Kompleksowy Plan Badawczo-Rozwojowy: Aplikacja Mobilna AI Sports Cameraman

Część 1: Raport z Głębokiego Researchu

Niniejsza sekcja stanowi fundament techniczny i akademicki dla projektu. Jej celem jest dostarczenie dogłębnej wiedzy, która umożliwi podejmowanie świadomych decyzji architektonicznych i technologicznych, a także posłuży jako solidna podstawa merytoryczna dla dokumentacji pracy inżynierskiej.

Sekcja 1: Architektura Śledzenia w Czasie Rzeczywistym na Urządzeniu Mobilnym

Rdzeniem aplikacji jest system zdolny do percepcji i śledzenia akcji w czasie rzeczywistym, działający w warunkach znacznych ograniczeń zasobów sprzętowych urządzenia mobilnego. Wymaga to zastosowania hybrydowej architektury, która w inteligentny sposób równoważy koszt obliczeniowy nowoczesnych modeli głębokiego uczenia z wydajnością klasycznych algorytmów wizji komputerowej.

1.1. Detekcja Obiektów: Analiza Porównawcza Modeli YOLOv8n i SSD MobileNetV2

Pierwszym i fundamentalnym krokiem w potoku przetwarzania jest precyzyjna lokalizacja kluczowego obiektu – w tym przypadku piłki. Zadanie to wymaga użycia modelu detekcji opartego na głębokim uczeniu, który jest na tyle lekki, aby mógł działać z akceptowalną wydajnością na smartfonie. Dwoma wiodącymi kandydatami w tej kategorii są najmniejszy

wariant modelu YOLOv8, czyli YOLOv8n, oraz architektura SSD (Single Shot Detector) z szkieletem (backbone) MobileNetV2.[1]

Analiza tych dwóch modeli ujawnia fundamentalny kompromis pomiędzy szybkością, dokładnością a zużyciem zasobów. Modele z rodziny YOLO są historycznie znane z wyjątkowej prędkości działania. YOLOv8 jest w stanie przetwarzać obrazy z szybkością od 40 do 155 klatek na sekundę (FPS) w zależności od konfiguracji sprzętowej, co znacząco przewyższa typowe osiągi modeli SSD, mieszczące się w zakresie 22-46 FPS.[3] Z drugiej strony, w niektórych specyficznych zadaniach, architektura SSD zoptymalizowana pod kątem urządzeń mobilnych (SSD-MobileNet) może osiągać wyższą dokładność detekcji. Jedno z badań porównawczych wykazało dokładność na poziomie 91.1% dla SSD-MobileNet w porównaniu do 86.4% dla YOLOv8.[4]

Jednakże, w kontekście aplikacji działającej nieprzerwanie przez długi czas (np. cały mecz), kluczowym, często niedocenianym parametrem jest zużycie pamięci RAM. W tym samym badaniu model SSD-MobileNet zużywał zaledwie 0.14 GB RAM, podczas gdy YOLOv8 wymagał 0.41 GB.[4] Ta trzykrotna różnica ma krytyczne znaczenie. Systemy operacyjne urządzeń mobilnych, takie jak Android i iOS, agresywnie zarządzają pamięcią i bez wahania zamykają aplikacje, które zużywają jej zbyt wiele, aby zapewnić płynne działanie systemu. Aplikacja, która przez 45 minut nagrywania meczu będzie utrzymywać wysokie zużycie RAM, jest narażona na nagłe zamknięcie przez system, co skutkowałoby utratą nagrania i frustracją użytkownika.

Wybór modelu nie jest zatem prostą decyzją "szybkość kontra dokładność". Należy go rozpatrywać w kontekście całego systemu. Chociaż YOLOv8n oferuje niższy czas inferencji, to całkowita latencja potoku przetwarzania (pobranie klatki, transfer danych, pre-processing, inferencja, post-processing) może sprawić, że różnica kilku milisekund w czasie samej inferencji stanie się nieistotna. W takim scenariuszu, znacznie niższe zużycie pamięci przez SSD-MobileNetV2 staje się decydującą zaletą, prowadząc do stworzenia bardziej stabilnej i niezawodnej aplikacji. Rekomendacją jest rozpoczęcie prototypowania z łatwiejszym w implementacji YOLOv8n, ale z gotowością do przejścia na SSD-MobileNetV2, jeśli testy długodystansowe wykażą problemy ze stabilnością aplikacji.

Tabela 1: Porównanie modeli detekcji obiektów na urządzeniach mobilnych

Model	Typ Architektury	Typowe mobilne FPS	Względna Dokładność	Zużycie RAM	Kluczowa Zaleta	Kluczowa Wada
YOLOv8	Jednoeta	40-155 [3]	Wysoka	Wysokie	Ekstremal	Znaczne

n	powy (Single-Pass)			(ok. 0.41 GB) [4]	na szybkość inferencji	zużycie pamięci RAM
SSD-MobileNetV2	Jednoetapowy (Multi-Scale)	22-46 [3]	Bardzo wysoka	Niskie (ok. 0.14 GB) [4]	Niskie zużycie RAM, stabilność	Niższa szybkość inferencji

1.2. Śledzenie Obiektów: Ocena Klasycznych Trackerów Opartych na Korelacji

Uruchamianie energochłonnego detektora na każdej klatce wideo jest nieefektywne i prowadziłoby do szybkiego przegrzania urządzenia. Zamiast tego, stosuje się strategię, w której lżejszy, klasyczny algorytm śledzący (tracker) przejmuje zadanie podążania za obiektem pomiędzy kolejnymi, rzadszymi detekcjami. Idealnymi kandydatami do tego zadania są trackery oparte na filtrach korelacyjnych, takie jak CSRT i KCF, dostępne w bibliotece OpenCV.

- **CSRT (Channel and Spatial Reliability Tracking):** Jest powszechnie uznawany za jeden z najdokładniejszych klasycznych trackerów.[5] Wykazuje dużą odporność na częściową okluzję (zasłonięcie obiektu) i rotację. Jego główną wadą jest jednak znacznie niższa prędkość działania w porównaniu do alternatyw.[5]
- **KCF (Kernelized Correlation Filters):** Ceniony jest przede wszystkim za wysoką prędkość, co czyni go doskonałym wyborem do aplikacji czasu rzeczywistego.[5] Jego dokładność jest niższa niż w przypadku CSRT, a algorytm ma większe trudności z radzeniem sobie ze znacznymi zmianami skali śledzonego obiektu oraz z długotrwałym dryfem, gdy obiekt jest zasłonięty.[7]

Ocena tych algorytmów musi uwzględniać ich rolę w szerszej architekturze. Nie działają one w izolacji, lecz w ramach potoku "Wykryj, a następnie Śledź" (Detect-then-Track). W tej architekturze, detektor (YOLO/SSD) jest uruchamiany okresowo, na przykład co 10-15 klatek, aby skorygować pozycję trackera. Oznacza to, że tracker nie musi być odporny na długoterminowy dryf czy poważne okluzje, ponieważ jego ewentualne błędy zostaną naprawione przy następnej detekcji. W tym specyficznym kontekście, główna słabość KCF (podatność na dryf) staje się nieistotna, a jego największa zaleta (prędkość) wysuwa się na pierwszy plan. Wybór szybszego KCF pozwoli na przetworzenie większej liczby klatek na sekundę, co przełoży się na płynniejszy obraz wynikowy, podczas gdy zadanie korekcji i ponownego znalezienia obiektu spoczywa na barkach znacznie dokładniejszego detektora.

1.3. Hybrydowa Strategia "Detect-then-Track": Architektura dla Wydajności i Stabilności

Ta hybrydowa strategia jest fundamentem nowoczesnych i wydajnych systemów śledzenia obiektów.[8] Łączy ona w sobie to, co najlepsze w obu światach: wysoką dokładność detektorów opartych na głębokim uczeniu oraz wysoką prędkość klasycznych algorytmów śledzących.

Architektura Potoku Przetwarzania:

1. **Klatka 0 (Inicjalizacja):** Uruchom kosztowny obliczeniowo detektor (np. YOLOv8n), aby znaleźć piłkę. Na podstawie uzyskanego prostokąta otaczającego (bounding box) zainicjuj szybki tracker (np. KCF). Zainicjuj również Filtr Kalmana, używając tej pozycji jako pierwszego pomiaru.
2. **Klatki 1 do N-1 (Śledzenie):** Dla każdej nowej klatki, użyj szybkiego trackera KCF do aktualizacji pozycji piłki. Użyj tej nowej, "surowej" pozycji, aby zaktualizować stan Filtra Kalmana, który zwróci wygładzoną i predykowaną pozycję.
3. **Klatka N (Korekta):** Ponownie uruchom kosztowny detektor.
4. **Asocjacja Danych:** Porównaj wynik nowej detekcji z aktualną pozycją zwróconą przez tracker (np. poprzez obliczenie metryki IoU - Intersection over Union).
    - **Dopasowanie:** Jeśli prostokąty w znacznym stopniu się pokrywają, oznacza to, że tracker poprawnie śledził obiekt. Zresetuj (ponownie zainicjuj) tracker, używając dokładniejszych współrzędnych z detektora. To koryguje wszelki dryf, który mógł nagromadzić się w ciągu ostatnich N klatek.
    - **Brak dopasowania:** Jeśli prostokąty się nie pokrywają, prawdopodobnie tracker zgubił obiekt (np. z powodu długiej okluzji). W takim przypadku odrzuć wynik trackera i użyj nowej detekcji do ponownego zainicjowania śledzenia.
5. **Powtórz:** Wróć do kroku 2.

Ta architektura to coś więcej niż tylko optymalizacja algorytmiczna. Jest to fundamentalna strategia zarządzania energią i zapobiegania dławieniu termicznemu (thermal throttling) na urządzeniu mobilnym. Ciągłe uruchamianie sieci neuronowej generuje ogromne obciążenie dla procesora (CPU), procesora graficznego (GPU) lub jednostki przetwarzania neuronowego (NPU), co prowadzi do gwałtownego drenażu baterii i wydzielania dużej ilości ciepła. Po osiągnięciu krytycznej temperatury, system operacyjny drastycznie obniży taktowanie procesorów, aby zapobiec uszkodzeniu, co spowoduje katastrofalny spadek wydajności aplikacji. Strategia "Detect-then-Track" działa jak przetwarzanie w trybie "burst" – intensywne obliczenia przez jedną klatkę, a następnie bardzo niskie zużycie energii przez N-1 kolejnych klatek. Rozkłada to obciążenie w czasie, utrzymując średnie zużycie energii i generowanie ciepła na znacznie niższym, zrównoważonym poziomie. Jest to zatem warunek konieczny do stworzenia stabilnej, długo działającej aplikacji AI na smartfonie.

1.4. Rozwiązywanie Kluczowych Wyzwań Wizyjnych: Okluzja, Rozmycie w Ruchu i Zmienność Skali

Boisko sportowe to dynamiczne i chaotyczne środowisko. Piłka będzie nieustannie zasłaniana przez graczy, rozmyta z powodu szybkiego ruchu i będzie drastycznie zmieniać swój pozorny rozmiar na ekranie.

- **Okluzja (Occlusion):** Hybrydowa architektura jest z natury odporna na ten problem.[10] Gdy tracker KCF zgubi piłkę, ponieważ ta została zasłonięta przez zawodnika, Filtr Kalmana, opierając się na modelu ruchu, będzie w stanie nadal predykować jej prawdopodobną pozycję przez kilka kolejnych klatek.[11] Kiedy piłka ponownie pojawi się w polu widzenia, następne uruchomienie pełnego detektora YOLO/SSD pozwoli na jej ponowne zlokalizowanie i skorygowanie ścieżki śledzenia.[9]
- **Rozmycie w Ruchu (Motion Blur):** Jest to fizyczne ograniczenie sensora kamery i czasu ekspozycji. Najlepszym sposobem na jego minimalizację jest praca z jak najwyższą liczbą klatek na sekundę.[13] Aplikacja powinna być skonfigurowana tak, aby żądać od API kamery najwyższego dostępnego trybu FPS (np. 60 lub 120 FPS). Nawet jeśli potok AI jest w stanie przetworzyć tylko 15-20 FPS, przechwytywanie obrazu z wyższą częstotliwością zwiększa prawdopodobieństwo, że klatki, które trafią do analizy, będą ostrzejsze. Predykcyjna natura Filtra Kalmana również pomaga "przetrwać" pojedynczą, bardzo rozmytą klatkę, w której detekcja mogłaby zawieść.
- **Zmienność Skali (Scale Variation):** Klasyczne trackery, w tym KCF, mają problemy z adaptacją do szybkich zmian rozmiaru śledzonego obiektu.[7] Jednak w naszej architekturze problem ten jest skutecznie rozwiązywany. Detektory takie jak YOLO i SSD doskonale radzą sobie z wykrywaniem obiektów w różnych skalach.[2] Okresowe uruchamianie detektora będzie więc nieustannie korygować nie tylko pozycję, ale również rozmiar prostokąta otaczającego, z którym inicjowany jest tracker.

Sekcja 2: Logika "Cyfrowego Operatora Kamery"

Posiadanie niezawodnego strumienia współrzędnych piłki to dopiero połowa sukcesu. Drugim, równie ważnym wyzwaniem jest przełożenie tych danych na płynne, inteligentne i estetyczne ruchy wirtualnej kamery, które naśladują pracę profesjonalisty.

2.1. Wygładzanie Ruchu i Predykcja Trajektorii: Implementacja Filtra Kalmana

Surowe współrzędne z detektora lub trackera będą z natury zaszumione i "drżące". Bezpośrednie mapowanie okna kadrującego na te współrzędne stworzyłoby rwany, nieprzyjemny dla oka i przyprawiający o mdłości obraz. Konieczne jest zastosowanie algorytmu wygładzającego i predykcyjnego.

Prostym podejściem jest **średnia ruchoma (moving average)**, która uśrednia N ostatnich pozycji obiektu.[15] Choć jest łatwa w implementacji, ma dwie poważne wady: wprowadza opóźnienie (lag), ponieważ reaguje na zmiany z pewną inercją, i jest czysto reaktywna – nie potrafi przewidywać przyszłej pozycji ani radzić sobie z brakującymi danymi (np. podczas okluzji).

Znacznie lepszym rozwiązaniem jest **Filtr Kalmana**. Jest to rekurencyjny estymator, który modeluje stan dynamicznego systemu (w naszym przypadku, pozycję i prędkość piłki) i optymalnie estymuje go na podstawie zaszumionych pomiarów. Działa on w dwuetapowym cyklu: **Predykcja** (szacuje następny stan na podstawie bieżącego stanu i modelu ruchu) oraz **Korekta/Aktualizacja** (poprawia predykcję, uwzględniając nowy, rzeczywisty pomiar z detektora/trackera).[12] Dzięki temu doskonale nadaje się do wygładzania niestabilnych danych z systemu wizyjnego.[17]

Rola Filtra Kalmana w tej aplikacji wykracza jednak daleko poza proste wygładzanie. Jest on kluczowym, wielofunkcyjnym komponentem spajającym całą architekturę. Potok AI, ze względu na optymalizacje (np. pomijanie klatek), nie będzie dostarczał nowych współrzędnych dla każdej klatki wyświetlanej na ekranie. Przykładowo, interfejs użytkownika może renderować się z prędkością 60 FPS, podczas gdy system AI dostarcza nowe dane tylko 15 razy na sekundę. Co dzieje się w międzyczasie? Aplikacja nie może zamrozić ruchu kamery. I tu właśnie kluczową rolę odgrywa krok **Predykcji** Filtra Kalmana. Na klatkach, dla których mamy nowy pomiar z AI, wykonujemy pełny cykl Predykcja-Korekta. Natomiast na klatkach "pustych", dla których nie ma nowych danych, wykonujemy *tylko* krok Predykcji. Pozwala to na wygenerowanie wysoce prawdopodobnej, estymowanej pozycji piłki i płynną aktualizację interfejsu użytkownika z pełną częstotliwością odświeżania ekranu.

W ten sposób Filtr Kalmana pełni trzy krytyczne funkcje:

1. **Wygładzanie:** Eliminuje drgania i szum z wyjścia trackera.
2. **Predykcja:** Przewiduje pozycję piłki podczas krótkich okluzji, pozwalając "przeczekać" moment zasłonięcia.
3. **Interpolacja:** Wypełnia luki pomiędzy klatkami przetworzonymi przez AI, umożliwiając płynny ruch wirtualnej kamery w interfejsie użytkownika o wysokiej częstotliwości odświeżania.

2.2. Predykcyjne Kadrowanie: Heurystyczne Podejście do Przewidywania Akcji

Profesjonalny operator kamery sportowej nie koncentruje się wyłącznie na piłce. Kadruje ujęcie tak, aby pokazać również graczy biorących udział w akcji, kontekst sytuacji i potencjalne kierunki rozwoju gry.[18] Stworzenie systemu, który w pełni replikuje tę ludzką inteligencję, jest niezwykle złożone. Zaawansowane systemy analityki sportowej wykorzystują dane o pozycji wszystkich graczy i piłki do analizy formacji taktycznych [20], a niektóre prace badawcze pokazują, że możliwe jest nawet przewidywanie przyszłej trajektorii gracza na podstawie jego postawy ciała i pozycji piłki.[22] Istnieją nawet badania, które z powodzeniem przewidują pozycję piłki, analizując *wyłącznie* ustawienie zawodników, co dowodzi silnej korelacji między tymi elementami.[19]

Implementacja takiego systemu, który śledziłby indywidualnie 22 graczy i piłkę w czasie rzeczywistym na jednym smartfonie, jest obliczeniowo niemożliwa. Należy zatem uprościć problem i stworzyć skuteczną heurystykę. Zamiast identyfikować poszczególnych graczy, można wytrenować detektor (YOLO/SSD) do rozpoznawania dwóch ogólnych klas: pilka i gracz. Jest to zadanie znacznie prostsze i obliczeniowo wykonalne.

W każdej przetworzonej klatce system otrzyma współrzędne piłki oraz listę współrzędnych wszystkich widocznych graczy. Na tej podstawie można zdefiniować "centrum akcji" nie jako pozycję samej piłki, ale jako **środek ciężkości (centroid) "klastra akcji"**. Klaster ten składa się z piłki oraz 2-3 graczy znajdujących się najbliżej niej. Wirtualna kamera powinna następnie kadrować ujęcie wokół tego dynamicznie zmieniającego się centrum. Taka heurystyka jest obliczeniowo tania i stanowi doskonałe przybliżenie tego, co robiłby ludzki operator, utrzymując w kadrze nie tylko piłkę, ale i najważniejszych uczestników akcji.

2.3. Logika Dynamicznego Zoomu Cyfrowego i Kadrowania: Algorytmy Kompozycji Ujęcia

Ostatnim elementem warstwy inteligencji jest decyzja o tym, jak szeroko lub wąsko kadrować ujęcie, czyli implementacja dynamicznego zoomu cyfrowego. Profesjonalne zautomatyzowane kamery sportowe wykorzystują kombinację szerokokątnych obiektywów i potężnego zoomu optycznego, aby śledzić akcję.[24] Ich logika często polega na identyfikacji wszystkich istotnych obiektów i takim dopasowaniu ogniskowej, aby wszyscy znaleźli się w kadrze w optymalny sposób.[26] Celem jest stworzenie naturalnego, telewizyjnego wrażenia, unikając gwałtownych i

rozpraszających zmian.[28]

Tę logikę można zaimplementować w formie prostej maszyny stanów, sterowanej rozmiarem wspomnianego wcześniej "klastra akcji".

1. **Obliczenie Klastra:** W każdej klatce, po zidentyfikowaniu piłki i najbliższych graczy, obliczany jest minimalny prostokąt otaczający (bounding box), który zawiera wszystkie te elementy.
2. **Analiza Rozmiaru:** Rozmiar (powierzchnia) tego prostokąta jest bezpośrednim wskaźnikiem tego, jak bardzo "rozproszona" jest akcja na boisku.
3. **Maszyna Stanów Zoomu:**
   - **Stan 1: "Szerokie Ujęcie" (Wide Shot):** Jeśli prostokąt klastra zajmuje dużą część ekranu (np. > 40%), oznacza to, że gracze są rozproszeni. Docelowy poziom zoomu jest ustawiany na niską wartość (np.).
   - **Stan 2: "Wąskie Ujęcie" (Tight Shot):** Jeśli prostokąt klastra jest mały (np. < 15%), oznacza to, że akcja jest bardzo zwarta (np. walka o piłkę). Docelowy poziom zoomu jest ustawiany na wysoką wartość (np.).
   - **Stan 3: "Utrzymanie" (Holding):** Jeśli rozmiar klastra mieści się w wartościach pośrednich, aktualny poziom zoomu jest utrzymywany.
4. **Płynne Przejścia:** Aby uniknąć skokowych zmian, aplikacja nie powinna natychmiastowo ustawiać docelowego zoomu. Zamiast tego, aktualna wartość zoomu powinna być płynnie animowana w kierunku wartości docelowej na przestrzeni kilkuset milisekund.

Taka oparta na prostych regułach maszyna stanów, sterowana geometrycznymi właściwościami "klastra akcji", stanowi solidną i łatwą w implementacji metodę inteligentnego sterowania zoomem cyfrowym, która naśladuje decyzje podejmowane przez ludzkiego operatora.

Sekcja 3: Stos Technologiczny dla Aplikacji Mobilnej (z Fokusem na Flutter)

Ta sekcja opisuje praktyczne aspekty implementacji, koncentrując się na kluczowym interfejsie pomiędzy warstwą interfejsu użytkownika napisaną we Flutterze a wysokowydajnym, natywnym kodem odpowiedzialnym za przetwarzanie AI.

3.1. Integracja Kamery: Dostęp do Surowych Strumieni Obrazu

Aby przeprowadzić inferencję, musimy w jak najwydajniejszy sposób dostarczyć surowe klatki obrazu z sensora kamery do naszego potoku AI.

- **Pakiety Fluttera:** Oficjalny pakiet camera jest standardowym wyborem w ekosystemie Fluttera. Udostępnia on metodę startImageStream, która dostarcza do kodu Dart strumień obiektów typu CameraImage.[30] Kluczową informacją jest to, że format obiektu CameraImage jest zależny od platformy – na Androidzie jest to zazwyczaj YUV_420_888, podczas gdy na iOS BGRA8888.[32] Ta niekompatybilność musi być obsłużona w kodzie natywnym. Alternatywą jest pakiet camerawesome, który oferuje bardziej gotowe komponenty UI, ale może zapewniać mniejszą kontrolę na niskim poziomie.[33]
- **Architektura Potoku Danych:** Najprostsza, intuicyjna implementacja polegałaby na pobraniu obiektu CameraImage w Dart, a następnie przesłaniu jego danych przez Platform Channel do kodu natywnego (Kotlin/Swift) w celu przetworzenia. Jednak doświadczenia deweloperów pokazują, że jest to **największe ryzyko wydajnościowe projektu**.[32] Serializacja i kopiowanie całego bufora obrazu z Dart do warstwy natywnej dla każdej klatki jest operacją niezwykle kosztowną i powolną, która może całkowicie zniweczyć korzyści płynące z szybkiego modelu AI.

Znacznie wydajniejszą i architektonicznie poprawną architekturą jest obsługa kamery i inferencji w całości po stronie natywnej. W tym podejściu potok wygląda następująco: Natywna Kamera -> Natywna Inferencja TFLite -> Przesłanie *jedynie wyników* (współrzędnych prostokąta) z powrotem do Fluttera przez Platform Channel. Taki model eliminuje kosztowny transfer danych obrazu. Rola aplikacji Flutter sprowadza się do warstwy prezentacji: odbiera współrzędne i na ich podstawie aktualizuje transformację widoku kamery.

W kontekście pracy inżynierskiej, należy rozważyć dwa plany:

- **Plan A (Prototyp):** Użycie standardowego pakietu camera i startImageStream, akceptując niższą wydajność. Jest to wystarczające do udowodnienia słuszności koncepcji (Proof of Concept).
- **Plan B (Architektura Docelowa):** Jeśli wydajność Planu A jest niezadowalająca, prawidłowym rozwiązaniem inżynierskim jest napisanie własnego pluginu Flutter, który hermetyzuje zarówno dostęp do kamery, jak i inferencję TFLite w kodzie natywnym, komunikując do Dart tylko finalne wyniki.

3.2. Inferencja na Urządzeniu: Integracja Modeli TensorFlow Lite przez Natywne Platform Channels

Niezależnie od wybranej architektury dostępu do kamery, potrzebujemy sposobu na uruchomienie naszego modelu .tflite.

- **Komunikacja Flutter-Natywny:** Standardowym mechanizmem komunikacji jest

MethodChannel (część Platform Channels). Flutter wywołuje metodę na kanale (np. o nazwie detect), przekazując dane. Kod natywny (w Kotlinie lub Swifcie) nasłuchuje na tym kanale, odbiera dane, wykonuje operacje (konwersja formatu obrazu, uruchomienie interpretera TFLite), a następnie zwraca wynik (współrzędne) z powrotem do Fluttera. Pakiet tflite_flutter jest popularnym wrapperem, ale jak wspomniano, jego bezpośrednie użycie na strumieniu wideo w Dart jest niezalecane dla aplikacji czasu rzeczywistego.[34]

- **Problem "Luki Asynchronicznej":** Wywołanie z Fluttera do kodu natywnego jest operacją asynchroniczną. Flutter wysyła klatkę do przetworzenia i nie czeka na wynik, kontynuując renderowanie interfejsu. Przetwarzanie natywne zajmuje pewien czas (np. 30-60 ms). Zanim wynik (współrzędne) wróci do Fluttera, na ekranie zostało już wyrenderowanych kilka nowych klatek. Otrzymany wynik jest więc "nieaktualny" (stale) względem tego, co użytkownik widzi w danej chwili. To ponownie podkreśla krytyczną rolę Filtra Kalmana, tym razem zaimplementowanego po stronie Dart. Strumień przychodzących (i lekko opóźnionych) współrzędnych z warstwy natywnej służy jako "pomiary" dla filtra. Wygładzone i predykowane wyjście filtra jest następnie używane do aktualizacji transformacji UI na *każdej* klatce renderowania, co niweluje lukę asynchroniczną i zapewnia iluzję płynnego ruchu w czasie rzeczywistym.

3.3. Krytyczna Optymalizacja Wydajności: Kwantyzacja, Akceleracja Sprzętowa i Zarządzanie Energią

Uruchomienie modelu głębokiego uczenia bez agresywnej optymalizacji jest niemożliwe w kontekście aplikacji mobilnej. Jest to krok obowiązkowy, a nie opcjonalny.

- **Kwantyzacja Modelu (Model Quantization):** Jest to proces redukcji precyzji wag w sieci neuronowej, najczęściej z 32-bitowych liczb zmiennoprzecinkowych (FP32) do 8-bitowych liczb całkowitych (INT8).[36] Kwantyzacja może zredukować rozmiar modelu na dysku o około 75% i znacząco przyspieszyć inferencję na CPU, ponieważ operacje na liczbach całkowitych są znacznie szybsze.[36]
- **Delegaty Sprzętowe (Hardware Delegates):** TensorFlow Lite może "delegować" zadanie obliczeń do wyspecjalizowanych komponentów sprzętowych. Na platformie Android dostępne są dwa główne delegaty:
  - **GPU Delegate:** Uruchamia model na procesorze graficznym (GPU) telefonu, który jest zoptymalizowany pod kątem masowych obliczeń równoległych.[34]
  - **NNAPI Delegate:** Wykorzystuje standardowe API Androida (Neural Networks API) do uruchamiania modelu na dedykowanych akceleratorach AI, takich jak NPU (Neural Processing Unit), jeśli urządzenie jest w nie wyposażone.[34]
- **Pomijanie Klatek (Frame Skipping):** Prosta, ale niezwykle skuteczna technika zarządzania energią. Zamiast przetwarzać każdą klatkę otrzymaną z kamery, potok AI jest uruchamiany tylko co N-tą klatkę. To bezpośrednio redukuje obciążenie obliczeniowe i

zużycie energii o czynnik N.

Optymalizacja nie jest procesem teoretycznym, lecz empirycznym zadaniem inżynierskim. Różne techniki są ze sobą powiązane. Na przykład, delegat NNAPI na wielu urządzeniach działa najlepiej (lub wyłącznie) z modelami skwantyzowanymi do INT8. Może się okazać, że skwantyzowany model INT8 działający na CPU (lub NPU przez NNAPI) jest szybszy i bardziej energooszczędny niż model FP32 na GPU. Optymalna konfiguracja jest silnie zależna od konkretnego modelu telefonu. Dlatego praca inżynierska musi zawierać rozdział poświęcony empirycznym testom wydajności. Należy zaimplementować w aplikacji mechanizm do pomiaru FPS i (pośrednio) zużycia baterii dla różnych kombinacji (model FP32 + GPU, model INT8 + NNAPI, model INT8 + CPU) i na tej podstawie wybrać najlepszą konfigurację dla urządzenia docelowego.

Część 2: Praktyczny Plan Rozwoju

Ta sekcja przekłada wiedzę teoretyczną i ustalenia badawcze z Części 1 na konkretny, podzielony na fazy plan działania, który poprowadzi od pustego projektu do działającego, wieloplatformowego prototypu.

Faza 0: Fundamenty (Tygodnie 1-2)

- **Zadanie 1: Konfiguracja środowiska badawczego**
  - Zainstaluj środowisko Python.
  - Zainstaluj kluczowe biblioteki: TensorFlow, OpenCV (pip install opencv-python) oraz Ultralytics (pip install ultralytics). To środowisko będzie służyć jako "piaskownica" do szybkiego testowania modeli i algorytmów przed ich przeniesieniem na platformę mobilną.
  - Skonfiguruj środowisko programistyczne Flutter dla platformy Android. Upewnij się, że jesteś w stanie skompilować i uruchomić standardową aplikację "Hello World" na swoim fizycznym urządzeniu z Androidem.
- **Zadanie 2: Pozyskanie i przygotowanie danych wideo**
  - Wyszukaj i pobierz publicznie dostępne nagrania wideo z amatorskich meczów piłki nożnej (np. z YouTube, Vimeo).
  - **Kluczowe działanie:** Użyj własnego smartfona, aby nagrać kilka 5-10 minutowych klipów z lokalnego meczu. Jest to niezwykle ważne, ponieważ charakterystyka tego wideo (jakość kamery, oświetlenie, kąt widzenia, dynamika ruchu) będzie idealnie odpowiadać docelowemu przypadkowi użycia aplikacji, co sprawi, że testy i prototyp

będą znacznie bardziej miarodajne.

Faza 1: Prototyp na PC - "Bezpieczna Piaskownica" (Tygodnie 3-6)

Celem tej fazy jest szybkie zweryfikowanie podstawowej koncepcji (Proof of Concept) w kontrolowanym środowisku PC, bez zmagania się ze złożonością programowania mobilnego.

- **Zadanie 1: Stworzenie skryptu do przetwarzania wideo**
  - Napisz skrypt w Pythonie, który używa biblioteki OpenCV (cv2.VideoCapture) do wczytywania klatek z przygotowanych wcześniej plików wideo.
- **Zadanie 2: Integracja detekcji i śledzenia obiektu**
  - Wykorzystaj bibliotekę ultralytics. Inicjalizacja modelu i śledzenia jest niezwykle prosta: model = YOLO('yolov8n.pt'), a następnie w pętli results = model.track(frame, persist=True). Metoda .track() automatycznie integruje wydajny algorytm śledzący (np. ByteTrack), który przypisuje stałe ID do wykrytych obiektów, co jest ogromnym ułatwieniem na etapie prototypowania.[38]
  - Skoncentruj się na detekcji obiektu klasy sports ball. Wyodrębnij z wyników współrzędne prostokąta otaczającego piłkę.
- **Zadanie 3: Implementacja logiki "Cyfrowego Operatora Kamery"**
  - Zaimplementuj w Pythonie prosty Filtr Kalmana do wygładzania współrzędnych piłki otrzymanych z modelu.
  - Na podstawie wygładzonych współrzędnych, oblicz docelowe okno kadrowania (crop window).
  - Użyj funkcji OpenCV do wycięcia tego okna z oryginalnej klatki.
  - Wyświetl finalny, wykadrowany obraz w oknie za pomocą cv2.imshow().
  - **Cel do osiągnięcia:** Działający skrypt, który wczytuje wideo i wyświetla wykadrowany, płynnie podążający za piłką obraz. To jest dowód słuszności koncepcji.

Faza 2: Aplikacja Mobilna - Najpierw Android (Tygodnie 7-12)

W tej fazie przenosimy sprawdzoną logikę do docelowego środowiska Flutter/Android.

- **Zadanie 1: Stworzenie szkieletu aplikacji z podglądem kamery**
  - Utwórz nowy projekt Flutter. Dodaj zależność do pakietu camera.[30]
  - Zaimplementuj kod niezbędny do inicjalizacji CameraController i wyświetlenia pełnoekranowego, nieprzetworzonego podglądu z kamery w widgecie CameraPreview. Prawidłowo obsłuż zapytania o uprawnienia dostępu do kamery.
- **Zadanie 2: Przeniesienie modelu AI i implementacja natywnego procesora**

    - Wyeksportuj model YOLOv8n do formatu TensorFlow Lite (.tflite) za pomocą narzędzia CLI od Ultralytics: yolo export model=yolov8n.pt format=tflite int8=True. Flaga int8=True dokona kluczowej kwantyzacji INT8.[36]
    - Dodaj plik .tflite do folderu assets w projekcie Flutter i zadeklaruj go w pubspec.yaml.
    - Po stronie Androida (w android/app/src/...), stwórz nową klasę w Kotlinie (np. ObjectDetector.kt). Będzie ona odpowiedzialna za załadowanie modelu .tflite z zasobów i uruchomienie na nim inferencji przy użyciu natywnej biblioteki TensorFlow Lite dla Androida.
- **Zadanie 3: Ustanowienie komunikacji przez Platform Channels**
    - W kodzie Dart, na CameraController wywołaj startImageStream. Wewnątrz callbacku strumienia, użyj MethodChannel do wywołania metody natywnej (np. o nazwie "runDetection"). Przekaż do niej dane obrazu z obiektu CameraImage (bufory bajtowe, wymiary, rotacja).
    - W głównej aktywności Androida (MainActivity.kt), zaimplementuj MethodChannel.MethodCallHandler. Po otrzymaniu wywołania "runDetection", odbierz dane obrazu, dokonaj niezbędnej konwersji z formatu YUV na Bitmap (format wymagany przez TFLite), a następnie przekaż go do instancji klasy ObjectDetector.
    - Po zakończeniu inferencji, ObjectDetector zwróci listę współrzędnych. Prześlij tę listę z powrotem do Fluttera jako wynik wywołania MethodChannel.
- **Zadanie 4: Implementacja logiki kadrowania i wygładzania w UI Fluttera**
    - W Dart, odbierz listę współrzędnych z warstwy natywnej.
    - Przekaż te współrzędne jako nowe pomiary do implementacji Filtra Kalmana w Dart.
    - **Ważne:** Nie renderuj wyciętego obrazu jako widget Image. Zamiast tego, umieść widget CameraPreview wewnątrz widgetu ClipRect. Następnie, używając widgetów Transform.scale i Transform.translate, dynamicznie przesuwaj i skaluj sam CameraPreview na podstawie wygładzonych współrzędnych z Filtra Kalmana. Jest to znacznie bardziej wydajne podejście.
- **Zadanie 5: Rygorystyczne testy i optymalizacja na fizycznym urządzeniu**
    - Zaimplementuj prosty mechanizm do mierzenia FPS (np. licznik klatek przetworzonych w ciągu sekundy).
    - Przeprowadź testy porównawcze różnych konfiguracji optymalizacyjnych: model INT8 z delegatem NNAPI, model Float16 z delegatem GPU.
    - Zaimplementuj logikę pomijania klatek (frame skipping) – w callbacku startImageStream dodaj licznik i wysyłaj do warstwy natywnej tylko co drugą lub co trzecią klatkę.
    - Przeprowadź testy 10-minutowego działania aplikacji, monitorując temperaturę urządzenia i zużycie baterii. Wybierz konfigurację, która zapewnia akceptowalną płynność (>15 FPS) bez nadmiernego przegrzewania telefonu.

Faza 3: Portowanie na iOS i Finalizacja (Tygodnie 13-15)

Ostatnia faza projektu skupia się na przeniesieniu aplikacji na platformę iOS i dopracowaniu szczegółów.

- **Zadanie 1: Konfiguracja środowiska deweloperskiego dla iOS**
  - Jest to wyzwanie logistyczne. Należy uzyskać dostęp do komputera z systemem macOS. Możliwe opcje to: skorzystanie z laboratorium na uczelni, użycie usługi chmurowej (np. MacinCloud) lub pożyczenie komputera. Ten krok jest absolutnie konieczny do skompilowania aplikacji na iOS.
- **Zadanie 2: Stworzenie natywnego kodu przetwarzającego w języku Swift**
  - Stwórz odpowiednik klasy ObjectDetector.kt w języku Swift (ObjectDetector.swift). Logika pozostaje identyczna: załadowanie modelu .tflite z zasobów i uruchomienie inferencji przy użyciu natywnej biblioteki TensorFlow Lite dla Swifta.
  - W pliku AppDelegate.swift, skonfiguruj FlutterMethodChannel w sposób analogiczny do implementacji w Kotlinie. Należy napisać kod konwertujący dane z CameraImage (które na iOS będą w formacie BGRA) na tensor wejściowy wymagany przez model.
- **Zadanie 3: Kompilacja, testowanie i wdrożenie na iOS**
  - Na komputerze Mac, uruchom w terminalu polecenie flutter build ios.
  - Otwórz wygenerowany projekt Runner.xcworkspace w środowisku Xcode. Skonfiguruj podpisywanie kodu (code signing) za pomocą swojego konta deweloperskiego Apple.
  - Skompiluj i uruchom aplikację najpierw na symulatorze iOS, a następnie na fizycznym iPhonie.
  - Zdiagnozuj i napraw wszelkie problemy specyficzne dla platformy iOS. Logika napisana w Dart powinna działać bez zmian; praca skupi się głównie na integracji natywnego kodu Swift.

Cytowane prace

1. YOLOv8 vs. MobileNet SSD v2: Compared and Contrasted - Roboflow, otwierano: września 30, 2025, https://roboflow.com/compare/yolov8-vs-mobilenet-ssd-v2
2. Comparison between YOLO and SSD MobileNet for Object Detection in a Surveillance Drone - ResearchGate, otwierano: września 30, 2025, https://www.researchgate.net/publication/355336797_Comparison_between_YOLO_and_SSD_MobileNet_for_Object_Detection_in_a_Surveillance_Drone
3. YOLOv8 vs SSD: Choosing the Right Object Detection Model - Keylabs, otwierano: września 30, 2025, https://keylabs.ai/blog/yolov8-vs-ssd-choosing-the-right-object-detection-model/
4. (PDF) Comparative Performance of Yolov8 and Ssd-mobilenet Algorithms for Road Damage Detection in Mobile Applications - ResearchGate, otwierano: września 30, 2025, https://www.researchgate.net/publication/394048272_Comparative_Performance_of_Yolov8_and_Ssd-mobilenet_Algorithms_for_Road_Damage_Detection_in_Mobile_Applications

5. Top OpenCV Tracking Algorithms Explained - Sighthound, otwierano: września 30, 2025, https://www.sighthound.com/blog/opencv-object-tracking-algorithms
6. Object Tracking Algorithms in OpenCV - Scaler Topics, otwierano: września 30, 2025, https://www.scaler.com/topics/object-tracking-algorithms/
7. OBJECT TRACKING. Techniques, Algorithms, and Practical... | by KHWAB KALRA | Medium, otwierano: września 30, 2025, https://medium.com/@khwabkalra1/object-tracking-2fe4127e58bf
8. Real-Time Target Detection and Tracking: A Comparative In-depth Review of Strategies, otwierano: września 30, 2025, https://www.researchgate.net/publication/256477560_Real-Time_Target_Detection_and_Tracking_A_Comparative_In-depth_Review_of_Strategies
9. (PDF) Real-time object detection and tracking for mobile robot using ..., otwierano: września 30, 2025, https://www.researchgate.net/publication/376199581_Real-time_object_detection_and_tracking_for_mobile_robot_using_YOLOv8_and_Strong_SORT
10. Object Detection Dilemmas: Conquering Occlusions, Scale, and Pose Variations | by Jing li, otwierano: września 30, 2025, https://medium.com/@jingli0155/object-detection-dilemmas-conquering-occlusions-scale-and-pose-variations-08d8e888b69e
11. TOTNet: Occlusion-Aware Temporal Tracking for Robust Ball Detection in Sports Videos - arXiv, otwierano: września 30, 2025, https://arxiv.org/pdf/2508.09650?
12. Real-Time Object Tracking Using YOLOv5, Kalman Filter & Hungarian Algorithm - Medium, otwierano: września 30, 2025, https://medium.com/@kevinnjagi83/real-time-object-tracking-using-yolov5-kalman-filter-hungarian-algorithm-9bd0e5a94c5a
13. Multi-Scale Attention-Augmented YOLOv8 for Real-Time Surface Defect Detection in Fresh Soybeans - MDPI, otwierano: września 30, 2025, https://www.mdpi.com/2227-9717/13/10/3040
14. High-Frame-Rate Global Shutter Cameras in Sports - Smartgiant, otwierano: września 30, 2025, https://www.smartgiant.com/high-frame-rate-global-shutter-cameras-in-sports/
15. The Method About Decreasing Vibration in Video With Two Ways: Simple Moving Average(SMA) and Kalman Filter(KF) | by ben60523 | Medium, otwierano: września 30, 2025, https://medium.com/@ben60523/the-method-about-decreasing-vibration-in-video-with-two-ways-simple-moving-average-sma-and-7f6f8030bb94
16. Time series filtering algorithms: a brief overview | by Vladislav Klekovkin | Deelvin Machine Learning | Medium, otwierano: września 30, 2025, https://medium.com/deelvin-machine-learning/time-series-filtering-algorithms-a-brief-overview-af2d3112cd03
17. Zero-Shot Player Tracking in Tennis with Kalman Filtering | Towards Data Science, otwierano: września 30, 2025, https://towardsdatascience.com/zero-shot-player-tracking-in-tennis-with-kalman-filtering-80bba73a4247/
18. AI-Powered Sports Video Analysis Software - Folio3 AI, otwierano: września 30,

2025, https://www.folio3.ai/ai-in-sports/video-analysis/
19. Prediction of the Ball Location on the 2D Plane in Football Using Optical Tracking Data - Open METU, otwierano: września 30, 2025, https://open.metu.edu.tr/bitstream/handle/11511/95277/10.21541-apjess.1060725-2203586.pdf
20. Player Tracking Data in Sports - Annual Reviews, otwierano: września 30, 2025, https://www.annualreviews.org/doi/10.1146/annurev-statistics-033021-110117
21. Validation of Player and Ball Tracking with a Local Positioning System - MDPI, otwierano: września 30, 2025, https://www.mdpi.com/1424-8220/21/4/1465
22. Using Transformers on Body Pose to Predict Tennis Player's Trajectory - arXiv, otwierano: września 30, 2025, https://arxiv.org/html/2411.04501v1
23. Ball Tracking in Association Football - Leveraging Player Detections to Locate the Ball Using Graph Attention Networks, otwierano: września 30, 2025, https://lup.lub.lu.se/student-papers/record/9172881/file/9172882.pdf
24. Top Sports Tracking Cameras in 2025 - OBSBOT, otwierano: września 30, 2025, https://www.obsbot.com/blog/sports/sports-tracking-cameras
25. The new S-Line POINT Zoom - Spiideo, otwierano: września 30, 2025, https://www.spiideo.com/news/the-new-s-line-point-zoom/
26. Stay in Focus: The Benefits of AI Auto-Framing in Conference Cameras, otwierano: września 30, 2025, https://funtechinnovation.com/blog-auto-framing/
27. Auto-Framing (Group Tracking) Feature - PTZOptics, otwierano: września 30, 2025, https://ptzoptics.com/auto-framing-group-tracking-feature/
28. Automatic Video Zooming for Sport Team Video Broadcasting on Smart Phones., otwierano: września 30, 2025, https://www.researchgate.net/publication/221415478_Automatic_Video_Zooming_for_Sport_Team_Video_Broadcasting_on_Smart_Phones
29. PTZ Auto Framing - Sony Pro, otwierano: września 30, 2025, https://pro.sony/en_DJ/products/ptz-network-cameras/ptz-auto-framing
30. Take a picture using the camera - Flutter Documentation, otwierano: września 30, 2025, https://docs.flutter.dev/cookbook/plugins/picture-using-camera
31. camera | Flutter package - Pub.dev, otwierano: września 30, 2025, https://pub.dev/packages/camera
32. Real-time Machine Learning with Flutter Camera | KBTG Life - Medium, otwierano: września 30, 2025, https://medium.com/kbtg-life/real-time-machine-learning-with-flutter-camera-bbcf1b5c3193
33. Top Flutter Camera Image Capture, Video Recorder packages - Flutter Gems, otwierano: września 30, 2025, https://fluttergems.dev/camera/
34. Integrating TensorFlow Lite Models for On-Device AI in Flutter, otwierano: września 30, 2025, https://vibe-studio.ai/insights/integrating-tensorflow-lite-models-for-on-device-ai-in-flutter
35. tflite_flutter | Flutter package - Pub.dev, otwierano: września 30, 2025, https://pub.dev/packages/tflite_flutter
36. On-Device AI: Revolutionizing Inference Speed with Optimization ..., otwierano:

września 30, 2025, https://zetic.ai/blog/on-device-ai-revolutionizing-inference-speed-with-optimization-techniques
37. How to Optimize AI Models for Mobile Devices Without Overloading ..., otwierano: września 30, 2025, https://medium.com/@sridinu03/how-to-optimize-ai-models-for-mobile-devices-without-overloading-them-305ca8093abe
38. YOLOv8 Object Tracking: A How-To Guide - Roboflow, otwierano: września 30, 2025, https://roboflow.com/how-to-track/yolov8
39. Multi-Object Tracking with Ultralytics YOLO, otwierano: września 30, 2025, https://docs.ultralytics.com/modes/track/