Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Wydział Informatyki

Projekt dyplomowy

Biblioteka Datasets dla Elixira

Datasets library for Elixir

Autorzy: Radosław Rolka, Weronika Wojtas
Kierunek studiów: Informatyka
Opiekun pracy: dr inż. Aleksander Smywiński-Pohl

Kraków, 2026

Spis treści

1 Cel prac i wizja produktu **5**
- 1.1 Opis dziedziny problemu . . . 5
- 1.2 Motywacja . . . 6
- 1.3 Rola produktu . . . 7
- 1.4 Obszary funkcjonalne . . . 7
- 1.5 Wymagania niefunkcjonalne . . . 8
- 1.6 Przegląd dostępnych rozwiązań . . . 8
- 1.7 Analiza technologiczna . . . 8
- 1.8 Analiza ryzyka . . . 9
- 1.9 Podsumowanie . . . 9

2 Zakres funkcjonalności **10**
- 2.1 Charakterystyka użytkownika . . . 10
- 2.2 Charakterystyka systemów współpracujących . . . 10
- 2.3 Wymagania funkcjonalne . . . 11
- 2.4 Wymagania jakościowe . . . 12
- 2.5 Scenariusze użytkowania . . . 13
- 2.6 Podsumowanie . . . 13

3 Wybrane aspekty realizacji **15**
- 3.1 Architektura systemu . . . 15
- 3.2 Stos technologiczny . . . 16
  - 3.2.1 Elixir i biblioteka standardowa . . . 16
  - 3.2.2 Explorer . . . 16
  - 3.2.3 Mix . . . 16
  - 3.2.4 ExDoc . . . 16
  - 3.2.5 ExUnit i ExCoveralls . . . 17
- 3.3 Przegląd poszczególnych komponentów . . . 17
  - 3.3.1 Moduł Interfejsu . . . 17
  - 3.3.2 Moduł HuggingFace . . . 17
  - 3.3.3 Moduł networkingowy . . . 18
- 3.4 Ciekawsze algorytmy i mechanizmy systemu . . . 18
  - 3.4.1 Cacheowanie danych . . . 19
  - 3.4.2 Pasek Postępu (ProgressBar) . . . 19
  - 3.4.3 Przykłady LiveBook . . . 20
- 3.5 Zapewnienie jakości . . . 20
  - 3.5.1 Testy jednostkowe . . . 20
  - 3.5.2 Analiza statyczna kodu . . . 20

3.5.3 Ciągła integracja (CI) . . . 22
3.5.4 Code review . . . 22
3.6 Podsumowanie . . . 22

4 Organizacja pracy **23**
4.1 Charakterystyka projektu . . . 23
4.2 Osoby w projekcie . . . 23
4.3 Zespół i podział obowiązków . . . 24
4.4 Organizacja prac i wykorzystane narzędzia . . . 25
4.4.1 Metodyka i Organizacja Pracy . . . 25
4.4.2 Komunikacja i Spotkania . . . 25
4.4.3 Narzędzia Wykorzystane w Projekcie . . . 25
4.5 Zastosowane techniki i praktyki . . . 26
4.5.1 Ciągła integracja (CI/CD) z GitHub Actions . . . 26
4.5.2 Moduledoc - Dokumentacja i przykłady w kodzie . . . 26
4.5.3 Refaktoryzacja kodu . . . 27
4.5.4 Pair programming . . . 27
4.5.5 Test Driven Development (TDD) . . . 27
4.6 Przebieg prac . . . 28
4.6.1 Faza Koncepcyjna i Planowania Architektury . . . 28
4.6.2 Faza Implementacji . . . 28
4.6.3 Faza Testowania, Optymalizacji i Dokumentacji . . . 28
4.7 Podsumowanie . . . 29

5 Wyniki projektu **30**
5.1 Zrealizowane funkcjonalnosci . . . 30
5.2 Główne scenariusze uzytkowania . . . 30
5.2.1 Ładowanie zbioru danych z repozytorium Hugging Face . . . 31
5.2.2 Ładowanie zbiorów danych z lokalnego systemu plików . . . 33
5.2.3 Wykorzystanie mechanizmu cache'owania i trybu offline . . . 33
5.2.4 Strumieniowe przetwarzanie dużych zbiorów danych . . . 35
5.2.5 Równoległe ładowanie i przetwarzanie danych . . . 35
5.2.6 Publikowanie zbiorów danych w repozytorium Hugging Face . . . 37
5.2.7 Pobieranie informacji i metadanych o zbiorach danych . . . 38
5.3 Dalszy rozwój projektu . . . 40
5.4 Subiektywna ocena projektu . . . 40
5.5 Podsumowanie . . . 41

Spis rysunków **43**

Spis tabel **44**

Spis listingów **45**

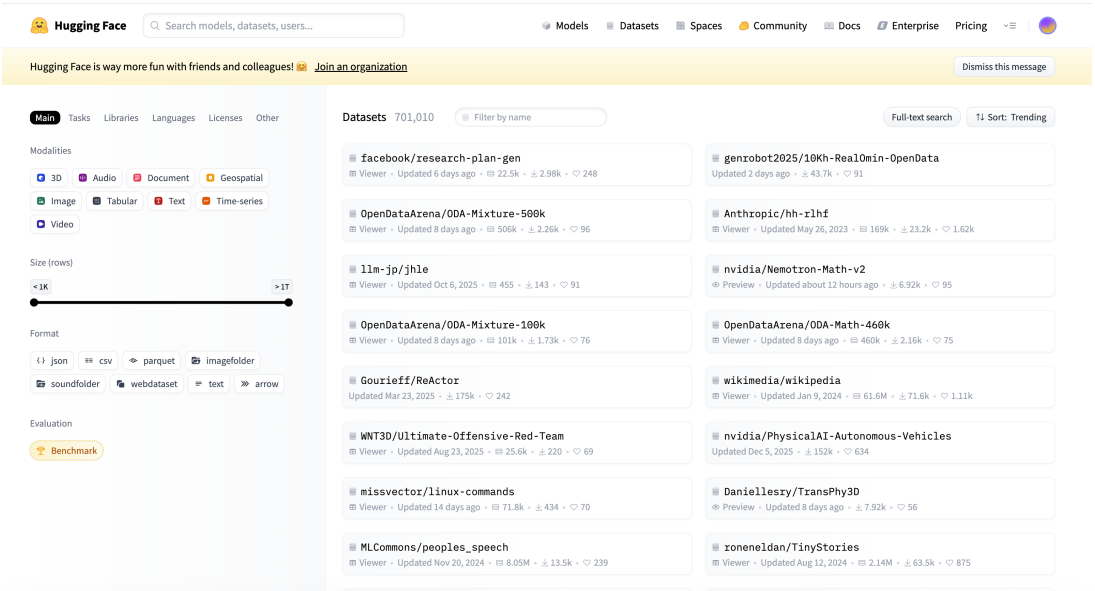
Rozdział 1

Cel prac i wizja produktu

Celem pracy jest stworzenie biblioteki w języku Elixir, która umożliwi łatwe pobieranie, przetwarzanie i zarządzanie zbiorami danych, które są powszechnie wykorzystane w uczeniu maszynowym. Biblioteka powinna oferować szeroki wybór gotowych zbiorów danych, a także możliwość dodawania własnych.

1.1. Opis dziedziny problemu

Praca z modelami uczenia maszynowego jest ściśle związana z wykorzystaniem danych, które stanowią podstawę procesów trenowania oraz walidacji algorytmów. Dostęp do odpowiednio przygotowanych i zróżnicowanych zbiorów danych ma kluczowe znaczenie dla jakości i skuteczności uczenia modeli. Dane muszą być nie tylko obszerne i reprezentatywne, lecz także poddane procesom czyszczenia, normalizacji oraz formatowania, co często wiąże się z dużym nakładem czasu i zasobów. Dodatkowym wyzwaniem jest różnorodność źródeł oraz formatów danych, która utrudnia ich bezpośrednie wykorzystanie w systemach uczenia maszynowego bez wcześniejszego przetwarzania. W odpowiedzi na te problemy coraz większe znaczenie zyskują publiczne repozytoria danych, takie jak Hugging Face Datasets. Platforma ta udostępnia bardzo dużą liczbę otwartych zbiorów danych obejmujących m.in. dane tekstowe, obrazy, nagrania dźwiękowe oraz zbiory multimodalne. Dane te są zazwyczaj ujednolicone strukturalnie i opisane metadanymi, co umożliwia ich szybkie wykorzystanie w procesie trenowania modeli. Integracja repozytorium z popularnymi bibliotekami uczenia maszynowego znacząco upraszcza pracę z danymi i redukuje czas potrzebny na ich przygotowanie.



Rysunek 1.1: Zrzut strony HuggingFace Datasets

1.2. Motywacja

Podczas studiów zainteresowaliśmy się językami programowania funkcyjnego, szczególnie Elixirem. Choć na początku jego podejście może wydawać się nietypowe, szybko dostrzegliśmy, jak prosty i elegancki jest ten język. Podczas pracy z Elixirem zauważyliśmy, że brakuje w nim biblioteki do zarządzania zbiorami danych, która w innych językach jest szeroko stosowana, szczególnie w sztucznej inteligencji.

Postanowiliśmy, że to będzie temat naszej pracy inżynierskiej. Chcemy stworzyć bibliotekę w Elixirze, inspirowaną Hugging Face Datasets [4], która pozwoli programistom łatwiej pracować ze zbiorami danych i ułatwi dostęp do nich.

1.3. Rola produktu

Głównym celem biblioteki jest uproszczenie i automatyzacja zarządzania, przetwarzania oraz optymalizacja zbiorów danych. Umożliwienie łatwego dostępu do różnorodnych zbiorów danych pozwoli na szybsze rozpoczęcie pracy nad projektami, eliminując konieczność manualnego zbierania i konfigurowania danych. Integracja funkcji automatycznego czyszczenia, normalizacji, skalowania i augmentacji danych znacząco zredukuje czasochłonne procesy przygotowywania danych, co jest zazwyczaj barierą w szybkim prototypowaniu i testowaniu modeli uczenia maszynowego. Użytkownikami końcowymi projektowanej biblioteki są przede wszystkim analitycy, specjaliści od uczenia maszynowego oraz studenci zajmujący się analizą danych i sztuczną inteligencją, którym to narzędzie ma za zadanie zwiększyć produktywność poprzez automatyzację rutynowych zadań.

1.4. Obszary funkcjonalne

1. **Pobieranie i zarządzanie zbiorami danych**

    - Pobieranie gotowych zbiorów danych z różnych źródeł: Implementacja mechanizmów umożliwiających pobieranie danych z platform takich jak Hugging Face Hub [1], Kaggle [3] czy innych repozytoriów.
    - Dodawanie własnych zbiorów danych: Możliwość integracji i zarządzania własnymi zestawami danych w systemie.

2. **Przeglądanie zbiorów danych**

    - Łatwe przeglądanie dostępnych zbiorów danych: Interfejsy umożliwiające szybki podgląd i analizę dostępnych danych.
    - Filtrowanie zbiorów danych: Narzędzia do selekcji danych na podstawie określonych kryteriów.

3. **Przetwarzanie i transformacja danych**

    - Czyszczenie danych: Funkcje do usuwania błędów i niekompletnych rekordów.
    - Normalizacja danych: Metody standaryzacji wartości w zbiorach danych.
    - Tokenizacja: Proces dzielenia tekstu na mniejsze jednostki, takie jak słowa czy zdania.
    - Tworzenie podzbiorów danych: Możliwość dzielenia większych zbiorów na mniejsze, bardziej zarządzalne części.

4. **Przykłady rozwiązań**

    - Przykłady użycia: Praktyczne scenariusze i case studies demonstrujące zastosowanie poszczególnych funkcji.

1.5. Wymagania niefunkcjonalne

Wymagania niefunkcjonalne odgrywają kluczową rolę w zapewnieniu, że stworzona biblioteka nie tylko spełni swoje zadania funkcjonalne, ale również będzie przyjazna dla użytkownika. Poniżej przedstawiono główne wymagania niefunkcjonalne dla projektu:

- Wydajność - Biblioteka powinna efektywnie zarządzać i przetwarzać duże zbiory danych z minimalnym opóźnieniem.
- Kompatybilność - Interfejs powinien być kompatybilny z różnymi systemami operacyjnymi i integrować się z istniejącymi popularnymi narzędziami i bibliotekami w ekosystemie Elixira.
- Dokumentacja - Kompletna i zrozumiała dokumentacja techniczna jest niezbędna, by użytkownicy mogli efektywnie wykorzystywać wszystkie funkcje biblioteki.

1.6. Przegląd dostępnych rozwiązań

Jednym z głównych narzędzi w tej dziedzinie rozwiązań jest biblioteka datasets od Hugging Face [4], która jest szeroko stosowana w społeczności uczenia maszynowego. Biblioteka datasets oferuje łatwy dostęp do szerokiej gamy zbiorów danych w różnych językach programowania. Oferuje ona również różnorodne narzędzia do przetwarzania i transformacji danych.

W kontekście Elixira, który nadal jest dynamicznie rozwijającym się językiem, nie istnieje jeszcze takie narzędzie, które w pełni odpowiadałoby potrzebom użytkowników w zakresie zarządzania i przetwarzania danych dla uczenia maszynowego, co tworzy przestrzeń na rynku dla nowego rozwiązania, które może lepiej odpowiadać na unikalne potrzeby społeczności Elixira, zwiększając efektywność ich pracy dzięki specjalizowanym narzędziom dostosowanym do ich środowiska i metod pracy.

1.7. Analiza technologiczna

Stos technologiczny został zaprojektowany z myślą o maksymalnym wykorzystaniu możliwości języka Elixir oraz jego ekosystemu. Do obliczeń numerycznych oraz operacji na tensorach wykorzystamy bibliotekę NX [5]. Biblioteka zapewnia wydajność w przeprowadzaniu operacji matematycznych, szczególnie w kontekście obliczeń związanych z dużymi zbiorami danych i sztuczną inteligencją. NX oferuje wsparcie dla operacji na tensorach, które są kluczowe w procesach uczenia maszynowego oraz analizy danych.

Zintegrowany z NX jest także Explorer [2], który będziemy wykorzystywać w naszej pracy do efektywnego zarządzania i analizy danych. Explorer to biblioteka, która umożliwia pracę z dwoma głównymi typami struktur danych: seriami, oraz dataframe'ami. Te struktury pozwalają na wygodne i szybkie eksplorowanie danych, co jest szczególnie istotne podczas analizy informacji. Explorer, jako backend, korzysta z Polars, biblioteki napisanej w języku Rust co przekłada się na znaczną poprawę wydajności w obliczeniach z dużymi zbiorami.

Do współpracy z modelami głębokiego uczenia maszynowego w naszym projekcie zastosujemy bibliotekę Bumblebee [8], która pozwala na łatwą integrację z pretrenowanymi modelami sieci neuronowych. Bumblebee umożliwia dostęp do popularnych modeli, które zostały udostępnione przez platformy sztucznej inteligencji, takie jak Hugging Face Transformers [9]. Ta

biblioteka umożliwi łatwą implementację i wykorzystanie zaawansowanych modeli AI w naszej pracy, co pozwoli na efektywne wdrożenie algorytmów uczenia maszynowego i głębokiego uczenia w środowisku Elixira.

1.8. Analiza ryzyka

W procesie projektowania i rozwijania nowej biblioteki istnieje wiele potencjalnych ryzyk, których zidentyfikowanie pozwala na lepsze przygotowanie, co z kolei zwiększa szanse na pomyślne zakończenie projektu. Są to między innymi:

- Adaptacja przez społeczność - Jako że Elixir jest stosunkowo mniej popularny niż inne języki wykorzystywane w dziedzinie uczenia maszynowego, takie jak Python, istnieje ryzyko, że biblioteka nie zyska szerokiego grona użytkowników. Promocja biblioteki i demonstrowanie jej wartości w rzeczywistych projektach będzie kluczowe.
- Integracja z istniejącymi narzędziami - Problemy z integracją nowej biblioteki z już istniejącymi ekosystemami i narzędziami, których niekompatybilność może powstrzymać potencjalnych użytkowników przed korzystaniem z biblioteki.
- Obsługą dużych zbiorów danych - Możliwe, że biblioteka nie będzie w stanie efektywnie procesować dużych zbiorów danych lub że wystąpią problemy z wydajnością.
- Niedostateczne testowanie - Niewystarczające testowanie w różnych środowiskach i scenariuszach użytkowania może prowadzić do niezauważonych błędów, które ujawnią się dopiero po wdrożeniu biblioteki.

1.9. Podsumowanie

Projekt ma na celu stworzenie biblioteki w języku Elixir, która będzie odpowiadała funkcjonalności biblioteki Hugging Face Datasets [4], umożliwiając łatwe pobieranie, przetwarzanie i zarządzanie zbiorami danych używanymi w uczeniu maszynowym. Biblioteka ta oferować będzie funkcje takie jak pobieranie gotowych zbiorów danych z różnych źródeł, możliwość dodawania własnych zbiorów danych, filtrowanie i przeglądanie dostępnych zbiorów, a także przetwarzanie danych (czyszczenie, normalizacja, tokenizacja). Użytkownicy będą mogli tworzyć podzbiory danych oraz integrować bibliotekę z innymi narzędziami w Elixirze, takimi jak Nx [5], Explorer [2] i Bumblebee [8]. Projekt zakłada również dostarczenie pełnej dokumentacji oraz przykładów użycia.

Rozdział 2

Zakres funkcjonalności

Celem niniejszego rozdziału jest przedstawienie specyfikacji funkcjonalnej projektowanej biblioteki. Specyfikacja została opracowana na podstawie analizy potrzeb użytkowników i rozmów konsultacyjnych z zainteresowanymi stronami.

2.1. Charakterystyka użytkownika

System zakłada istnienie jednego głównego rodzaju użytkownika – **programisty pracującego z językiem Elixir**, który zajmuje się tworzeniem, trenowaniem lub walidacją modeli uczenia maszynowego. Użytkownicy ci mogą być częścią większych zespołów badawczo-rozwojowych lub niezależnymi deweloperami.

Zakłada się, że użytkownik:

- posiada podstawową lub zaawansowaną znajomość języka Elixir,
- zna podstawy uczenia maszynowego oraz pracy z danymi,
- wymaga efektywnego i prostego dostępu do przygotowanych zbiorów danych,
- oczekuje narzędzi ułatwiających wstępne przetwarzanie danych, takich jak czyszczenie, normalizacja, tokenizacja.

2.2. Charakterystyka systemów współpracujących

Tworzona biblioteka do zarządzania zbiorami danych w języku Elixir zakłada ścisłą współpracę z wybranym zestawem zewnętrznych narzędzi i bibliotek, które pełnią kluczową rolę w zapewnieniu pełnej funkcjonalności systemu. Integracja tych komponentów pozwala na maksymalne wykorzystanie potencjału języka Elixir w kontekście przetwarzania danych na potrzeby uczenia maszynowego. Poniżej przedstawiono charakterystykę najważniejszych współpracujących systemów:

- **Źródła zbiorów danych** – Biblioteka umożliwia pobieranie popularnych zbiorów danych wykorzystywanych w uczeniu maszynowym z różnych publicznych repozytoriów, takich jak Hugging Face Datasets [4]. W związku z tym wymagana jest integracja z usługami HTTP i obsługą różnorodnych formatów danych.

- **Nx** [5] – Biblioteka opiera się na integracji z narzędziem Nx, które zapewnia funkcje numeryczne i przetwarzanie tensorów. Współpraca z Nx pozwala na bezproblemowe przygotowanie danych do trenowania modeli uczenia maszynowego w Elixirze.
- **Explorer** [2] – Do eksploracji, filtrowania i transformacji danych tabularycznych wykorzystywana jest biblioteka Explorer , która umożliwia przetwarzanie danych w sposób zbliżony do narzędzi takich jak Pandas [7] w Pythonie. Dzięki temu użytkownik może łatwo analizować i przygotowywać dane w ramach jednego ekosystemu.
- **Bumblebee** [8] – W celu dalszego wykorzystania przetworzonych danych, możliwa jest integracja z biblioteką Bumblebee, która dostarcza gotowe modele i narzędzia do pracy z NLP i uczeniem głębokim.
- **Lokalna pamięć masowa** – System wspiera lokalne przechowywanie danych oraz cache'owanie zbiorów, aby zminimalizować potrzebę wielokrotnego pobierania tych samych danych i zwiększyć wydajność pracy z dużymi zestawami.

2.3. Wymagania funkcjonalne

W poniższym rozdziale szczegółowo opisano wymagania funkcjonalne dla projektowanej biblioteki w języku Elixir, klasyfikując je zgodnie z metodologią MoSCoW [6] ukierunkowaną na najbardziej krytyczne aspekty funkcjonalności systemu.

Must Have

- Możliwość pobierania datasetów z zewnętrznych źródeł – biblioteka musi umożliwić użytkownikowi łatwy dostęp do zasobów danych oferowanych przez różne repozytoria.
- Integracja z narzędziami Elixir takimi jak Nx i Explorer – niezbędne jest zapewnienie kompatybilności i efektywnej współpracy narzędziowej.
- Pełna dokumentacja funkcji biblioteki – użytkownicy muszą mieć dostęp do jasnych i zrozumiałych instrukcji korzystania z biblioteki.

Should Have

- Możliwość dodawania własnych datasetów do biblioteki – funkcja ta pozwala użytkownikom na personalizację i rozszerzenie bazy danych.
- Narzędzia do czyszczenia i normalizacji danych – choć nie krytyczne, znacząco podnoszą wartość użytkową biblioteki.

Could Have

- Rozbudowane funkcje tokenizacji danych – ułatwiłyby przetwarzanie tekstu, zwiększając potencjalne obszary zastosowań biblioteki.
- Wtyczki wspierające nowsze frameworki i biblioteki w ekosystemie Elixir – mogą zwiększyć atrakcyjność biblioteki dla szerokiej grupy użytkowników.

Won't Have

- Automatyczne tłumaczenia dokumentacji na różne języki – choć przydatne w przyszłości, nie będą dostępne w pierwszej wersji produktu.
- Zaawansowane algorytmy sztucznej inteligencji do analizy danych – nie są planowane w aktualnym zakresie projektu.

2.4. Wymagania jakościowe

Poniżej przedstawione zostały wymagania niefunkcjonalne, które mają na celu zapewnienie wysokiej jakości tworzonej biblioteki oraz komfortu jej użytkowania. Odpowiednie spełnienie tych wymagań wpłynie pozytywnie na łatwość integracji, rozwój i utrzymanie projektu w dłuższym okresie.

- **Czytelny i spójny interfejs API** – Interfejs biblioteki powinien być intuicyjny i dobrze zaprojektowany, umożliwiając użytkownikowi szybkie rozpoczęcie pracy bez konieczności zapoznawania się z nadmiernie rozbudowaną dokumentacją. Nazewnictwo funkcji, struktur i modułów powinno być spójne i zgodne z konwencjami języka Elixir.
- **Wydajność** – Biblioteka powinna umożliwiać efektywne operacje na dużych zbiorach danych, takich jak filtrowanie, czyszczenie czy transformacja. Szczególny nacisk powinien zostać położony na optymalizację operacji na dużych zbiorach dancyh oraz minimalizację zużycia pamięci i czasu przetwarzania.
- **Skalowalność** – Projekt powinien być skalowalny zarówno pod względem wielkości obsługiwanych danych. Powinien umożliwiać bezproblemowe dodawanie nowych źródeł danych, funkcjonalności i formatów danych bez konieczności istotnej przebudowy istniejącej architektury.
- **Bezpieczeństwo danych** – W przypadku integracji z zewnętrznymi źródłami danych, komunikacja powinna być realizowana z użyciem bezpiecznych protokołów (np. HTTPS). System powinien być odporny na wstrzykiwanie niepoprawnych danych oraz błędne formaty plików.
- **Odporność na błędy** – Biblioteka powinna zawierać mechanizmy wykrywania i obsługi błędów, umożliwiające użytkownikowi uzyskanie jasnych komunikatów w przypadku problemów z danymi, połączeniem lub działaniem funkcji.
- **Kompatybilność z ekosystemem Elixira** – Projekt musi zapewniać pełną kompatybilność z innymi bibliotekami wykorzystywanymi w uczeniu maszynowym w języku Elixir, w szczególności Nx [5], Explorer [2] i Bumblebee [8]. Powinien też działać na różnych platformach systemowych wspierających środowisko Elixira.
- **Dokumentacja** – Biblioteka powinna być opatrzona pełną dokumentacją techniczną, zawierającą opisy funkcji, struktur danych, przykłady użycia oraz wskazówki dotyczące integracji z innymi narzędziami. Dokumentacja powinna być dostępna zarówno w kodzie (np. jako modułowe `@doc`), jak i w formie zewnętrznej (np. README, przewodniki).
- **Łatwość w utrzymaniu i rozwoju** – Kod źródłowy powinien być przejrzysty, modularny, zgodny z dobrymi praktykami programistycznymi i łatwy do testowania oraz rozszerzania. Projekt powinien uwzględniać przyszłą rozbudowę, np. o nowe formaty danych, dodatkowe transformacje lub integracje.

- **Testowalność** – System powinien być w pełni testowalny. Moduły powinny być projektowane w sposób umożliwiający tworzenie testów jednostkowych oraz testów integracyjnych, co ułatwi utrzymanie wysokiej jakości kodu i szybką detekcję błędów w przyszłości.

2.5. Scenariusze użytkowania

Poniżej przedstawiono przykładowe scenariusze użytkowania systemu, które ilustrują typowe sytuacje, w jakich programista może korzystać z biblioteki. Scenariusze te odzwierciedlają główne funkcjonalności systemu i obrazują sposób interakcji użytkownika z interfejsem programistycznym (API) biblioteki.

- **Pobranie gotowego zbioru danych**

  Użytkownik chce szybko pobrać popularny zbiór danych w celu przetestowania modelu klasyfikacji tekstu. W tym celu korzysta z funkcji udostępnianych przez bibliotekę, które automatycznie pobierają dane z repozytorium, zapisują je lokalnie i przygotowują do dalszego przetwarzania.

- **Dodanie własnego zbioru danych**

  Użytkownik posiada własny zbiór danych zapisany w formacie CSV. Chce go załadować, wstępnie przefiltrować oraz znormalizować. Korzystając z biblioteki, użytkownik definiuje strukturę zbioru danych, wskazuje lokalizację pliku, a następnie stosuje dostępne funkcje do oczyszczenia danych i konwersji ich do odpowiedniego formatu.

- **Filtrowanie danych na podstawie kryteriów**

  Użytkownik analizuje zbiór danych zawierający opinie klientów i chce utworzyć podzbiór, który zawiera wyłącznie opinie pozytywne. Biblioteka umożliwia zastosowanie funkcji filtrowania na podstawie warunków logicznych, co pozwala szybko uzyskać interesujący użytkownika fragment danych.

- **Integracja danych z modelem uczenia maszynowego**

  Po przygotowaniu danych, użytkownik chce przesłać je jako tensory do modelu zaimplementowanego w bibliotece Bumblebee [8]. Biblioteka wspiera konwersję danych do struktury kompatybilnej z Nx[5] oraz umożliwia bezpośrednie przekazanie ich do dalszego przetwarzania przez model.

- **Przegląd dostępnych zbiorów danych**

  Użytkownik nie jest jeszcze zdecydowany, z jakim zbiorem chce pracować. Korzysta z funkcji przeglądania dostępnych zbiorów, które zawierają metadane, takie jak: źródło, liczba rekordów, typ danych (tekst, obraz, liczby), wymagania wstępne itp. Po zapoznaniu się z informacjami, wybiera odpowiedni zbiór i rozpoczyna pracę.

2.6. Podsumowanie

Ten rozdział dostarcza wszechstronnej analizy projektowanego systemu, obejmującej szczegółowy opis funkcji z uwzględnieniem priorytetów, opis procesów biznesowych oraz scenariuszy użytkowania, co pozwala lepiej zrozumieć i wizualizować funkcjonalności projektu. Ta

całościowa prezentacja ułatwia zarządzanie oczekiwaniami i planowanie dalszych etapów rozwoju systemu, uwzględniając ustalony zakres prac i zasoby, co jest kluczowe dla skutecznego planowania przyszłych etapów wdrożenia.

Rozdział 3

Wybrane aspekty realizacji

Niniejszy rozdział poświęcony jest praktycznym aspektom implementacji biblioteki. Celem tego rozdziału jest przedstawienie kluczowych decyzji projektowych, które miały wpływ na strukturę i funkcjonalności finalnego produktu. Rozdział ten stanowi podstawę do głębszego zrozumienia technicznego podejścia przyjętego podczas tworzenia biblioteki oraz wyjaśnia, jakie specyficzne problemy zostały rozwiązane w trakcie pracy nad projektem.

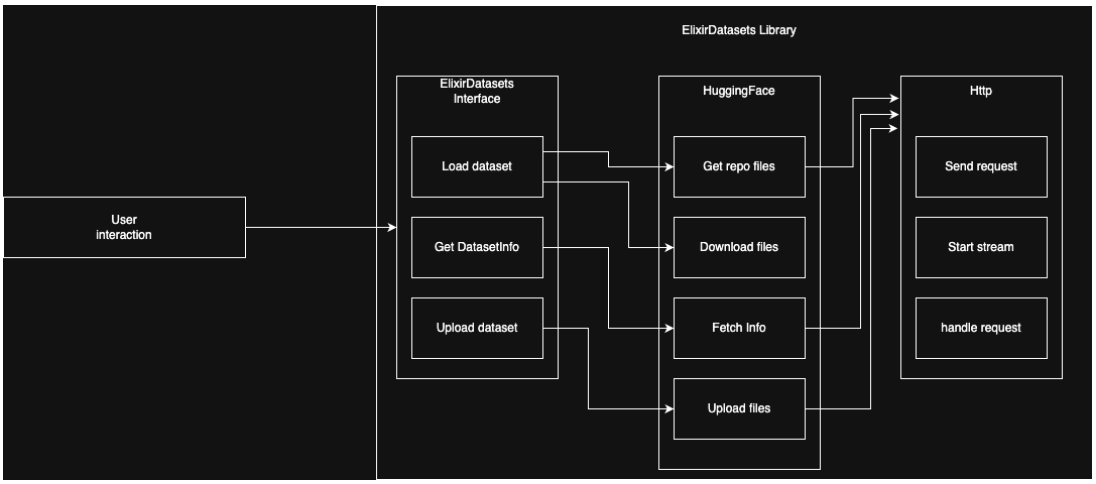
3.1. Architektura systemu

System opracowany w ramach tej pracy inżynierskiej został skonstruowany tak, aby zapewnić użytkownikowi łatwy dostęp do popularnych zestawów danych wykorzystywanych w uczeniu maszynowym oraz umożliwić dodawanie i zarządzanie własnymi zbiorami danych. Aby osiągnąć te cele, architektura systemu została podzielona na trzy główne moduły.

Moduł networkingowy jest odpowiedzialny za wszystkie operacje wymagające komunikacji z zewnętrznymi serwerami za pomocą protokołu HTTP. Jego głównymi zadaniami są: pobieranie danych z zewnętrznych źródeł, przesyłanie żądań o dostęp do zbiorów danych i odbieranie odpowiedzi.

Moduł HuggingFace odpowiada za integrację z API Hugging Face, co pozwala na wyszukiwanie, pobieranie i zarządzanie zbiorami danych dostępnymi na platformie oraz autoryzację do materiałów z ograniczonym dostępem.

Interfejs biblioteki ElixirDatasets to komponent, z którym bezpośrednio wchodzi w interakcję użytkownik końcowy. Stanowi on "fasadę"dla wszystkich operacji dostępnych w bibliotece, ukrywając za sobą złożoność modułów niższego poziomu. Każdy z tych komponentów pełni specyficzne funkcje, które razem tworzą spójny i efektywny system.



Rysunek 3.1: Schemat systemu

3.2. Stos technologiczny

Niniejszy rozdział opisuje technologie wybrane do budowy i wsparcia rozwoju naszej biblioteki w Elixirze. Przedstawione zostaną zarówno główne składniki stosu technologicznego, jak i przyczyny ich wyboru, co umożliwi zrozumienie, dlaczego stanowią one optymalne rozwiązanie.

3.2.1. Elixir i biblioteka standardowa

Język programowania Elixir został wybrany jako główny język programowania ze względu na jego skalowalność, wydajność oraz wyjątkowe wsparcie dla programowania współbieżnego. Biblioteka standardowa Eliksira dostarcza szeroką gamę modułów i funkcji, które pomagają w efektywnym budowaniu aplikacji, w tym w obsłudze sieciowej, przetwarzaniu danych i konstrukcjach współbieżnych.

3.2.2. Explorer

Współpraca z danymi wymaga używania narzędzi, które umożliwiają łatwą manipulację i transformację zbiorów danych. Biblioteka Explorer służy do łatwego zarządzania danymi w Elixirze, oferując funkcje podobne do tych z pakietu pandas w Pythonie. Umożliwia ona efektywne przetwarzanie dużych zbiorów danych.

3.2.3. Mix

Mix jest narzędziem służącym do zarządzania projektami w Elixirze, które umożliwia kompilację kodu, jego testowanie oraz zarządzanie zależnościami. Mix jest integralną częścią ekosystemu Elixir i stanowi fundament pod względem konstrukcji i administracji projektami.

3.2.4. ExDoc

Dokumentacja jest niezmiernie ważna dla utrzymania i rozwijania projektów programistycznych, zwłaszcza tych otwartych, gdzie inni programiści mogą wnosić wkład. ExDoc to narzędzie

do generowania dokumentacji w Elixirze, które pozwala na tworzenie przejrzystej i łatwo przeszukiwalnej dokumentacji dla projektów.

3.2.5. ExUnit i ExCoveralls

Zapewnienie jakości kodu poprzez testy jest fundamentem stabilnego oprogramowania. ExUnit to framework testowy dostarczany razem z Elixirem, który umożliwia pisanie czytelnych i efektywnych testów jednostkowych. ExCoveralls z kolei to narzędzie, które pozwala na mierzenie pokrycia kodu testami, co jest ważnym wskaźnikiem jakości projektu programistycznego.

3.3. Przegląd poszczególnych komponentów

W tej sekcji dokonujemy szczegółowego przeglądu kluczowych komponentów systemu, które mają zasadnicze znaczenie dla działania i efektywności naszej biblioteki. Każdy z komponentów odpowiada za specyficzne funkcje i jest istotny zarówno z punktu widzenia realizacji podstawowych zadań, jak i możliwości rozbudowy czy utrzymania całego systemu.

3.3.1. Moduł Interfejsu

Moduł interfejsu odgrywa kluczową rolę w naszej bibliotece, zapewniając interaktywny dostęp do funkcjonalności platformy HuggingFace/datasets. Ten komponent ma za zadanie odtworzenie interfejsu tej platformy, co umożliwia użytkownikom łatwe pobieranie zbiorów danych, ich udostępnianie oraz przeglądanie szczegółowych informacji o danych przed ich pobraniem.

Interaktywność modułu interfejsu znacząco ułatwia korzystanie z bogatych zasobów dostępnych na platformie Hugging Face, pozwalając na bezpośrednie i intuicyjne wykorzystanie dostępnych zbiorów w procesach analitycznych i badawczych. Dzięki temu użytkownicy mogą nie tylko efektywnie zarządzać danymi, ale również optymalizować czas potrzebny na przygotowanie i przetwarzanie informacji niezbędnych do analiz lub treningów modeli uczenia maszynowego.

Podsumowując, moduł interfejsu jest niezbędnym elementem biblioteki, umożliwiającym integrację z zewnętrznymi źródłami danych i zapewniającym użytkownikowi dostęp do funkcjonalności kluczowych dla efektywnego wykorzystania dostępnych zbiorów danych.

3.3.2. Moduł HuggingFace

Moduł HuggingFace w bibliotece ElixirDatasets pełni kluczową rolę w zapewnianiu dostępu do szerokiej gamy zbiorów danych dostępnych na platformie Hugging Face. Ten komponent modułu jest odpowiedzialny za interakcję z API Hugging Face, co ułatwia pobieranie danych, ich udostępnianie, a także podgląd informacji o zbiorach przed ich pobraniem.

Dzięki wykorzystaniu funkcji ‘file_url‘ i ‘file_listing_url‘, moduł umożliwia uzyskanie URL-i do konkretnych plików i listowania zawartości repozytorium na platformie Hugging Face. Obejmuje to zarówno publiczne zbiory danych, jak i te prywatne, dostępne tylko dla

autoryzowanych użytkowników. Proces ten jest wspomagany przez mechanizm cache'owania, gdzie każde pobrane pliki są zapisywane lokalnie wraz z ich metadanymi, co pozwala na optymalizację kolejnych zapytań.

Dodatkowo, rozszerzona funkcja ‘cached_download‘ pozwala na pobieranie plików z zastosowaniem cache bazującego na ETagach (Entity Tags). Jeśli zasób nie uległ zmianie, dane mogą być serwowane bezpośrednio z lokalnego cache, co znacznie przyspiesza dostęp i redukuje zużycie zasobów sieciowych.

Użytkownik może również korzystać z opcjonalnych ustawień, takich jak tryb offline, który zapewnia dostęp do zasobów nawet w przypadku braku połączenia z Internetem, pod warunkiem że dane zostały wcześniej pobrane i zapisane w cache'u.

3.3.3. Moduł networkingowy

Moduł networkingowy w projekcie ElixirDatasets pełni fundamentalną rolę w umożliwianiu komunikacji sieciowej, stosując się do wiodących standardów praktyk internetowych. Jest odpowiedzialny za mechaniczne aspekty pobierania danych, obsługę protokołów HTTP/HTTPS oraz zarządzanie połączeniami sieciowymi. Moduł ten, zaimplementowany w ‘ElixirDatasets.Utils.HTTP‘, służy jako centralny punkt dla wszystkich operacji sieciowych wykonanych przez bibliotekę.
Fundamentalne funkcje modułu networkingowego obejmują:

- **Pobieranie danych:** Moduł potrafi wykonywać bezpieczne żądania do zdalnych serwerów, pobierając dane bezpośrednio na lokalne dyski użytkownika. To mechanizm, który stoi za funkcją ‘download‘, pozwala na efektywne zarządzanie przepływem danych.

- **Zarządzanie nagłówkami HTTP:** Za pomocą funkcji ‘get_header‘, moduł umożliwia manipulację i odczyt nagłówków HTTP, co jest kluczowe do prawidłowego rozumienia i kontrolowania odpowiedzi serwera.

- **Opcje konfiguracyjne:** Użytkownicy mogą dostosować wiele aspektów żądań HTTP, takich jak nagłówki, ciało żądania, czy timeout, co daje elastyczność potrzebną do obsługi różnorodnych scenariuszy użytecznych.

Jedną z istotnych cech modułu jest jego zdolność do obsługi przekierowań i autoryzacji, co czyni go potężnym narzędziem do integracji z różnymi API oferującymi dane. Ponadto, moduł zapewnia mechanizmy bezpieczeństwa, takie jak weryfikacja certyfikatów SSL czy obsługa etykiet ETag, które minimalizują ryzyko przechwycenia danych i umożliwiają inteligentne zarządzanie cache'owaniem.

3.4. Ciekawsze algorytmy i mechanizmy systemu

Poniższa sekcja koncentruje się na przedstawieniu najbardziej istotnych komponentów opracowywanego systemu. Zostaną one omówione nie tylko pod kątem szczegółowego działania, ale także w kontekście ich wpływu na całość tworzonego systemu.

3.4.1. Cacheowanie danych

Cacheowanie danych jest krytycznym aspektem naszego systemu, zwiększającym jego efektywność poprzez redukcję liczby koniecznych operacji wejścia/wyjścia, zewnętrznych zapytań API oraz pobrań danych, które już wcześniej zostały załadowane. Aby zminimalizować niepotrzebne operacje, szczególnie w przypadku wielokrotnych uruchomień kodu przez użytkownika, implementujemy mechanizm etag (entity tag) oraz OId (Object Identifier). Etag i OId to unikalne identyfikatory przypisane do każdej wersji zasobu, które umożliwiają jednoznaczne stwierdzenie, czy przechowywana wersja zasobu nadal jest aktualna.

Przykładem wykorzystania etagu oraz oidu może być kontrola stanu zbiorów danych na zdalnym repozytorium. Poniżej przedstawiono przykłady odpowiedzi serwera, ilustrującej zastosowanie etagu i oidu:

```
{
  "etag": "W/\"129-cBr/GjmAu235zSmcAE8hRzjbA90\"",
  "url": "https://huggingface.co/api/datasets/fka/awesome-chatgpt-prompts"
}%
```

Listing 3.1: Podgląd odpowiedzi serwera z etagiem

```
[
    {
        "type": "file",
        "oid": "f4f3945bd7150d3e12988485c42da1f8c29c59f8",
        "size": 2265,
        "path": ".gitattributes"
    },
    {
        "type": "file",
        "oid": "d3dc1fd0061ff5ebb4c34451bd6c17d4547b6612",
        "size": 339,
        "path": "README.md"
    },{
        "type": "file",
        "oid": "6df098dd5d2ff6d9fedd3aa052e6fd49c3389b77",
        "size": 104186,
        "path": "prompts.csv"
    }
]%
```

Listing 3.2: Podgląd odpowiedzi serwera z oid

3.4.2. Pasek Postępu (ProgressBar)

Aby ułatwić użytkownikom wizualizację postępu pobierania dużych zbiorów danych, w bibliotece został zaimplementowany pasek postępu (ProgressBar). Jest to narzędzie graficzne, które dynamicznie aktualizuje się w trakcie pobierania danych, pokazując użytkownikowi w

sposób bezpośredni ile danych już zostało pobrane, a ile pozostało do końca procesu. Taka wizualizacja jest szczególnie przydatna w przypadku pobierania dużych zbiorów danych, gdzie proces może trwać znaczną ilość czasu.

Rysunek 3.2: Przykład wizualizacji paska postępu

3.4.3. Przykłady LiveBook

LiveBook to narzędzie interaktywne, zaprojektowane do tworzenia dokumentów, które mogą zawierać zarówno treść edukacyjną jak i wykonywalny kod. W ramach naszej biblioteki w języku Elixir, zintegrowaliśmy przykłady LiveBook, które umożliwiają użytkownikom eksperymentowanie z kodem na żywo, bezpośrednio w przeglądarce.

Te interaktywne dokumenty są niezwykle przydatne w edukacji i prezentacji możliwości biblioteki, ponieważ pozwalają na natychmiastowe obserwowanie wyników działania kodu. Użytkownicy mogą modyfikować lub rozwijać przykłady, co zwiększa ich zrozumienie działania biblioteki i zachęca do głębszego eksplorowania jej funkcji.

3.5. Zapewnienie jakości

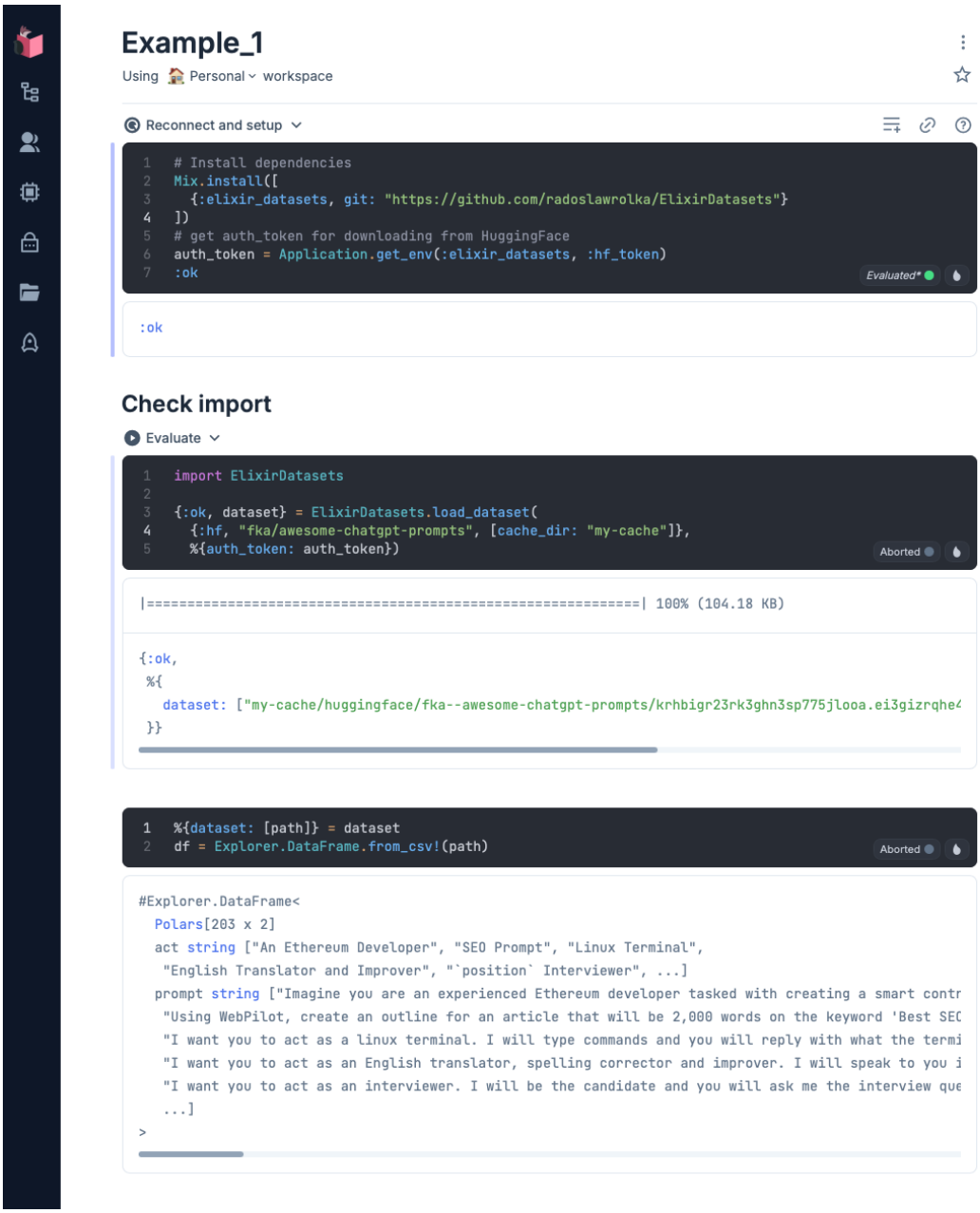
W projektowaniu oprogramowania, zapewnienie jakości odgrywa kluczową rolę w sprawdzaniu, czy produkt spełnia oczekiwania użytkowników i wewnętrzne standardy jakości przed jego wydaniem. W kontekście rozwijania biblioteki przedstawione zostaną techniki testowania automatycznego oraz utrzymywania jakości oprogramowania.

3.5.1. Testy jednostkowe

Testy jednostkowe koncentrują się na pojedynczych, izolowanych fragmentach kodu, takich jak funkcje lub metody, sprawdzając, czy zachowują się one zgodnie z oczekiwaniami na podstawie zdefiniowanych przypadków testowych. W projektowanej bibliotece w języku Elixir, testy te są realizowane z wykorzystaniem frameworka ExUnit, który oferuje wsparcie dla asercji, testowania równoległego, a także umożliwia łatwą integrację z narzędziami do pomiaru pokrycia kodu testami, takimi jak ExCoveralls, którego pokrycie testami zostało ustawione na 90%. Efektywne wykorzystanie testów jednostkowych pozwala na szybką identyfikację i izolację błędów oraz zapewnia stabilność kodu poprzez ciągłą weryfikację jego poprawności w miarę jego rozwoju.

3.5.2. Analiza statyczna kodu

W napisanej bibliotece zostały wykorzystane narzędzia analizy statycznej kodu, które przeprowadzało rygorystyczne sprawdzanie typów oraz wykrywanie nieosiągalnych fragmentów



Rysunek 3.3: Przykład wykorzystania LiveBook

kodu. Z uwagi na dynamiczną naturę Elixira, narzędzie to stanowi kluczowy element zapewniający dodatkowe bezpieczeństwo typów, które nie jest domyślnie ścisłe w tym języku. Ponadto, w procesie tworzenia biblioteki wykorzystano narzędzia zapewniające przestrzeganie ujednoliconego stylu kodowania, co znacząco przyczyniło się do poprawy czytelności kodu. Te narzędzia automatycznie egzekwowały zasady dotyczące formatowania i struktury kodu, co ułatwiło współpracę w zespole i zwiększyło efektywność w utrzymaniu oraz rozwijaniu projektu. Implementacja analizy statycznej pozwala na wcześniejsze zidentyfikowanie problemów oraz ułatwia utrzymanie wysokiej jakości kodu.

3.5.3. Ciągła integracja (CI)

Kluczowym narzędziem usprawniającym proces ciągłej integracji jest GitHub Actions. Ta platforma CI/CD pozwala na automatyczne wykonywanie różnorodnych operacji w arbitralnie skonfigurowanym środowisku wirtualnym. W naszym przypadku, każde zgłoszenie do repozytorium inicjuje serię zadań w GitHub Actions, które obejmują kompilację kodu w trybie ścisłym, uruchomienie testów, zgodność z ustanowionymi standardami formatowania, a także analizę pokrycia testami.

3.5.4. Code review

Code review, czyli przegląd kodu prez innego członka zespołu, jest niezastąpionym procesem w cyklu rozwoju oprogramowania. Proces ten polegał na ocenie kodu przez osobę niezwiązaną bezpośrednio z implementacją przy utworzeniu żądania dodania nowych funkcjonalności do oficjalnej gałęzi repozytorium. Dopiero po pozytywnym rozpatrzeniu takiego wniosku możliwe było połączenie dwóch wersji.

3.6. Podsumowanie

Biblioteka została zaprojektowana zgodnie z zasadami modularności, co klarownie definiuje rolę i odpowiedzialności poszczególnych segmentów kodu. Narzędzia oraz technologie użyte do jej rozwoju są powszechnie akceptowane i standardowe w społeczności deweloperów Elixira. Dla zapewnienia niezawodności systemu stosowane są różnorodne formy testowania, a także regularne recenzje kodu wewnątrz zespołu. Jakość kodu jest dodatkowo monitorowana przez narzędzia do statycznej analizy, co potwierdza wysokie standardy jego utrzymania.

Rozdział 4

Organizacja pracy

Ten rozdział poświęcony jest szczegółowemu przedstawieniu metodyki i technik, które zostały wykorzystane w trakcie realizacji projektu. Obejmuje on dokładny opis kolejnych etapów prac, a także sposób podziału odpowiedzialności i zadań w zespole projektowym. Ponadto, wskazuje i charakteryzuje narzędzia, które wspierały komunikację i zarządzanie organizacją pracy w trakcie procesu tworzenia.

4.1. Charakterystyka projektu

Tworzona praca inżynierska ma charakter rozwojowy. Jej celem było stworzenie wyspecjalizowanej biblioteki w języku Elixir, koncentrującej się na efektywnym pozyskiwaniu, zarządzaniu i ładowaniu zbiorów danych powszechnie stosowanych w dziedzinie uczenia maszynowego.

Projekt od początku zakładał stworzenie solidnego fundamentu do pracy z danymi, opierając się na architekturze języka Elixir. Wymagania dotyczące kluczowych obszarów funkcjonalnych, takich jak mechanizmy pobierania danych z zewnętrznych repozytoriów (*np*. Hugging Face Hub) oraz metody ich efektywnego ładowania do pamięci, były znane i stanowiły trzon planu prac. Projekt ten koncentrował się na dostarczeniu narzędzia umożliwiającego szybki start pracy analitycznej, automatyzując etap pozyskiwania i przygotowania danych źródłowych.

Realizacja projektu przebiegała zgodnie z procesem przyrostowo-iteracyjnym. Prace rozpoczęto od fazy badawczo-prototypowej, której celem było potwierdzenie wykonalności kluczowych założeń architektonicznych i technicznych w środowisku Elixir. Następnie, implementacja właściwa została podzielona na logiczne przyrosty (moduły), obejmujące kolejno funkcjonalności związane z pobieraniem i zarządzaniem danymi oraz ich ładowaniem i wstępnym przeglądaniem. Każdy moduł był rozwijany i testowany w krótkich iteracjach, co pozwoliło na sukcesywne budowanie stabilnej biblioteki i minimalizację wystąpienia błędów. Takie podejście umożliwiło optymalne wykorzystanie zasobów i skupienie się na dostarczeniu stabilnego rdzenia systemu.

4.2. Osoby w projekcie

Projekt był realizowany przez zespół składający się z dwóch członków:

- Radosław Rolka
- Weronika Wojtas

Opiekę merytoryczną oraz rolę klienta pełnił dr inż. Aleksander Smywiński-Pohl.

4.3. Zespół i podział obowiązków

Projekt realizowany był przez zespół dwuosobowy, w którym każdy z członków odpowiadał za określone obszary funkcjonalności biblioteki. Podział obowiązków był zaplanowany w taki sposób, aby maksymalnie wykorzystać umiejętności każdego członka zespołu oraz zapewnić efektywny przebieg prac.

Radosław Rolka odpowiadał za następujące obszary projektu:

- **Architektura systemu** – projektowanie ogólnej struktury biblioteki oraz definiowanie interakcji między poszczególnymi modułami.
- **Moduł networkingowy** – implementacja funkcji odpowiedzialnych za pobieranie danych z zewnętrznych źródeł poprzez protokół HTTP/HTTPS.
- **Moduł HuggingFace** – integracja z API platformy Hugging Face Hub, obsługa autoryzacji oraz zarządzanie dostępem do zbiorów danych.
- **System cacheowania** – implementacja mechanizmu cacheowania danych bazującego na etagach i OId, optymalizacja wydajności pobierania.
- **Transformacja danych** – implementacja funkcji do wczytywania i przetwarzania danych z różnych formatów (TXT, CSV, JSON, Parquet).
- **Główny interfejs biblioteki** – tworzenie publicznego API biblioteki.
- **Ciągła integracja (CI/CD)** – konfiguracja GitHub Actions, automatyzacja testów i kontrola jakości kodu.

Weronika Wojtas odpowiadała za następujące obszary projektu:

- **Architektura systemu** – projektowanie ogólnej struktury biblioteki oraz definiowanie interakcji między poszczególnymi modułami.
- **Integracja z Explorer** – zapewnienie kompatybilności biblioteki z narzędziem Explorer do pracy z dataframe'ami i seriami.
- **Przetwarzanie strumieniowe** – dodanie możliwości ładowania zbiorów strumieniowo
- **Wielowątkowe pobieranie zbiorów danych** – dodanie obsługi wielu wątków przy równoległym pobieraniu zbiorów
- **Rozszerzone funkcjonalności ładowania zbiorów danych** – dodanie dodatkowych opcji ładowania, takich jak filtrowanie zbiorów danych na podstawie nazwy, pobieranie wybranych podzbiorów danych oraz weryfikacja integralności i kompletności pobranych zbiorów
- **Testy jednostkowe** – pisanie kompleksowych testów jednostkowych dla wszystkich modułów biblioteki z użyciem frameworka ExUnit.
- **Dokumentacja** – tworzenie dokumentacji technicznej i przykładów użycia kodu w formie LiveBook.
- **Pokrycie testami** – monitorowanie i utrzymanie wysokiego poziomu pokrycia kodu testami (target: 80%) z wykorzystaniem ExCoveralls.

4.4. Organizacja prac i wykorzystane narzędzia

W tej sekcji opisane zostały metody i narzędzia wykorzystane do organizacji pracy nad tworzoną biblioteką. Zawiera ona opis przyjętej metodyki oraz kanałów komunikacji, a także listę kluczowych narzędzi wspierających proces programowania, kontroli wersji oraz zarządzania zadaniami.

4.4.1. Metodyka i Organizacja Pracy

W projekcie zastosowano metodykę iteracyjną w zakresie planowania i przeglądu postępów. Kluczowe elementy organizacji pracy obejmowały:

- **Zarządzanie Zadaniami**: Do tworzenia, przypisywania i śledzenia zadań wykorzystywano wbudowany w repozytorium system GitHub Issues.
- **Kontrola Wersji**: Do zarządzania kodem źródłowym oraz współpracy używano systemu kontroli wersji Git oraz platformy GitHub. Dzięki temu możliwe było tworzenie rozdzielnych gałęzi roboczych (branches), przeprowadzanie Code Review oraz weryfikowanie zmian przed ich włączeniem do głównej gałęzi (*main*).

Podział i przydział zadań odbywał się na regularnych spotkaniach, najczęściej zaraz po omówieniu postępów i rozwiązaniu napotkanych problemów.

4.4.2. Komunikacja i Spotkania

Komunikacja była realizowana na dwóch płaszczyznach: wewnątrz zespołu oraz z opiekunem (klientem).

Komunikacja codzienna i omawianie pilnych spraw odbywało się głównie za pośrednictwem komunikatorów Facebook Messenger oraz Discord. Kanały te służyły do szybkiego rozwiązywania bieżących problemów technicznych i organizacyjnych. Uzupełnieniem komunikacji zdalnej były regularne spotkania stacjonarne, które odbywały się co kilka tygodni. Spotkania te były kluczowe dla synchronizacji strategicznych działań, dogłębnego omówienia architektury systemu, a także wspólnego pisania i edytowania tekstu pracy inżynierskiej.

Kontakt z opiekunem odbywał się w kluczowych momentach realizacji projektu. Pierwsze spotkanie miało charakter konsultacyjny, służąc ostatecznemu sprecyzowaniu celów i zakresu funkcjonalnego biblioteki na etapie definiowania wymagań. Kolejne kontakty były wykorzystywane do cyklicznych prezentacji postępu prac. Taki rytm współpracy pozwalał na wczesne uzyskanie informacji zwrotnej i weryfikację, czy kierunek rozwoju systemu jest zgodny z pierwotnymi założeniami.

4.4.3. Narzędzia Wykorzystane w Projekcie

Do stworzenia, zarządzania i dokumentowania projektu wykorzystano szereg narzędzi, które można podzielić na kilka kategorii:

Obszar	Narzędzia	Opis
Kontrola wersji i zarządzanie	Git / GitHub	Hostowanie kodu źródłowego, kontrola wersji, tworzenie pull requestów oraz zarządzanie zadaniami poprzez GitHub Issues.
Środowisko programistyczne	IntelliJ IDEA / Visual Studio Code	Główne środowiska programistyczne (*IDE*) wykorzystywane do tworzenia kodu w języku Elixir.
Dokumentacja i testowanie	Livebook	Narzędzie do tworzenia interaktywnych notatników w Elixirze, wykorzystane do prezentacji i tworzenia przykładów zastosowania biblioteki.
Dokumentacja końcowa	Overleaf	Platforma chmurowa wykorzystana do wspólnego pisania i formatowania niniejszej pracy inżynierskiej w LaTeX-u.

Tabela 4.1: Zestawienie wykorzystanych narzędzi projektowych

Wykorzystanie natywnych funkcji platformy GitHub (Issues i Git) do zarządzania zadaniami oraz kontroli wersji okazało się wystarczające dla projektu o ograniczonym, jednoosobowym/dwuosobowym zespole, eliminując konieczność stosowania bardziej złożonych narzędzi typu Jira czy Trello.

4.5. Zastosowane techniki i praktyki

W trakcie realizacji projektu zespół stosował szereg technik i praktyk mających na celu zapewnienie wysokiej jakości kodu, efektywności pracy oraz łatwości utrzymania projektu w przyszłości. Poniżej opisano najważniejsze z nich.

4.5.1. Ciągła integracja (CI/CD) z GitHub Actions

Projekt wykorzystuje platformę GitHub Actions do automatyzacji procesów ciągłej integracji. Każde push'owanie do głównej gałęzi repozytorium oraz każde pull request'owanie inicjuje serię testów i kontroli jakości w arbitralnie skonfigurowanym środowisku wirtualnym. Główne zadania wykonywane w ramach CI/CD to:

- **Kompilacja i testy** – Kod jest kompilowany, a następnie uruchamiane są wszystkie testy jednostkowe.
- **Kontrola formatowania** – Automatycznie sprawdzane jest, czy kod spełnia standardy formatowania.
- **Sprawdzanie ostrzeżeń** – Kompilacja w trybie surowym, co oznacza, że wszelkie ostrzeżenia są traktowane jako błędy i uniemożliwiają pomyślne ukończenie pipeline'u.

4.5.2. Moduledoc - Dokumentacja i przykłady w kodzie

Każdy moduł publiczny oraz funkcja publiczna jest opatrzona dokumentacją w kodzie oraz przykładami użycia. Podane opisy są automatycznie przetwarzane do postaci ustandaryzowanej

dokumentacji w ekosystemie, a następnie jest weryfikowane ich działanie oraz zgodność z implementacją, co zapobiega powstawaniu nieaktualnych lub błędnych informacji w dokumentacji. Przykładowa dokumentacja funkcji zawiera następujące elementy:

- Opis przeznaczenia i zastosowania
- Podpisy funkcji ze specyfikacją typów
- Przykłady użycia
- Informacje o błędach, które funkcja może zwrócić

4.5.3. Refaktoryzacja kodu

Regularna refaktoryzacja kodu była integralną częścią procesu rozwoju biblioteki. Zespół dążył do utrzymania kodu w czystej i zrozumiałej formie, co ułatwiało jego dalszy rozwój i utrzymanie. Refaktoryzacja obejmowała między innymi:

- Uproszczenie złożonych funkcji i modułów
- Usuwanie redundantnego kodu
- Poprawę czytelności poprzez lepsze nazewnictwo i strukturę kodu

4.5.4. Pair programming

W niektórych kluczowych momentach projektu, zespół stosował technikę pair programming, gdzie dwóch programistów wspólnie pracowało nad jednym zadaniem. Ta praktyka pozwalała na szybsze rozwiązywanie problemów, wymianę wiedzy oraz poprawę jakości kodu poprzez bieżącą weryfikację i dyskusję nad implementowanymi rozwiązaniami. Dzięki temu zespół mógł efektywniej radzić sobie z trudnymi wyzwaniami technicznymi oraz zwiększyć spójność kodu.

4.5.5. Test Driven Development (TDD)

Zespół stosował podejście Test Driven Development (TDD), które polega na pisaniu testów jednostkowych przed implementacją funkcjonalności. Ta praktyka pozwalała na lepsze zrozumienie wymagań oraz zapewniała, że każda nowa funkcjonalność była odpowiednio przetestowana od samego początku. TDD pomagało również w utrzymaniu wysokiej jakości kodu oraz ułatwiało refaktoryzację, ponieważ istniała pewność, że zmiany nie wprowadzą regresji. Przebieg TDD obejmował następujące kroki:

- **Red Phase**: Napisanie testu jednostkowego, który definiuje oczekiwaną funkcjonalność
- **Green Phase**: Uruchomienie testu, który początkowo powinien zakończyć się niepowodzeniem, a następnie implementacja minimalnej ilości kodu potrzebnej do przejścia testu
- **Refactor Phase**: Refaktoryzacja kodu w celu poprawy jego struktury i czytelności, przy jednoczesnym zachowaniu przejścia wszystkich testów

4.6. Przebieg prac

Realizacja projektu została podzielona na trzy główne, sekwencyjne fazy, które odzwierciedlają przyjętą metodykę przyrostowo-iteracyjną oraz specyfikę tworzenia biblioteki programistycznej.

4.6.1. Faza Koncepcyjna i Planowania Architektury

Faza ta obejmowała wstępny okres projektu, koncentrując się na badaniach, analizie i planowaniu strategicznym:

- Definicja Zakresu i Celu: Ostateczne sprecyzowanie celu pracy jako stworzenia biblioteki do pobierania, zarządzania i ładowania zbiorów danych dla uczenia maszynowego w języku Elixir.
- Wybór Stosu Technologicznego: Zatwierdzenie języka Elixir jako głównego narzędzia oraz wybór kluczowych bibliotek i technik niezbędnych do efektywnej obsługi operacji I/O, zarządzania pamięcią, oraz przetwarzania zbiorów danych.
- Opracowanie modułowej struktury biblioteki. Zdecydowano o podziale na odrębne moduły odpowiedzialne za integrację z API zewnętrznymi (pobieranie), oraz parsowanie i ładowanie danych do wewnętrznego formatu.

4.6.2. Faza Implementacji

Faza implementacyjna była realizowana iteracyjnie, skupiając się na budowie i testowaniu dwóch głównych przyrostów funkcjonalnych:

- **Implementacja modułu pobierania i zarządzania** - prace rozpoczęto od budowy fundamentu biblioteki, koncentrując się na interakcji ze światem zewnętrznym. Pierwszy etap polegał na integracji ze źródłami danych. Opracowano mechanizmy komunikacji z zewnętrznymi repozytoriami. Równolegle wdrożono funkcje zarządzania plikami, tworząc mechanizmy do obsługi pobranych zbiorów danych na dysku, w tym weryfikacji integralności plików oraz podstawowego zarządzania ścieżkami dostępu.
- **Implementacja modułu ładowania i parsowania** - po zapewnieniu możliwości dostępu do danych, skupiono się na ich efektywnym wczytywaniu. Opracowano parsery danych, wdrażając wydajne funkcje do parsowania popularnych formatów. Fazę tę zakończyło stworzenie przykładów użycia i funkcjonalności w środowisku Livebook oraz na serwerze Elixira, które miały posłużyć do weryfikacji i demonstracji kluczowych funkcji biblioteki.

4.6.3. Faza Testowania, Optymalizacji i Dokumentacji

Ostatnia faza koncentrowała się na stabilizacji, weryfikacji i finalizacji produktu:

- **Testowanie funkcjonalne**: przeprowadzono testy jednostkowe kluczowych modułów (pobieranie, parsowanie) oraz testy integracyjne, sprawdzające spójne działanie całej biblioteki.

- **Optymalizacja wydajności**: skupiono się na identyfikacji i usunięciu potencjalnych wąskich gardeł w operacjach I/O, dążąc do jak najwydajniejszego ładowania dużych zbiorów danych.
- **Dokumentacja**: całość prac została podsumowana w niniejszej pracy inżynierskiej, a dodatkowo utworzono szczegółową dokumentację techniczną biblioteki, opisującą interfejsy modułów i sposób instalacji.

4.7. Podsumowanie

Prace nad projektem były starannie zaplanowane i zorganizowane, co pozwoliło na efektywne wykorzystanie zasobów zespołu oraz osiągnięcie założonych celów w określonym czasie. Wykorzystanie narzędzi organizujących i synchronizujących pracę zespołu było kluczowe dla sukcesu projektu, umożliwiając płynną komunikację i skuteczne zarządzanie zadaniami. Przyjęte techniki i praktyki programistyczne zapewniły wysoką jakość kodu oraz stabilność finalnego produktu.

Rozdział 5

Wyniki projektu

Rozdział ten podsumowuje prace wykonane w ramach realizacji systemu oraz prezentuje osiągnięte rezultaty. Ponadto zawiera ocenę uzyskanych wyników dokonaną przez członków zespołu, a także wskazuje możliwe kierunki dalszego rozwoju opracowanego rozwiązania.

5.1. Zrealizowane funkcjonalnosci

Zgodnie z przyjętą metodyką realizacji projektu, w pierwszej kolejności zaimplementowano funkcjonalności o najwyższym priorytecie, stanowiące podstawę działania tworzonej biblioteki. Obejmują one możliwość ładowania zbiorów danych zarówno z repozytorium Hugging Face Hub, jak i z lokalnego systemu plików, z automatycznym wykrywaniem obsługiwanych formatów danych oraz wsparciem dla różnych konfiguracji i podziałów zbiorów danych.

Istotnym elementem systemu jest mechanizm cache'owania, który umożliwia lokalne przechowywanie pobranych danych i ogranicza konieczność ich ponownego pobierania. Zaimplementowano różne tryby pracy z cache oraz możliwość konfiguracji katalogu przechowywania danych, a uzupełnieniem tej funkcjonalności jest tryb pracy offline.

Biblioteka oferuje również wsparcie dla strumieniowego przetwarzania danych, umożliwiającego efektywną pracę z dużymi zbiorami bez konieczności ładowania ich w całości do pamięci operacyjnej. W celu zwiększenia wydajności zaimplementowano mechanizmy przetwarzania równoległego, pozwalające na szybsze ładowanie zbiorów danych składających się z wielu plików.

System zapewnia integrację z interfejsem programistycznym Hugging Face Hub, umożliwiając pobieranie informacji i metadanych dotyczących zbiorów danych, a także obsługę autoryzacji, w tym dostęp do prywatnych zasobów. Rozszerzeniem funkcjonalności biblioteki jest możliwość wysyłania zbiorów danych do repozytorium Hugging Face, co pozwala na ich publikowanie i zarządzanie bezpośrednio z poziomu języka Elixir.

Dane mogą być automatycznie konwertowane do struktury DataFrame z wykorzystaniem biblioteki Explorer. Całość została uzupełniona o mechanizmy walidacji parametrów, obsługę błędów oraz zestaw testów i dokumentację API. Szczegółowe scenariusze użycia biblioteki zostały opisane w kolejnych rozdziałach pracy.

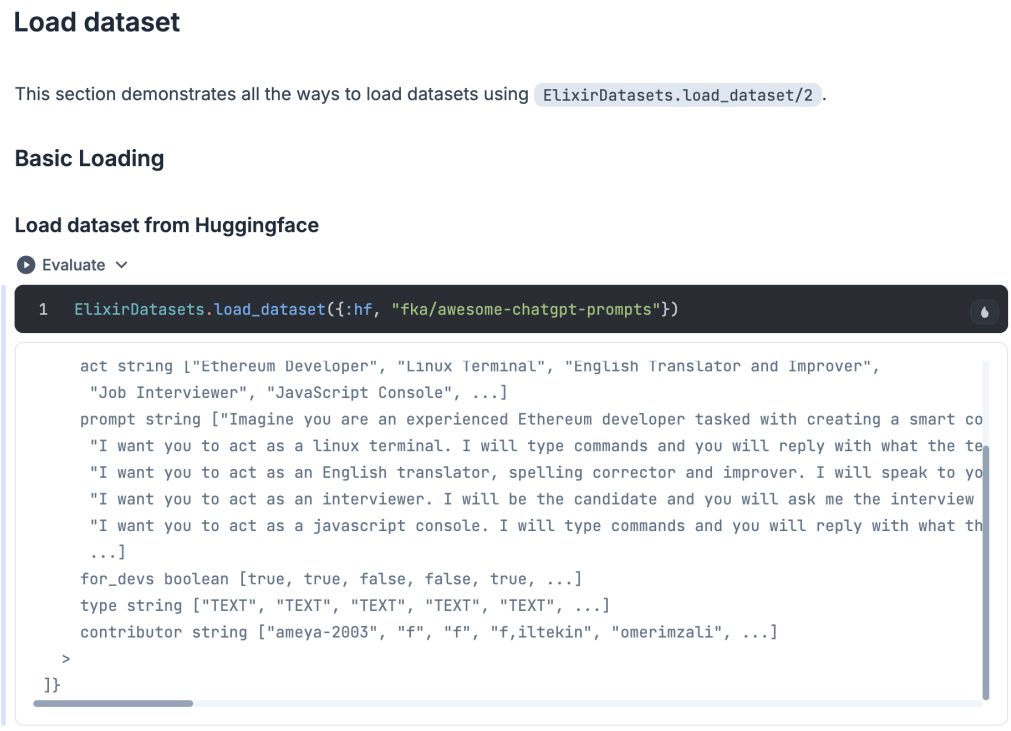
5.2. Główne scenariusze uzytkowania

Poniższa sekcja przedstawia najważniejsze scenariusze użycia biblioteki ElixirDatasets z perspektywy programisty korzystającego z niej w środowisku języka Elixir. Opisane scenariu-

sze ilustrują typowe sposoby pracy ze zbiorami danych pochodzącymi z repozytorium Hugging Face oraz z lokalnego systemu plików, a także prezentują mechanizmy optymalizacji wydajności, takie jak cache'owanie, przetwarzanie strumieniowe oraz równoległe ładowanie danych. Dla wybranych scenariuszy dołączono fragmenty kodu demonstrujące sposób wykorzystania interfejsu API biblioteki.

5.2.1. Ładowanie zbioru danych z repozytorium Hugging Face

Podstawowym scenariuszem użycia biblioteki jest pobranie zbioru danych bezpośrednio z repozytorium Hugging Face Hub. Użytkownik inicjuje ten proces poprzez wywołanie funkcji load dataset, przekazując jako parametr nazwę zbioru danych. Biblioteka automatycznie pobiera wymagane pliki, wykrywa obsługiwany format danych oraz zapisuje je w lokalnym cache. Użytkownik może dodatkowo określić konfigurację zbioru danych oraz wybrany podział, na przykład zbiór treningowy lub testowy. Po zakończeniu operacji dane zwracane są w postaci struktury DataFrame, gotowej do dalszego przetwarzania.



Rysunek 5.1: Przykład z livebook - ładowanie z Hugging Face

Load dataset from Huggingface from given subdir

```
ElixirDatasets.load_dataset(
  {:hf, "stanfordnlp/imdb", subdir: "plain_text"})
```

```
    label s64 [0, 0, 0, 0, 0, ...]
  >,
  #Explorer.DataFrame<
    Polars[50000 x 2]
    text string ["This is just a precious little diamond. The play, the script are excellent. I cant
     "When I say this is my favourite film of all time, that comment is not to be taken lightly. I pr
     "I saw this movie because I am a huge fan of the TV series of the same name starring Roy Dupuis
     "Being that the only foreign films I usually like star a Japanese person in a rubber suit who cr
     "After seeing Point of No Return (a great movie) and being told that the original was better, I
     ...]
    label s64 [-1, -1, -1, -1, -1, ...]
  >
]}
```

Rysunek 5.2: Przykład z livebook - ładowanie z podanego folderu

Load dataset from Huggingface with auth token as option

```
ElixirDatasets.load_dataset!(
  {:hf, "cornell-movie-review-data/rotten_tomatoes"},
  %{auth_token: auth_token})
```

```
    ...]
  label s64 [1, 1, 1, 1, 1, ...]
>,
#Explorer.DataFrame<
  Polars[1066 x 2]
  text string ["compassionately explores the seemingly irreconcilable situation between conservative
   "the soundtrack alone is worth the price of admission .",
   "rodriguez does a splendid job of racial profiling hollywood style--casting excellent latin actor
   "beneath the film's obvious determination to shock at any cost lies considerable skill and determ
   "bielinsky is a filmmaker of impressive talent .", ...]
  label s64 [1, 1, 1, 1, 1, ...]
>
]
```

Rysunek 5.3: Przykład z livebook - ładowanie z tokenem autoryzacyjnym

5.2.2. Ładowanie zbiorów danych z lokalnego systemu plików

Możliwa jest też praca z danymi zapisanymi lokalnie. W tym przypadku użytkownik wskazuje ścieżkę do katalogu zawierającego pliki danych, a biblioteka automatycznie rozpoznaje ich format oraz filtruje je zgodnie z wybraną konfiguracją i podziałem. Rozwiązanie to umożliwia jednolity sposób pracy z danymi niezależnie od ich źródła pochodzenia.

Rysunek 5.4: Przykład z livebook

5.2.3. Wykorzystanie mechanizmu cache'owania i trybu offline

W celu zwiększenia wydajności oraz ograniczenia liczby operacji sieciowych biblioteka automatycznie wykorzystuje mechanizm cache'owania pobranych zbiorów danych. Użytkownik może skonfigurować katalog cache za pomocą zmiennej środowiskowej oraz wybrać tryb pobierania danych, w tym ponowne użycie istniejącego cache lub wymuszenie ponownego pobrania plików. Dodatkowo możliwe jest uruchomienie trybu offline, w którym biblioteka korzysta wyłącznie z lokalnie zapisanych danych, co pozwala na pracę bez aktywnego połączenia z Internetem.

Offline mode

Work with cached datasets without network access:

```
case ElixirDatasets.load_dataset(
  {:hf, "cornell-movie-review-data/rotten_tomatoes"},
  split: "train",
  offline: true
) do
  {:ok, [offline_data]} ->
    IO.puts("✓ Loaded from cache: #{Explorer.DataFrame.n_rows(offline_data)} rows")

  {:error, reason} ->
    IO.puts("✗ Not in cache: #{reason}")
end
```

```
✓ Loaded from cache: 8530 rows
```

```
:ok
```

Rysunek 5.5: Przykład z livebook - offline mode

Using custom cache directory

Control where downloaded files are stored:

```
custom_cache = "/tmp/my_datasets_cache"

{:ok, [cached_data]} = ElixirDatasets.load_dataset(
  {:hf, "cornell-movie-review-data/rotten_tomatoes"},
  split: "train",
  cache_dir: custom_cache
)

IO.puts("Dataset cached in: #{custom_cache}")
IO.puts("Loaded #{Explorer.DataFrame.n_rows(cached_data)} rows")
```

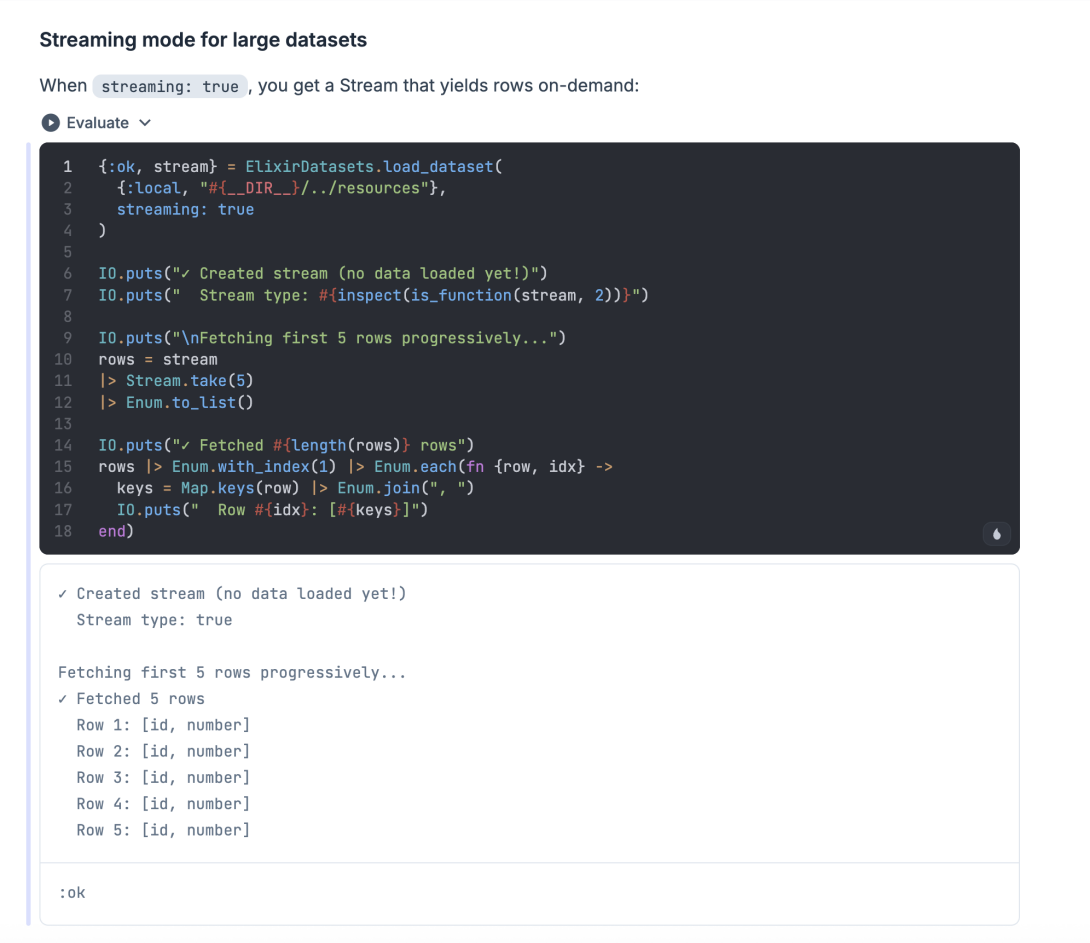
```
Dataset cached in: /tmp/my_datasets_cache
Loaded 8530 rows
```

```
:ok
```

Rysunek 5.6: Przykład z livebook - używanie podanego folderu jako cache

5.2.4. Strumieniowe przetwarzanie dużych zbiorów danych

W przypadku pracy z bardzo dużymi zbiorami danych użytkownik może skorzystać z trybu strumieniowego. Po włączeniu tej opcji biblioteka zwraca strumień danych zamiast pełnego zbioru w pamięci operacyjnej. Dane są pobierane i przetwarzane progresywnie w konfigurowalnych paczkach, co umożliwia efektywną pracę z danymi przekraczającymi rozmiar dostępnej pamięci RAM. Strumień może być dalej przetwarzany przy użyciu standardowych mechanizmów języka Elixir, takich jak moduły Stream i Enum.



Rysunek 5.7: Przykład z livebook - wykorzystanie streamingu

5.2.5. Równoległe ładowanie i przetwarzanie danych

Biblioteka umożliwia również wykorzystanie przetwarzania równoległego podczas ładowania zbiorów danych składających się z wielu plików. Użytkownik może określić liczbę procesów wykorzystywanych do tego celu, co pozwala na znaczące skrócenie czasu ładowania danych na systemach wielordzeniowych. Mechanizm ten jest w pełni transparentny dla użytkownika i zachowuje kolejność przetwarzanych danych.

Parallel processing with num_proc

Use `num_proc` to load multiple files in parallel:

Reevaluate

```elixir
IO.puts("Loading with num_proc: 1 (sequential)...")
{time_seq, {:ok, datasets_seq}} = :timer.tc(fn ->
  ElixirDatasets.load_dataset(
    {:hf, "aaaaa32r/elixirDatasets"},
    num_proc: 1
  )
end)

IO.puts("Loading with num_proc: 4 (parallel)...")
{time_par, {:ok, datasets_par}} = :timer.tc(fn ->
  ElixirDatasets.load_dataset(
    {:hf, "aaaaa32r/elixirDatasets"},
    num_proc: 4
  )
end)

time_seq_sec = time_seq / 1_000_000
time_par_sec = time_par / 1_000_000
speedup = time_seq / time_par

IO.puts("  Performance Comparison:")
IO.puts("  Sequential (num_proc: 1): #{Float.round(time_seq_sec, 3)}s")
IO.puts("  Parallel (num_proc: 4):   #{Float.round(time_par_sec, 3)}s")
IO.puts("  Speedup:                  #{Float.round(speedup, 2)}x")
IO.puts("  Datasets loaded:          #{length(datasets_par)}")
```

Evaluated

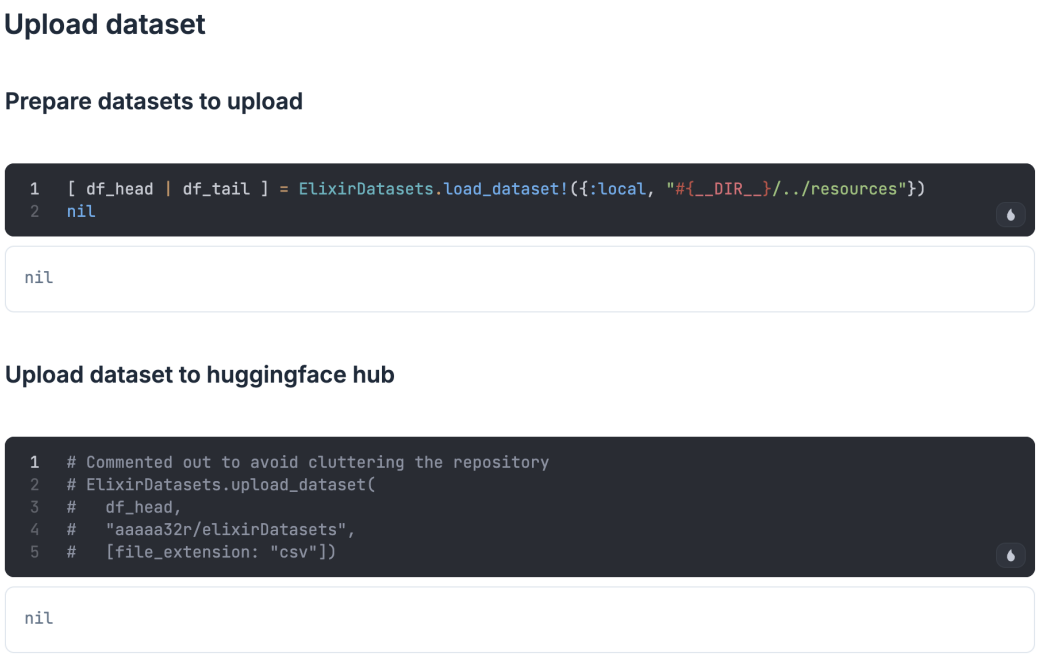
```
Loading with num_proc: 1 (sequential)...
Loading with num_proc: 4 (parallel)...
  Performance Comparison:
  Sequential (num_proc: 1): 0.282s
  Parallel (num_proc: 4):   0.284s
  Speedup:                  0.99x
  Datasets loaded:          4
```

Rysunek 5.8: Przykład z livebook - ładowanie wielowątkowe

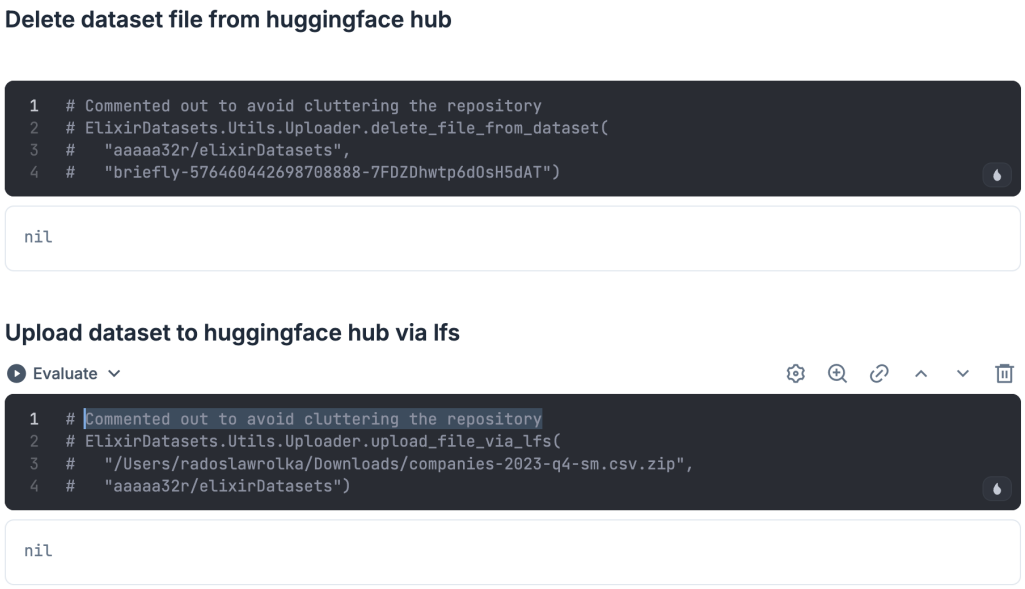
5.2.6. Publikowanie zbiorów danych w repozytorium Hugging Face

Biblioteka ElixirDatasets umożliwia publikowanie zbiorów danych w repozytorium Hugging Face bezpośrednio z poziomu języka Elixir. Użytkownik przygotowuje dane w postaci struktury DataFrame, a następnie inicjuje proces ich wysyłania do wskazanego repozytorium. Biblioteka automatycznie obsługuje zapis danych w wybranym formacie oraz komunikację z interfejsem API Hugging Face, w tym proces autoryzacji i tworzenie commitów.

Dla większych zbiorów danych dostępne jest wsparcie dla przesyłania plików z wykorzystaniem technologii Git LFS, co umożliwia wydajny upload danych przekraczających limity standardowego API. Dostępna jest również możliwość zarządzania zawartością repozytorium, w tym usuwania nieaktualnych plików. Funkcjonalność ta pozwala traktować Hugging Face Hub nie tylko jako źródło danych, ale także jako platformę do ich publikowania i współdzielenia.



Rysunek 5.9: Przykład z livebook - publikowanie zbioru danych



Rysunek 5.10: Przykład z livebook - publikowanie zbioru danych

5.2.7. Pobieranie informacji i metadanych o zbiorach danych

Biblioteka zapewnia dostęp do szczegółowych informacji i metadanych dotyczących zbiorów danych przechowywanych w repozytorium Hugging Face. Użytkownik może pobrać informacje o dostępnych konfiguracjach, podziałach zbiorów, liczbie przykładów oraz podstawowe dane opisowe.

Pozyskane metadane mogą być wykorzystywane do analizy struktury zbioru danych przed jego załadowaniem lub do automatycznego doboru konfiguracji i podziałów w dalszych etapach przetwarzania. Informacje te mogą być również zapisywane lokalnie i ponownie wykorzystywane bez konieczności każdorazowej komunikacji z interfejsem API, co zwiększa przejrzystość i efektywność pracy z danymi.

Get dataset infos

```
1  ElixirDatasets.get_dataset_infos("cornell-movie-review-data/rotten_tomatoes")
```

```
features: [
  %{"dtype" => "string", "name" => "text"},
  %{
    "dtype" => %{"class_label" => %{"names" => %{"0" => "neg", "1" => "pos"}}},
    "name" => "label"
  }
],
splits: [
  %{"name" => "train", "num_bytes" => 1074810, "num_examples" => 8530},
  %{"name" => "validation", "num_bytes" => 134679, "num_examples" => 1066},
  %{"name" => "test", "num_bytes" => 135972, "num_examples" => 1066}
],
description: nil
```

Get dataset split names

```
1  ElixirDatasets.get_dataset_split_names("cornell-movie-review-data/rotten_tomatoes")
```

```
{:ok, ["train", "validation", "test"]}
```

Get dataset config names

```
1  ElixirDatasets.get_dataset_config_names("aaaaa32r/elixirDatasets")
```

Rysunek 5.11: Przykład z livebook - używanie podanego folderu jako cache

5.3. Dalszy rozwój projektu

Potencjalny dalszy rozwój biblioteki ElixirDatasets może obejmować rozszerzenia zwiększające jej funkcjonalność oraz użyteczność w zastosowaniach związanych z analizą danych i uczeniem maszynowym. Jednym z istotnych kierunków rozwoju jest wprowadzenie mechanizmów wstępnego przetwarzania danych, takich jak normalizacja i standaryzacja, obejmujących m.in. ujednolicanie danych tekstowych oraz skalowanie danych numerycznych. Naturalnym uzupełnieniem tych funkcjonalności byłaby implementacja tokenizacji danych tekstowych, stanowiącej podstawowy etap przygotowania danych w zadaniach przetwarzania języka naturalnego.

Dalsze prace mogłyby również dotyczyć rozszerzenia wsparcia dla danych multimodalnych, takich jak obrazy czy nagrania dźwiękowe, a także pogłębionej integracji z bibliotekami uczenia maszynowego dostępnymi w ekosystemie języka Elixir. W obszarze wydajności możliwe jest dalsze rozwijanie mechanizmów przetwarzania strumieniowego i równoległego oraz rozbudowa systemu cache'owania, w tym automatyczne zarządzanie przechowywanymi danymi. Realizacja tych kierunków rozwoju mogłaby znacząco zwiększyć skalowalność biblioteki oraz jej przydatność w środowiskach produkcyjnych i badawczych.

- **Rozszerzenie obsługi formatów danych** – dodanie wsparcia dla kolejnych popularnych formatów, takich jak Avro, ORC czy formaty binarne wykorzystywane w zadaniach uczenia głębokiego, co zwiększyłoby uniwersalność biblioteki.
- **Zaawansowana walidacja i analiza jakości danych** – implementacja mechanizmów sprawdzania spójności schematów, wykrywania brakujących lub niepoprawnych wartości oraz podstawowej analizy statystycznej zbiorów danych przed ich dalszym przetwarzaniem.
- **Integracja z pipeline'ami uczenia maszynowego** – umożliwienie bezpośredniego łączenia procesów ładowania i przetwarzania danych z etapami trenowania modeli, np. poprzez integrację z bibliotekami opartymi na Nx i Axon.
- **Interfejs wiersza poleceń (CLI)** – stworzenie narzędzia CLI pozwalającego na pobieranie, publikowanie i zarządzanie zbiorami danych bez konieczności pisania kodu, co ułatwiłoby wykorzystanie biblioteki w procesach automatyzacji.
- **Rozbudowa mechanizmów zarządzania cache** – wprowadzenie limitów rozmiaru cache, automatycznego usuwania nieużywanych danych oraz możliwości współdzielenia cache pomiędzy różnymi projektami.
- **Obsługa wersjonowania zbiorów danych** – umożliwienie łatwego przełączania się pomiędzy różnymi wersjami zbiorów danych oraz śledzenia zmian, co jest szczególnie istotne w projektach badawczych i produkcyjnych.
- **Rozszerzenie dokumentacji i przykładów użycia** – dodanie bardziej rozbudowanych przykładów, scenariuszy zastosowań oraz integracji z popularnymi narzędziami do analizy danych.

5.4. Subiektywna ocena projektu

większość kluczowych funkcjonalności, dzięki czemu powstała biblioteka umożliwiająca sprawną i wydajną obsługę zbiorów danych z repozytorium Hugging Face w języku Elixir. Zaim-

plementowane rozwiązania spełniają założone wymagania funkcjonalne i pozwalają na wygodną pracę zarówno z małymi, jak i dużymi zbiorami danych.

Przeprowadzone testy jednostkowe i integracyjne przyczyniły się do zapewnienia stabilności oraz wysokiej jakości implementacji. Zastosowane technologie i narzędzia programistyczne okazały się trafnym wyborem i dobrze sprawdziły się w kontekście realizowanego projektu.

Podsumowując, projekt spełnił zakładane cele i stanowi solidną podstawę do dalszego rozwoju biblioteki oraz jej wykorzystania w kolejnych projektach związanych z analizą danych i uczeniem maszynowym w języku Elixir.

5.5. Podsumowanie

Niniejsza praca inżynierska przedstawia projekt i implementację biblioteki ElixirDatasets, której celem jest umożliwienie wygodnej i wydajnej pracy ze zbiorami danych pochodzącymi z repozytorium Hugging Face w języku Elixir. W ramach pracy zaprojektowano i zaimplementowano kluczowe mechanizmy ładowania danych, cache'owania, przetwarzania strumieniowego oraz równoległego, a także integrację z interfejsem API Hugging Face.

Zrealizowane rozwiązanie spełnia założone wymagania funkcjonalne i pozwala na efektywną obsługę zarówno małych, jak i dużych zbiorów danych. Przeprowadzone testy potwierdziły poprawność i stabilność działania biblioteki. Uzyskane rezultaty stanowią solidną podstawę do dalszego rozwoju projektu oraz jego wykorzystania w kolejnych pracach związanych z analizą danych i uczeniem maszynowym w ekosystemie języka Elixir.

Bibliografia

[1] H. Face. *Hugging Face Hub*. 2025. URL: https://huggingface.co/.

[2] C. Grainger i J. Valim. *Explorer: Fast and Elegant Data Exploration in Elixir*. 2025. URL: https://github.com/elixir-explorer/explorer.

[3] Kaggle. *Kaggle: Your Machine Learning and Data Science Community*. 2025. URL: https://www.kaggle.com.

[4] Q. Lhoest i in. „Datasets: A Community Library for Natural Language Processing". W: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online i Punta Cana, Dominican Republic: Association for Computational Linguistics, list. 2021, s. 175–184. arXiv: 2109.02846 [cs.CL]. URL: https://aclanthology.org/2021.emnlp-demo.21.

[5] S. Moriarity, J. Valim i P. O. L. Valente. *Nx - Numerical Elixir*. 2022. URL: https://github.com/elixir-nx/nx.

[6] P. Podgórni. *Metoda MoSCoW – co to jest?* 2024. URL: https://www.itvision.pl/experts/model-moscow-czym-jest-jak-korzystac/.

[7] T. pandas development team. *pandas-dev/pandas: Pandas*. Wer. latest. Lut. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[8] J. Valim i contributors. *Bumblebee: Pre-trained Neural Network Models in Elixir*. 2025. URL: https://github.com/elixir-nx/bumblebee.

[9] T. Wolf i in. „Transformers: State-of-the-Art Natural Language Processing". W: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, paź. 2020, s. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Spis rysunków

1.1 Zrzut strony HuggingFace Datasets 6

3.1 Schemat systemu 16
3.2 Zas 20
3.3 Przykład wykorzystania LiveBook 21

5.1 Przykład z livebook 31
5.2 Przykład z livebook 32
5.3 Przykład z livebook 32
5.4 Przykład z livebook 33
5.5 Przykład z livebook 34
5.6 Przykład z livebook 34
5.7 Przykład z livebook 35
5.8 Przykład z livebook 36
5.9 Przykład z livebook 37
5.10 Przykład z livebook 38
5.11 Przykład z livebook 39

Spis tabel

4.1 Zestawienie wykorzystanych narzędzi projektowych 26

Spis listingów

3.1 Podgląd odpowiedzi serwera z etagiem 19
3.2 Podgląd odpowiedzi serwera z oid 19