające rzeczywistych integracji są uruchamiane oddzielnie. Wymagają działającego DaVinci Resolve, mostu Lua, przygotowanego projektu testowego, a w wybranych przypadkach również dostępu do usług modeli.

Do częściowej automatyzacji testów integracyjnych wykorzystywane jest DaVinci Resolve Scripting API [63]. Edytor nie zapewnia jednak dedykowanego środowiska do testowania całej aplikacji ani automatycznej oceny jakości montażu. Z tego względu pełny przepływ działania oraz rezultat audiowizualny są dodatkowo sprawdzane ręcznie, co opisano szerzej w rozdziale 6.

5. Implementacja prototypu systemu Cuttie-ai

Tworzenie aplikacji wykorzystujących duże modele językowe jest stosunkowo nowym i dynamicznie rozwijającym się obszarem inżynierii oprogramowania. Dostępne biblioteki zapewniają mechanizmy komunikacji z modelami oraz odbierania odpowiedzi o określonej strukturze. Istotnym elementem implementacji jest zatem określenie zakresu informacji opisujących zadanie i środowisko, które powinny zostać przekazane agentowi przed podjęciem decyzji. Przejście od pojedynczego wywołania modelu do działającego systemu wymagało przy tym podjęcia szeregu decyzji projektowych i rozwiązania problemów implementacyjnych związanych między innymi z niedeterministycznym charakterem działania modeli językowych.

Zrealizowane rozwiązanie ma postać lokalnego prototypu współpracującego z DaVinci Resolve 21 Free. Opis implementacji obejmuje organizację kodu źródłowego oraz najważniejsze mechanizmy składające się na reprezentatywną ścieżkę działania. Przedstawiono przejście od wskazania i analizy materiałów, przez przygotowanie kontekstu i interpretację polecenia, po utworzenie oraz kontrolowane wykonanie zatwierdzonego planu edycji.

5.1. Zakres implementacji prototypu

Zaimplementowane rozwiązanie ma postać wtyczki do DaVinci Resolve 21 Free. Bezpłatna wersja edytora ogranicza możliwość sterowania programem za pomocą skryptów zewnętrznych. Z tego względu interfejs użytkownika i warstwa serwerowa działają poza edytorem, natomiast dostęp do projektu zapewnia most Lua działający w jego środowisku. Rozpoczęcie pracy z projektem wymaga otwarcia DaVinci Resolve oraz wybrania odpowiedniej pozycji w menu skryptów. Uruchomiony w ten sposób most wykonawczy odczytuje stan projektu oraz odbiera i wykonuje zatwierdzone operacje. Bez tego komponentu dostępne pozostają funkcje niewymagające połączenia z edytorem, w tym analiza plików lokalnych.

Interfejs użytkownika obejmuje cztery główne obszary: rozmowę, materiały, stan połączenia z DaVinci Resolve oraz ustawienia. Materiały mogą zostać wskazane na dysku albo pobrane z biblioteki otwartego projektu (*Media Pool*). Obsługiwane są pliki wideo, audio, obrazy i dokumenty tekstowe. Wyniki analizy opisują zawartość materiałów w postaci możliwej do wykorzystania podczas planowania montażu. Interfejs pozwala je przeglądać oraz poprawiać wybrane informacje, takie jak podsumowanie, rola materiału, transkrypcja i opisy fragmentów.

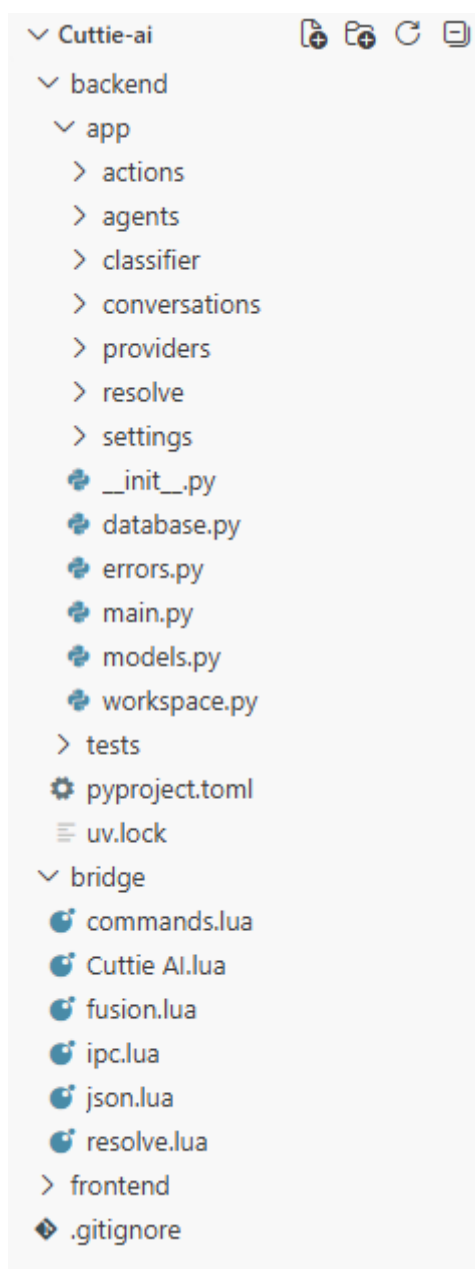
Polecenia w języku naturalnym są interpretowane z uwzględnieniem historii rozmowy, wyników analizy oraz stanu aktywnej osi czasu. Zakres zaimplementowanych działań obejmuje tworzenie nowych osi czasu i wykonywanie wybranych zmian w istniejącym montażu. Program może dobierać i rozmieszczać fragmenty, dodawać muzykę, napisy i lektora, stosować efekty oraz zmieniać proporcje obrazu. Obsługiwane są również wyszukiwanie wypowiedzi, usuwanie ciszy i renderowanie. Każda ścieżka prowadząca do zmiany projektu kończy się utworzeniem typowanego planu edycji (*EditPlan*). Plan zawiera wyłącznie operacje obsługiwane przez aplikację, podlega walidacji i jest przedstawiany użytkownikowi do zatwierdzenia albo odrzucenia. Dopiero zaakceptowane polecenia są przekazywane do mostu i wykonywane w DaVinci Resolve.

Program wykorzystuje jeden wspólny obszar roboczy i umożliwia przechowywanie jednego planu oczekującego. Historia rozmów i ustawienia są zapisywane trwale. Materiały, wyniki analizy, ręczne poprawki oraz aktywny plan należą natomiast do stanu bieżącej sesji.

Wybrane funkcje mogą korzystać ze skonfigurowanych usług modeli działających poza komputerem użytkownika.

5.2. Organizacja kodu źródłowego i zakres opisu implementacji

Organizacja repozytorium odzwierciedla podział odpowiedzialności pomiędzy głównymi częściami systemu. Kod umieszczono w trzech katalogach odpowiadających aplikacji desktopowej, lokalnej warstwie serwerowej oraz mostowi uruchamianemu wewnątrz DaVinci Resolve. Oddzielnie przechowywane są testy oraz skrypty służące do uruchamiania i weryfikacji rozwiązania. Strukturę najważniejszych katalogów i plików przedstawia Rysunek 11.



Rysunek 11. Struktura katalogów i najważniejszych plików projektu. Źródło: opracowanie własne.

Katalog frontend zawiera aplikację desktopową. Proces główny środowiska Electron odpowiada między innymi za cykl życia programu, wybór plików oraz komunikację z lokalną warstwą serwerową. Warstwa preload udostępnia ograniczony interfejs komunikacyjny, natomiast renderer zawiera interfejs użytkownika zbudowany w bibliotece React. W części shared umieszczono współdzielone definicje tras, typów danych i komunikatów.

Główna logika aplikacji znajduje się w katalogu backend/app. Moduł classifier odpowiada za rejestrowanie i analizę materiałów. W module agents zaimplementowano interpretację polecenia oraz sterowanie procesem planowania, natomiast actions buduje i waliduje plan edycji. Moduł resolve obsługuje stan edytora oraz komunikację z mostem. Pozostałe części odpowiadają między innymi za integrację z usługami modeli, ustawienia oraz historię rozmów. Wspólne kontrakty danych znajdują się w pliku models.py, a stan bieżącej sesji jest przechowywany przez workspace.py.

Katalog bridge zawiera kod Lua wykonywany w środowisku DaVinci Resolve. Obejmuje skrypt uruchamiający, obsługę plikowego kanału komunikacji, rozpoznawanie poleceń oraz ich odwzorowanie na operacje udostępniane przez Scripting API i Fusion API. Deterministyczne testy warstwy serwerowej umieszczono w katalogu backend/tests, natomiast scenariusze wymagające rzeczywistych integracji są uruchamiane za pomocą oddzielnych skryptów weryfikacyjnych.

Opis implementacji nie stanowi kompletnej dokumentacji kodu źródłowego. Do szczegółowego przedstawienia wybrano mechanizmy realizujące główne etapy przepływu, definiujące granice pomiędzy komponentami albo oddzielające decyzje modeli od operacji wykonywanych deterministycznie. Pominięto powtarzalną obsługę tras HTTP, konfigurację narzędzi, proste komponenty interfejsu, modele pomocnicze oraz warianty poszczególnych poleceń montażowych. Elementy te pozostają częścią kompletnej implementacji, lecz ich przedstawienie nie jest konieczne do prześledzenia działania rozwiązania.

Kolejność dalszego opisu odpowiada przepływowi danych, a nie układowi katalogów. Rozpoczyna się od analizy materiałów i przygotowania kontekstu, a kończy przekazaniem zatwierdzonego planu do warstwy wykonawczej.

5.3. Implementacja analizy materiałów i przygotowania kontekstu

Pierwszym etapem opisywanego przepływu jest przygotowanie reprezentacji materiałów. Jej model logiczny przedstawiono w części poświęconej architekturze systemu. Na poziomie implementacji proces obejmuje rejestrację pliku, wybór odpowiedniej ścieżki analizy, połączenie wyników cząstkowych oraz ograniczenie ilości danych przekazywanych do procesu planowania.

Pliki wskazane na dysku oraz materiały pobrane z biblioteki projektu trafiają do wspólnego modułu klasyfikacji. Podczas rejestracji ścieżka jest normalizowana, co pozwala rozpoznać ponowne wskazanie tego samego pliku. System zapisuje również jego rozmiar i czas ostatniej modyfikacji. Jeżeli materiał pochodzi z kilku źródeł, informacje o pochodzeniu oraz odwołania do elementów *Media Pool* są łączone w jednym wpisie. Zmiana właściwości pliku powoduje natomiast usunięcie wcześniejszego wyniku analizy.

Analiza materiałów jest wykonywana sekwencyjnie i chroniona przed równoczesnym uruchomieniem kolejnej operacji tego rodzaju. Błąd dotyczący jednego pliku nie przerywa przetwarzania pozostałych elementów. Poprawne wyniki są zapisywane w stanie bieżącej sesji, natomiast informacje o niepowodzeniach trafiają do odpowiedzi całej operacji.

Rodzaj materiału jest sprawdzany ponownie bezpośrednio przed rozpoczęciem analizy. Pozwala to uwzględnić zmianę pliku, która mogła nastąpić po jego wcześniejszym

zarejestrowaniu. Na podstawie rozpoznanego rodzaju wybierana jest ścieżka przeznaczona dla dokumentu tekstowego, obrazu statycznego albo materiału czasowego, czyli wideo lub audio. Sposób realizacji tego podziału przedstawiono w Listing 1.

Listing 1. Wybór ścieżki analizy materiału. Źródło: opracowanie własne.

```
def _classify_one(
    self, asset: MediaAsset, mode: AnalysisMode, warnings: list[str] |
    None = None
) -> MediaClassification:
    kind, metadata = probe_media(Path(asset.path))
    if kind != asset.kind:
        asset = asset.model_copy(update={"kind": kind})
        self._workspace.media[asset.id] = asset
    if kind == "text":
        return self._classify_text(asset, metadata, mode, warnings)
    with TemporaryDirectory(prefix="cuttie-classifier-") as tmp:
        tmp_dir = Path(tmp)
        if kind == "image":
            return self._classify_image(asset, metadata, mode, tmp_dir)
        return self._classify_timed(asset, kind, metadata, mode,
        tmp_dir)
```

Dokument tekstowy jest odczytywany z uwzględnieniem konfigurowalnego limitu długości. Dla obrazu wykonywane są normalizacja formatu, pomiar wybranych cech jakościowych oraz opis zawartości. Materiały wideo i audio korzystają ze wspólnej ścieżki przeznaczonej dla danych posiadających przebieg czasowy. Jeżeli materiał zawiera ścieżkę dźwiękową, zostaje ona wyodrębniona w postaci pliku roboczego, a następnie poddana transkrypcji i analizie cech sygnału. Dla wideo wykrywano również granice scen i wybierane klatki przeznaczone do dalszej analizy. Liczba próbek zależy od wybranego trybu analizy. Pliki pośrednie są zapisywane w katalogu tymczasowym i usuwane po zakończeniu przetwarzania.

Wyniki cząstkowe są następnie łączone we wspólną reprezentację `MediaClassification`. W przypadku wideo i audio tworzone są przedziały `Moment` o długości nieprzekraczającej dziesięciu sekund. Dla materiału wideo podstawę podziału stanowią granice wykrytych scen. Każdy przedział może zawierać wypowiedziane słowa, opis obrazu, znaczniki oraz lokalne cechy dźwięku. Powstała reprezentacja jest zapisywana w obszarze roboczym bieżącej sesji.

Pełny wynik analizy nie jest przekazywany bezpośrednio do modelu językowego. Po otrzymaniu polecenia system wybiera fragmenty, które mogą być związane z oczekiwanym montażem. Aktualna implementacja wykorzystuje deterministyczne dopasowanie leksykalne. Wyrazy występujące w poleceniu są porównywane z nazwami plików, transkrypcjami, opisami obrazu oraz znacznikami przypisanymi do poszczególnych momentów. Kandydaci są porządkowani według liczby znalezionych wystąpień.

Dopasowanie leksykalne jest uzupełniane regułami zapobiegającymi pominięciu materiałów wskazanych bezpośrednio przez użytkownika. Jeżeli taki plik został wcześniej przeanalizowany, jego reprezentacja zostaje dołączona nawet wtedy, gdy żadne słowo nie zostało dopasowane. Dla przeanalizowanego materiału czasowego zarejestrowanego jako plik dodany przez użytkownika zachowywany jest ponadto kandydat obejmujący całe nagranie. Mechanizm ten przedstawiono na Listing 2.

Listing 2. Dobór fragmentów materiałów przekazywanych do procesu planowania.
Źródło: opracowanie własne.

```
def _build_candidates(self, message, limit):
    attached_ids = [
        mid for mid, asset in self._workspace.media.items() if
        "attached_file" in asset.origins
    ]
    result = list(self._classifier.find_candidates(message,
limit=limit))
    matched_media_ids = {candidate.media_id for candidate in result}
    for mid in attached_ids:
        if mid in matched_media_ids:
            continue
        classification = self._workspace.classifications.get(mid)
        if classification is not None:
            result.extend(self._candidates_for_classification(classifica
tion))
    if not result:
        for classification in self._workspace.classifications.values():
            result.extend(self._candidates_for_classification(classifica
tion))

    result = self._with_full_material(result, attached_ids)
    if len(result) <= limit:
        return result
    attached_set = set(attached_ids)
    attached_cands = [c for c in result if c.media_id in attached_set]
    other_cands = [c for c in result if c.media_id not in attached_set]
    return attached_cands + other_cands[: max(0, limit -
len(attached_cands))]
```

Jeżeli dopasowanie nie zwróci żadnego wyniku, lista jest uzupełniana na podstawie dostępnych klasyfikacji. Ustawienia określają docelową liczbę kandydatów, jednak materiały dodane bezpośrednio przez użytkownika mogą spowodować przekroczenie tego limitu. Zapobiega to odrzuceniu materiałów, które użytkownik wskazał do wykorzystania w montażu.

Wybrane fragmenty są łączone z pozostałymi informacjami potrzebnymi podczas bieżącej interakcji. Warstwa serwerowa pobiera aktualny stan osi czasu, przygotowuje skrócone opisy zarejestrowanych materiałów oraz tworzy oddzielną listę plików audio. Dołącza również ograniczoną historię rozmowy i informacje o planie oczekującym na zatwierdzenie. Zestawienie tych danych w strukturze TurnContext przedstawiono w Listing 3.

Listing 3. Budowanie kontekstu bieżącej interakcji. Źródło: opracowanie własne.

```
def _build_context(self, message, limits) -> TurnContext:
    active_timeline = self._load_active_timeline()
    candidates = self._build_candidates(message, limits.candidate_top_n)
    media = [self._media_brief(a) for a in
self._workspace.media.values()]
```

```

        music_candidates = [
            self._music_candidate(a) for a in self._workspace.media.values()
        ]
        if a.kind == "audio"
    ]
    history = [
        ChatMessage(role=m.role, content=m.content)
        for m in self._conversations.tail(
            self._workspace.conversation_id, limits.max_history_messages
        )
    ]
    if self._workspace.conversation_id else []
    return TurnContext(
        message=message,
        history=history,
        media=media,
        candidates=candidates,
        music_candidates=music_candidates,
        pending_plan_summary=self._pending_plan_summary(),
        active_timeline=active_timeline,
    )

```

Skrócone opisy materiałów nie zawierają ścieżek systemowych, danych identyfikujących wersję pliku ani pełnych wyników klasyfikacji. Zachowują identyfikator, nazwę, rodzaj, czas trwania, podsumowanie, sugerowaną rolę i informację o dostępności transkrypcji. W przypadku dokumentów tekstowych mogą również zawierać ich treść ograniczoną skonfigurowanym limitem. Ustawienia ograniczają liczbę wiadomości historycznych i wyznaczają docelową liczbę kandydatów, którą mogą przekroczyć materiały dodane bezpośrednio przez użytkownika. Stan aktywnej osi czasu jest odczytywany przed każdą interakcją. Jeżeli edytor jest niedostępny, kontekst powstaje bez informacji o osi czasu, a system zapisuje odpowiednie ostrzeżenie.

Tak przygotowany kontekst stanowi wejście dla **Orchestrator**, którego implementację oraz sposób sterowania pozostałymi rolami przedstawiono w następnym podrozdziale.

5.4. Implementacja interpretacji polecenia i sterowania rolami

Obiekt `TurnContext` przygotowany na podstawie wyników analizy materiałów jest przekazywany w metodzie `_run_turn` do roli **Orchestrator**. Do wywołania dołączany jest rejestr operacji obsługiwanych przez aplikację. Sposób przekazania danych do programu DSP-y przedstawiono w Listing 4.

Listing 4. Przekazanie kontekstu bieżącej interakcji do roli Orchestratora. Źródło: opracowanie własne.

```

def decide(lm, context: TurnContext, available_actions: list[dict]) ->
    DelegationDecision:

    program = _get_program()
    with dspy.context(lm=lm):
        prediction = program(
            message=context.message,

```

```

        history=context.history,
        media=context.media,
        candidates=context.candidates,
        music_candidates=context.music_candidates,
        available_actions=available_actions,
        allowed_effects=list(ALLOWED_FREE_EFFECTS),
        pending_plan_summary=context.pending_plan_summary,
        active_timeline=context.active_timeline,
    )
    return prediction.decision

```

Katalog `available_actions` zawiera opisy dostępnych operacji i ich parametrów. Model nie otrzymuje bezpośredniego dostępu do metod warstwy serwerowej ani warstwy wykonawczej. Lista `allowed_effects` ogranicza natomiast wybór gotowych efektów do zbioru obsługiwanego przez aplikację.

Wynikiem wywołania jest obiekt `DelegationDecision`. Zawiera on rozpoznane wymagania, ewentualne pytanie doprecyzowujące oraz typowane opisy dalszych działań. Mogą one dotyczyć przygotowania nowego montażu, zmiany aktywnej osi czasu, wyszukania fragmentów, renderowania albo wykonania pojedynczej operacji. Wynik **Orchestrator** nie zawiera poleceń `EditPlan` i nie jest przekazywany do DaVinci Resolve.

Model danych `DelegationDecision` zabrania dodawania pól spoza zdefiniowanego schematu. Zgodność strukturalna wyniku nie przesądza jednak o poprawności wybranej ścieżki, którą kontroluje kod klasy `AgentService`. Jeżeli komunikatu modelu nie można przekształcić do wymaganego typu, bieżąca interakcja kończy się stanem `blocked`.

Metoda `_run_turn` odczytuje poszczególne pola decyzji i jawnie wybiera dalszy przebieg. Pytanie doprecyzowujące powoduje zwrócenie stanu `needs_user`. Niewykonalne wymaganie oznaczone jako obowiązkowe zatrzymuje planowanie, natomiast pominięcie elementu opcjonalnego jest zapisywane jako ostrzeżenie. Jeżeli rozpoznano kilka wymaganych operacji prostych, metoda zwraca pytanie o ich kolejność i nie przygotowuje planu w tej samej interakcji. Po doprecyzowaniu może utworzyć plan dla jednej operacji.

Pełna sekwencja wyspecjalizowanych ról jest wykorzystywana podczas tworzenia nowego montażu. Steruje nią metoda `_run_designers`. Role odpowiedzialne za narrację i dźwięk są uruchamiane warunkowo, zależnie od decyzji orkiestratora. Ich wyniki trafiają następnie do roli projektującej oś czasu. Po określeniu segmentów kod obsługuje żądania efektów niestandardowych. Wygaszenia, ruch kadru i wybrane zmiany kolorystyczne są odwzorowywane deterministycznie. Dla pozostałych żądań uruchamiana jest rola projektanta efektów. Jeżeli efekt dotyczy kilku segmentów, jego projekt jest przygotowywany oddzielnie dla każdego z nich.

Każda rola otrzymuje ograniczony opis zadania zamiast całego kontekstu interakcji. Rola odpowiedzialna za narrację korzysta z podsumowań i fragmentów transkrypcji. Rola dźwiękowa otrzymuje cel, dodatkowe wskazówki oraz listę kandydatów muzycznych. Projektant osi czasu wykorzystuje wybrane fragmenty oraz wcześniejsze wyniki dotyczące narracji i muzyki. Projektowanie pozostałych efektów odbywa się na podstawie katalogu narzędzi Fusion odczytanego z bieżącego środowiska DaVinci Resolve. Role nie wywołują się wzajemnie, a ich kolejność ustala kod napisany w Pythonie.

Wyniki ról są zwracane za pomocą wspólnej struktury `SpecialistOutcome`. Stan `completed` oznacza dostępność poprawnie zbudowanego wyniku. Stany `needs_context` i

needs_user powodują przedstawienie użytkownikowi pytania, natomiast unsupported oraz failed zatrzymują dalsze planowanie.

Otrzymanie danych zgodnych ze schematem jest uzupełniane kontrolą merytoryczną wykonywaną przez kod. Warstwa serwerowa sprawdza, czy wskazane materiały istnieją, odpowiadają wymaganemu rodzajowi oraz należą do zbioru kandydatów przekazanych roli. Zakresy czasowe muszą mieścić się w długości pliku i w przedziale udostępnionym podczas wyboru fragmentów. Osobno weryfikowany jest wybór materiału dźwiękowego oraz realizacja wymaganych efektów. Format osi czasu ustalony przez aplikację zastępuje wartości formatu zwrócone przez model. Kontrole te dotyczą wyników pracy ról i poprzedzają walidację końcowego planu poleceń.

Po uzyskaniu poprawnych wyników ActionService buduje i waliduje obiekt EditPlan. Dopiero wtedy **PlanReviewer** ocenia zgodność przygotowanego rozwiązania z poleceniem użytkownika. Recenzent nie modyfikuje planu. Zwraca typowany wynik zawierający werdykt, opis problemów oraz wskazanie roli odpowiedzialnej za ewentualną poprawkę. Sterowanie tym procesem przedstawiono w Listing 5.

Listing 5. Sterowanie oceną planu i pojedynczą falą poprawy. Źródło: opracowanie własne.

```
def _produce_montage(self, decision, context, fmt, resolved_render,
warnings) -> AgentTurnResult:
    artifacts = self._run_designers(decision, context, fmt, owner=None,
prev=None, guidance=None)
    if isinstance(artifacts, AgentTurnResult):
        return artifacts
    narrative, audio, timeline_design = artifacts[:3]
    assembled = self._assemble_and_review(decision, artifacts,
resolved_render, warnings)
    if isinstance(assembled, AgentTurnResult):
        return assembled
    plan, review = assembled

    owner = _blocking_owner(review)
    if review.verdict != "repair" or owner not in ("narrative", "audio",
"timeline", "effect"):
        return _review_objection(plan, review, warnings)

    guidance = "; ".join(i.message for i in review.issues if i.severity
== "blocking")
    artifacts2 = self._run_designers(
        decision, context, fmt, owner=owner, prev=(narrative, audio,
timeline_design), guidance=guidance
    )
    if isinstance(artifacts2, AgentTurnResult):
        return artifacts2
    assembled2 = self._assemble_and_review(decision, artifacts2,
resolved_render, warnings)
    if isinstance(assembled2, AgentTurnResult):
```

```
        return assembled2
    plan, review2 = assembled2
    return _review_objection(plan, review2, warnings)
```

Automatyczna poprawka jest wykonywana tylko dla problemów przypisanych do jednej z czterech ról projektowych. Uwagi blokujące są przekazywane jako dodatkowa wskazówka. Poprawne wyniki wcześniejszych ról są przy tym zachowywane. Zmiana narracji albo muzyki powoduje jednak kolejne uruchomienie **TimelineDesigner** oraz ponowne przygotowanie zależnych efektów. Korekta osi czasu również wymaga ich odtworzenia, natomiast błąd przypisany do efektu nie wpływa na narrację, muzykę ani układ montażu.

Kod dopuszcza jedną taką falę poprawy. Po jej zakończeniu warstwa serwerowa buduje nową wersję planu, a następnie ją waliduje i ocenia. Jeżeli zastrzeżenia pozostają, kolejne wywołania nie są wykonywane automatycznie. System przedstawia uwagi użytkownikowi i oczekuje na jego decyzję. Do warstwy wykonawczej nie trafia bezpośrednia odpowiedź żadnej z ról. Podstawą dalszego działania pozostaje typowany i zweryfikowany **EditPlan**.

5.5. Implementacja typowanego planu edycji

Ustrukturyzowane wyniki pracy ról są przekazywane do deterministycznej części warstwy serwerowej skupionej w **ActionService**. Modele uczestniczą w wyborze materiałów, ustalaniu ich kolejności i określaniu oczekiwanych zmian. Nie tworzą jednak poleceń wykonywanych bezpośrednio w **DaVinci Resolve**. Za przekształcenie decyzji montażowych w plan wykonawczy odpowiada kod aplikacji.

Sposób przygotowania planu zależy od rodzaju działania. Metoda **build_edit_plan** obsługuje utworzenie nowego montażu na podstawie projektu osi czasu oraz opcjonalnych danych dotyczących narracji, muzyki, efektów i renderowania. Operacje rejestrowane jako pojedyncze działania, takie jak dodanie napisów, muzyki lub lektora oraz renderowanie, korzystają z **build_action_plan**. Zmiany istniejących klipów, w tym zastosowanie efektu, są przekształcane przez **build_edit_in_place_plan**, natomiast zmianę proporcji obrazu i przenikanie obsługują oddzielne konstruktory. Każda z tych ścieżek prowadzi do utworzenia obiektu **EditPlan**.

W przypadku nowego montażu opis osi czasu jest przekształcany w listę segmentów zawierających wybór materiału, zakres źródłowy oraz ustawienia dotyczące efektów i zachowania mowy. Kolejność segmentów określa ich układ, natomiast czasy umieszczenia na osi i łączna długość montażu są obliczane deterministycznie przez funkcję **assemble_new_timeline**. Listing 6 przedstawia główny fragment budowania planu, obejmujący utworzenie uporządkowanej listy poleceń oraz umieszczenie jej w obiekcie **EditPlan**. Obsługę opcjonalnych efektów i renderowania pominięto na listingu, ponieważ nie zmienia ona przedstawionego sposobu konstruowania planu.

Listing 6. Deterministyczne utworzenie poleceń i końcowej struktury **EditPlan. Źródło: opracowanie własne.**

```
result: BuildResult = assemble_new_timeline(
    ctx,
    name=timeline_design.name,
    width=timeline_design.width,
    height=timeline_design.height,
```

```

        fps=timeline_design.fps,
        segments=_segment_plans(timeline_design),
        music=music,
        generate_subtitles=timeline_design.generate_subtitles,
        fill=timeline_design.fill,
    )

    commands = list(result.commands)
    timeline_ref = next(
        command
        for command in commands
        if command.type == "create_timeline"
    ).timeline_ref
    all_warnings = list(result.warnings) + list(warnings)

    ...

    return EditPlan(
        id=uuid.uuid4().hex,
        summary=_compose_summary(
            result.summary, narrative_design, resolved_render_settings
        ),
        target_project_id=result.target_project_id,
        target_timeline_id=result.target_timeline_id,
        estimated_duration_s=result.estimated_duration_s,
        commands=commands,
        warnings=all_warnings,
    )

```

Funkcja `assemble_new_timeline` ustala kolejność operacji wymaganych do utworzenia osi czasu. Przygotowuje między innymi polecenia utworzenia osi, ustawienia jej formatu, dodania materiałów, muzyki i napisów. Obliczenia dotyczące czasu rozpoczęcia klipów, długości segmentów oraz dopasowania ścieżki dźwiękowej są wykonywane przez kod deterministyczny. W tej części odbywa się również przygotowanie plików pomocniczych, jeżeli materiał wymaga przekształcenia przed dodaniem do projektu.

Obiekt `EditPlan` zawiera identyfikator planu, jego podsumowanie, docelowy projekt i oś czasu, przewidywaną długość montażu, ostrzeżenia oraz uporządkowaną listę poleceń. Polecenia należą do zamkniętego zbioru typów rozróżnianych przez pole `type`. Obejmuje on między innymi tworzenie osi czasu, dodawanie klipów i napisów, stosowanie efektów oraz renderowanie.

Zależności pomiędzy planowanymi elementami są reprezentowane przez lokalne odwołania. Polecenie utworzenia osi czasu definiuje odwołanie `timeline_ref`, z którego korzystają następne operacje. Dodany klip otrzymuje analogiczne odwołanie `item_ref`, umożliwiające wskazanie go przed uzyskaniem rzeczywistego identyfikatora z DaVinci Resolve. Elementy istniejące już w projekcie są wskazywane za pomocą ich właściwych identyfikatorów.

Walidacja planu odbywa się na dwóch poziomach. Poszczególne modele danych sprawdzają typy i zakresy własnych parametrów, natomiast walidator klasy `EditPlan` kontroluje zależności pomiędzy poleceniami. Weryfikuje między innymi unikalność odwołań, kolejność ich

definiowania i wykorzystywania oraz położenie polecenia renderowania. Na Listing 7 przedstawiono kontrolę odwołań `timeline_ref` oraz położenia polecenia renderowania. Analogiczne reguły dotyczące `command_ref` i `item_ref` pominięto, aby zachować czytelność fragmentu.

Listing 7. Wybrane reguły walidacji kolejności poleceń planu edycji. Źródło: opracowanie własne.

```
timeline_refs: set[str] = set()

for command in self.commands:
    target = getattr(command, "timeline", None)

    if (
        target is not None
        and target.timeline_ref is not None
        and target.timeline_ref not in timeline_refs
    ):
        raise ValueError(
            f"timeline_ref '{target.timeline_ref}' "
            "uzyty przed create_timeline"
        )

    if isinstance(command, CreateTimelineCommand):
        if command.timeline_ref in timeline_refs:
            raise ValueError(
                f"zduplikowany timeline_ref "
                f"'{command.timeline_ref}'"
            )
        timeline_refs.add(command.timeline_ref)

for index, command in enumerate(self.commands):
    if (
        command.type == "render_timeline"
        and index != len(self.commands) - 1
    ):
        raise ValueError(
            "render_timeline może być wyłącznie "
            "ostatnią komendą planu"
        )
```

Błąd naruszający strukturę planu uniemożliwia utworzenie obiektu `EditPlan`. Ostrzeżenia dotyczące warunków, które nie blokują wykonania, są zapisywane w planie i mogą zostać przedstawione użytkownikowi. Poprawny plan trafia do stanu bieżącej sesji jako plan oczekujący. Na tym etapie nie wprowadza jeszcze żadnych zmian w projekcie DaVinci Resolve.

5.6. Implementacja zatwierdzania i wykonania planu

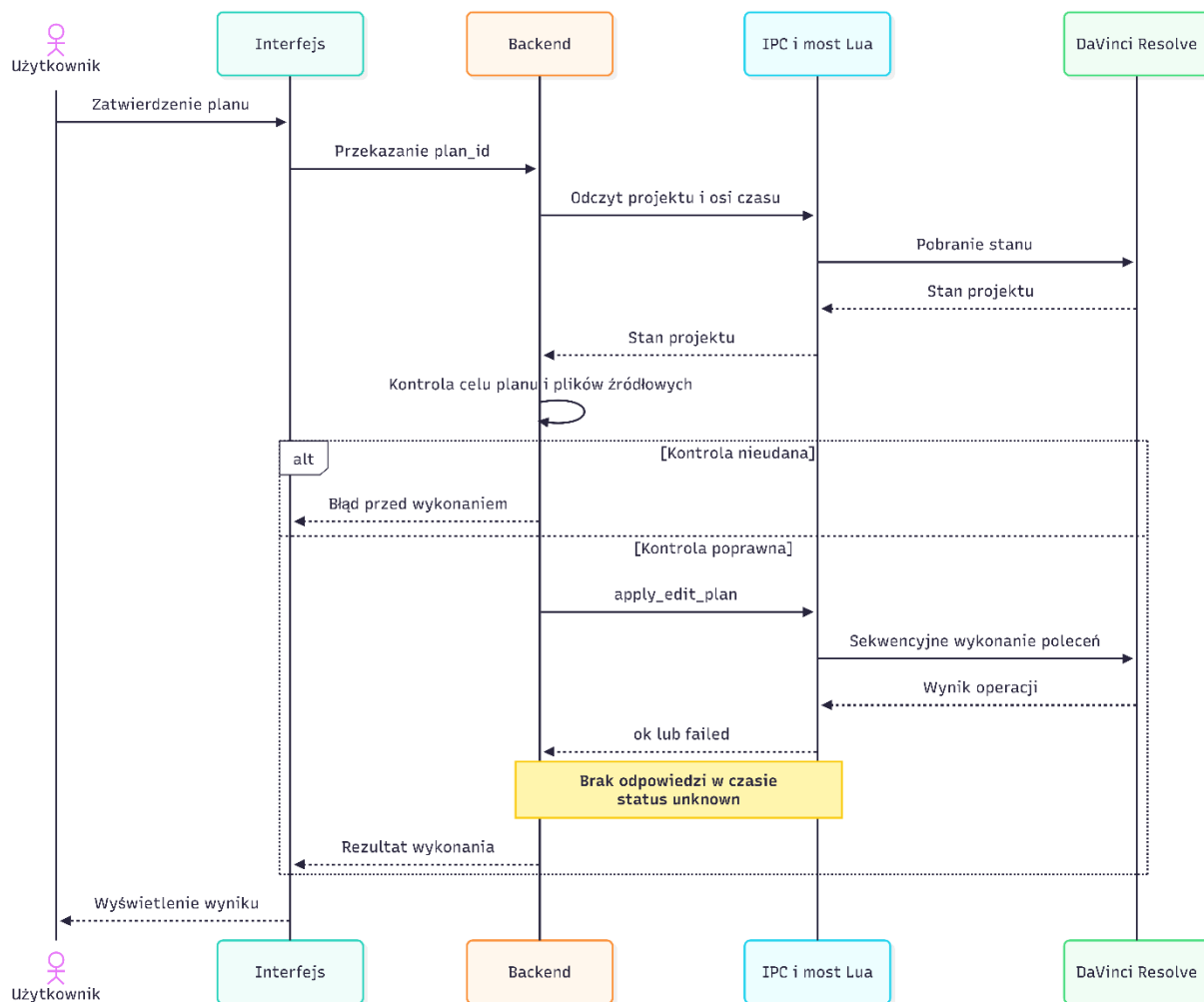
Plan przygotowany w poprzednim etapie jest przechowywany w stanie bieżącej sesji warstwy serwerowej. Aplikacja stacjonarna przedstawia jego podsumowanie, ostrzeżenia oraz uporządkowaną listę poleceń. Użytkownik może odrzucić cały plan albo zatwierdzić go i rozpocząć wykonanie. Odrzucenie usuwa plan bez komunikacji z DaVinci Resolve.

Podczas zatwierdzania frontend przesyła wyłącznie identyfikator `plan_id`, a nie ponownie pełną zawartość planu. Warstwa serwerowa pobiera właściwy obiekt z pola `Workspace.pending_plan` i porównuje jego identyfikator z wartością otrzymaną w żądaniu. Zapobiega to wykonaniu planu, który został zastąpiony po wyświetleniu go w interfejsie. Operacja jest chroniona blokadą wspólną dla działań dotyczących planu. Listing 8 przedstawia kontrolę identyfikatora planu oraz rozpoczęcie jego wykonania.

Listing 8. Kontrola identyfikatora i rozpoczęcie wykonania planu oczekującego. Źródło: opracowanie własne.

```
def apply_pending(self, plan_id: str) -> ApplyOutcome:
    with self._workspace.plan_transaction():
        plan = self._workspace.pending_plan
        if plan is None:
            raise NoPendingPlanError("Brak oczekującego planu do
zastosowania.")
        if plan.id != plan_id:
            raise PlanMismatchError(
                f"Podany plan_id nie odpowiada oczekujacemu planowi
(oczekiwany: {plan.id})."
            )
        outcome = self.execute_plan(plan)
        retryable = outcome.status != "ok" and not
outcome.completed_command_refs
        self._workspace.pending_plan = plan if retryable else None
        self._workspace.last_apply_result = outcome
    return outcome
```

Przed przekazaniem planu do mostu `ResolveService` ponownie sprawdza jego cel. Identyfikator otwartego projektu musi odpowiadać polu `target_project_id`. Jeżeli operacje dotyczą istniejącej osi czasu, warstwa serwerowa sprawdza również jej obecność w projekcie. Kontrolowane są także pliki źródłowe. Ich ścieżki, rozmiary i czasy ostatniej modyfikacji są porównywane z wartościami zapisanymi podczas budowania planu. Wykrycie zmiany zatrzymuje wykonanie przed rozpoczęciem modyfikowania projektu. Sekwencję kontroli oraz wykonania zatwierdzonego planu przedstawiono na rysunku Rysunek 12.



Rysunek 12. Diagram sekwencji przedstawiający kontrolę i wykonanie zatwierdzonego planu edycji. Źródło: opracowanie własne.

Poprawny plan jest umieszczany w jednym poleceniu `apply_edit_plan` i przekazywany przez plikowy kanał IPC. Polecenie zawiera wersję schematu, własny identyfikator oraz pełny obiekt `EditPlan`. Most ponownie sprawdza typ żądania, podstawową strukturę planu, wymagane pola poleceń i identyfikator projektu. Do mostu napisanego w języku Lua nie trafia kod ani swobodne polecenie przeznaczone do interpretacji. Most odbiera wyłącznie zwalidowane dane przypisane do pól zdefiniowanych typów poleceń, w tym wartości tekstowe, takie jak nazwy lub treść napisów.

Typy poleceń są odwzorowywane na jawnie zdefiniowane funkcje korzystające z Scripting API albo Fusion API. Lokalne odwołania `timeline_ref` i `item_ref` są podczas wykonania wiązane z obiektami utworzonymi w DaVinci Resolve. Pozwala to kolejnym poleceniom wskazywać oś czasu lub klip utworzony wcześniej w ramach tego samego planu. Listing 9 przedstawia sekwencyjne wykonywanie poleceń oraz zatrzymanie planu po wystąpieniu pierwszego błędu.

Listing 9. Sekwencyjne wykonanie poleceń i zatrzymanie planu po wystąpieniu błędu.
Źródło: opracowanie własne.

```
for _, cmd in ipairs(plan.commands or {}) do
    if M.render_heartbeat then
        M.render_heartbeat("applying", nil)
    end
    local handler = apply_handlers[cmd.type]
    if not handler then
        return {
            completed_command_refs = ctx.completed,
            failed_command_ref = cmd.command_ref,
            error = "bridge nie obsługuje komendy: " .. tostring(cmd.type),
            data = ctx.data,
        }
    end
    local ok, err = pcall(handler, ctx, cmd)
    if not ok then
        return {
            completed_command_refs = ctx.completed,
            failed_command_ref = cmd.command_ref,
            error = tostring(err),
            data = ctx.data,
        }
    end
    ctx.completed[#ctx.completed + 1] = cmd.command_ref
end
return { completed_command_refs = ctx.completed, data = ctx.data }
end
```

Polecenia są wykonywane w kolejności zapisanej w planie. Po zakończeniu operacji jej `command_ref` trafia do listy `completed_command_refs`. Pierwszy błąd zatrzymuje dalsze wykonanie, a odpowiedź wskazuje polecenie, przy którym wystąpił problem. Operacje zakończone wcześniej pozostają w projekcie, ponieważ system nie zapewnia transakcyjnego wycofania zmian.

Rezultat ma postać `ApplyOutcome` ze stanem `ok`, `failed` albo `unknown`. Ostatni z nich oznacza brak wiarygodnej odpowiedzi w wymaganym czasie i nie powoduje automatycznego ponowienia planu. Warstwa serwerowa zapisuje również listę ukończonych poleceń, identyfikator operacji zakończonej błędem i jego opis.

Plan jest usuwany ze stanu oczekującego po wyniku `ok` oraz po rezultacie zawierającym co najmniej jedno ukończone polecenie. Zapobiega to powtórzeniu przeprowadzonych operacji. Jeżeli wynik `failed` nie zawiera ukończonych poleceń, plan pozostaje dostępny do ponownego zatwierdzenia. Jeżeli wynik `unknown` również nie zawiera ukończonych poleceń, warstwa serwerowa zachowuje plan, jednak interfejs nie pozwala wykonać go ponownie. Brak odpowiedzi nie potwierdza bowiem, że żadna operacja nie została przeprowadzona. Rezultat jest zapisywany w polu `last_apply_result` i przedstawiany użytkownikowi.

6. Weryfikacja założeń projektowych i analiza wyników

Podczas implementacji aplikacji pojawiło się wiele problemów, których nie można było przewidzieć wyłącznie na podstawie dokumentacji i przyjętej architektury. Pierwsza grupa dotyczyła modeli wykorzystywanych do analizy i interpretacji. Rozwiązania lokalne nie zapewniły na docelowym komputerze wymaganej szybkości i jakości, natomiast odpowiedzi zgodne ze schematem nadal mogły zawierać niewłaściwe decyzje. Druga grupa problemów wynikała z ograniczeń DaVinci Resolve 21 Free. Część potrzebnych operacji nie była dostępna, inne wymagały przygotowania technicznych plików pośrednich, a zachowanie zakresów czasowych i elementów osi czasu trzeba było ustalać eksperymentalnie.

Osobną trudność stanowiło testowanie rozwiązania złożonego z aplikacji desktopowej, warstwy serwerowej i mostu Lua działającego wewnątrz edytora. Testy automatyczne pozwalają potwierdzić poprawność typów danych, obliczeń i walidacji, lecz nie pokazują, czy wykonana operacja rzeczywiście zmienia projekt w oczekiwany sposób. Zdarzało się również, że struktura projektu wskazywała na poprawne nałożenie efektu, ale zmiana nie była widoczna w renderze. Dlatego próby integracyjne uzupełniono odczytem stanu osi czasu, porównywaniem obrazu przed operacją i po niej oraz kontrolą ręczną.

Weryfikacja gotowego prototypu została oparta na doświadczeniach zebranych podczas tych prób. Jej celem jest ustalenie, które elementy systemu działają powtarzalnie, które pozostają zależne od decyzji modeli, a które można ocenić wyłącznie w rzeczywistym środowisku montażowym. Pozwala to przedstawić wyniki razem z problemami, które wpłynęły na ostateczną postać rozwiązania, bez utożsamiania poprawności technicznej z artystyczną jakością montażu.

6.1. Najważniejsze problemy implementacyjne

Pierwotnym założeniem projektu było możliwie szerokie wykorzystanie modeli uruchamianych lokalnie. Rozważano również podział analizy na mniejsze zadania, dla których można było dobrać wyspecjalizowane modele o niższych wymaganiach sprzętowych. Przeprowadzone próby wykazały jednak, że sprawdzone rozwiązania nie zapewniały na sprzęcie dostępnym podczas realizacji pracy jednocześnie wymaganej szybkości, jakości wyników i zgodności z przyjętymi strukturami danych. Z tego względu zadania wymagające interpretacji powierzono usługom zewnętrznym, natomiast lokalnie pozostawiono analizę deterministyczną, budowanie i walidację planu oraz wykonanie operacji.

Dobór sposobu wykorzystania modeli nie był jedynym założeniem wymagającym korekty. Kolejne problemy wynikały z ograniczeń interfejsów programistycznych DaVinci Resolve 21 Free, niedokumentowanego zachowania części operacji oraz trudności w testowaniu systemu złożonego z kilku współpracujących procesów. Ich rozwiązanie wymagało przygotowywania prób wykonalności, odczytywania stanu projektu i sprawdzania rezultatów bezpośrednio w edytorze.

6.1.1 Dobór sposobu wykorzystania modeli

Początkowo planowano powierzyć modelom lokalnym znaczną część zadań. Podejście to ograniczałoby zależność od zewnętrznych usług oraz konieczność przekazywania do nich informacji opisujących materiały i projekt montażowy. Do uruchamiania modeli wykorzystano Ollama [70], [71], natomiast próby skoncentrowano na modelach z rodziny Gemma 4.

Podstawową konfigurację oparto na modelu `gemma4:e2b`. Sprawdzono również większy wariant `gemma4:e4b`. Oba modele były zapisane w kwantyzacji `Q4_K_M`, ograniczającej ilość pamięci potrzebnej do ich uruchomienia. Mniejszy wariant wybrano jako podstawowy ze względu na niższe wymagania sprzętowe. Rozmiar większego modelu przekraczał pojemność pamięci zastosowanej karty graficznej, dlatego jego uruchomienie wymagało wykorzystania również pamięci operacyjnej komputera.

Próby miały charakter testów wykonalności, a nie systematycznego porównania dostępnych modeli. Przeprowadzono je na komputerze wyposażonym w sześciordzeniowy procesor AMD Ryzen 5 5600X, 16 GB pamięci RAM oraz kartę graficzną NVIDIA GeForce RTX 3060 Ti z 8 GB pamięci graficznej. Wykorzystano te same polecenia oraz opisy materiałów, które później zastosowano w próbach z modelami chmurowymi. W każdym przypadku sprawdzano możliwość uruchomienia modelu, czas przetwarzania, przydatność uzyskanego wyniku oraz jego zgodność ze strukturą wymaganą przez aplikację.

Modele nie otrzymywały bezpośrednio plików multimedialnych. Ich wejście obejmowało polecenie użytkownika, wyniki wcześniejszej analizy materiałów, historię rozmowy oraz informacje o stanie projektu. Próby dotyczyły przede wszystkim rozpoznawania zamiaru użytkownika, wyboru odpowiedniej ścieżki działania i przygotowywania ustrukturyzowanych decyzji wykorzystywanych przez rolę uczestniczącą w planowaniu montażu.

Wariant `gemma4:e2b` można było uruchomić na dostępnym komputerze, jednak nie zapewniał jednocześnie wymaganej jakości odpowiedzi i czasu działania. Problem był szczególnie widoczny podczas planowania wieloetapowego, ponieważ obsługa pojedynczego polecenia mogła wymagać kolejnych wywołań modelu przez rolę rozpoznającą zamiar, rolę przygotowującą poszczególne części montażu oraz rolę oceniającą wynik. Opóźnienia każdego wywołania sumowały się, przez co pełny przebieg trwał zbyt długo dla interaktywnej pracy. Większy wariant `gemma4:e4b` wymagał więcej zasobów, a uzyskana poprawa nie usuwała problemów związanych z jakością i powtarzalnością decyzji.

Na podstawie przeprowadzonych prób zrezygnowano z wykorzystania modeli lokalnych jako podstawy interpretacji i planowania montażu. Zadania te powierzono modelom udostępnianym przez usługi zewnętrzne. Lokalnie pozostawiono operacje deterministyczne, obejmujące analizę techniczną materiałów, przygotowanie kontekstu, budowanie i walidację planu oraz wykonanie zatwierdzonych poleceń. Wniosek ten odnosi się do sprawdzonych wariantów Gemma 4 i sprzętu wykorzystanego podczas realizacji pracy, a nie do wszystkich modeli, które mogą być uruchamiane lokalnie.

6.1.2 Niedeterministyczność modeli i kontrola wykonania

Decyzje modeli językowych pozostają zmienne, dlatego architekturę aplikacji zaprojektowano tak, aby ograniczyć skutki błędnej odpowiedzi. Model przygotowuje propozycję montażu, natomiast kod sprawdza jej strukturę, wskazane materiały i dopuszczalne operacje. Powstały plan jest przedstawiany użytkownikowi i wymaga zatwierdzenia przed wykonaniem. Takie podejście nie gwarantuje uzyskania identycznego planu przy każdym wywołaniu, ale uniemożliwia bezpośrednie wykonanie niezwyfikowanej odpowiedzi w projekcie DaVinci Resolve.

Źródłem zmienności jest również wieloetapowy przebieg planowania. **Orchestrator** rozpoznaje zamiar użytkownika, po czym kod uruchamia wyłącznie role potrzebne do wykonania danego zadania. Każda z nich podejmuje osobną decyzję dotyczącą między innymi narracji, dźwięku, wyboru fragmentów lub efektów. Błąd może więc wystąpić na dowolnym etapie, nawet gdy wcześniejsze rozpoznanie polecenia było poprawne.

Instrukcje przygotowane dla poszczególnych ról napisano po polsku i uzupełniono przykładami typowych poleceń. Określają one między innymi różnicę pomiędzy utworzeniem nowego montażu a zmianą istniejącej osi czasu, zasady wyboru materiałów oraz sytuacje wymagające zadania pytania użytkownikowi. Nazwy techniczne pozostają w języku angielskim tam, gdzie odpowiadają identyfikatorom operacji i elementom interfejsów programistycznych. Polskie instrukcje ograniczają potrzebę samodzielnego przekładania intencji użytkownika na język angielski, lecz nadal są wskazówkami dla modelu, a nie regułami wymuszonymi przez kod.

Zakres danych przekazywanych modelom również został ograniczony. Zamiast pełnych wyników analizy otrzymują one skrócone opisy materiałów, wybrane fragmenty, ograniczoną historię rozmowy oraz informacje o elementach aktywnej osi czasu. Model może posługiwać się identyfikatorami występującymi w tym kontekście. Jeżeli mimo to wskaże nieistniejący materiał, niewłaściwy rodzaj pliku albo zakres wykraczający poza udostępniony fragment, wynik zostaje odrzucony.

Odpowiedzi ról są zapisywane w typowanych strukturach danych. Schematy ograniczają dopuszczalne wartości, wymagają obecności określonych pól i odrzucają pola nieprzewidziane w kontrakcie. Sama zgodność ze schematem nie potwierdza jednak poprawności decyzji. Nazwa wybranej operacji jest dlatego sprawdzana w rejestrze funkcji obsługiwanych przez aplikację, a jej parametry podlegają oddzielnej walidacji. Plan pełnego montażu ocenia dodatkowo rola recenzująca, która może wskazać element wymagający jednokrotnego ponownego przygotowania. Ponieważ recenzent również korzysta z modelu językowego, jego ocena stanowi dodatkową kontrolę merytoryczną, a nie gwarancję poprawności.

Oddzielnego rozwiązania wymagały polskie nazwy efektów zestawiane z angielskimi nazwami narzędzi Fusion. Kod wyodrębnia z opisu wybrane rdzenie, takie jak „rozmy-”, „nasyc-” i „przenikan-”, a następnie przypisuje im terminy używane w katalogu, między innymi blur, saturation, merge i dissolve. Mechanizm nie tłumaczy całego polecenia. Służy jedynie do wstępnego wyboru narzędzi, których rzeczywiste identyfikatory i parametry zostały odczytane z DaVinci Resolve.

Dla często wykorzystywanych efektów przygotowano rozwiązania deterministyczne. Zestaw ustawień obrazu czarno-białego ustawia nasycenie na wartość zero, natomiast wygaszenia, podstawowy ruch kadru oraz ciepły i zimny wygląd obrazu korzystają z ustalonych grafów i parametrów. Po rozpoznaniu takiego przypadku model nie projektuje samodzielnie struktury efektu. Pozostałe efekty obsługuje wyspecjalizowana rola, której udostępnia się ograniczony katalog narzędzi.

Powtarzalność dotyczy zatem technicznego przekształcenia zaakceptowanych decyzji w operacje montażowe, a nie samej interpretacji polecenia i kreatywnego wyboru materiałów. Mimo zastosowania wielowarstwowych mechanizmów kontroli nie udało się zapewnić stuprocentowej skuteczności części systemu zależnej od modeli językowych. Walidacja wykrywa błędy strukturalne i część niespójności, lecz technicznie poprawny plan nadal może nie odpowiadać intencji użytkownika. Ocenę tych elementów należy więc opierać na serii powtórzeń i odsetku poprawnych wyników, a nie na pojedynczym teście zakończonym powodzeniem.

6.1.3 Ograniczenia integracji z DaVinci Resolve 21 Free

Interfejsy programistyczne udostępnione w badanej instalacji DaVinci Resolve 21 Free nie odzwierciedlają pełnego zestawu operacji dostępnych w graficznym interfejsie edytora. Część działań można wykonać bezpośrednio, natomiast pozostałe wymagają złożenia z prostszych operacji albo przygotowania technicznych plików pośrednich. Ograniczenia te wpłynęły na architekturę integracji oraz zakres funkcji prototypu.

W badanym środowisku zewnętrzne skrypty nie mogły uzyskać dostępu do działającego edytora. Most napisany w języku Lua musi zostać ręcznie uruchomiony wewnątrz programu, a warstwa serwerowa komunikuje się z nim za pomocą plikowego kanału komunikacji międzyprocesowej IPC. Odczyt projektu, pobieranie materiałów z *Media Pool*, rozpoznawanie narzędzi Fusion oraz wykonanie planu są możliwe tylko wtedy, gdy DaVinci Resolve i most są aktywne. Bez edytora mogą natomiast działać interfejs aplikacji, warstwa serwerowa oraz analiza plików lokalnych. Przyjęty sposób integracji uniemożliwia przeprowadzenie pełnych testów całego przepływu w środowisku pozbawionym interfejsu graficznego.

Jednym z istotnych ograniczeń funkcjonalnych jest brak sterowania prędkością materiałów. Interfejs programistyczny nie udostępnia operacji pozwalającej przyspieszyć lub zwolnić klip wideo albo audio. Nie zapewnia również regulacji głośności klipów i ścieżek. Prototyp może wyłączyć całą ścieżkę dźwiękową. W przypadku dodawania muzyki warstwa serwerowa przygotowuje przed importem techniczny plik dźwiękowy z ustalonym poziomem oraz automatycznym ściszeniem w przedziałach zawierających mowę. Nie jest jednak możliwa zmiana głośności istniejącego klipu ani wprowadzenie automatyki poziomu w wybranym miejscu osi czasu.

Niektóre podstawowe operacje edycyjne również wymagają zastosowania mechanizmów zastępczych. Przycinanie, dzielenie i przesuwanie materiałów jest realizowane przez usunięcie klipu, a następnie ponowne umieszczenie odpowiedniego fragmentu na osi czasu. W przypadku materiału zawierającego powiązany obraz i dźwięk odtwarzane są oba elementy. Konsekwencją takiego działania jest możliwość utraty dodanych wcześniej efektów, korekcji barwnej i kompozycji Fusion. Operacje te nie są zatem równoważne ich wykonaniu za pomocą graficznego interfejsu edytora.

Obrazy statyczne są przekształcane w techniczne pliki wideo o określonym czasie trwania, ponieważ interfejs programistyczny ignoruje długość podawaną podczas ich umieszczania na osi czasu. Brak bezpośredniej funkcji dodawania przejść częściowo zastąpiono własnymi mechanizmami przenikania oraz przejścia przez czerń. Rozwiązania te umożliwiają wykonanie podstawowych zmian, lecz nie zapewniają pełnego zakresu narzędzi dostępnych podczas ręcznej pracy w edytorze.

Integracja z Fusion wymagała rozpoznania rzeczywistego katalogu narzędzi i ich parametrów w uruchomionym środowisku DaVinci Resolve. W pomiarze wykonanym na badanej instalacji rejestr zawierał 1209 wpisów, a niektóre narzędzia udostępniały dziesiątki parametrów. Rejestr obejmował również elementy wewnętrzne oraz narzędzia niedostępne w używanej wersji programu, dlatego nie mógł zostać bezpośrednio potraktowany jako katalog obsługiwanych efektów. Sama lista nazw nie dostarczała też modelowi informacji wystarczających do prawidłowego określenia przeznaczenia węzłów i sposobu użycia ich parametrów.

Osobnym problemem było wykonanie przygotowanego grafu. Kompozycja utworzona bezpośrednio za pomocą metody `AddFusionComp` mogła zostać zapisana w projekcie i ponownie odczytana, lecz nie powodowała widocznej zmiany wyrenderowanego obrazu. Działający rezultat uzyskano dopiero po wyeksportowaniu kompozycji, a następnie ponownym zaimportowaniu jej jako pliku `.comp`. Przypadek ten pokazał, że poprawna struktura projektu i pozytywny wynik odczytu nie stanowią wystarczającego potwierdzenia działania efektu.

W aktualnej implementacji rejestr Fusion jest ograniczany do 30 sprawdzonych narzędzi. Model wskazuje elementy odpowiadające opisowi efektu, a następnie otrzymuje rzeczywiste identyfikatory oraz parametry tych węzłów. Grafy przygotowane przez rolę projektującą efekty podlegają deterministycznej walidacji. Proste wygaszenia, ruch kadru i określone zmiany kolorystyczne są natomiast budowane bez udziału modelu według ustalonych reguł. Nadal nie można zagwarantować wykonania dowolnego efektu opisanego w języku naturalnym.

Obsługiwany zakres zależy od narzędzi umieszczonych w katalogu, możliwości ich bezpiecznego połączenia oraz zachowania konkretnej wersji DaVinci Resolve.

6.1.4 Trudności związane z testowaniem

W początkowym okresie implementacji deterministyczne testy automatyczne warstwy serwerowej przechodziły poprawnie i potwierdzały działanie modeli danych, walidatorów, konstruktorów planu oraz mechanizmów komunikacji. Korzystały jednak z wcześniej przygotowanych danych wejściowych. Dzięki temu pozwalały sprawdzić, czy określony plan zostanie prawidłowo przekształcony w polecenia, ale nie obejmowały sposobu, w jaki taki plan powstaje na podstawie odpowiedzi modelu językowego.

Po połączeniu wszystkich elementów aplikacji okazało się, że najwięcej problemów występuje na styku modeli i kodu wykonawczego. Model mógł nie uruchomić właściwej ścieżki, pominąć część polecenia albo dobrać niewłaściwe narzędzie i parametry efektu. W rezultacie mechanizm sprawdzony wcześniej w testach automatycznych nie zawsze był prawidłowo wykorzystywany w rzeczywistym scenariuszu. Dotyczyło to szczególnie efektów Fusion, dla których poprawnie przygotowany plan nie zawsze prowadził do widocznej zmiany w DaVinci Resolve.

Z tego powodu podstawą weryfikacji stały się testy całego przepływu *end-to-end* oraz testy typu *smoke*. Każdy taki scenariusz rozpoczynał się od polecenia sformułowanego w języku naturalnym. Następnie sprawdzano decyzje modeli, zawartość przygotowanego planu, jego wykonanie przez most Lua oraz rezultat widoczny w projekcie montażowym. W ten sposób test obejmował aplikację w postaci zbliżonej do tej, z której korzysta użytkownik.

Taki sposób weryfikacji znacznie wydłużał testowanie całej aplikacji. Każdy przypadek wymagał przeprowadzenia pełnego przepływu, a następnie ręcznej oceny wyniku w edytorze. Z tego względu nie dało się dokładnie sprawdzić każdego efektu dla wielu sformułowań tego samego polecenia. Wariantów wypowiedzi było znacznie więcej, niż można było uwzględnić w testach opartych na ustalonych wcześniej danych wejściowych.

Kontrola efektów wymagała dodatkowo uruchomienia DaVinci Resolve oraz ręcznego sprawdzenia podglądu lub renderu. Sam brak błędu wykonania nie potwierdzał, że zmiana była rzeczywiście widoczna. Nie dało się przygotować przypadków dla wszystkich wariantów opisu efektów, ponieważ ten sam zamiar bywa wyrażany na wiele sposobów, a model może dobrać różne zestawy narzędzi i parametrów. Próby obejmowały więc wybrane, reprezentatywne scenariusze kończące się sprawdzeniem finalnego materiału.

6.2. Wyniki weryfikacji

Weryfikację pełnego przepływu aplikacji oparto na wcześniej określonych scenariuszach. Każdy z nich wykonano w pięciu wariantach różniących się sposobem sformułowania polecenia oraz wykorzystanymi materiałami źródłowymi. Próba rozpoczynała się od analizy materiałów i przekazania polecenia w języku naturalnym. Następnie sprawdzano przygotowany plan, jego wykonanie oraz rezultat widoczny w DaVinci Resolve. Za poprawny uznawano przebieg, w którym system wybrał właściwy rodzaj działania, uwzględnił wymagane operacje i wprowadził oczekiwane zmiany w projekcie.

Warianty ustalano przed rozpoczęciem końcowej serii prób. Scenariusz uznawano za zaakceptowany, jeżeli wszystkie pięć wariantów spełniło przyjęte kryteria. Odmienny sposób montażu nie był traktowany jako błąd, jeśli nadal odpowiadał poleceniu i wykorzystywał właściwe materiały. W przypadku celowo niejednoznacznego żądania prawidłowym

zachowaniem mogło być również zadanie pytania doprecyzowującego bez wprowadzania zmian w projekcie.

Tabela 3 zawiera reprezentatywne scenariusze uzyskane przez pogrupowanie zbliżonych operacji.

Tabela 3. Scenariusze pełnego przepływu

Scenariusz	Zakres weryfikacji
Kompilacja zdjęć wakacyjnych	Utworzenie osi czasu, wybór i uporządkowanie zdjęć, zastosowanie ruchu kadru oraz podstawowych przejść
Film pożegnalny	Wybór wskazanych nagrań lub fragmentów, zastosowanie obrazu czarno-białego oraz przejść pomiędzy ujęciami
Krótki film zsynchronizowany z muzyką	Analiza rytmu, rozmieszczenie klipów względem muzyki, dodanie podkładu oraz utworzenie edytowalnej osi czasu
Edycja istniejącej osi czasu	Przycinanie, dzielenie i przesuwanie wskazanych klipów bez niezamierzonego modyfikowania pozostałych elementów
Napisy na podstawie mowy lub tekstu	Zgodność treści, kolejność i zakresy czasowe napisów oraz brak ich wzajemnego nakładania
Dodanie lektora	Przygotowanie wypowiedzi, utworzenie osobnej ścieżki dźwiękowej, zachowanie właściwego języka i położenia na osi czasu
Wybór fragmentów materiału	Utworzenie montażu zawierającego fragmenty dotyczące wskazanego tematu albo wybrane najciekawsze momenty
Oczyszczanie nagrania mowy	Usunięcie wykrytych słów-wypełniaczy i powtórzeń przy zachowaniu pozostałej części wypowiedzi
Efekty obrazu i przejścia	Zastosowanie wygaszania, ruchu kadru, korekcji kolorystycznej, przenikania lub efektu Fusion oraz potwierdzenie jego widoczności
Zmiana formatu i renderowanie	Utworzenie wersji 9:16, 16:9 albo 1:1, zachowanie zawartości montażu oraz przygotowanie pliku wynikowego
Obsługa konfliktu efektów	Rozpoznanie istniejącego efektu, uzyskanie decyzji o jego zastąpieniu albo rozszerzeniu i wykonanie wybranej operacji

W końcowej serii prób uzyskano poprawny przebieg wszystkich wymienionych scenariuszy. Poszczególne realizacje różniły się sposobem sformułowania polecenia oraz wykorzystanymi materiałami źródłowymi. Zestawienie ma charakter jakościowy i obejmuje zakres sprawdzonych funkcji prototypu. Nie pozwala natomiast ocenić powtarzalności odpowiedzi modelu dla identycznych danych ani określić procentowej niezawodności systemu.

Poza końcową serią akceptacyjną wykonywano również próby eksploracyjne. Zaobserwowano w nich, że niewielka zmiana treści polecenia mogła spowodować wybór innej ścieżki planowania, pominięcie części wymagania albo przygotowanie planu niezgodnego z oczekiwaniem. W takim przypadku próba kończyła się wynikiem negatywnym, mimo że deterministyczne moduły systemu działały poprawnie. Problemy wynikały zarówno z decyzji modelu, jak i z normalizacji tekstu oraz reguł kierujących polecenie do właściwego modułu.

Poprawne wykonanie jednego polecenia nie gwarantowało zatem równoważnej obsługi wszystkich jego parafraz.

Uzupełnieniem weryfikacji była pilotażowa próba zadaniowa z udziałem trzech osób bez doświadczenia w obsłudze DaVinci Resolve. Każdy uczestnik otrzymał ten sam zestaw materiałów oraz opis filmu, który należało przygotować, lecz samodzielnie formułował polecenia dla aplikacji. Wszystkie trzy osoby ukończyły zadanie i uzyskały materiał odpowiadający przedstawionemu celowi. Ze względu na niewielką liczbę uczestników rezultat ten należy traktować jako obserwację pilotażową, a nie jako reprezentatywne badanie użyteczności.

Wyniki odnoszą się do przygotowanych scenariuszy, wykorzystanych materiałów i konfiguracji modeli. Potwierdzają możliwość przeprowadzenia pełnego przepływu dla badanego zakresu, lecz nie gwarantują uzyskania takiego samego rezultatu dla dowolnego polecenia, materiału źródłowego ani kolejnego wywołania modelu.

6.3. Ocena realizacji założeń projektowych

Realizację założeń projektowych można ocenić na dwóch poziomach. Pierwszy obejmuje techniczne działanie systemu. W tym zakresie prototyp realizuje przewidziany przepływ: analizuje obsługiwane materiały, interpretuje polecenie użytkownika, przygotowuje typowany plan i przekazuje go do wykonania dopiero po zatwierdzeniu. Operacje są wykonywane w natywnym projekcie DaVinci Resolve, dzięki czemu uzyskany montaż pozostaje edytowalny.

Drugi poziom dotyczy praktycznej użyteczności rozwiązania. Wyniki przeprowadzonej weryfikacji wskazują, że przyjęty sposób interakcji pozwala osobie bez doświadczenia montażowego przejść przez podstawowy proces przygotowania krótkiego filmu. Wstępnie potwierdza to zasadność zastosowania języka naturalnego jako warstwy pośredniej pomiędzy oczekiwanym rezultatem a operacjami wykonywanymi w edytorze. Zakres weryfikacji nie pozwala jednak uogólnić tego wniosku na wszystkie grupy użytkowników, rodzaje materiałów i sposoby formułowania poleceń.

Za zrealizowane można więc uznać założenia dotyczące architektury, kontrolowanego wykonania oraz zachowania edytowalnego wyniku. Badany scenariusz dostarczył również pierwszej pozytywnej przesłanki dotyczącej dostępności rozwiązania dla osób nietechnicznych. Nie przeprowadzono natomiast porównawczego pomiaru czasu montażu. Możliwość skrócenia pracy pozostaje zatem przewidywaną korzyścią rozwiązania, a nie wynikiem potwierdzonym w ramach wykonanej weryfikacji.

6.4. Potencjalne zastosowania

Najbardziej naturalnym obszarem zastosowania prototypu jest tworzenie krótkich filmów do publikacji w mediach społecznościowych. Odbiorcami mogą być osoby prywatne, początkujący twórcy, małe przedsiębiorstwa oraz organizacje publikujące treści okazjonalnie i niedysponujące osobą zajmującą się montażem. Źródłem mogą być zdjęcia i krótkie nagrania wykonane telefonem, przedstawiające wydarzenie, podróż, produkt, świadczoną usługę albo działalność organizacji.

System może posłużyć do opracowania pierwszej wersji filmu, w której materiały zostają uporządkowane w spójną sekwencję i uzupełnione o podstawowe elementy obrazu oraz dźwięku. Uzyskany projekt pozostaje dostępny do samodzielnej poprawy w DaVinci Resolve albo przekazania bardziej doświadczonej osobie. Prototyp sprawdza się przede wszystkim tam, gdzie

istotne jest szybkie utworzenie krótkiej publikacji, natomiast nie jest wymagany rozbudowany montaż narracyjny, zaawansowana korekcja obrazu ani złożona obróbka dźwięku.

7. Podsumowanie

W ramach pracy zaprojektowano i zaimplementowano prototyp umożliwiający obsługę wybranych funkcji DaVinci Resolve 21 Free za pomocą poleceń w języku naturalnym. Rozwiązanie obejmuje analizę materiałów multimedialnych, interpretację zamiaru użytkownika, przygotowanie planu montażu oraz wykonanie zatwierdzonych operacji w edytowalnym projekcie. Zrealizowano tym samym pełny przepływ prowadzący od wskazania materiałów do utworzenia lub zmodyfikowania osi czasu.

Najważniejszym założeniem rozwiązania było oddzielenie niedeterministycznych decyzji modeli od operacji zmieniających projekt. Modele językowe uczestniczą w rozpoznawaniu intencji, wyborze materiałów i przygotowaniu koncepcji montażu. Końcowy plan jest jednak budowany oraz sprawdzany przez deterministyczne moduły aplikacji. Użytkownik może zapoznać się z planem, a następnie go zatwierdzić albo odrzucić. Takie podejście ogranicza ryzyko wykonania nieobsługiwanej operacji i pozwala zachować kontrolę nad wprowadzanymi zmianami.

Przeprowadzona weryfikacja potwierdziła możliwość zastosowania prototypu w przygotowanych scenariuszach obejmujących podstawowy montaż krótkich materiałów. Rozwiązanie może obniżyć barierę wejścia dla osób, które nie znają interfejsu i terminologii edytora. W wybranych przypadkach, takich jak kompilacja zdjęć lub przygotowanie krótkiego filmu ze wskazanych nagrań, użytkownik może uzyskać gotowy rezultat bez samodzielnego wykonywania poszczególnych operacji montażowych. Nie oznacza to jednak, że system poprawnie obsłuży dowolne polecenie ani że zastąpi montażystę w bardziej złożonych zadaniach.

Wnioski z pracy odnoszą się do modeli, usług oraz wersji DaVinci Resolve dostępnych podczas realizacji projektu. Rozwiązania wykorzystujące sztuczną inteligencję zmieniają się szybko, dlatego możliwości uznawane obecnie za ograniczone mogą zostać rozszerzone w ciągu kilku miesięcy. Rozwój modeli nie eliminuje jednak potrzeby walidacji, ponieważ odpowiedź zgodna z wymaganym schematem nadal może zawierać niewłaściwą decyzję.

Jednym z możliwych kierunków rozwoju jest zastosowanie FFmpeg do deterministycznego wykonywania operacji niedostępnych przez interfejsy programistyczne DaVinci Resolve 21 Free. Wstępne próby wskazywały, że modele łatwiej dobierały operacje oparte na FFmpeg niż rozwiązania zależne od specyfiki Scripting API i Fusion API. Przetworzony materiał mógłby być następnie importowany do projektu. Ścieżka ta wymagałaby rozbudowania obsługi plików pośrednich i powinna uzupełniać obecny mechanizm, ponieważ import całego filmu jako jednego klipu ograniczałby możliwość jego dalszej edycji.

Prace cytowane

- [1] „Introduction to video editing,” Adobe, [Online]. Available: <https://www.adobe.com/creativecloud/video/discover/edit-a-video.html>. [Data uzyskania dostępu: 18 08 2026].
- [2] M. Pierce, „Video Editing Workflow,” TechSmith, [Online]. Available: <https://www.techsmith.com/blog/video-editing-workflow/>. [Data uzyskania dostępu: 18 08 2026].
- [3] O. Abtahi, „Fine cuts: Making your video screen-ready,” Descript, 29 07 2026. [Online]. Available: <https://www.descript.com/blog/article/fine-cuts-making-your-video-screen-ready>. [Data uzyskania dostępu: 18 08 2026].
- [4] The YouTube Team, „From epic edits to quick clips: Transitioning your long-form content to YouTube Shorts,” YouTube Official Blog, 16 04 2025. [Online]. Available: <https://blog.youtube/creator-and-artist-stories/transitioning-your-long-form-content-to-youtube-shorts>. [Data uzyskania dostępu: 18 08 2026].
- [5] E. Seo, „Creating Made Easier,” TikTok for Business, 21 11 2023. [Online]. Available: https://ads.tiktok.com/business/creativecenter/quicktok/online/Creating_Made_Easier/pad/en. [Data uzyskania dostępu: 18 08 2026].
- [6] „The Ultimate Video Editing Workflow for Content Creators,” TechSmith, 15 08 2025. [Online]. Available: <https://www.techsmith.com/blog/how-to-edit-a-video/>. [Data uzyskania dostępu: 18 08 2026].
- [7] „Overview of Text-Based Editing,” Adobe, 21 01 2026. [Online]. Available: <https://helpx.adobe.com/pl/premiere/desktop/edit-projects/edit-video-using-text-based-editing/overview-of-text-based-editing.html>. [Data uzyskania dostępu: 18 08 2026].
- [8] „User Needs + Defining Success,” Google, [Online]. Available: <https://pair.withgoogle.com/chapter/user-needs/>. [Data uzyskania dostępu: 18 08 2026].
- [9] „Feedback + Control,” Google, [Online]. Available: <https://pair.withgoogle.com/chapter/feedback-controls/>. [Data uzyskania dostępu: 18 08 2026].
- [10] P. Chang, „A Beginner's Guide to Video File Formats: MP4s,” Mux, 11 01 2021. [Online]. Available: <https://www.mux.com/blog/a-beginners-guide-to-video-file-formats>. [Data uzyskania dostępu: 19 08 2026].
- [11] A. Huamán, „Histogram Comparison,” OpenCV, [Online]. Available: https://docs.opencv.org/4.x/d8/dc8/tutorial_histogram_comparison.html. [Data uzyskania dostępu: 20 08 2026].
- [12] „skimage.metrics,” scikit-image, [Online]. Available: <https://scikit-image.org/docs/stable/api/skimage.metrics.html>. [Data uzyskania dostępu: 20 08 2026].
- [13] J. L. Pech-Pacheco, G. Cristóbal, J. Chamorro-Martínez i J. Fernández-Valdivia, „Diatom autofocusing in brightfield microscopy: a comparative study,” w *15th International Conference on Pattern Recognition (ICPR 2000)*, 2000.
- [14] R. Ramnauth, „Extracting & Interpreting Audio Features in Python,” Garage, 25 05 2020. [Online]. Available: <https://rramnauth2220.github.io/blog/posts/code/200525-feature-extraction.html>. [Data uzyskania dostępu: 20 08 2026].

- [15] C. Stryker, „What Are Large Language Models (LLMs)?,” IBM Think, [Online]. Available: <https://www.ibm.com/think/topics/large-language-models>. [Data uzyskania dostępu: 23 08 2026].
- [16] OpenAI, „Introducing Structured Outputs in the API,” OpenAI, 06 08 2024. [Online]. Available: <https://openai.com/index/introducing-structured-outputs-in-the-api/>. [Data uzyskania dostępu: 23 08 2026].
- [17] Google Cloud, „Content generation parameters,” Google Cloud, 21 08 2026. [Online]. Available: <https://docs.cloud.google.com/gemini-enterprise-agent-platform/models/capabilities/content-generation-parameters>. [Data uzyskania dostępu: 23 08 2026].
- [18] IBM, „What is Multimodal AI?,” IBM, [Online]. Available: <https://www.ibm.com/think/topics/multimodal-ai>. [Data uzyskania dostępu: 23 08 2026].
- [19] Microsoft, „AI workloads on Azure,” Microsoft, 07 05 2026. [Online]. Available: <https://learn.microsoft.com/en-us/azure/well-architected/ai/get-started>. [Data uzyskania dostępu: 23 08 2026].
- [20] Y. Kim, J. Russell, M. Karpinska i M. Iyyer, „One ruler to measure them all: Benchmarking multilingual long-context language models,” w *2nd Conference on Language Modeling (COLM 2025)*, Montreal, 2025.
- [21] OpenAI, „What are tokens and how to count them?,” OpenAI, 19 08 2026. [Online]. Available: <https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them>. [Data uzyskania dostępu: 23 08 2026].
- [22] Anthropic, „Building Effective Agents,” Anthropic, 19 12 2024. [Online]. Available: <https://www.anthropic.com/engineering/building-effective-agents>. [Data uzyskania dostępu: 23 08 2026].
- [23] OpenAI, „Multi-agent,” OpenAI API Documentation, [Online]. Available: <https://developers.openai.com/api/docs/guides/responses-multi-agent>. [Data uzyskania dostępu: 23 08 2026].
- [24] N. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni i P. Liang, „Lost in the Middle: How Language Models Use Long Contexts,” *Transactions of the Association for Computational Linguistics*, tom 12, pp. 157-173, 2024.
- [25] OpenAI, „GPT-5.6 Sol,” Open AI, [Online]. Available: <https://developers.openai.com/api/docs/models/gpt-5.6-sol>. [Data uzyskania dostępu: 28 08 2026].
- [26] Google, „Gemini 3.7 Flash,” Google, [Online]. Available: <https://ai.google.dev/gemini-api/docs/models/gemini-3.7-flash>. [Data uzyskania dostępu: 28 08 2026].
- [27] Anthropic, „Claude Fable 5,” Anthropic, [Online]. Available: <https://platform.claude.com/docs/en/models/fable-5/overview>. [Data uzyskania dostępu: 28 08 2026].
- [28] OpenAI, „Pricing,” OpenAI API, [Online]. Available: <https://platform.openai.com/pricing>. [Data uzyskania dostępu: 23 08 2026].
- [29] Adobe, „Explore Adobe Premiere Features,” Adobe, [Online]. Available: <https://www.adobe.com/products/premiere/features.html>. [Data uzyskania dostępu: 24 08 2026].

- [30] Adobe, „Premiere AI Assistant (beta) – overview and FAQs,” Adobe Help Center, 20 07 2026. [Online]. Available: <https://helpx.adobe.com/premiere/desktop/premiere-ai-assistant/assistant-faq.html>. [Data uzyskania dostępu: 24 08 2026].
- [31] Blackmagic Design, „DaVinci Resolve 21 Reference Manual,” Blackmagic Design, [Online]. Available: https://documents.blackmagicdesign.com/UserManuals/DaVinci_Resolve_21_Reference_Manual.pdf. [Data uzyskania dostępu: 24 08 2026].
- [32] AutoCut, „AutoCut AI Plugin – Edit Faster in Premiere Pro & DaVinci Resolve,” AutoCut, [Online]. Available: <https://www.autocut.com/en/>. [Data uzyskania dostępu: 24 08 2026].
- [33] Eddie AI, „Install and enable Eddie's DaVinci Resolve Extension,” Eddie AI, 29 10 2025. [Online]. Available: <https://help.heyeddie.ai/en/articles/11272954-install-and-enable-eddie-s-davinci-resolve-extension>. [Data uzyskania dostępu: 24 08 2026].
- [34] Jumper, „Compatibility,” Jumper, 16 03 2026. [Online]. Available: <https://docs.getjumper.io/compatibility>. [Data uzyskania dostępu: 24 08 2026].
- [35] CapCut, „CapCut Desktop — AI-powered Video Editor,” CapCut, [Online]. Available: <https://www.capcut.com/tools/desktop-ai-power>. [Data uzyskania dostępu: 24 08 2026].
- [36] Descript, „Underlord — Your AI Video & Podcast Editing Assistant,” Descript, [Online]. Available: <https://www.descript.com/underlord>. [Data uzyskania dostępu: 24 08 2026].
- [37] OpusClip, „Introduction to OpusClip,” OpusClip Help Center, [Online]. Available: <https://help.opus.pro/docs/introduction>. [Data uzyskania dostępu: 24 08 2026].
- [38] M. Sandoval-Castañeda, B. C. Russell, J. Sivic, G. Shakhnarovich i F. Caba Heilbron, „EditDuet: A Multi-Agent System for Video Non-Linear Editing,” Vancouver, 2025.
- [39] L. Yan, Y. Zhang, X. Xu, J. Lin, B. Pan, X. Zheng, J. Qian, A. Wu, J. Geng, R. Li, F. Cai, J. Niu, R. Li, W. Li i C. Lyu, „Crayotter: Traceable Multi-Agent Workflows for Long-Form Video Editing,” 2026.
- [40] FastAPI, „Features,” FastAPI Documentation, [Online]. Available: <https://fastapi.tiangolo.com/features/>. [Data uzyskania dostępu: 25 08 2026].
- [41] Pydantic, „Models,” Pydantic Validation Documentation, [Online]. Available: <https://pydantic.dev/docs/validation/latest/concepts/models/>. [Data uzyskania dostępu: 25 08 2026].
- [42] Pydantic, „Unions,” Pydantic Validation Documentation, [Online]. Available: <https://pydantic.dev/docs/validation/latest/concepts/unions/#discriminated-unions>. [Data uzyskania dostępu: 25 08 2026].
- [43] SQLite Project, „SQLite Is Serverless,” SQLite, [Online]. Available: <https://www.sqlite.org/serverless.html>. [Data uzyskania dostępu: 25 08 2026].
- [44] keyring project, „Welcome,” keyring documentation, [Online]. Available: <https://keyring.readthedocs.io/en/latest/>. [Data uzyskania dostępu: 25 08 2026].
- [45] O. Khattab, A. Singhvi, P. Maheshwari, Z. Zhang, K. Santhanam, S. Vardhamanan, S. Haq, A. Sharma, T. T. Joshi, H. Moazam, H. Miller, M. Zaharia i C. Potts, „DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines,” w *The Twelfth International Conference on Learning Representations (ICLR 2024)*, Vienna, 2024.

- [46] Stanford NLP / DSPy Project, „Class-based signatures,” DSPy Documentation, [Online]. Available: <https://dspy.ai/getting-started/class-based-signatures/>. [Data uzyskania dostępu: 25 08 2026].
- [47] Stanford NLP / DSPy Project, „Adapters: how signatures become prompts,” DSPy Documentation, [Online]. Available: <https://dspy.ai/diving-deeper/adapters/>. [Data uzyskania dostępu: 25 08 2026].
- [48] Stanford NLP / DSPy Project, „Image,” DSPy Documentation, [Online]. Available: <https://dspy.ai/api/primitives/Image/>. [Data uzyskania dostępu: 25 08 2026].
- [49] Stanford NLP / DSPy Project, „dspy.Tool,” DSPy Documentation, [Online]. Available: <https://dspy.ai/api/primitives/Tool/>. [Data uzyskania dostępu: 25 08 2026].
- [50] Stanford NLP / DSPy Project, „LM,” DSPy Documentation, [Online]. Available: <https://dspy.ai/api/models/LM/>. [Data uzyskania dostępu: 25 08 2026].
- [51] LiteLLM / BerriAI, „LiteLLM – Getting Started,” LiteLLM Documentation, [Online]. Available: <https://docs.litellm.ai/>. [Data uzyskania dostępu: 25 08 2026].
- [52] PyAV Project, „PyAV Documentation,” PyAV, [Online]. Available: <https://pyav.org/docs/stable/>. [Data uzyskania dostępu: 25 08 2026].
- [53] Meta, „FFmpeg at Meta: Media Processing at Scale,” Engineering at Meta, 02 03 2026. [Online]. Available: <https://engineering.fb.com/2026/03/02/video-engineering/ffmpeg-at-meta-media-processing-at-scale/>. [Data uzyskania dostępu: 25 08 2026].
- [54] FFmpeg Project, „About FFmpeg,” FFmpeg, [Online]. Available: <https://www.ffmpeg.org/about.html>. [Data uzyskania dostępu: 25 08 2026].
- [55] PySceneDetect Project, „Detectors – PySceneDetect documentation,” PySceneDetect Documentation, [Online]. Available: <https://www.scenedetect.com/docs/latest/api/detectors.html#contentdetector>. [Data uzyskania dostępu: 25 08 2026].
- [56] OpenCV Project, „Image Processing,” OpenCV Documentation, [Online]. Available: https://docs.opencv.org/4.13.0/d7/da8/tutorial_table_of_content_imgproc.html. [Data uzyskania dostępu: 25 08 2026].
- [57] SYSTRAN, „Faster Whisper transcription with CTranslate2,” GitHub, [Online]. Available: <https://github.com/SYSTRAN/faster-whisper>. [Data uzyskania dostępu: 25 08 2026]
- [58] B. McFee, C. Raffel, D. Liang, D. P. W. Ellis, M. McVicar, E. Battenberg i O. Nieto, „librosa: Audio and Music Signal Analysis in Python,” w *Proceedings of the 14th Python in Science Conference*, 2015.
- [59] Electron, „Why Electron,” Electron Documentation, [Online]. Available: <https://www.electronjs.org/docs/latest/why-electron>. [Data uzyskania dostępu: 25 08 2026].
- [60] Electron, „Apps showcase,” Electron Documentation, [Online]. Available: <https://www.electronjs.org/apps/>. [Data uzyskania dostępu: 25 08 2026].
- [61] React Team, „Your First Component,” React, [Online]. Available: <https://react.dev/learn/your-first-component>. [Data uzyskania dostępu: 25 08 2026].
- [62] Discord, „Discord Patch Notes: June 3, 2025,” Discord Blog, 03 06 2025. [Online]. Available: <https://discord.com/blog/discord-patch-notes-june-3-2025>. [Data uzyskania dostępu: 25 08 2026].

- [63] Blackmagic Design, „Fusion 8 Scripting Guide,” Blackmagic Design, [Online]. Available:
https://documents.blackmagicdesign.com/UserManuals/Fusion8_Scripting_Guide.pdf.
[Data uzyskania dostępu: 25 08 2026]
- [64] OpenAI, „Models,” OpenAI API Documentation, [Online]. Available:
<https://developers.openai.com/api/docs/models>. [Data uzyskania dostępu: 25 08 2026].
- [65] OpenAI, „Audio and speech,” OpenAI API Documentation, [Online]. Available:
<https://developers.openai.com/api/docs/guides/audio>. [Data uzyskania dostępu: 25 08 2026].
- [66] OpenAI, „Structured model outputs,” OpenAI API Documentation, [Online]. Available:
<https://developers.openai.com/api/docs/guides/structured-outputs>. [Data uzyskania dostępu: 25 08 2026].
- [67] pytest development team, „pytest: helps you write better programs,” pytest documentation, [Online]. Available: <https://docs.pytest.org/en/stable/index.html>. [Data uzyskania dostępu: 25 08 2026].
- [68] FastAPI, „Test Client – TestClient,” FastAPI Documentation, [Online]. Available:
<https://fastapi.tiangolo.com/reference/testclient/>. [Data uzyskania dostępu: 25 08 2026].
- [69] Microsoft, „TSConfig Option: strict,” TypeScript, [Online]. Available:
<https://www.typescriptlang.org/tsconfig/strict>. [Data uzyskania dostępu: 25 08 2026].
- [70] Ollama, Ollama, [Online]. Available: <https://ollama.com/>. [Data uzyskania dostępu: 27 08 2026].
- [71] Ollama, „gemma4,” Ollama, [Online]. Available: <https://ollama.com/library/gemma4>.
[Data uzyskania dostępu: 29 08 2026].

Dodatki

Skróty

AI – ang. *artificial intelligence*; sztuczna inteligencja, czyli dziedzina obejmująca metody umożliwiające systemom komputerowym wykonywanie zadań wymagających analizy, rozpoznawania lub podejmowania decyzji.

API – ang. *application programming interface*; interfejs programistyczny aplikacji określający sposób komunikacji pomiędzy programami lub ich komponentami.

BPM – ang. *beats per minute*; liczba uderzeń rytmicznych przypadających na minutę, stosowana do określania tempa utworu muzycznego.

CPU – ang. *central processing unit*; procesor główny komputera odpowiedzialny za wykonywanie instrukcji programu.

CSS – ang. *Cascading Style Sheets*; język arkuszy stylów wykorzystywany do określania wyglądu elementów interfejsu.

CUDA – ang. *Compute Unified Device Architecture*; platforma obliczeń równoległych firmy NVIDIA umożliwiająca wykorzystanie karty graficznej do wykonywania obliczeń.

DSPy – ang. *Declarative Self-improving Python*; środowisko służące do deklaratywnego definiowania zadań wykonywanych przez modele językowe i multimodalne. W formach odmienionych w pracy stosowany jest zapis „DSP-y”.

HTML – ang. *HyperText Markup Language*; język znaczników służący do opisywania struktury dokumentów i interfejsów webowych.

HTTP – ang. *Hypertext Transfer Protocol*; protokół komunikacyjny wykorzystywany do przesyłania żądań i odpowiedzi pomiędzy aplikacją desktopową a lokalną warstwą serwerową.

HSV – ang. *hue, saturation, value*; przestrzeń barw opisująca kolor za pomocą odcienia, nasycenia i wartości odpowiadającej jasności.

IPC – ang. *inter-process communication*; komunikacja międzyprocesowa. W projekcie wykorzystano jej odmianę opartą na wymianie plików pomiędzy warstwą serwerową a mostem działającym w DaVinci Resolve.

JSON – ang. *JavaScript Object Notation*; tekstowy format zapisu danych ustrukturyzowanych, wykorzystywany między innymi podczas komunikacji pomiędzy komponentami aplikacji.

LLM – ang. *large language model*; duży model językowy przetwarzający polecenia i kontekst zapisane w języku naturalnym oraz generujący na ich podstawie odpowiedzi.

MCP – ang. *Model Context Protocol*; protokół umożliwiający udostępnianie modelom i agentom kontekstu, narzędzi oraz zewnętrznych źródeł danych.

MP4 – format kontenera multimedialnego zgodny ze standardem MPEG-4 Part 14, umożliwiający przechowywanie obrazu, dźwięku, napisów i metadanych.

NLE – ang. *non-linear editing*; nieliniowa edycja wideo umożliwiająca swobodną zmianę kolejności i parametrów fragmentów materiału bez modyfikowania plików źródłowych.

OCR – ang. *optical character recognition*; optyczne rozpoznawanie znaków, czyli przekształcanie tekstu widocznego na obrazie do postaci możliwej do dalszego przetwarzania przez program.

RAM – ang. *random access memory*; pamięć operacyjna wykorzystywana do przechowywania danych i programów podczas ich wykonywania.

RMS – ang. *root mean square*; wartość skuteczna sygnału obliczana jako pierwiastek ze średniej kwadratów wartości próbek. W pracy służy między innymi do analizy poziomu sygnału dźwiękowego i wykrywania ciszy.

SSIM – ang. *structural similarity index measure*; wskaźnik podobieństwa strukturalnego obrazów uwzględniający ich jasność, kontrast i lokalną strukturę.

XML – ang. *Extensible Markup Language*; rozszerzalny język znaczników wykorzystywany do zapisu i wymiany danych o określonej strukturze.

Terminologia

Agent – komponent wykorzystujący model językowy do wykonania określonego zadania, na przykład rozpoznania intencji, zaplanowania osi czasu lub oceny przygotowanego rozwiązania.

EditPlan – typowana struktura danych zawierająca uporządkowaną listę operacji przeznaczonych do wykonania w projekcie DaVinci Resolve. W pracy nazwę należy zapisywać czcionką kodową.

FFmpeg – zestaw bibliotek i narzędzi służących do odczytywania, przekształcania oraz zapisywania materiałów audio i wideo.

Fusion – środowisko kompozycji i efektów wizualnych zintegrowane z DaVinci Resolve, oparte na grafie połączonych narzędzi.

Fusion API – interfejs programistyczny umożliwiający tworzenie i modyfikowanie kompozycji oraz efektów Fusion.

Media Pool – biblioteka materiałów należących do otwartego projektu DaVinci Resolve.

Model multimodalny – model przetwarzający więcej niż jeden rodzaj danych, na przykład tekst i obraz.

Most Lua – skrypt uruchamiany wewnątrz DaVinci Resolve, który odbiera zwalidowane polecenia i odwzorowuje je na operacje dostępne w interfejsach programistycznych edytora.

Orkiestracja – sterowanie kolejnością wykonywania zadań oraz przekazywaniem danych pomiędzy rolami agentowymi i deterministycznymi modułami aplikacji.

Polecenie modelu – instrukcja wraz z kontekstem przekazywana do modelu językowego. Angielski termin *prompt* może zostać podany w nawiasie przy pierwszym użyciu.

Scripting API – interfejs programistyczny DaVinci Resolve służący między innymi do odczytywania projektu oraz wykonywania operacji na materiałach i osi czasu.

Test pełnego przepływu – ang. *end-to-end test*; test obejmujący cały przebieg działania aplikacji, od polecenia użytkownika do rezultatu widocznego w projekcie montażowym.

Test typu smoke – podstawowy test sprawdzający, czy najważniejszy scenariusz działania systemu może zostać wykonany bez błędu.

Ustrukturyzowana odpowiedź – wynik modelu zapisany zgodnie z wcześniej określonym schematem pól, typów i dopuszczalnych wartości.

Walidacja deterministyczna – sprawdzanie danych za pomocą jednoznacznych reguł wykonywanych przez kod, bez powierzania decyzji modelowi językowemu.

Spis rysunków

Rysunek 1. Wykrywanie przedziału ciszy na podstawie wartości RMS. Źródło: opracowanie własne na podstawie [14].	15
Rysunek 2. Porównanie tokenizacji polecenia w języku polskim i angielskim. Źródło: opracowanie własne.	17
Rysunek 3. Przykład wykorzystania odpowiedzi ustrukturyzowanej do wywołania narzędzia. Źródło: opracowanie własne.	20
Rysunek 4. Panel Premiere AI Assistant przedstawiający wykonane czynności i propozycje kolejnych działań. Źródło: Adobe [30]	24
Rysunek 5. Panel wtyczki AutoCut po zakończeniu automatycznego usuwania ciszy w programie Adobe Premiere Pro. Źródło: AutoCut [32]	25
Rysunek 6. Architektura logiczna systemu i przepływ danych pomiędzy jego komponentami. Źródło: opracowanie własne	33
Rysunek 7. Model reprezentacji materiałów i wykorzystanie wyników analizy podczas planowania. Źródło: opracowanie własne.	35
Rysunek 8. Diagram aktywności przedstawiający przebieg interpretacji polecenia i przygotowania planu edycji. Źródło: opracowanie własne.	37
Rysunek 9. Przepływ danych pomiędzy sygnaturą, adapterem i modelem w DSP-y. Źródło: opracowanie własne.	44
Rysunek 10. Zakres odpowiedzialności DaVinci Resolve Scripting API i Fusion API w warstwie wykonawczej. Źródło: opracowanie własne.	50
Rysunek 11. Struktura katalogów i najważniejszych plików projektu. Źródło: opracowanie własne.	54
Rysunek 12. Diagram sekwencji przedstawiający kontrolę i wykonanie zatwierdzonego planu edycji. Źródło: opracowanie własne.	65

Spis tabel

Tabela 1. Porównanie analizowanych rozwiązań wspomagających montaż wideo.....	28
Tabela 2. Rozmieszczenie stanu systemu	33
Tabela 3. Scenariusze pełnego przepływu	72

Spis listingów

Listing 1. Wybór ścieżki analizy materiału. Źródło: opracowanie własne.	56
Listing 2. Dobór fragmentów materiałów przekazywanych do procesu planowania. Źródło: opracowanie własne.	57
Listing 3. Budowanie kontekstu bieżącej interakcji. Źródło: opracowanie własne.	57
Listing 4. Przekazanie kontekstu bieżącej interakcji do roli Orchestratora. Źródło: opracowanie własne.	58
Listing 5. Sterowanie oceną planu i pojedynczą falą poprawy. Źródło: opracowanie własne.	60
Listing 6. Deterministyczne utworzenie poleceń i końcowej struktury EditPlan. Źródło: opracowanie własne.	61
Listing 7. Wybrane reguły walidacji kolejności poleceń planu edycji. Źródło: opracowanie własne.	63
Listing 8. Kontrola identyfikatora i rozpoczęcie wykonania planu oczekującego. Źródło: opracowanie własne.	64
Listing 9. Sekwencyjne wykonanie poleceń i zatrzymanie planu po wystąpieniu błędu. Źródło: opracowanie własne.	66