

OKIEM EKSPERTA

Transformery w przetwarzaniu języka naturalnego i widzenia komputerowego

Generatywna AI oraz modele LLM z wykorzystaniem Hugging Face, ChatGPT, GPT-4V i DALL-E 3

Wydanie III

Denis Rothman



Helion 

<packt>

Tytuł oryginału: Transformers for Natural Language Processing and Computer Vision:
Explore Generative AI and Large Language Models with Hugging Face,
ChatGPT, GPT-4V, and DALL-E 3, 3rd Edition

Tłumaczenie: Radosław Meryk

ISBN: 978-83-289-2050-7

Copyright © Packt Publishing 2024. First published in the English language
under the title 'Transformers for Natural Language Processing and Computer Vision -
Third Edition - (9781805128724)'

Polish edition copyright © 2025 by Helion S.A.

All rights reserved. No part of this book may be reproduced or transmitted in any
form or by any means, electronic or mechanical, including photocopying, recording
or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu
niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii
metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym,
magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi
bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje
były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich
wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych
lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności
za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

helion.pl/user/opinie/trawpr

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 230 98 63

e-mail: helion@helion.pl

WWW: helion.pl (księgarnia internetowa, katalog książek)

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- **Lubię to! » Nasza społeczność**

Spis treści |

O autorze	17
O korektorze merytorycznym	18
Przedmowa	19
ROZDZIAŁ 1.	
Czym są transformery?	27
Stała złożoność czasowa $O(1)$, która na zawsze zmieniła nasze życie	29
Uwaga $O(1)$ pokonuje rekurencyjne metody $O(n)$	30
Magia obliczeniowej złożoności czasowej warstwy uwagi	32
Krótka podróż od rekurencji do uwagi	41
Od jednego tokena do rewolucji w dziedzinie sztucznej inteligencji	44
Od jednego tokena do całości	47
Modele podstawowe	49
Od zadań ogólnych do zadań specjalistycznych	50
Rola specjalistów AI	55
Przyszłość specjalistów AI	56
Jakich zasobów powinniśmy używać?	57
Wytyczne dotyczące podejmowania decyzji	58
Rozwój łatwych do integracji interfejsów API i asystentów	59
Wybieranie gotowych do użycia bibliotek opartych na API	61
Wybór platformy chmurowej i modelu transformera	62
Podsumowanie	63
Pytania	64
Odnosiniki	64
Lektura uzupełniająca	65

ROZDZIAŁ 2.**Wprowadzenie do architektury modelu transformera 66**

Powstanie transformera — uwaga to wszystko, czego potrzebujesz	67
Stos kodera	70
Stos dekodera	93
Szkolenie i wydajność	96
Transformery Hugging Face	97
Podsumowanie	98
Pytania	99
Odnośniki	100
Lektura uzupełniająca	100

ROZDZIAŁ 3.**Emergencja a zadania końcowe — niewidoczne****głębiny transformerów 101**

Zmiana paradygmatu: czym jest zadanie NLP?	102
Wewnątrz głowicy podwarstwy uwagi transformera	103
Analiza emergencji z użyciem ChatGPT	107
Badanie potencjału modelu w zakresie wykonywania	
zadań końcowych	110
Ocena modeli za pomocą wskaźników	111
Ocena dokonywana przez człowieka	112
Uruchamianie zadań końcowych	124
CoLA	124
SST-2	125
MRPC	126
WSC	127
Podsumowanie	127
Pytania	128
Odnośniki	129
Lektura uzupełniająca	130

ROZDZIAŁ 4.**Postępy w tłumaczeniach z wykorzystaniem Google Trax,****Tłumacza Google i Gemini 131**

Definicja tłumaczenia maszynowego	132
Transdukcje i tłumaczenia wykonywane przez ludzi	133
Transdukcje i tłumaczenia maszynowe	134

Ocena tłumaczeń maszynowych	135
Wstępne przetwarzanie zbioru danych WMT	135
Ocena tłumaczeń maszynowych według BLEU	141
Tłumaczenia z wykorzystaniem Google Trax	144
Instalowanie biblioteki Trax	145
Tworzenie modelu oryginalnego transformera	145
Inicjalizowanie modelu z wykorzystaniem wyuczonych wag	147
Tokenizowanie zdania	147
Dekodowanie wyjścia z transformera	147
Detokenizowanie i wyświetlanie tłumaczenia	148
Tłumaczenie za pomocą Tłumacza Google	149
Tłumaczenie z wykorzystaniem wrappera interfejsu Google Translate Ajax API	150
Tłumaczenie z wykorzystaniem systemu Gemini	152
Potencjał systemu Gemini	153
Podsumowanie	154
Pytania	155
Odnosiniki	155
Lektura uzupełniająca	156

ROZDZIAŁ 5.

Szczegóły dostrajania z wykorzystaniem modelu BERT	157
Architektura BERT	158
Stos kodera	159
Dostrajanie modelu BERT	166
Określanie celu	167
Ograniczenia sprzętowe	167
Instalowanie transformerów Hugging Face	168
Importowanie modułów	168
Określanie CUDA jako urządzenia dla modułu torch	169
Ładowanie zestawu danych CoLA	169
Tworzenie zdań i list etykiet oraz dodawanie tokenów BERT	171
Aktywowanie tokenizera BERT	171
Przetwarzanie danych	172
Tworzenie masek uwagi	172
Dzielenie danych na zbiór szkoleniowy i zbiór walidacyjny	173
Konwertowanie danych na tensory torch	173
Wybieranie rozmiaru partii i tworzenie iteratora	173

Konfigurowanie modelu BERT	174
Ładowanie bazowego modelu Hugging Face bert-base-uncased	176
Pogrupowane parametry optymalizatora	177
Hiperparametry pętli szkoleniowej	179
Pętla szkolenia	180
Ocena szkolenia	181
Prognozowanie i ocena z użyciem wydzielonego zbioru danych	181
Ocena modelu z wykorzystaniem współczynnika korelacji Matthews'a	184
Ocena za pomocą współczynnika korelacji Matthews'a całego zestawu danych	185
Budowanie interfejsu Pythona do interakcji z modelem	186
Zapisywanie modelu	186
Tworzenie interfejsu dla przeszkolonego modelu	187
Podsumowanie	189
Pytania	190
Odnośniki	191
Lektura uzupełniająca	191

ROZDZIAŁ 6.

Wstępne szkolenie transformera od podstaw

z wykorzystaniem modelu RoBERTa

Szkolenie tokenizera i wstępne szkolenie transformera	194
Budowanie modelu KantaiBERT od podstaw	195
Krok 1. Ładowanie zbioru danych	196
Krok 2. Instalowanie transformerów Hugging Face	197
Krok 3. Szkolenie tokenizera	198
Krok 4. Zapisywanie plików na dysku	200
Krok 5. Ładowanie plików tokenizera po przeszkoleniu	201
Krok 6. Sprawdzanie ograniczeń zasobów: GPU i CUDA	202
Krok 7. Definiowanie konfiguracji modelu	203
Krok 8. Ponowne ładowanie tokenizera w module transformers	203
Krok 9. Inicjalizowanie modelu od podstaw	203
Krok 10. Tworzenie zbioru danych	209
Krok 11. Definiowanie mechanizmu zbierania danych	210
Krok 12. Inicjalizowanie trenera	210
Krok 13. Wstępne szkolenie modelu	212

Krok 14. Zapisywanie przeszkolonego modelu (+ tokenizer + konfiguracja) na dysku	212
Krok 15. Modelowanie języka za pomocą potoku FillMaskPipeline	213
Wstępne szkolenie modelu obsługi klienta generatywnej sztucznej inteligencji na danych pochodzących z serwisu X	215
Krok 1. Pobieranie zbioru danych	216
Krok 2. Instalowanie bibliotek Hugging Face: transformers i datasets	216
Krok 3. Ładowanie i filtrowanie danych	216
Krok 4. Sprawdzanie ograniczeń zasobów: układ GPU i CUDA	218
Krok 5. Definiowanie konfiguracji modelu	218
Krok 6. Tworzenie i przetwarzanie zbioru danych	219
Krok 7. Inicjalizowanie obiektu trenera	220
Krok 8. Wstępne szkolenie modelu	221
Krok 9. Zapisywanie modelu	221
Krok 10. Interfejs użytkownika do czatu z agentem generatywnej AI	222
Dalsze szkolenie wstępne	224
Ograniczenia	224
Następne kroki	224
Podsumowanie	225
Pytania	226
Odnosniki	226
Lektura uzupełniająca	227

ROZDZIAŁ 7.

ChatGPT — rewolucja w generatywnej sztucznej inteligencji

Model GPT jako technologia ogólnego przeznaczenia	229
Udoskonalenia	230
Rozpowszechnianie	231
Wszechobecność	232
Architektura modeli transformerów GPT firmy OpenAI	234
Rozwój modeli transformerów o miliardach parametrów	235
Coraz większe rozmiary modeli transformerów	235
Rozmiar kontekstu i maksymalna długość ścieżki	237
Od dostrajania do modeli zero-shot	238
Stos warstw dekodera	240
Modele GPT	241

Modele OpenAI w roli asystentów	244
ChatGPT udostępnia kod źródłowy	244
Asystent tworzenia kodu GitHub Copilot	245
Przykłady promptów ogólnego przeznaczenia	247
Rozpoczęcie pracy z ChatGPT — GPT-4 w roli asystenta	249
Rozpoczęcie pracy z API modelu GPT-4	254
Uruchomienie pierwszego zadania NLP z użyciem modelu GPT-4	254
Uruchamianie wielu zadań NLP	257
Wykorzystanie techniki RAG z GPT-4	258
Instalacja	258
Odzyskiwanie informacji z dokumentów	259
Zastosowanie techniki RAG	260
Podsumowanie	263
Pytania	264
Odnośniki	264
Lektura uzupełniająca	265

ROZDZIAŁ 8.

Dostrajanie modeli GPT OpenAI	266
Zarządzanie ryzykiem	267
Dostrajanie modelu GPT do wykonywania (generatywnego) zadania uzupełniania	268
1. Przygotowywanie zbioru danych	270
1.1. Przygotowywanie danych w formacie JSON	270
1.2. Konwertowanie danych do formatu JSONL	272
2. Dostrajanie oryginalnego modelu	275
3. Uruchamianie dostrojonego modelu GPT	277
4. Zarządzanie zadaniami dostrajania i dostrojonymi modelami	280
Przed zakończeniem	281
Podsumowanie	283
Pytania	283
Odnośniki	284
Lektura uzupełniająca	284

ROZDZIAŁ 9.**Rozbijanie czarnej skrzynki za pomocą narzędzi****do interpretacji działania transformerów 285**

Wizualizacja działania transformera z użyciem BertViz	287
Uruchamianie BertViz	287
Interpretacja działania transformerów Hugging Face za pomocą narzędzia SHAP	299
Podstawowe informacje o SHAP	299
Wyjaśnienie wyników transformerów Hugging Face z użyciem SHAP	302
Wizualizacja transformera poprzez uczenie słownikowe	304
Współczynniki transformera	304
Wprowadzenie do LIME	306
Interfejs wizualizacji	307
Inne narzędzia interpretacji mechanizmów AI	309
LIT	309
Modele LLM OpenAI wyjaśniają działanie neuronów w transformerach	313
Ograniczenia i kontrola ze strony człowieka	316
Podsumowanie	317
Pytania	317
Odnośniki	318
Lektura uzupełniająca	318

ROZDZIAŁ 10.**Badanie roli tokenizerów w kształtowaniu modeli transformerów ... 319**

Dopasowywanie zbiorów danych i tokenizerów	320
Najlepsze praktyki	321
Tokenizacja Word2Vec	325
Badanie tokenizerów zdań i tokenizerów WordPiece w celu zrozumienia wydajności tokenizerów podwyrazów w kontekście ich wykorzystania przez transformery	334
Tokenizerzy wyrazów i zdań	335
Tokenizerzy oparte na podwyrazach	339
Badanie tokenizerów w kodzie	343
Podsumowanie	348
Pytania	349
Odnośniki	349
Lektura uzupełniająca	349

ROZDZIAŁ 11.**Wykorzystanie osadzeń LLM jako alternatywy****dla precyzyjnego dostrajania 350**

Osadzenia LLM jako alternatywa dla precyzyjnego dostrajania 352

Od projektowania promptów do inżynierii promptów 352

Podstawy osadzania tekstu za pomocą NLTK i Gensim 353

Instalowanie bibliotek 353

1. Odczytywanie pliku tekstowego 353

2. Tokenizacja tekstu z użyciem tokenizera Punkt 354

3. Osadzanie tekstu za pomocą Gensim i Word2Vec 356

4. Opis modelu 357

5. Dostęp do słowa i wektora słów 359

6. Analiza przestrzeni wektorowej Gensim 360

7. TensorFlow Projector 363

Implementacja systemów pytań i odpowiedzi z użyciem technik
opartych na osadzeniach 367

1. Instalowanie bibliotek i wybór modeli 367

2. Implementacja modelu osadzeń i modelu GPT 368

3. Przygotowywanie danych do wyszukiwania 372

4. Wyszukiwanie 373

5. Zadawanie pytania 375

Uczenie transferowe z użyciem osadzeń Ada 379

1. Zbiór danych Amazon Fine Food Reviews 379

2. Obliczanie osadzeń Ada i zapisywanie ich w celu
ponownego wykorzystania w przyszłości 381

3. Klasteryzacja 382

4. Próbkę tekstu w klastrach i nazwy klastrów 384

Podsumowanie 386

Pytania 387

Odnosińniki 387

Lektura uzupełniająca 387

ROZDZIAŁ 12.**Oznaczanie ról semantycznych bez analizy składniowej****z wykorzystaniem modelu GPT-4 i ChatGPT 388**

Rozpoczynanie pracy z technikami SRL 390

Wprowadzenie do świata AI bez składni 391

Definicja SRL 392

Wizualizacja SRL 393

Eksperymenty SRL z ChatGPT z modelem GPT-4	393
Prosty przykład	394
Trudny przykład	398
Kwestionowanie zakresu SRL	399
Wyzwania związane z analizą orzeczeń	400
Ponowna definicja SRL	401
Od technik SRL specyficznych dla zadania do emergencji z wykorzystaniem ChatGPT	403
1. Instalowanie OpenAI	404
2. Tworzenie funkcji dialogu z GPT-4	404
3. Uruchamianie żądań SRL	405
Podsumowanie	412
Pytania	413
Odnośniki	414
Lektura uzupełniająca	414

ROZDZIAŁ 13.

Zadania generowania streszczeń z użyciem modeli T5 i ChatGPT 415

Projektowanie uniwersalnego modelu tekst – tekst	417
Powstanie modeli transformerów tekst – tekst	418
Prefiks zamiast formatów specyficznych dla zadań	419
Model T5	421
Tworzenie streszczeń tekstu z użyciem modelu T5	423
Hugging Face	423
Inicjalizowanie modelu transformera T5	425
Tworzenie streszczeń dokumentów z użyciem modelu T5	431
Od transformera tekst – tekst do prognoz nowych słów z użyciem systemu ChatGPT firmy OpenAI	438
Porównanie metod tworzenia streszczeń modelu T5 i systemu ChatGPT	438
Tworzenie streszczeń z użyciem ChatGPT	439
Podsumowanie	444
Pytania	444
Odnośniki	445
Lektura uzupełniająca	445

ROZDZIAŁ 14.**Najnowocześniejsze modele LLM Vertex AI i PaLM 2 446**

Architektura	448
Pathways	448
PaLM	451
PaLM 2	452
Asystenty AI	454
Gemini	456
Google Workspace	457
Google Colab Copilot	460
Interfejs Vertex AI modelu PaLM 2	462
API PaLM 2 Vertex AI	468
Odpowiadanie na pytania	469
Zadanie typu pytanie – odpowiedź	471
Podsumowanie dialogu	472
Analiza tonu	473
Zadania wielokrotnego wyboru	475
Kod	477
Dostrajanie	482
Utworzenie kontenera	483
Dostrajanie modelu	484
Podsumowanie	486
Pytania	487
Odnośniki	487
Lektura uzupełniająca	487

ROZDZIAŁ 15.**Pilnowanie gigantów, czyli łagodzenie zagrożeń związanych****z użyciem modeli LLM 488**

Powstanie funkcjonalnej sztucznej inteligencji ogólnej (AGI)	490
Ograniczenia instalacji najnowocześniejszych platform	492
Auto-BIG-bench	495
WandB	501
Kiedy agenty AI zaczną się replikować?	503
Zarządzanie zagrożeniami	505
Halucynacje i zapamiętywanie	506
Ryzykowne zachowania emergentne	511

Dezinformacja	513
Wywieranie wpływu na opinię publiczną	514
Treści szkodliwe	516
Prywatność	518
Cyberbezpieczeństwo	519
Narzędzia do łagodzenia zagrożeń z RLHF i RAG	520
1. Moderowanie wejścia i wyjścia za pomocą transformerów i bazy reguł	521
2. Budowanie bazy wiedzy dla systemu ChatGPT i modelu GPT-4	525
3. Parsowanie żądań użytkownika i korzystanie z bazy wiedzy	527
4. Generowanie zawartości ChatGPT z funkcją obsługi dialogu	528
Podsumowanie	530
Pytania	531
Odnosińki	531
Lektura uzupełniająca	532

ROZDZIAŁ 16.

Nie tylko tekst — transformery wizyjne u progu

rewolucyjnej sztucznej inteligencji 533

Od modeli niezależnych od zadań do multimodalnych transformerów wizyjnych	534
Transformery wizyjne (ViT)	536
Podstawowa architektura ViT	536
Transformery wizyjne w kodzie	539
CLIP	549
Podstawowa architektura modelu CLIP	549
CLIP w kodzie	550
DALL-E 2 i DALL-E 3	553
Podstawowa architektura DALL-E	554
Wprowadzenie w tematykę API modeli DALL-E 2 i DALL-E 3	555
GPT-4V, DALL-E 3 i rozbieżne skojarzenia semantyczne	561
Definicja rozbieżnego skojarzenia semantycznego	561
Tworzenie obrazu z użyciem systemu ChatGPT Plus z DALL-E	563
Wykorzystanie API modelu GPT-4V i eksperymenty z zadaniami DAT ...	565
Podsumowanie	570
Pytania	571
Odnosińki	571
Lektura uzupełniająca	572

ROZDZIAŁ 17.**Przekraczanie granic między obrazem a tekstem**

z użyciem modelu Stable Diffusion	573
Przekraczanie granic generowania obrazu	574
Część I. Zamiana tekstu na obraz z użyciem modelu Stable Diffusion	576
1. Osadzanie tekstu za pomocą kodera transformera	577
2. Tworzenie losowych obrazów z szumami	578
3. Próbkowanie w dół modelu Stable Diffusion	579
4. Próbkowanie w górę na poziomie dekodera	581
5. Wynikowy obraz	582
Uruchamianie implementacji Keras modelu Stable Diffusion	582
Część II. Zamiana tekstu na obraz za pomocą API Stable Diffusion	584
Wykorzystanie modelu Stable Diffusion generatywnej sztucznej inteligencji do wykonania zadania z zakresu skojarzeń rozbieżnych (DAT)	586
Część III. Zamiana tekstu na wideo	588
Zamiana tekstu na wideo z użyciem modeli animacji Stability AI	588
Zamiana tekstu na wideo z użyciem odmiany modelu CLIP firmy OpenAI	590
Zamiana wideo na tekst z użyciem modelu TimeSformer	592
Przygotowywanie klatek wideo	593
Wykorzystanie modelu TimeSformer do tworzenia prognoz na podstawie klatek wideo	595
Podsumowanie	596
Pytania	597
Odnośniki	597
Lektura uzupełniająca	597

ROZDZIAŁ 18.**AutoTrain na platformie Hugging Face —**

szkolenie modeli wizyjnych bez kodowania	598
Cel i zakres tego rozdziału	600
Pierwsze kroki	601
Przesyłanie zestawu danych	602
Bez kodowania?	605
Szkolenie modeli za pomocą mechanizmu AutoTrain	606
Wdrażanie modelu	607

Uruchamianie modeli w celu wnioskowania	609
Pobieranie obrazów walidacyjnych	609
Wnioskowanie: klasyfikacja obrazów	611
Eksperymenty walidacyjne na przeszkolonych modelach	613
Wypróbowywanie skuteczności najlepszego modelu ViT dla korpusu obrazów	626
Podsumowanie	627
Pytania	628
Odnosińki	629
Lektura uzupełniająca	629

ROZDZIAŁ 19.

Na drodze do funkcjonalnej ogólnej AI z systemem HuggingGPT

i jego odpowiednikami	630
Definicja systemu F-AGI	632
Instalowanie i importowanie bibliotek	635
Zbiór walidacyjny	635
Poziom 1 — łatwy obraz	636
Poziom 2 — trudny obraz	636
Poziom 3 — bardzo trudny obraz	637
HuggingGPT	638
Poziom 1 — łatwy	640
Poziom 2 — trudny	642
Poziom 3 — bardzo trudny	645
CustomGPT	649
Google Cloud Vision	651
Łączenie modeli: Google Cloud Vision z ChatGPT	656
Łączenie modeli z użyciem systemu Runway Gen-2	658
Midjourney: wyobraź sobie okręt płynący w przestrzeni galaktycznej	659
Gen-2: niech ten statek pływa po morzu	660
Podsumowanie	661
Pytania	662
Odnosińki	663
Lektura uzupełniająca	663

ROZDZIAŁ 20.**Nie tylko prompty projektowane przez człowieka —****generatywne kreowanie pomysłów 664**

Część I. Definicja generatywnego kreowania pomysłów 666

Zautomatyzowana architektura kreowania pomysłów 666

Zakres i ograniczenia 668

Część II. Automatyzacja projektowania promptów

na potrzeby generatywnego projektowania obrazów 668

Prezentacja HTML systemu opartego na ChatGPT z modelem GPT-4 669

Llama 2 674

Wykorzystanie modelu Llama 2 z modelem Hugging Face 675

Midjourney 680

Microsoft Designer 687

Część III. Zautomatyzowane generatywne kreowanie pomysłów

z użyciem modelu Stable Diffusion 690

1. Brak promptu, automatyczne instrukcje dla modelu GPT-4 691

2. Generowanie promptu przez generatywną sztuczną inteligencję
z użyciem ChatGPT z modelem GPT-4 6943. i 4. Generowanie obrazów przez generatywną sztuczną
inteligencję z użyciem modelu Stable Diffusion i ich wyświetlanie 696

Przyszłość należy do Ciebie! 698

Przyszłość programistów dzięki technikom VR-AI 698

Podsumowanie 703

Pytania 704

Odnosińniki 704

Lektura uzupełniająca 705

Dodatek. Odpowiedzi na pytania 706**Skorowidz 728**

Czym są transformery?

Rozdział 1

Transformery to powszechne w branży AI, ustandaryzowane modele językowe **LLM** (ang. *Large Language Models* — duże modele językowe) przeznaczone do obliczeń równoległych. Model transformera pozwala wykonywać szeroki zakres zadań bez konieczności dostrajania. Transformery mogą przeprowadzać samonadzorowane uczenie na miliardach nieprzetworzonych rekordów nieoznakowanych danych z miliardami parametrów. Z tych modeli o miliardach parametrów wyłoniły się architektury multimodalne, umożliwiające przetwarzanie tekstu, zdjęć, dźwięku i wideo.

Wykorzystanie architektur transformerów spopularyzował ChatGPT. Od momentu jego pojawienia się stały się one technologiami ogólnego przeznaczenia, wykorzystywanymi w takich dziedzinach jak branża wydawnicza, elektronika i informatyka.

Transformery są stosowane niemal wszędzie! Wykorzystują je Google Cloud AI, Amazon Web Services (AWS), Microsoft Azure, OpenAI, Google Workspace, Microsoft 365, Google Colab Copilot, GitHub Copilot, Hugging Face, Meta i niezliczone inne usługi.

Funkcjonalność modeli transformerów przeniknęła do wielu aspektów ludzkiej działalności. Wykorzystuje się je między innymi w takich technologiach, jak generatywna sztuczna inteligencja przetwarzania tekstu (ang. *Generative AI for text*), generatywna sztuczna inteligencja przetwarzania obrazów (ang. *Generative AI for Images*), dyskryminatywna sztuczna inteligencja, modele specyficzne dla konkretnych zadań, uczenie nie-nadzorowane, uczenie nadzorowane, projektowanie i inżynieria odpowiedzi, zamiana tekstu na kod oraz kodu na tekst. Czasami model podobny do stosowanego w GPT obejmuje wszystkie te pojęcia!

Wpływ na społeczeństwo jest ogromny. Programowanie aplikacji w wielu przypadkach stało się ćwiczeniem edukacyjnym. Menedżer projektu może teraz przejść do platformy chmurowej OpenAI, zarejestrować się, uzyskać klucz API i w ciągu kilku minut rozpocząć pracę. Następnie użytkownicy mogą wprowadzić tekst, określić zadanie NLP (ang. *natural language processing* — przetwarzania języka naturalnego) jako Google Workspace lub Microsoft 365 i uzyskać odpowiedź utworzoną przez Google Vertex AI lub model transformera ChatGPT. Użytkownicy mogą również skorzystać z platformy Google Gen App Builder i tworzyć aplikacje bez znajomości programowania lub uczenia maszynowego.

Liczbę przyprawiają o zawrót głowy. Bommasani i współpracownicy (2023) opisali ekosystem Foundation Model zawierający 128 modeli podstawowych, 70 aplikacji i 64

zestawy danych. W artykule wspomniano również o ponad 150 000 modeli Hugging Face i ponad 20 000 zestawów danych! Lista ta rozszerza się z tygodnia na tydzień i rozprzestrzenia na każdy obszar społecznej działalności.

Gdzie w tym ekosystemie funkcjonuje specjalista AI lub ktoś, kto chce nim zostać?

Czy menedżer projektu powinien pracować lokalnie? A może implementacja powinna być wykonana bezpośrednio na platformach Google Cloud, Microsoft Azure lub AWS? Czy zespół programistów powinien wybrać Hugging Face, Google Trax, OpenAI czy AllenNLP? Czy specjalista AI powinien korzystać z API, które praktycznie nie wymaga umiejętności programowania w dziedzinie sztucznej inteligencji? Czy użytkownik powinien skorzystać z platformy Google Gen App Builder i zbudować aplikację AI techniką bez kodowania oraz bez wiedzy w dziedzinie ML?

Odpowiedź na *wszystkie* powyższe pytania brzmi „tak”! Nie wiadomo, czego może chcieć lub oczekiwać przyszły pracodawca, klient bądź użytkownik. Dlatego przygotuj się na dostosowanie się do każdej potrzeby, która się pojawi — na poziomie zbioru danych, modelu i aplikacji. Niniejsza książka nie opisuje wszystkich technologii dostępnych na rynku. Nie można nauczyć się wszystkich modeli i wszystkich dostępnych platform, które istnieją na rynku. Jeśli będziesz próbować nauczyć się wszystkiego, niczego nie zapamiętasz. Trzeba wiedzieć, od czego zacząć i kiedy przestać. Dzięki lekturze tej książki zdobędziesz wystarczającą wiedzę do tego, by potrafić dostosować się do stale zmieniającego się rynku.

Niniejszy rozdział rozpocznę od zaprezentowania niesamowitych możliwości modeli transformerów o zwoźniczo prostej złożoności czasowej $O(1)$, które zmieniły wszystko. Aby zobaczyć, jak transformery wykorzystują akceleratory sprzętowe, zbudujemy notatnik z wykorzystaniem Pythona, PyTorch i TensorFlowa. Następnie pokażę, jak pojedynczy token (najmniejsza część słowa) doprowadził do rewolucji AI, którą obserwujemy.

W dalszej części zaprezentuję mało znany algorytm transformera z 2017 roku, który zdominował wiele dziedzin. Byłem zmuszony znaleźć dla niego nową nazwę: określiłem go jako model podstawowy (ang. *Foundation Model*). Za pomocą modeli podstawowych można w AI zrobić prawie wszystko! Wystarczy wygodnie usiąść i obserwować, jak ChatGPT wyjaśnia, analizuje i pisze w notatniku Pythona program klasyfikujący i wyświetla drzewo decyzyjne.

Na koniec tego rozdziału przedstawię rolę specjalisty w dziedzinie sztucznej inteligencji na stale zmieniającym się rynku pracy. Zajmę się problemem wyboru odpowiednich zasobów.

Przed rozpoczęciem eksploracyjnej podróży przez opisane w tej książce różnorodne implementacje modeli transformerów trzeba zapoznać się z kluczowymi pojęciami.

Oto zagadnienia, na których skoncentruję się w tym rozdziale.

- Jak jeden wynalazek $O(1)$ zmienił bieg historii sztucznej inteligencji
- W jaki sposób modele transformerów przejęły akceleratory sprzętowe
- Jak jeden token obalił setki aplikacji AI

- Wiele aspektów modelu transformera
- Generatywna sztuczna inteligencja kontra dyskryminatywna sztuczna inteligencja
- Uczenie nienadzorowane i samonadzorowane a uczenie nadzorowane
- Modele ogólnego przeznaczenia a modele specyficzne dla zadania
- W jaki sposób ChatGPT zmienił znaczenie automatyzacji
- Prezentacja tworzenia i dokumentowania programu klasyfikacyjnego przez ChatGPT
- Rola specjalistów ds. sztucznej inteligencji
- Płynne API transformerów
- Wybór modelu transformera

Uwaga

Ze względu na wiele innowacji i aktualizacji bibliotek w tej najnowocześniejszej technologii pakiety i modele często się zmieniają. Najnowsze przykłady instalacji i kodu można znaleźć w repozytorium GitHub (<https://github.com/Denis2054/Transformers-for-NLP-and-Computer-Vision-3rd-Edition/tree/main/Chapter01>).

W przypadku jakichkolwiek problemów z uruchomieniem kodu w tym lub innym rozdziale możesz również opublikować wiadomość w społeczności Discord (<https://www.packt.link/Transformers>).

Na początek przyjrzymy się, jak doszło do rewolucji wywołanej przez transformery.

Stała złożoność czasowa $O(1)$, która na zawsze zmieniła nasze życie

W jaki sposób ta zводniczo prosta klasa złożoności czasowej $O(1)$ mogła na zawsze zmienić sztuczną inteligencję i nasze codzienne życie? W jaki sposób złożoność $O(1)$ mogła przyczynić się do wyjaśnienia głębokich zmian architektonicznych, dzięki którym ChatGPT stał się tak potężny i wprawił świat w osłupienie? Jak coś tak prostego jak $O(1)$ mogło pozwolić systemom podobnym do ChatGPT rozprzestrzenić się na każdą dziedzinę i setki zadań?

Odpowiedzią na te pytania jest to, że jedynym sposobem na odnalezienie drogi w rosnącym labiryncie zbiorów danych, modeli i aplikacji jest *skupienie się* na pojęciach leżących u podstaw tysięcy zasobów. Pojęcia te doprowadzą Cię do sedna funkcjonalności, których potrzebujesz w swoich projektach.

W tym podrozdziale najpierw udzielę odpowiedzi na te ważne pytania, a następnie przejdę do wyjaśnienia, jak jeden token, czyli minimalny fragment słowa, rozpoczął

rewolucję AI. Ta rewolucja rozprzestrzeniła się na całym świecie i doprowadziła do automatyzacji, jakiej nie widziano nigdy wcześniej.

Spróbujmy dotrzeć do sedna chaosu i rewolucji wywołanej przez transformery.

Aby osiągnąć ten cel, w tym podrozdziale wykorzystam naukę i technologię. Pozwolę nam one zrozumieć, jak to wszystko się zaczęło. Na podstawie notatnika zbudowanego z wykorzystaniem Pythona i biblioteki PyTorch najpierw przyjrzymy się złożoności $O(1)$, a następnie złożoności warstwy.

Zacznę od wyjaśnienia podstawowych pojęć i terminologii dotyczącej złożoności $O(1)$.

Uwaga $O(1)$ pokonuje rekurencyjne metody $O(n)$

$O(1)$ to zapis z użyciem notacji „wielkiego O ”. „Wielkie O ” oznacza rząd (ang. *order*). W tym przypadku $O(1)$ oznacza stałą złożoność czasową. Można powiedzieć, że operacja ma złożoność $O(1)$ lub że ma złożoność rzędu 1.

Możesz w to wierzyć lub nie, ale jesteś w samym sercu rewolucji!

Architekturę modelu transformera objaśnię w rozdziale 2. „Wprowadzenie do architektury modelu transformera”.

W tym podrozdziale i rozdziale skupię się najpierw na tym, co doprowadziło do rozpowszechnienia się sztucznej inteligencji poprzez transformery i szłał ChatGPT: wykładniczym wzroście wydajności sprzętu spowodowanym zastosowaniem techniki *samo-uwagi* (ang. *self-attention*). Pokażę, w jaki sposób warstwa uwagi wykorzystuje sprzęt i otwiera drzwi do niesamowitych nowych technik uczenia maszynowego.

Poniższe zdanie zawiera 9 słów:

Jerzy lubi pomarańcze rano, ale nie lubi ich wieczorem.

Długość tego zdania wynosi $n = 9$.

Problem rozumienia języka można podsumować jednym słowem: kontekst. Słowo rzadko można zdefiniować bez kontekstu. Znaczenie słowa może się zmieniać w różnych kontekstach poza jego słownikową definicją.

Zacznijmy od pojęciowego podejścia do warstwy uwagi.

Warstwa uwagi

Jeśli spojrzymy na to, co lubi Jerzy, możemy wskazać na kilka zależności wymiarów (lub parametrów) słowa *pomarańcze*:

Wymiar 1.: związek między słowami *pomarańcze* i *wieczór*

Wymiar 2.: związek między słowami *pomarańcze* i *rano*

Wymiar 3.: związek między słowami *Jerzy* i *pomarańcze*

Wymiar 4.: związek między słowami *Jerzy* i *rano*

...

Wymiar z

Zauważ, że związki są definiowane parami: jedno słowo do jednego słowa. Tak właśnie działa *samouwaga* w transformerze.

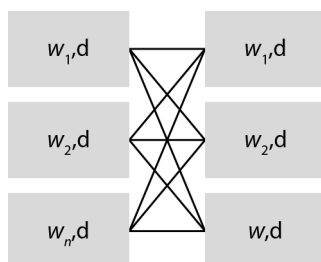
Jeśli przedstawimy to za pomocą notacji wielkiego O , otrzymamy $O(1)$. „ O ” czytamy jako „rzędu”. Na przykład $O(1)$ jest klasą złożoności „rzędu 1”, co oznacza stałą złożoność czasową.

Aby znaleźć związek z innym słowem w analizie parami, wykonujemy dla każdego słowa jedną operację $O(1)$.

W przełożeniu na liczby:

- n = długość sekwencji, która w tym przypadku wynosi 11 słów.
- d = liczba wymiarów wyrażona za pomocą liczb zmiennoprzecinkowych. W uczeniu maszynowym wymiary są wyrażane w liczbach zmiennoprzecinkowych. Na przykład jeśli x jest słowem, wartości mogą być następujące: $[-0.2333, 03.8559, 0.9844 \dots 0394]$. Model nauczy się tych wartości na podstawie miliardów punktów danych tekstowych.

Omawiany związek par to obliczenie $n * n$, co zilustrowałem na rysunku 1.1.



Rysunek 1.1. Związki między słowami

W tym przypadku $O(1)$ reprezentuje złożoność pamięciową.

Złożoność obliczeniowa warstwy uwagi wynosi zatem $O(n^2 \cdot d)$. Złożoność n^2 ma operacja parowania (słowo – słowo) całej sekwencji n . Z kolei d reprezentuje wymiary, których model się uczy.

Zobaczmy różnicę w porównaniu z warstwą rekurencyjną.

Warstwa rekurencyjna

Warstwy rekurencyjne nie działają w ten sposób. Mają złożoność $O(n)$, czyli rzędu n , co oznacza liniową złożoność czasową. Im dłuższa sekwencja, tym zajmuje więcej pamięci. Dlaczego? Warstwy rekurencyjne nie uczą się wymiarów z wykorzystaniem związków parami. Uczą się w określonej kolejności. Na przykład:

Wymiar a: *Jerzy*

Wymiar b: *lubi i Jerzy*

Wymiar c: *pomarańcze i lubi i Jerzy*

Wymiar d: *w i pomarańcze i lubi i Jerzy*

...

Wymiar z

Jak łatwo zauważyć, dla określonego słowa nie jest analizowany związek z innym słowem, lecz z kilkoma innymi słowami jednocześnie!

Liczbę wymiarów d jednego słowa trzeba pomnożyć przez wymiary poprzedniego słowa, dlatego w rzeczywistości mamy d^2 wymiarów. Złożoność obliczeniowa warstwy rekurencyjnej wynosi zatem:

$$O(n \cdot d^2)$$

Przyjrzyjmy się bliżej tej magii.

Magia obliczeniowej złożoności czasowej warstwy uwagi

Warstwa uwagi ma złożoność czasową dostępu do pamięci $O(1)$. Z tego wynika, że obliczenie iloczynu skalarnego między każdym słowem wymaga operacji o złożoności czasowej $O(n^2 \cdot d)$. W uczeniu maszynowym oznacza to pomnożenie reprezentacji d każdego słowa przez inne słowo. Warstwę uwagi można zatem nauczyć wszystkich związków za pomocą jednej operacji mnożenia macierzy!

Warstwa rekurencyjna o złożoności czasowej obliczeń $O(n \cdot d^2)$ jest ograniczona przez jej liniowy, sekwencyjny proces $O(n)$. Wykonanie tego samego zadania co w warstwie uwagi zajmie więcej operacji. Przeprowadzimy teraz symulację modelu złożoności czasowej uwagi i rekurencyjnego modelu złożoności czasowej w notatniku Pythona, z wykorzystaniem bibliotek PyTorch i TensorFlow.

Obliczenia mają charakter koncepcyjny. W tym podrozdziale przeprowadzimy symulację z użyciem układów CPU, GPU i TPU:

- **Procesor główny** (ang. *Central Processing Unit* — CPU) — podstawowy komponent obliczeniowy komputera.
- **Procesor graficzny** (ang. *Graphics Processing Unit* — GPU) — wyspecjalizowana jednostka obliczeniowa. Układy GPU były początkowo wykorzystywane do renderowania obrazów 3D, a następnie ewoluowały do wykonywania zadań uczenia maszynowego, takich jak mnożenie macierzy.
- **Procesor tensorowy** (ang. *Tensor Processing Unit* — TPU) — procesor przyspieszający uczenie maszynowe opracowany przez Google'a i zoptymalizowany pod kątem biblioteki TensorFlow.

Otwórz notatnik *0-1_and_Accelerators.ipynb* z katalogu repozytorium niniejszego rozdziału.

Zacniemy od obliczeń z wykorzystaniem układu CPU.

Czasowa złożoność obliczeniowa przy użyciu CPU

CPU to procesor obliczeniowy ogólnego przeznaczenia. Nie jest specjalnie zaprojektowany do mnożenia macierzy. Pozwala wykonywać złożone operacje, ale tylko na pewnym poziomie wydajności.

Przed uruchomieniem pierwszej komórki notatnika trzeba sprawdzić, czy używamy procesora CPU. Aby to zrobić, przejdź do menu *Środowisko wykonawcze*, wybierz opcję *Zmień typ środowiska wykonawczego* i upewnij się, że parametr *Akcelerator sprzętowy* jest ustawiony na *CPU* (rysunek 1.2).

Zmień typ środowiska wykonawczego

Typ środowiska wykonawczego

Python 3

Akcelerator sprzętowy ?



CPU



T4 GPU



A100 GPU



L4 GPU



TPU v2-8

Potrzebujesz dostępu do lepszych GPU?

[Kup dodatkowe jednostki obliczeniowe](#)

Anuluj

Zapisz

Rysunek 1.2. Wybór typu środowiska wykonawczego

W gruncie rzeczy akceleratorem sprzętowym jest procesor graficzny (GPU) lub procesor tensorowy (TPU), który wykonuje określone zadania obliczeniowe (na przykład mnożenie macierzy) wydajniej niż procesor główny (CPU) ogólnego przeznaczenia. Ten ostatni nie jest akceleratorem sprzętowym.

Notatnik rozpoczyna się tabelką, w której zestawiono złożoności czasowe omówione przed chwilą (rysunek 1.3).

Celem notatnika jest przedstawienie złożoności poszczególnych warstw samouwagi i warstw rekurencyjnych, a nie samych algorytmów. Złożoność czasowa obliczeń mechanizmu samouwagi zależy od operacji mnożenia macierzy.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	Recurrent $O(n \cdot d^2)$	$O(n)$	$O(n)$

Rysunek 1.3. Złożoności czasowe według książki Attention is All You Need (Vaswani et al., 2017, str. 6)

Rekurencyjny czas obliczeniowy wynika z wykorzystania pętli symulującej podejście sekwencyjne. Model rekurencyjny do obliczania wartości wykorzystuje macierze, jednak jego ogólna złożoność obliczeniowa jest rzędu n i d . Warto zauważyć, że obliczenia nie odzwierciedlają rzeczywistych algorytmów wewnątrz warstw samouwagi i warstw rekurencyjnych. Na wydajność wpływa również wiele innych czynników: sprzęt, dane, hiperparametry, czas szkolenia itp.

Celem tego notatnika jest pokazanie ogólnej złożoności obliczeniowej. W tym przypadku symulacje są wystarczające. Najpierw zdefiniujemy framework ewaluacji:

```
# Czasy obliczeniowe złożoności na warstwę
# Porównanie czasu obliczeniowego pomiędzy:
# samouwaga =  $O(n^2 * d)$ 
# oraz
# warstwa rekurencyjna =  $O(n * d^2)$ 
```

Następnie importujemy pakiety numpy i time:

```
import numpy as np
import time
```

Najpierw definiujemy długość sekwencji (liczbę słów w sekwencji) i wymiary (wektor liczbowy reprezentujący cechy słowa):

```
# Zdefiniuj długość sekwencji i wymiarowość reprezentacji
n = 512
d = 512
```

Zauważysz, że $n = d$, co oznacza, że $O(n^2 \cdot d) = O(n \cdot d^2) = 512 \cdot 512 \cdot 512 = 134\,217\,728$ operacji. W tym przypadku zarówno warstwa uwagi, jak i warstwa rekurencyjna z perspektywy czasowej złożoności obliczeniowej mają do wykonania tyle samo operacji. Zdefiniujemy sekwencję danych wejściowych z losowymi wartościami dla wymiarów (d):

```
# Zdefiniuj wejścia
```

Najpierw zasymulujemy złożoność czasową warstwy samouwagi za pomocą mnożenia macierzy, z określeniem czasu początku operacji `start_time`:

```
# Symulacja warstwy samouwagi  $O(n^2 \cdot d)$ 
start_time = time.time()
for i in range(n):
    for j in range(n):
        _ = np.dot(input_seq[i], input_seq[j])
```


Po zakończeniu mnożenia macierzy obliczamy czas trwania tych obliczeń i wyświetlamy go:

```
at=time.time()-start_time
print(f"Czas obliczeń warstwy samouwagi: {time.time() - start_time} sekund")
```

Wyświetlamy czas:

```
Czas obliczeń warstwy samouwagi: 0.44140100479125977 sekund
```

Następnie uruchamiamy symulację złożoności czasowej funkcji warstwy rekurencyjnej bez mnożenia macierzy. Określamy czas początkowy i końcowy:

```
# Symulacja warstwy rekurencyjnej  $O(n*d^2)$ 
start_time = time.time()
hidden_state = np.zeros(d)
for i in range(n):
    for j in range(d):
        for k in range(d):
            hidden_state[j] += input_seq[i, j] * hidden_state[k]
rt=time.time()-start_time
print(f"Czas obliczeń warstwy rekurencyjnej: {time.time() - start_time}
↳sekund")
```

W wyniku wyświetlamy czas wykonania:

```
Czas obliczeń warstwy rekurencyjnej: 71.78978061676025 sekund
```

W kolejnym kroku obliczymy procentową złożoność czasową obliczeń warstwy uwagi i złożoność czasową obliczeń warstwy rekurencyjnej. W ten sposób możemy mierzyć procent wydajności. Takie podejście pozwala uzyskać ogólny pogląd na możliwości warstw uwagi. Następnie wyświetlamy wynik:

```
# Obliczenie podsumowania

# Obliczenie procenta wartości dla warstwy uwagi (at)
percentage_at = round((at / total) * 100,2)
# Wyświetlenie wyniku
print(f"Procentowy udział 'czasu obliczeniowego uwagi' w sumie 'warstwa
↳uwagi' plus 'warstwa rekurencyjna' wynosi {percentage_at}%")
```

Z wyników widać, że warstwa uwagi charakteryzuje się mniejszą czasową złożonością obliczeniową, zatem jest bardziej wydajna:

```
Procentowy udział 'czasu obliczeniowego uwagi' w sumie 'warstwa uwagi' plus
↳'warstwa rekurencyjna' wynosi 0.61%.
```

W tej konfiguracji, z wykorzystaniem CPU, warstwa uwagi charakteryzuje się niższą złożonością obliczeniową niż warstwa rekurencyjna.

Przejdźmy teraz do układu GPU.

Czasowa złożoność obliczeniowa przy użyciu GPU

Przed uruchomieniem pierwszej komórki notatnika należy sprawdzić, czy używamy układu GPU. Aby to zrobić, przejdź do menu *Środowisko wykonawcze*, wybierz opcję *Zmień typ środowiska wykonawczego* i upewnij się, że parametr *Akcelerator sprzętowy* jest ustawiony na *GPU* (rysunek 1.4).

Zmień typ środowiska wykonawczego

Typ środowiska wykonawczego

Python 3 ▼

Akcelerator sprzętowy (?)

☐ CPU
 ☒ T4 GPU
 ☐ A100 GPU
 ☐ L4 GPU

☐ TPU v2-8

Potrzebujesz dostępu do lepszych GPU?

[Kup dodatkowe jednostki obliczeniowe](#)

Anuluj Zapisz

Rysunek 1.4. Zmiana ustawień w celu wykorzystania układu GPU

W tym przypadku zastosowany typ GPU to NVIDIA A100. Układy GPU dobrze radzą sobie z algorytmami wymagającymi wielu obliczeń, szczególnie mnożenia macierzy, takimi jak obliczenia realizowane przez mechanizmy warstwy uwagi, którą symulujemy w tym notatniku.

Aby wykorzystać możliwości układów GPU, zaimplementujemy symulację z wykorzystaniem biblioteki PyTorch:

```
# Wersja PyTorch
import torch
import time
```

Użyjemy tych samych parametrów co w przypadku wykorzystania układu GPU:

```
# Zdefiniowanie długości sekwencji i wymiarowości reprezentacji
n = 512
d = 512
```

Teraz aktywujemy GPU i definiujemy dane wejściowe:

```
# Jeśli to możliwe, użyj GPU, w przeciwnym razie korzystaj z CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
# Zdefiniuj wejście
input_seq = torch.rand(n, d, device=device)
```

Uruchamiamy tę samą symulację co dla CPU, tym razem z wykorzystaniem GPU:

```
# Symulacja warstwy samouwagi  $O(n^2 \cdot d)$ 
start_time = time.time()
_ = torch.mm(input_seq, input_seq.t())
at = time.time() - start_time
print(f"Czas obliczeń warstwy samouwagi: {at} sekund")
```

Wynik nie jest imponujący, ponieważ nie wykonywaliśmy ogromnej liczby operacji mnożenia macierzy, co pozwoliłoby w pełni wykorzystać potencjał układu GPU. Możliwości GPU stają się wyraźniejsze w przypadku bardziej złożonych obliczeń:

```
cuda
Czas obliczeń warstwy samouwagi: 2.887202501296997 sekund
```

Teraz uruchomimy funkcję warstwy rekurencyjnej, ale ograniczymy czas jej działania do około dziesięciokrotności czasu wykorzystanego przez warstwę samouwagi (w zależności od aktywności GPU). To wystarczy, by zaprezentować wyniki. Aby przerwać obliczenia, skorzystamy z instrukcji `if ct>at*10`:

```
# Symulacja warstwy rekurencyjnej  $O(n \cdot d^2)$ 
start_time = time.time()
hidden_state = torch.zeros(d, device=device)
for i in range(n):
    for j in range(d):
        for k in range(d):
            hidden_state[j] += input_seq[i, j] * hidden_state[k]
            ct = time.time() - start_time
            if ct>at*10:
                break
```

Obliczamy ograniczony czas, jaki daliśmy funkcji na działanie i go wyświetlamy:

```
rt = time.time() - start_time
print(f"Czas obliczeń warstwy rekurencyjnej: {rt} sekund")
```

Wyniki pokazują, że uzyskana wydajność nie jest zbyt wysoka:

```
Czas obliczeń warstwy rekurencyjnej: 36.3216814994812 sekund
```

Obliczamy procentowy udział warstwy uwagi w czasie całkowitym:

```
# Obliczenie sumy
total = at + rt

# Obliczenie procentowego udziału warstwy at
percentage_at = round((at / total) * 100, 2)
# Wyświetlenie wyniku
print(f"Procentowy udział czasu obliczeń warstwy samouwagi w sumie obliczeń
↳ warstwy samouwagi i warstwy rekurencyjnej wynosi {percentage_at}%")
```

```
Procentowy udział czasu obliczeń warstwy samouwagi w sumie obliczeń warstwy
↳ samouwagi i warstwy rekurencyjnej wynosi 0.01%
```

Możemy sprawdzić informacje o GPU:

```
!nvidia-smi
```

W wyniku zostaną wyświetlone informacje o bieżącym procesorze GPU (rysunek 1.5).

NVIDIA-SMI 535.104.05				Driver Version: 535.104.05				CUDA Version: 12.2			
GPU	Name			Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC		
Fan	Temp	Perf		Pwr:Usage/Cap		Memory-Usage	GPU-Util	Compute M.			
								MIG	M.		
0	Tesla T4			Off	00000000:00:04.0	Off		0			
N/A	72C	P0		32W / 70W	155MiB / 15360MiB		0%	Default			
								N/A			

Processes:											
GPU	GI	CI		PID	Type	Process name				GPU Memory	
	ID	ID								Usage	

Rysunek 1.5. Informacje o GPU

Na koniec przyjrzymy się symulacji z wykorzystaniem układu TPU.

Czasowa złożoność obliczeniowa przy użyciu TPU

Google zaprojektowało układy Cloud TPU specjalnie na potrzeby obliczeń macierzowych i sieci neuronowych. Ponieważ podstawowym zadaniem TPU jest mnożenie macierzy, nie ma sensu wykonywać operacji sekwencyjnych. Ograniczymy czas działania warstwy rekurencyjnej do czasu 10-krotnie dłuższego niż czas działania warstwy uwagi. Przed uruchomieniem pierwszej komórki notatnika należy sprawdzić, czy używamy układu TPU. Aby to zrobić, przejdź do menu Środowisko wykonawcze, wybierz opcję Zmień typ środowiska wykonawczego i upewnij się, że parametr Akcelerator sprzętowy jest ustawiony na TPU (rysunek 1.6).

W tej symulacji, aby w pełni wykorzystać zalety układu TPU, skorzystamy z biblioteki TensorFlow:

```
import tensorflow as tf
import numpy as np
import time
```

Program jest taki sam jak w przypadku oceny układów CPU i GPU, z tą różnicą, że tym razem działa na TPU z użyciem TensorFlow:

```
# Zdefiniowanie długości sekwencji i wymiarowości reprezentacji
n = 512
d = 512

# Zdefiniowanie wejść
input_seq = tf.random.normal((n, d), dtype=tf.float32)
```

Zmień typ środowiska wykonawczego

Typ środowiska wykonawczego

Python 3 ▼

Akcelerator sprzętowy ?

☐ CPU
 ☐ T4 GPU
 ☐ A100 GPU
 ☐ L4 GPU
☒ TPU v2-8

Potrzebujesz dostępu do lepszych GPU?

[Kup dodatkowe jednostki obliczeniowe](#)

Anuluj Zapisz

Rysunek 1.6. Zmiana ustawień notatnika w celu użycia TPU

Wykonujemy mnożenie macierzy i mierzymy czas trwania obliczeń:

```
# Symulacja warstwy samouwagi  $O(n^2 \cdot d)$ 
start_time = time.time()
_ = tf.matmul(input_seq, input_seq, transpose_b=True)
at = time.time() - start_time
print(f"Czas obliczeń warstwy samouwagi: {at} sekund")
```

Wynik pokazuje, że zastosowanie TPU zapewniło wysoką wydajność:

```
Czas obliczeń warstwy samouwagi: 0.04089665412902832 sekund
```

Teraz uruchomimy warstwę rekurencyjną:

```
# Symulacja warstwy rekurencyjnej  $O(n \cdot d^2)$ 
start_time = time.time()
hidden_state = np.zeros((n, d), dtype=np.float32)
for i in range(n):
    for j in range(d):
        for k in range(d):
            hidden_state[i, j] += input_seq[i, j].numpy() * hidden_
                ↳ state[min(i,k),j]
            ct = time.time() - start_time
            if ct>at*10:
                break
```

Wynik nie wskazuje na wysoką wydajność obliczeń, co jest normalne:

```
rt = time.time() - start_time
print(f"Czas obliczeń warstwy rekurencyjnej: {rt} sekund")
```

```
Czas obliczeń warstwy rekurencyjnej: 56.08650851249695 sekund
```

Teraz obliczymy procent całkowitego czasu:

```
# Obliczenie sumy
total = at + rt

# Obliczenie procentowego udziału warstwy at
percentage_at = round((at / total) * 100, 2)

# Wyświetlenie wyniku
print(f"Procentowy udział czasu obliczeń warstwy samouwagi w sumie obliczeń
↳warstwy samouwagi i warstwy rekurencyjnej wynosi {percentage_at}%"):
```

Wyniki potwierdzają, że warstwy uwagi gwarantują wyższą wydajność:

```
Procentowy udział czasu obliczeń warstwy samouwagi w sumie obliczeń warstwy
↳samouwagi i warstwy rekurencyjnej wynosi 0.07%
```

W kolejnym punkcie przeniesiemy symulację TPU do świata modeli LLM.

TPU-LLM

Przeprowadzimy teraz taką samą symulację jak dla TPU z wykorzystaniem dość typowych parametrów dla warstwy uwagi. Wymiarowość modelu LLM może wynosić $d=12288$. Limit liczby wejść dla modelu LLM może wynosić $n=32728$. Powyższe wartości są jedynie przykładem uruchomienia warstwy rekurencyjnej z wykorzystaniem układu TPU. Wartości są takie same jak w przypadku modelu LLM:

```
import tensorflow as tf
import numpy as np
import time

# Definicja długości sekwencji i wymiarowości reprezentacji
n = 32768
d = 12288

# Definicja wejść
input_seq = tf.random.normal((n, d), dtype=tf.float32)
```

Spróbujmy uruchomić tę samą funkcję co wcześniej i wyświetlić czas jej wykonania:

```
# Symulacja warstwy samouwagi  $O(n^2*d)$ 
start_time = time.time()
_ = tf.matmul(input_seq, input_seq, transpose_b=True)

at = time.time() - start_time
print(f"Czas obliczeń warstwy samouwagi: {at} sekund")
```

Czas wskazuje na bardzo wysoką wydajność:

```
Czas obliczeń warstwy samouwagi: 6.904115200042725 sekund
```

Możemy też wyświetlić informacje na temat TPU:

```
import os
from tensorflow.python.profiler import profiler_client
```

```
tpu_profile_service_address = os.environ['COLAB_TPU_ADDR'].replace('8470',  
↳ '8466')  
print(profiler_client.monitor(tpu_profile_service_address, 100, 2))
```

Możemy zauważyć, że jednostki TPU Matrix były rzadko wykorzystywane:

```
Timestamp: 10:20:30  
TPU type: TPU v2  
Utilization of TPU Matrix Units (higher is better): 0.000%
```

Wnioski z tej oceny można sformułować w następujących punktach:

- Dzięki wyeliminowaniu rekurencyjności wydajność warstwy uwagi jest wyższa niż wydajność warstwy rekurencyjnej.
- Analiza słów jeden do jednego w warstwie uwagi pozwala łatwiej wykryć długoterminowe zależności.
- Architektura warstwy uwagi umożliwia mnożenie macierzy, co pozwala w pełni wykorzystać potencjał nowoczesnych układów GPU i TPU.
- Dzięki wykorzystaniu możliwości układów graficznych (GPU) (a także układów TPU) oraz warstwy uwagi algorytmy mogą przetwarzać więcej danych i przyswajać więcej informacji.

Zaskakująco prosta złożoność $O(1)$, która doprowadziła do złożoności $O(n^2 \cdot d)$, zmieniła codzienne życie na zawsze i pozwoliła przenieść sztuczną inteligencję na wyższy poziom.

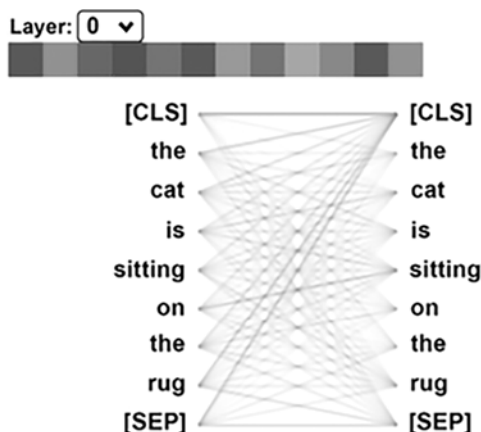
Jak to się stało?

Krótką podróż od rekurencji do uwagi

W sekwencyjnych modelach przetwarzania języka naturalnego **rekurencyjne sieci neuronowe** (ang. *Recurrent Neural Networks* — **RNN**), w tym modele LSTM, są stosowane od dziesięcioleci. Jednak w przypadku długich sekwencji i dużej liczby parametrów rekurencja osiąga swoje granice. Dlatego obecnie dominują najnowocześniejsze modele oparte na transformerach.

W tym punkcie krótko omówię wprowadzenie do przetwarzania języka naturalnego (ang. *Natural Language Processing* — **NLP**). W szczególności zwrócę uwagę na działania, które doprowadziły do powstania transformerów. Te ostatnie opiszę bardziej szczegółowo w rozdziale 2. „Wprowadzenie do architektury modelu transformera”. Najpierw jednak przyjrzymy się głowicy uwagi transformera, która zastąpiła warstwy RNN sieci neuronowej modeli NLP.

Podstawowe działanie transformera można w skrócie podsumować jako „mieszanie tokenów”. Modele NLP najpierw konwertują sekwencje słów na tokeny. Sieci RNN analizują tokeny w funkcjach rekurencyjnych. Transformery nie analizują tokenów w sekwencjach, lecz obliczają związek każdego tokena z innymi tokenami w sekwencji, tak jak pokazałem na rysunku 1.7.



Rysunek 1.7. Głowica uwagi warstwy transformera

Funkcję uwagi omówię szczegółowo w rozdziale 2. Wniosek z rysunku 1.7 na ten moment jest taki, że w przypadku innowacyjnych modeli LLM dla *każdego słowa (tokena) sekwencji jest obliczany związek ze wszystkimi innymi słowami sekwencji*.

Przyjrzyjmy się z grubsza historii transformerów.

Krótką historia transformerów

Nad wzorcami sekwencji i modelowaniem języka naturalnego pracowało wiele wybitnych umysłów. W rezultacie maszyny stopniowo uczyły się przewidywać prawdopodobne sekwencje słów. Aby wymienić wszystkie najważniejsze osoby, które się do tego przyczyniły potrzeba by całej książki.

W tym punkcie podzielę się pracami kilku moich ulubionych badaczy, którzy uTOROWALI drogę sztucznej inteligencji do miejsca, w jakim jest dziś, i którzy położyli podwaliny pod nadejście transformerów.

Pod koniec XIX i na początku XX wieku Andriej Markow wprowadził koncepcję wartości losowych i stworzył teorię procesów stochastycznych (zobacz podrozdział „Lektura uzupełniająca”). W dziedzinie sztucznej inteligencji znamy je jako **proces decyzyjny Markowa** (ang. *Markov Decision Process* — **MDP**), **łańcuchy Markowa** i **procesy Markowa**. Na początku XX wieku Markow wykazał, że można przewidzieć następny element łańcucha, wykorzystując jedynie najnowsze elementy tego łańcucha. Zastosował tę teorię do liter i przeprowadził wiele eksperymentów. Należy wziąć pod uwagę, że choć nie miał komputera, udowodnił teorię, która w dziedzinie sztucznej inteligencji jest stosowana do dziś.

W 1948 roku Claude Shannon opublikował książkę *The Mathematical Theory of Communication*. Opisał w niej podwaliny modelu komunikacji opartego na koderze źródłowym, nadajniku i odbiorniku, zwanym również dekoderem semantycznym. Stworzył teorię informacji w takiej postaci, jaką znamy dzisiaj. Oczywiście Shannon wspominał o teoriach Markowa.

W 1950 roku Alan Turing opublikował swój przełomowy artykuł: *Computing Machinery and Intelligence*. Turing oparł swój artykuł na temat inteligencji maszyn na przykładzie udanej maszyny, która pozwoliła odszyfrowywać niemieckie wiadomości w czasie II wojny światowej. Wiadomości składały się z ciągów słów i liczb.

W 1954 roku w ramach eksperymentu Georgetown-IBM wykorzystano komputery do tłumaczenia zdań z języka rosyjskiego na angielski z zastosowaniem systemu reguł. System reguł to program, który uruchamiał listę reguł analizujących struktury danych. Systemy reguł nadal istnieją i są w powszechnym użyciu. Jednak w niektórych przypadkach, dzięki automatycznemu uczeniu się wzorców, sztuczna inteligencja może zastąpić listy reguł dla miliardów kombinacji językowych.

Terminu *AI* po raz pierwszy użył John McCarthy, w 1956 roku, przy okazji badań nad możliwością uczenia się maszyn.

W 1982 roku John Hopfield wprowadził pojęcie **RNN** znanych jako sieci Hopfielda lub „asocjacyjne” sieci neuronowe. Inspiracją dla Hopfielda była książka W.A. Little’a z 1974 roku zatytułowana *The Existence of Persistent States in the Brain*. Książka ta na całe dekady stworzyła teoretyczne podstawy procesów uczenia się. Sieci RNN ewoluowały i w wyniku tej ewolucji powstały sieci LSTM w formie, jaką znamy dzisiaj.

RNN skutecznie zapamiętuje trwałe stany sekwencji (rysunek 1.8).



Rysunek 1.8. Działanie sieci RNN

Każdy stan S_n przechwytyuje informację S_{n-1} . Po dotarciu do końca sieci funkcja F wykonuje działanie: transdukcję, modelowanie lub dowolny inny typ zadania opartego na sekwencji.

W latach 80. Yann LeCun zaprojektował wielofunkcyjną **konwolucyjną sieć neuronową** (ang. *Convolutional Neural Network* — **CNN**). Zastosował sieci CNN do sekwencji tekstowych, a także do transdukcji i modelowania sekwencji. Sieci CNN również opierają się na opisanych przez W.A. Little’a trwałych stanach, które przetwarzają informacje warstwa po warstwie. W latach 90. ubiegłego wieku wynikiem kilku lat pracy Yanna LeCuna był system LeNet-5, który doprowadził do powstania wielu modeli CNN w postaci, jaką znamy dzisiaj. Jednakże wydajna w wielu zastosowaniach architektura CNN napotyka problemy przy obsłudze długoterminowych zależności w długich i złożonych sekwencjach.

Mógłbym wymienić wiele innych znakomitych nazwisk, prac naukowych i modeli, które mogłyby zawstydić każdego specjalistę zajmującego się sztuczną inteligencją. Wszyscy w branży AI przez wszystkie te lata byli na dobrej drodze. Pola Markowa, sieci RNN i CNN wyewoluowały w wiele innych modeli. Pojawiło się pojęcie uwagi: uwzględniania innych tokenów w sekwencji, a nie tylko ostatniego. Warstwę tę wprowadzono do modeli RNN i CNN.

Następnie, gdy modele sztucznej inteligencji musiały analizować dłuższe sekwencje wymagające coraz większej mocy obliczeniowej, twórcy sztucznej inteligencji zaczęli używać bardziej wydajnych maszyn i znaleźli sposoby na optymalizację gradientów.

Przeprowadzano również badania nad modelami sekwencja – sekwencja, ale nie spełniły one oczekiwań. Wydawało się, że nie da się już niczego zrobić, aby osiągnąć większy postęp, i tak minęło 30 lat. Wprowadzono warstwę uwagi, ale modele nadal opierały się na rekurencji. I wreszcie, pod koniec 2017 roku, pojawił się wyposażony w podwarstwy głowic uwagi i wiele innych mechanizmów nowoczesny transformer. Sieci neuronowe RNN nie są już konieczne do modelowania sekwencji.

Zanim zagłębię się w architekturę oryginalnego transformera, co nastąpi w rozdziale 2. „Wprowadzenie do architektury modelu transformera”, rozpocznę od ogólnego opisu zmiany paradygmatu wywołanej przez jeden, niepozorny token.

Od jednego tokena do rewolucji w dziedzinie sztucznej inteligencji

Jak wkrótce się przekonasz, nie ma tu pomyłki — tytuł jest poprawny. Rewolucję w dziedzinie sztucznej inteligencji wywołał jeden token, który otworzył drzwi do sztucznej inteligencji w każdej dziedzinie i w każdym zastosowaniu.

ChatGPT z GPT-4, PaLM 2 i inne modele LLM charakteryzują się unikatowym sposobem generowania tekstu.

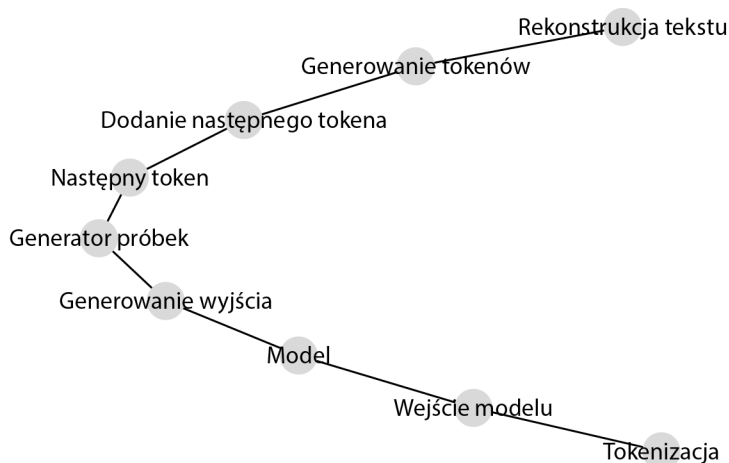
W modelach LLM token stanowi minimalną część słowa. *Token to pojęcie, od którego model LLM zaczyna się i na którym się kończy.*

Na przykład słowo włączając mogłoby składać się z dwóch tokenów: włączaj + ąc. Modele GPT prognozują tokeny na podstawie setek miliardów tokenów znajdujących się w zbiorze danych szkoleniowych. Przyjrzyjmy się pokazanemu na rysunku 1.9 wykresowi przedstawiającemu model GPT OpenAI, który wyciąga wnioski w celu wygenerowania tokena.

Dość zaskakująco jedynymi elementami tego grafu zarządzanymi przez model są Model i Generowanie wyjścia. W tych fazach generowane są nieprzetworzone logity. Za całą resztę odpowiada potok.

Aby zrozumieć działanie potoku, przyjrzyjmy się opisowi następujących kroków:

- 1. Tokenizacja.** Potok za pomocą tokenizera konwertuje sekwencję wejściową, np. „Czy słońce znajduje się w Układzie Słonecznym?”, na tokeny. Kluczowej roli tokenizerów w modelach transformerów przyjrzymy się bliżej w rozdziale 10. „Badanie roli tokenizerów w kształtowaniu modeli transformerów”.
- 2. Wejście modelu.** W kolejnym kroku potok przekazuje podzieloną na tokeny sekwencję do przeszkolonego modelu GPT.



Rysunek 1.9. Graf wnioskowania GPT zbudowany w Pythonie z wykorzystaniem pakietu NetworkX

3. **Model.** Model przetwarza wejście w różnych warstwach — od warstwy wejściowej, poprzez wiele warstw transformerów, aż do warstwy wyjściowej. Tę architekturę szczegółowo opisałem w rozdziale 2. „Wprowadzenie do architektury modelu transformera”.
4. **Generowanie wyjścia.** Na podstawie sekwencji wejściowej model generuje nieprzetworzone logity wyjściowe.
5. **Generator próbek** (ang. *sampler*). Generator próbek przekształca logity na prawdopodobieństwa. W rozdziale 7. „ChatGPT — rewolucja w generatywnej sztucznej inteligencji” wprowadzę pojęcie hiperparametrów, które wpływają na wynik modelu. Więcej informacji na temat procesu próbkowania obejmującego główne hiperparametry można znaleźć w podrozdziale „Interfejs Vertex AI PaLM 2” rozdziału 14. „Najnowocześniejsze modele LLM Vertex AI i PaLM 2”.
6. **Wybór następnego tokena (następny token).** Na podstawie prawdopodobieństwa generatora próbek wybierany jest następny token.
7. **Dodawanie następnego tokena.** Wybrany następny token jest dodawany do sekwencji wejściowej (w postaci tokena) i proces powtarza się od kroku 3. aż do osiągnięcia maksymalnego limitu tokenów.
8. **Zakończenie generowania tokenów (generowanie tekstu).** Generowanie tekstu zakończy się po osiągnięciu maksymalnego limitu tokenów lub wykryciu tokena końca sekwencji.
9. **Rekonstrukcja tekstu.** Tokenizer konwertuje końcową sekwencję tokenów z powrotem na ciąg tekstowy. Ten krok obejmuje ponowne scalenie wszystkich tokenów reprezentujących części słów w celu utworzenia całych słów.

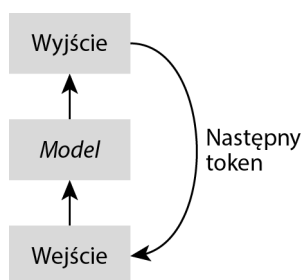
Magia się powtarza! Model pobiera sekwencję danych wejściowych i generuje jeden token.

Ten token jest dodawany do sekwencji i model zaczyna pracę od początku w celu wygenerowania kolejnego tokena.

Kroki opisanego procesu można podsumować następująco:

1. Wejście = sekwencja wejść wyrażona za pomocą tokenów.
2. Model przetwarza dane wejściowe.
3. Wyjście = jeśli zostanie osiągnięta maksymalna liczba żądanych tokenów, kolejny token jest dodawany na końcu danych wejściowych.

Omówiony powyżej proces — generowanie wyjścia modelu w postaci jednego tokena — okazał się przełomowy (rysunek 1.10).



Rysunek 1.10. Model przetwarza sekwencję wejść, generuje nieprzetworzone dane wyjściowe i wybiera jeden token

Jedno przejście do przodu można zapisać za pomocą następującej notacji:

$$t = f(n)$$

Oto objaśnienia poszczególnych elementów powyższego wzoru:

- t = jeden token.
- f = model i kontroler wyjścia, który wnioskuje następny token.
- n = początkowa sekwencja tokenów + każdy kolejny token, który jest do niej dodawany aż do osiągnięcia maksymalnej liczby tokenów lub wykrycia tokena końca sekwencji.

Powyższy proces określiliśmy jako $f(n)$. Pod maską oznaczenia $f(n)$ kryje się proces dodawania kolejnych prognozowanych tokenów. Funkcję $f(n)$ można zinterpretować następująco:

$$t_{-i} = f(t_{-1}, t_{-2}, \dots, t_{-i-1})$$

Wyobraź sobie: jeden token! A jednak przyczynił się do zmian niemal w każdej dziedzinie: tłumaczenia, podsumowania, odpowiadanie na pytania, generowanie obrazów, wideo... lista wydłuża się z dnia na dzień.

W kolejnym punkcie przejdziemy od jednego tokena do całości!

Od jednego tokena do całości

W poprzednim punkcie pokazałem, że model generatywny można podsumować wzorem $t = f(n)$. Model przetwarza n i generuje t , po jednym tokenie.

Konsekwencje są niepojęte i niezwykle.

Łatwo zauważyć, że jednotokenowe podejście, podsumowane wzorem $t = f(n)$, ma następujące skutki:

- *Dynamiczna* natura modelu transformera. Model dostosowuje swoje wyjścia na podstawie przyrostowych danych wejściowych reprezentowanych w następujący sposób:

$$t_i = f(t_1, t_2, \dots, t_{i-1})$$

- Model dostosuje się i wygeneruje wyniki przy użyciu zupełnie nowych danych wejściowych!
- *Ukryta* natura modelu transformera. Model koduje związki między tokenami i zapisuje je w postaci wag i odchyleń. Nie ma wyraźnych wytycznych. Po prostu ciągle produkuje tokeny na podstawie dynamicznych wejść opartych na milionach próbek danych tekstowych, graficznych i audio!
- Niesamowita *elastyczność* systemu wynika z jego dynamicznych i niejawnych właściwości. Model transformera podobny do GPT może wnioskować sensowne wyniki dla szerokiego zakresu zróżnicowanych kontekstowo danych wejściowych.

Oto implikacje podejścia modeli opartych na jednym tokenie z różnych perspektyw:

- **Nadzorowane i nienadzorowane**

Niektórzy mogliby powiedzieć, że model serii GPT, taki jak ChatGPT, jest poddawany szkoleniu nienadzorowanemu. To stwierdzenie jest prawdziwe tylko do pewnego stopnia. Model podobny do GPT uzyskuje dokładność, token po tokenie, za pośrednictwem uczenia samonadzorowanego, podczas którego prognozuje każdy kolejny token na podstawie poprzednich tokenów w sekwencji. Osiąga to poprzez oddziaływanie na reprezentacje wszystkich pozostałych tokenów w sekwencji.

Model GPT można również dostroić poprzez dodanie do wejścia (zapytania) i wyjścia (uzupełnienia) etykiet! Możemy dostarczyć tysiące wejść (zapytań) generujących jako wyjście (uzupełnienie) jeden token. Na przykład możemy utworzyć jako dane wejściowe tysiące pytań, których wyniki będą zawierać tylko odpowiedzi „prawda” lub „fałsz”. Na tym polega niejawne uczenie nadzorowane. Ponadto model jawnie nie zapamiętuje poprawnych prognoz, lecz po prostu uczy się wzorców tokenów.

- **Dyskryminatywne i generatywne**

Od czasu pojawienia się ChatGPT do opisu modeli podobnych do GPT często jest używany termin „sztuczna inteligencja generatywna”. Należy zauważyć, że modele GPT nie były pierwszymi, które można było klasyfikować jako

generatywną sztuczną inteligencję — rekurencyjne sieci neuronowe również były generatywne.

Ponadto twierdzenie, że modele podobne do GPT wykonują zadania generatywnej sztucznej inteligencji, z punktu widzenia wiedzy naukowej nie jest do końca prawdą. W poprzednim punkcie, „Nadzorowane i nienadzorowane”, wspominałem, że model „generatywny” może wykonywać nadzorowane zadanie wnioskowania o wynikach takich jak „prawda” lub „fałsz”, gdy danymi wejściowymi są stwierdzenia lub pytania. To nie jest zadanie dla generatywnej sztucznej inteligencji! To jest zadanie dyskryminatywnej sztucznej inteligencji.

Inny problem dotyczy zadań podsumowywania. Częścią podsumowania jest wyszukiwanie słów kluczowych, tematów i nazw. Inna część, gdy system wnioskuje nowe tokeny w celu tworzenia streszczenia tekstu, może mieć charakter generatywny.

Pełne możliwości natury autoregresyjnej (generowanie tokena i dodawanie go do danych wejściowych) w modelowaniu języka (dodawanie nowego tokena na końcu sekwencji) są osiąmane wtedy, gdy model LLM wymyśla historię — to jest generatywna sztuczna inteligencja.

W zależności od zadania, jakie ma wykonać model podobny do GPT, może być on dyskryminatywny, generatywny lub zarówno dyskryminatywny, jak i generatywny.

■ Modele ukierunkowane na specyficzne zadania i zadania szczegółowe

Modele ukierunkowane na konkretne zadania są często przeciwstawiane modelom ogólnego przeznaczenia, takim jak LLM podobne do GPT. Modele ukierunkowane na określone zadanie są szkolone w celu uzyskania bardzo dobrego wykonania takiego zadania. W przypadku niektórych zadań do pewnego stopnia jest to prawda. Jednakże modele LLM przeszkolone pod kątem wykonywania zadań ogólnego przeznaczenia zaskakująco dobrze radzą sobie ze specyficznymi zadaniami, co zostało udowodnione w ocenach uzyskanych na różnych egzaminach (np. prawniczych i medycznych). Ponadto modele transformerów potrafią teraz rozpoznawać i generować obrazy.

■ Interakcja i automatyzacja

Kluczowym punktem w analizie relacji między ludźmi a modelami LLM jest automatyzacja. W jakim stopniu powinniśmy automatyzować wykonywane zadania? Niektórych zadań nie da się zautomatyzować za pomocą modeli LLM. ChatGPT i inne modele spełniające funkcje asystentów nie mogą podejmować decyzji dotyczących życia i śmierci, moralności, etyki ani biznesu. Żądanie od systemu wykonania takiego działania naraża organizację na niebezpieczeństwo.

Przed wdrożeniem modeli LLM należy dokładnie przeanalizować automatyzację.

W kolejnych rozdziałach tej książki zapoznasz się z zadaniami nadzorowanymi, samonadzorowanymi, nienadzorowanymi, dyskryminatywnymi, generatywnymi, zadaniami

specyficznymi dla konkretnej dziedziny, zadaniami szczegółowymi oraz interakcjami z rozwiniętą sztuczną inteligencją i systemami automatycznymi. Czasami wszystko to można znaleźć w architekturze modelu LLM ogólnego przeznaczenia!

Przygotuj się na zapoznanie się z różnymi podejściami!

Modele podstawowe

Zaawansowane, rozbudowane, wielofunkcyjne modele transformerów stanowią tak dużą zmianę paradygmatu, że do ich opisania potrzebna jest nowa nazwa: **modele podstawowe** (ang. *Foundation Models*). W związku z tym na Uniwersytecie Stanforda powstało **Centrum Badań nad Modelami Podstawowymi** (ang. *Center for Research on Foundation Models* — **CRFM**). W sierpniu 2021 roku CRFM opublikowało dwustronicowy artykuł (zobacz podrozdział „Oдноśniki”), którego autorami było ponad stu naukowców i specjalistów: *On the Opportunities and Risks of Foundation Models*.

Modele podstawowe nie powstały w środowisku akademickim, lecz w wielkiej branży technologicznej. Google opracował model transformera, który doprowadził do powstania takich programów jak Google BERT, LaMBDA, PaLM 2 i inne. Microsoft nawiązał współpracę z OpenAI, by stworzyć ChatGPT z GPT-4, a wkrótce także inne modele.

Aby sprostać wykładniczemu wzrostowi petabajtów danych napływających do centrów danych wielkich firm technologicznych, powstała potrzeba znalezienia lepszego modelu. Transformery narodziły się więc z konieczności.

Aby zrozumieć potrzebę powstania wykorzystywanych w branży modeli AI, przyjrzyjmy się ewolucji LLM.

Transformery mają dwie charakterystyczne cechy: wysoki stopień homogenizacji i niezwykle zdolności emergencji (czyli wyłaniania się nowych właściwości). Dzięki *homogenizacji* jeden model może być wykorzystywany do realizacji wielu różnych zadań. Tego typu możliwości *wyłaniają się* w wyniku szkolenia modeli składających się z miliardów parametrów na superkomputerach.

Obecny ekosystem modeli transformerów różni się od wszystkich innych ewolucji w dziedzinie sztucznej inteligencji i można go podsumować za pomocą następujących czterech właściwości:

- **Architektura modelu.** Model ma charakter branżowy. Warstwy modelu są identyczne i zostały specjalnie zaprojektowane do przetwarzania równoległego. Szczegółowy opis architektury transformerów znajdziesz w rozdziale 2. „Wprowadzenie do architektury modelu transformera”.
- **Dane.** Wielkie firmy technologiczne dysponują największym źródłem danych w historii ludzkości, generowanym głównie przez ludzkie działania i interakcje w sieci. Obejmują one nawyki przeglądania stron internetowych, zapytania wpisywane w wyszukiwarkach, posty w mediach społecznościowych i zakupy online.

- **Moc obliczeniowa.** Wielkie firmy technologiczne dysponują mocą obliczeniową na niespotykaną dotąd skalę. Na przykład model GPT-3 został przeszkolony z wydajnością około 50 **PetaFLOPS** (operacji zmiennoprzecinkowych na sekundę), a Google dysponuje obecnie superkomputerami specyficznymi dla dziedziny, których wydajność przekracza 80 PetaFLOPS. Ponadto GPT-4, PaLM 2 i inne modele LLM wykorzystują do szkolenia tysiące układów GPU.
- **Inżynieria zapytań.** Przeszkolonym transformerom można zlecić wykonanie zadania za pomocą zapytania (ang. *prompt*). Zapytanie wprowadza się w języku naturalnym. Jednakże użyte słowa wymagają pewnej struktury, co sprawia, że zapytania stają się metajęzykiem.

Model podstawowy jest zatem modelem transformera, który przeszkolono na superkomputerach z wykorzystaniem miliardów rekordów danych i miliardów parametrów. Taki model może wykonywać zadania z szerokiego zakresu bez konieczności dalszego dostrajania. Dlatego skala modeli podstawowych jest wyjątkowa. W pełni przeszkolone modele są często nazywane silnikami. GPT-4, Google BERT, PaLM 2 i wiele modeli transformerów można teraz uznać za modele podstawowe. Bommasani i współpracownicy (2023), aby śledzić rosnącą liczbę zasobów na rynku (zestawów danych, modeli i aplikacji), stworzyli tzw. grafy ekosystemów (ang. *ecosystem graphs*).

W kolejnym punkcie zaprezentuję przykład działania modeli podstawowych i zmian, jakie zaszły w sposobie tworzenia programów.

Od zadań ogólnych do zadań specjalistycznych

W tym punkcie pokażę, jak sztuczna inteligencja oparta na transformerach na zawsze zmieniła dotychczasowy sposób pracy. Wykorzystam ChatGPT, który automatycznie zaprojektuje, napisze i wyjaśni program klasyfikacyjny. W tym rozdziale, w kontekście konkretnego zadania, które zlecimy programowi ChatGPT, zaprezentuję zadania dyskryminatywne, generatywne, zadania specyficzne dla dziedziny oraz zadania ogólne modeli transformerów.

Nie daj się zwieść! Sztuczna inteligencja oparta na transformerach jest na wczesnym etapie rozwoju. Jednak w miarę rozwoju modeli i pojawiania się nowych sztuczna inteligencja będzie wykonywać coraz bardziej złożone zadania.

Zacznijmy od przyjrzenia się ChatGPT, który pisze i wyjaśnia algorytm klasyfikacji!

Otwórz plik *ChatGPT_Plus_writes_and_explains_AL.ipynb* w katalogu z przykładami do tego rozdziału w repozytorium GitHub tej książki.

Do ChatGPT Plus przekazałem dwa polecenia (prompty):

1. Stwórz klasyfikację zbioru danych Iris z wykorzystaniem biblioteki `sklearn`; przedstaw wyniki za pomocą wykresu w `matplotlib`; nie używaj API OpenAI.
2. Teraz napisz szczegółowe objaśnienie tego kodu dla początkującego programisty w Pythonie.

Cały ten notatnik wraz z objaśnieniami napisał ChatGPT, korzystając z modelu GPT-3.5 Turbo.

Jedyną ingerencją człowieka było utworzenie notatnika oraz skopiowanie tekstu i kodu dostarczonego przez ChatGPT. Najpierw zamieszczę objaśnienia programu:

Ten program to prosty projekt uczenia maszynowego, który pokazuje, jak użyć

- ↳ scikit-learn – popularnej biblioteki uczenia maszynowego w Pythonie –
- ↳ do stworzenia klasyfikatora drzewa decyzyjnego dla zbioru danych Iris.
- ↳ Zbiór danych Iris to słynny zbiór danych znany w społeczności uczenia
- ↳ maszynowego i często używany jako benchmark dla algorytmów
- ↳ klasyfikacyjnych.

W tym programie najpierw załadujesz zbiór danych Iris i podzielisz go

- ↳ na zbiór szkoleniowy i testowy. Następnie utworzysz model klasyfikatora
- ↳ drzewa decyzyjnego za pomocą klasy DecisionTreeClassifier z biblioteki
- ↳ scikit-learn i przeskolisz go na danych szkoleniowych. Potem ocenisz
- ↳ dokładność modelu na zbiorze testowym i obliczysz macierz pomyłek, która
- ↳ jest tabelą podsumowującą wydajność modelu. Na koniec wyświetlisz macierz
- ↳ pomyłek za pomocą Matplotlib.

Ten program to świetny sposób dla początkujących, aby nauczyć się uczenia

- ↳ maszynowego oraz korzystania z biblioteki scikit-learn. Dzięki analizie
- ↳ kodu i objaśnień zdobędziesz podstawową wiedzę o tym, jak korzystać
- ↳ z biblioteki scikit-learn w celu budowania i oceny modeli uczenia
- ↳ maszynowego. Nauczysz się również, jak używać Matplotlib do wizualizacji
- ↳ wyników swoich modeli.

Ogólnie rzecz biorąc, ten program to zabawny i edukacyjny sposób na naukę

- ↳ uczenia maszynowego i Pythona. Niezależnie od tego, czy jesteś
- ↳ początkującym, czy doświadczonym programistą, znajdziesz w tym programie
- ↳ coś wartościowego. Zatem zacznijmy i dowiedzmy się, jak zbudować
- ↳ klasyfikator drzewa decyzyjnego dla zbioru danych Iris!¹

ChatGPT wyjaśnia podstawową konfigurację programu klasyfikacyjnego. Na początku trudno w to uwierzyć, ale przyzwyczaisz się!

Instalacja scikit-learn:
pip install -U scikit-learn

1.Importowanie bibliotek

W tym kroku importujemy kilka bibliotek, z których będziemy korzystać

- ↳ w kodzie. NumPy i Pandas są używane do operowania danymi, Matplotlib
- ↳ do wizualizacji danych, a Scikit-learn do uczenia maszynowego. Importujemy
- ↳ także konkretne funkcje, takie jak load_iris, train_test_split,
- ↳ DecisionTreeClassifier, confusion_matrix, oraz plot_tree z odpowiednich
- ↳ modułów Scikit-learn.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

¹ Zamieszczone w książce wyniki są tłumaczeniami uzyskanych przez Autora odpowiedzi ChatGPT — przyp. tłum.

```

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.tree import plot_tree

```

Następnie ChatGPT ładuje zbiór danych i dzieli go na zestaw szkoleniowy i testowy:

2. Załadowanie zbioru danych Iris:

```
iris = load_iris()
```

Używamy funkcji `load_iris` z modułu `datasets` w `Scikit-learn`, aby załadować
 ↳zbiór danych Iris do zmiennej o nazwie `iris`.

3. Podział zbioru danych na zestawy szkoleniowy i testowy:

```
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
↳test_size=0.2, random_state=42)
```

Używamy funkcji `train_test_split` z `Scikit-learn`, aby podzielić zbiór danych
 ↳na zestawy szkoleniowy i testowy. Przekazujemy `iris.data` jako cechy
 ↳i `iris.target` jako etykiety. Ustawiamy wielkość zestawu testowego na 20%,
 ↳co oznacza, że 20% danych zostanie użytych do testowania, a 80%
 ↳do szkolenia. Ustawiamy również `random_state` na 42, aby wyniki były
 ↳odtwarzalne.

W tym momencie, gdy patrzyłem, jak ChatGPT wypisuje wszystko to, co do tej pory napisał, i wciąż gorączkowo pisze, zacząłem się zastanawiać, co ja tu właściwie robię. Jaka była moja rola w tym wszystkim? Czy sztuczna inteligencja zabierze mi pracę? Czy zajmie moje stanowisko programisty? Chociaż zdaję sobie sprawę z licznych ograniczeń sztucznej inteligencji generatywnej, AI nadal mnie zachwyca.

Teraz ChatGPT konfiguruje, wyświetla i szkoli model klasyfikatora drzewa decyzyjnego (rysunek 1.11):

4. Szkolenie modelu klasyfikatora drzewa decyzyjnego:

```
model = DecisionTreeClassifier(random_state=42)
```

```
model.fit(X_train, y_train)
```

Tworzymy obiekt `DecisionTreeClassifier` z modułu `tree` biblioteki `Scikit-learn`
 ↳i ustawiamy stan losowy na 42. Następnie szkolimy model na danych
 ↳szkoleniowych przy użyciu metody `fit`.

Co teraz? ChatGPT stosuje nawet standardowe procedury uczenia maszynowego, ocenia model, a następnie oblicza macierz pomyłek (rysunek 1.12) i ją wykreśla!

5. Ocena modelu na zestawie testowym:

```
score = model.score(X_test, y_test)
```

```
print(f"Dokładność: {score:.2f}")
```

Dokładność 1.00

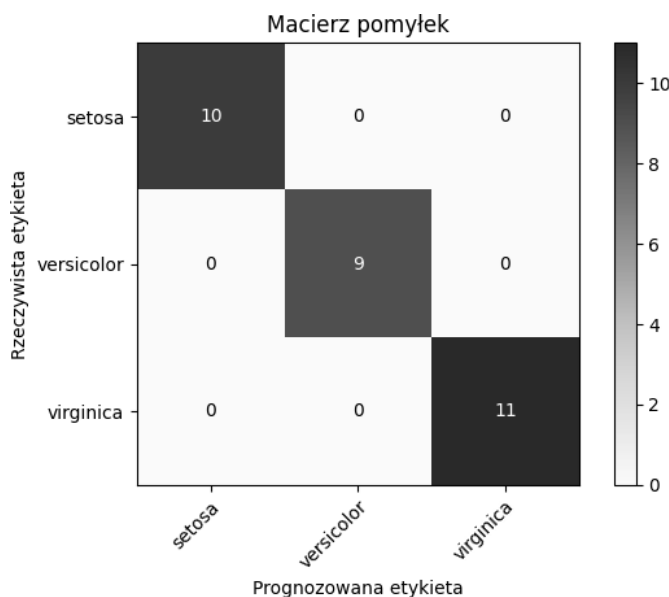
Do oceny dokładności modelu na zestawie testowym używamy metody `score`
 ↳przeszkolonego modelu. Następnie wypisujemy dokładność na konsoli.

```

▼ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)

```

Rysunek 1.11. ChatGPT tworzy klasyfikator drzewa decyzyjnego



Rysunek 1.12. ChatGPT tworzy macierz pomyłek wyników modelu

6. Obliczanie macierzy pomyłek:

```

y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)

```

7.1. Wykreślenie macierzy pomyłek:

```

fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation="nearest", cmap=plt.cm.Blues)
ax.figure.colorbar(im, ax=ax)
ax.set(xticks=np.arange(cm.shape[1]),
       yticks=np.arange(cm.shape[0]),
       xticklabels=iris.target_names, yticklabels=iris.target_names,
       title="Macierz pomyłek",
       xlabel="Prognoszowana etykieta",
       ylabel="Rzeczywista etykieta")
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
        ↳rotation_mode="anchor")
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j, i, format(cm[i, j], "d"), ha="center", va="center",
              color="white" if cm[i, j] > cm.max() / 2 else "black")
↳fig.tight_layout()

```

Na koniec ChatGPT tworzy drzewo decyzyjne (rysunek 1.13):

7.2. Wykreślenie drzewa decyzyjnego:

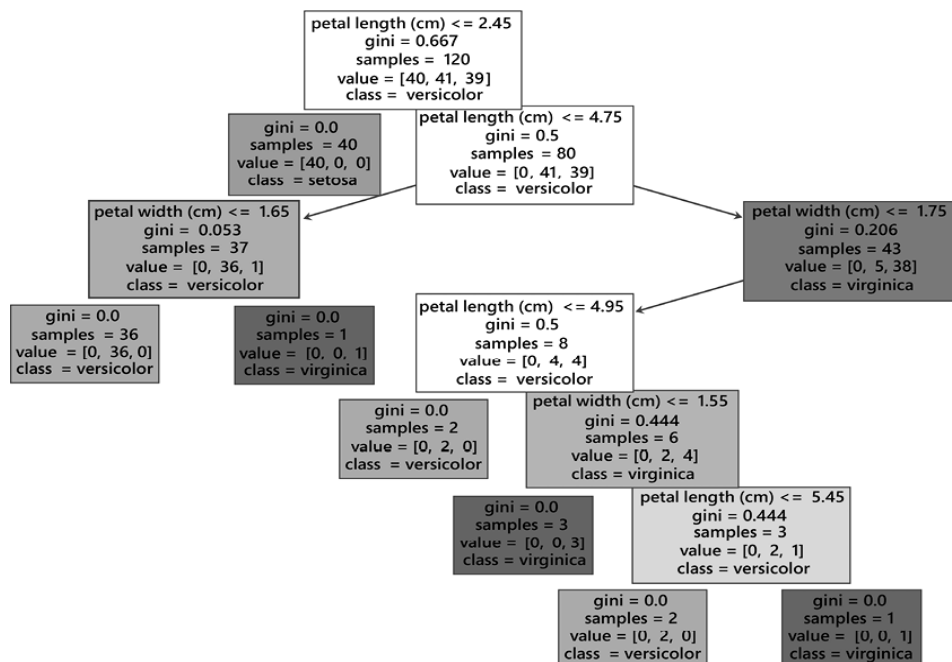
Wykreślenie drzewa decyzyjnego

```
plt.figure(figsize=(20,10))
```

```
plot_tree(model, feature_names=iris.feature_names, class_names=iris.target_
```

```
names, filled=True)
```

```
plt.show()
```



Rysunek 1.13. Fragment drzewa decyzyjnego wykreślonego przez ChatGPT

Weź głęboki oddech i pozwól, aby to, co właśnie zobaczyłeś, do Ciebie dotarło. Program sztucznej inteligencji samodzielnie wykonał 100% pracy. Więc oczywiście pierwszą reakcją mogłoby być: *Ojej, sztuczna inteligencja zabierze mi pracę! Generatywna sztuczna inteligencja z pewnością zwiększy swój zasięg, jeśli złożone problemy będzie można rozbić na małe elementy!*

Drugą reakcją może być: *Hm, będę testować zaproponowane przez ChatGPT rozwiązania do granic możliwości, aż popełni błędy, które udowodnią, że nie jest wart całego tego szumu! Sztuczna inteligencja nigdy nie będzie w stanie zastąpić złożoności tego, co robię.*

Takie reakcje będą się nasilać w miarę rozprzestrzeniania się sztucznej inteligencji opartej na transformerach na coraz szerszą gamę codziennych zastosowań.

Odpowiedzialność specjalistów w dziedzinie sztucznej inteligencji sprowadza się do zdobycia wszechstronnej wiedzy na temat transformerów i ich funkcjonalności, oceny ich ograniczeń oraz doskonalenia naszych umiejętności.

Rola specjalistów AI

Sztuczna inteligencja oparta na transformerach łączy wszystko ze wszystkim i wszędzie. Maszyny komunikują się bezpośrednio z innymi maszynami. Sygnały urządzeń IoT korzystających ze sztucznej inteligencji podejmują automatyczne decyzje bez udziału człowieka. Algorytmy NLP wysyłają automatyczne raporty, podsumowania, e-maile, reklamy i inne treści.

Specjaliści w dziedzinie sztucznej inteligencji muszą dostosować się do nowej ery coraz bardziej zautomatyzowanych zadań, w tym implementacji modeli transformerów. Specjaliści w dziedzinie sztucznej inteligencji będą mieli nowe zadania. Jeśli wypiszemy zadania z zakresu przetwarzania języka naturalnego, jakie będzie musiał wykonać specjalista ds. AI, od najważniejszych do najmniej ważnych, okaże się, że niektóre wysokopoziomowe zadania wymagają od specjalisty AI niewielkiego zaangażowania lub zgoła żadnego. Specjalista AI może być guru w dziedzinie sztucznej inteligencji, dostarczać pomysły projektowe, wyjaśnienia i implementacje.

Praktyczna definicja tego, co dla specjalisty AI oznacza transformer, będzie się różnić w zależności od ekosystemu.

Przyjrzyjmy się kilku przykładom:

- **API.** Korzystanie z API OpenAI nie wymaga obecności programisty AI. Projektant stron internetowych może utworzyć formularz, a językoznawca lub **ekspert merytoryczny** (ang. *subject matter expert* — **SME**) może przygotować teksty zapytań. Podstawowa rola specjalisty ds. sztucznej inteligencji będzie wymagać umiejętności lingwistycznych pozwalających pokazać, a nie tylko opowiedzieć, jak modele ChatGPT/GPT-4 i ich następcy mają zrealizować określone zadanie. Na przykład pokazanie modelowi, co ma zrobić, wiąże się z pracą nad kontekstem danych wejściowych. To nowe zadanie nazwano *inżynierią zapytań* (ang. *prompt engineering*). *Inżynier zapytań* ma zapewnioną przyszłość w dziedzinie sztucznej inteligencji!
- **Biblioteka.** Rozpoczęcie korzystania z gotowych modeli oferowanych przez bibliotekę Google Trax wymaga niewielkiego nakładu pracy programistycznej. Specjalista AI dysponujący wiedzą lingwistyczną i umiejętnościami przetwarzania języka naturalnego może pracować na zbiorach danych i wynikach. Specjalista AI może również wykorzystać zestaw narzędzi Trax, aby stworzyć modele dostosowane do konkretnego projektu.
- **Szkolenie i dostrajanie.** Niektóre funkcjonalności Hugging Face wymagają ograniczonych prac programistycznych. Dostarczają zarówno interfejsów API, jak i bibliotek. Czasami jednak musimy pobrudzić sobie ręce. W takim przypadku szkolenie, dostrajanie modeli i znajdowanie właściwych hiperparametrów wymaga wiedzy specjalisty w dziedzinie sztucznej inteligencji.
- **Umiejętności na poziomie programowania.** W niektórych projektach tokenizery i zestawy danych nie pasują do siebie. Albo osadzenia modelu

(ang. *embeddings*) mogą nie pasować do projektu. W tym przypadku kluczową rolę może odegrać na przykład programista AI współpracujący z lingwistą. Dlatego na tym poziomie bardzo przydatne może się okazać szkolenie w dziedzinie lingwistyki komputerowej.

Ostatnią ewolucję sztucznej inteligencji przetwarzającej język naturalny można określić mianem „wbudowanych transformatorów”. Są one umieszczone w asystentach, pomocniczych pilotach (ang. *copilots*) i codziennym oprogramowaniu. To stanowi znaczącą zmianę w ekosystemie programowania systemów sztucznej inteligencji:

- Transformery LLM z miliardami parametrów i wieloma warstwami do szkolenia, takie jak OpenAI GPT-4, są obecnie zintegrowane z kilkoma aplikacjami Microsoft Azure, w tym z GitHub Copilot.
- Wbudowane transformery nie są dostępne bezpośrednio, ale zapewniają automatyczne wsparcie programistyczne, np. do automatycznego generowania kodu. Oferują również funkcje podsumowujące, mechanizmy zadawania pytań i udzielania odpowiedzi oraz wiele innych zadań w coraz większej liczbie aplikacji.
- Zastosowania wbudowanych transformerów są nieograniczone, a dzięki funkcjom uzupełniania tekstu korzystanie z nich nie stanowi problemu dla użytkownika końcowego.

W dalszej części tej książki postaram się dokładniej omówić ten fascynujący nowy świat wbudowanych transformatorów.

Zestaw umiejętności specjalisty AI w dziedzinie modeli LLM wymaga zdolności adaptacji, wiedzy interdyscyplinarnej i przede wszystkim *elastyczności*. Niniejsza książka wyposaży specjalistów AI w różnorodne ekosystemy transformerów, które pomogą im dostosować się do nowych paradygmatów rynku. Te przeciwstawne i często sprzeczne strategie pozostawiają wiele możliwych sposobów implementacji.

Przyszłość specjalistów AI to praca w wielu różnych specjalizacjach.

Przyszłość specjalistów AI

Nie należy lekceważyć wpływu modeli podstawowych na społeczeństwo. Inżynieria zapytań stała się umiejętnością wymaganą od specjalistów zajmujących się sztuczną inteligencją.

Specjalista AI będzie również zajmował się algorytmami maszyna – maszyna z wykorzystaniem klasycznej sztucznej inteligencji, urządzeń IoT, przetwarzania brzegowego i innych technologii. Specjalista AI będzie również wykorzystywał klasyczne algorytmy podczas projektowania i implementacji fascynujących połączeń między botami, robotami, serwerami i wszelkiego rodzaju urządzeniami podłączonymi do sieci.

Książka ta nie ogranicza się zatem do bieżącej inżynierii, lecz obejmuje szeroki zakres umiejętności projektowych niezbędnych do tego, aby zostać specjalistą AI w dobie modeli LLM.

Inżynieria zapytań to podzbiór umiejętności projektowych, które musi rozwinąć specjalista zajmujący się sztuczną inteligencją. Oto wybrane dziedziny, w które może zainwestować specjalista AI:

- **Specjaliści AI** posiadają specjalistyczną wiedzę lub umiejętności w jednej z dziedzin sztucznej inteligencji. Do interesujących obszarów należy dostosowywanie modeli, utrzymanie i wsparcie systemów AI.
- **Architekci AI** projektują architektury oraz dostarczają rozwiązań do skalowania i innych informacji dotyczących wdrażania i produkcji.
- **Eksperci AI** posiadają zaawansowaną wiedzę i umiejętności w dziedzinie sztucznej inteligencji. Ekspert AI może przyczynić się do rozwoju tego obszaru poprzez badania, artykuły naukowe i innowacyjne rozwiązania.
- **Analitycy AI** koncentrują się na danych, dużych zbiorach danych i architekturze modeli, aby dostarczać informacje innym zespołom.
- **Badacze AI** skupiają się na badaniach na uniwersytetach lub w prywatnych firmach.
- **Inżynierowie AI** projektują systemy AI, budują je i implementują modele AI.
- **Menedżerowie AI** wykorzystują kluczowe umiejętności zarządzania projektami, zarządzania produktami oraz zarządzania zasobami (ludźmi, maszynami i oprogramowaniem).
- Pojawi się wiele innych obszarów, co pokaże, że w dziedzinie sztucznej inteligencji istnieją liczne możliwości rozwijania zasobów, takich jak zestawy danych, modele i aplikacje.

W obliczu rosnącej obecności transformerów we wszelkich zastosowaniach ważne jest znalezienie odpowiednich zasobów do realizacji projektów.

Jakich zasobów powinniśmy używać?

Generatywna sztuczna inteligencja zaciera granice między platformami chmurowymi, frameworkami, bibliotekami, językami i modelami. Transformery to nowość, a różnorodność i liczba ekosystemów, w jakich one występują, jest oszałamiająca. Google Cloud udostępnia gotowe do użycia modele transformerów.

OpenAI wdrożyło API transformera, który praktycznie nie wymaga programowania. Hugging Face udostępnia usługę biblioteki w chmurze. Mogłbym wymienić jeszcze wiele przykładów.

Microsoft 365 i Google Workspace dysponują narzędziami generatywnej sztucznej inteligencji opartymi na najnowocześniejszych transformerach. Dzięki temu każdy użytkownik pakietu Microsoft 365 i Google Workspace ma sztuczną inteligencję na wyciągnięcie ręki!

Wybór zasobów do wdrożenia transformerów w zadaniach przetwarzania języka naturalnego ma kluczowe znaczenie. Od tego zależy powodzenie projektu. Wyobraź sobie

prawdziwą rozmowę kwalifikacyjną lub prezentację. Wyobraź sobie, że rozmawiasz ze swoim przyszłym pracodawcą, swoim zespołem lub klientem.

Na przykład zaczynasz przedstawianie założeń od świetnej prezentacji PowerPoint na temat Hugging Face. Możesz spotkać się z negatywną reakcją menedżera, który mówi: „Przykro mi, ale w tego typu projektach używamy Google Cloud AI, a nie Hugging Face. Czy mógłbyś zastosować Google Cloud AI?”. Jeśli nie odpowiesz twierdząco, gra jest dla Ciebie skończona.

Ten sam problem mógłby się pojawić, gdybyś specjalizował się w Google Cloud AI. Mógłbyś spotkać menedżera, który chce używać modeli ChatGPT/GPT-4 firmy OpenAI z API i bez konieczności programowania. Jeśli specjalizujesz się w silnikach GPT OpenAI z API bez konieczności programowania, mógłbyś trafić na menedżera projektu lub klienta, który woli AutoML lub HuggingGPT firmy Hugging Face!

Najgorsze, co może się zdarzyć, to sytuacja, kiedy menedżer zaakceptuje Twoje rozwiązanie, ale ostatecznie okaże się, że w ogóle nie nadaje się ono do realizacji zadań NLP w danym projekcie.

Najważniejszą zasadą, o której należy pamiętać, jest to, że jeśli skupisz się tylko na tym rozwiązaniu, które Ci odpowiada, najprawdopodobniej w pewnym momencie zatkniesz razem ze statkiem.

Skup się na systemie, którego potrzebujesz, a nie na tym, który lubisz.

Celem tej książki nie jest omówienie wszystkich dostępnych na rynku rozwiązań opartych na transformerach, lecz opisanie wystarczającej liczby ekosystemów transformatorów, abyś mógł elastycznie dostosowywać się do każdej sytuacji, z jaką przyjdzie Ci się zmierzyć w projekcie NLP.

Przyjrzyjmy się kilku podejściom, które mogą pomóc Ci podjąć decyzję.

Wytyczne dotyczące podejmowania decyzji

Moje osobiste doświadczenia opierają się na trzech prostych zasadach:

- Postaraj się zrozumieć na poziomie eksperta w danej dziedzinie zadania, które ma wykonać model sztucznej inteligencji. Zapisz je na papierze bez pomocy ChatGPT. Zrób to samodzielnie. Następnie stwórz szczegółowy schemat blokowy.
- Aby przetestować wydajność sprzętu (oprogramowania), zbuduj prototypowy zestaw danych, który można powielić w dowolnej skali.
- Aby sprawdzić, który z modeli NLP (CV) odpowiada Twoim potrzebom, wypróbuj jeden lub kilka z nich: może to być oprogramowanie open source lub komercyjne, API, instalacja lokalna, platforma w chmurze. Pamiętaj o uwzględnieniu ograniczeń Twojego projektu.

Nie spiesz się. Nie pozwól, by ktokolwiek na Ciebie naciskał. Kieruj się dowodami, a nie opiniami.

Następnie zwróć się do ChatGPT Plus o dodatkowe wskazówki dotyczące trzech zasad, o których wspominałem przed chwilą. ChatGPT przekazuje kilka ciekawych porad:

1. **Ocena problemu.** Zaczynij od dokładnego zrozumienia problemu. Jakie są najważniejsze wyzwania? Z jaką skalą danych masz do czynienia?
2. **Ocena zestawu umiejętności.** Jaki jest poziom zaawansowania zespołu? Znajomość niektórych platform może przyspieszyć rozwój systemu.
3. **Skalowalność.** Weź pod uwagę trajektorię wzrostu. Czy wybrane rozwiązanie będzie skalowalne w miarę gromadzenia się danych i zwiększania się złożoności problemu?
4. **Spółeczność i wsparcie.** Narzędzia i modele z aktywnymi społecznościami mogą być wygodne ze względu na dostępne zasoby, samouczki i natychmiastowe rozwiązania typowych problemów.
5. **Interoperacyjność.** W jakim stopniu rozwiązanie integruje się z istniejącymi systemami lub potencjalnymi przyszłymi narzędziami?

Pamiętaj, że narzędzie lub model jest na tyle dobry, na ile zespół rozumie jego zasady i na ile potrafi je zastosować w rozwiązywaniu danego problemu.

Jednym z wyzwań projektu przetwarzania języka naturalnego jest ocena rozwiązania, które ma być wdrożone. API sztucznej inteligencji to jedno z narzędzi na rynku, które stale się rozwijają i które należy zbadać.

Rozwój łatwych do integracji interfejsów API i asystentów

Wkroczyliśmy już w erę industrializacji sztucznej inteligencji. Między innymi takie usługi jak Microsoft Azure, Google Cloud, Amazon Web Services (AWS) i IBM dostarczają usług AI, których nie jest w stanie dostarczyć żaden pojedynczy programista lub zespół programistów. Giganci technologiczni dysponują superkomputerami wartymi miliony dolarów i dysponującymi ogromnymi zbiorami danych, co pozwala im szkolić modele transformerów i ogólnie modele sztucznej inteligencji.

Wielkie firmy technologiczne mają wielu klientów korporacyjnych, którzy korzystają z ich usług w chmurze. W rezultacie dodanie API transformera do istniejącej architektury chmury wymaga mniejszego wysiłku niż w przypadku jakiegokolwiek innego rozwiązania.

Mała firma, a nawet osoba prywatna, za pośrednictwem API może uzyskać dostęp do najpotężniejszych modeli transformerów, praktycznie bez konieczności inwestowania w programowanie. API w ciągu kilku dni może wdrożyć nawet stażysta. Aby wykonać tak prostą implementację, nie trzeba mieć tytułu doktora, nie trzeba nawet być inżynierem.

Przykładowo platforma OpenAI dla niektórych z najefektywniejszych modeli transformerów dostępnych na rynku oferuje teraz API typu **Software as a Service (SaaS)**.

Modele transformerów OpenAI są tak skuteczne i tak bardzo przypominają ludzkie zachowania, że obecna polityka wymaga od potencjalnego użytkownika wypełnienia formularza zgłoszeniowego. Po zaakceptowaniu żądania użytkownik uzyskuje dostęp do wszechświata narzędzi przetwarzania języka naturalnego!

Prostota korzystania z API OpenAI jest zaskakująca:

1. Jednym kliknięciem uzyskaj klucz API.
2. Za pomocą jednej linijki zaimportuj pakiet OpenAI w notatniku.
3. Wprowadź w *wierszu poleceń* dowolne zadanie NLP.
4. Otrzymasz odpowiedź bez konieczności pisania żadnego dodatkowego kodu.

Na przykład, jak wyjaśniono na platformie OpenAI (<https://platform.openai.com/examples/default-sql-translate>), możesz przetłumaczyć język naturalny na zapytanie SQL (rysunki 1.14 i 1.15).

Prompt

```
### Postgres SQL tables, with their properties:
#
# Employee(id, name, department_id)
# Department(id, name, address)
# Salary_Payments(id, employee_id, amount, date)
#
### A query to list the names of the departments which employed more than
10 employees in the last 3 months
SELECT
```

Rysunek 1.14. Prośba o utworzenie kwerendy SQL w OpenAI

Sample response

```
SELECT d.name
FROM Department d
INNER JOIN Employee e ON d.id = e.department_id
INNER JOIN Salary_Payments sp ON e.id = sp.employee_id
WHERE sp.date > NOW() - INTERVAL '3 months'
GROUP BY d.name
HAVING COUNT(*) > 10
```

Rysunek 1.15. Odpowiedź na prośbę utworzenie zapytania SQL

To wszystko! Zapytania nie zawsze są idealne i mogą wymagać poprawek, ale ogólnie rzecz biorąc, wykonanie tego zadania pokazuje olbrzymie możliwości modeli LLM. Witamy w świecie generatywnej sztucznej inteligencji!

Programiści skupiający się na rozwiązaniach opartych wyłącznie na kodzie wyewoluują w pokolenie programistów o interdyscyplinarnym podejściu, wykorzystujących potencjał asystentów AI.

Specjaliści AI nauczą się, jak *pokazać* modelowi transformera, czego się od niego oczekuje, zamiast intuicyjnie *opowiadać* mu, co ma zrobić. W trakcie lektury tej książki poznasz kilka metod kontrolowania zachowania tych najnowocześniejszych modeli sztucznej inteligencji.

Chociaż API mogą zaspokoić wiele potrzeb, mają też ograniczenia. Uniwersalne API może sprawdzać się w różnorodnych zadaniach, ale może być niewystarczające do konkretnego zadania przetwarzania języka naturalnego. Na przykład tłumaczenie za pomocą transformerów nie jest łatwym zadaniem. W takim przypadku programista AI, konsultant lub kierownik projektu musi udowodnić, że samo API nie jest w stanie rozwiązać wymaganego zadania NLP. W związku z tym może zaistnieć konieczność poszukania solidnej biblioteki, alternatywnych rozwiązań lub opracowania odpowiednich narzędzi.

Wybieranie gotowych do użycia bibliotek opartych na API

W tej książce przyjrzymy się kilku bibliotekom. Google ma jedno z najnowocześniejszych laboratoriów sztucznej inteligencji na świecie. Aplikację Google Trax można zainstalować w usłudze Google Colab za pomocą kilku linijek. Możemy wybierać między usługami bezpłatnymi i płatnymi. Możemy modyfikować kod źródłowy, dostrajając modele, a nawet szkolić je na własnych serwerach lub w chmurze Google Cloud. Przykładowo przejście od gotowych do użycia API do spersonalizowanego modelu transformera na potrzeby zadań tłumaczeniowych wydaje się krokiem wstecz.

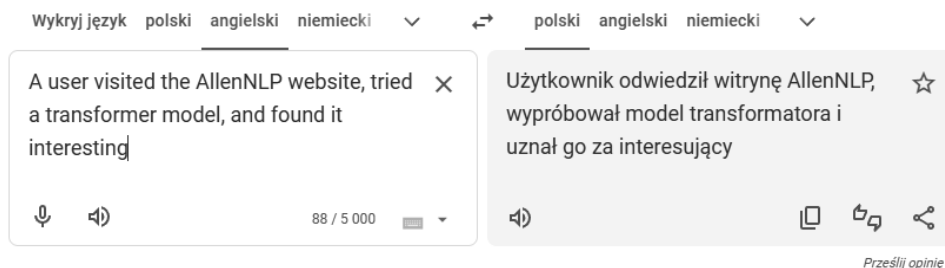
Jednak w niektórych przypadkach może się okazać, że zarówno ma to walory edukacyjne, jak i jest skuteczne. W rozdziale 4. „Postępy w tłumaczeniach dzięki Google Trax, Tłumacza Google i Gemini” przyjrzymy się ostatnim zmianom w usługach Google w zakresie tłumaczeń i pokażę sposób korzystania z Google Trax.

API takie jak OpenAI wymagają ograniczonych umiejętności programistycznych, natomiast takie biblioteki, jak Google Trax wymagają nieco więcej kodowania. Oba podejścia pokazują, że API sztucznej inteligencji wymagają większego nakładu pracy ze strony edytora API, ale znacznie mniejszego wysiłku przy implementacji transformerów.

Tłumacz Google to jedna z najpopularniejszych aplikacji internetowych, która obok kilku algorytmów wykorzystuje także transformery. Z Tłumacza Google można korzystać online lub za pośrednictwem API.

Spróbuj przy użyciu Tłumacza Google przetłumaczyć zdanie wymagające rozwiązania koreferencji z języka angielskiego na polski. Zdanie brzmi: *A user visited the AllenNLP*

website, tried a transformer model, and found it interesting. Tłumacz Google wygenerował tłumaczenie przedstawione na rysunku 1.16.



Rysunek 1.16. Rozwiązywanie koreferencji w tłumaczeniu przy użyciu Tłumacza Google

Wygląda na to, że Tłumacz Google rozwiązał problem koreferencji, ale słowo *transformator* po polsku oznacza urządzenie elektryczne. Słowo *transformer* jest w języku polskim zapożyczeniem. Od specjalisty AI może być wymagane posiadanie umiejętności językowych niezbędnych do realizacji konkretnego projektu. W tym przypadku nie są potrzebne zaawansowane umiejętności programowania. Projekt może jednak wymagać doprecyzowania danych wejściowych przed poproszeniem systemu AI o przetłumaczenie.

Ten przykład pokazuje, że aby móc pracować nad kontekstem wejściowym, być może będziesz musiał nawiązać współpracę z językoznawcą lub nabyć umiejętności lingwistycznych. Ponadto może być konieczne wykorzystanie kodu, aby ulepszyć wejście za pomocą interfejsu do obsługi kontekstów.

Być może Google udoskonali swój system do czasu opublikowania tej książki, ale w obecnej postaci ma on tysiące ograniczeń.

Aby zatem skorzystać z Tłumacza Google, być może nadal będziesz zmuszony do zastosowania dodatkowych skryptów. Możesz również być konieczne znalezienie modelu transformera dla potrzeb konkretnego tłumaczenia, np. BERT, T5 lub innego modelu, które omówię w dalszej części tej książki.

Jednym z zadań specjalisty AI jest wybór API. Biorąc pod uwagę stale rosnącą liczbę rozwiązań, znalezienie odpowiedniego modelu transformera nie jest łatwe.

Wybór platformy chmurowej i modelu transformera

Na rynku przetwarzania języka naturalnego dominują duże korporacje technologiczne. Google, Facebook i Microsoft codziennie uruchamiają miliardy procedur przetwarzania języka naturalnego, co dzięki gromadzonym danym zwiększa możliwości modeli AI wykorzystywanych przez te firmy. Wielcy giganci oferują obecnie szeroką gamę modeli transformerów i stosują najwyższej klasy modele podstawowe.

Próg modeli podstawowych stanowią w pełni przeszkolone na superkomputerach transformery, np. OpenAI GPT-4, Google PaLM, a także nowe modele, które będą powstawać przez cały czas. Modele podstawowe są często zastrzeżone, co oznacza, że nie mamy dostępu do ich kodu, choć niektóre z nich możemy dostrajać dzięki wykorzystaniu usługi w chmurze.

Hugging Face stosuje inne podejście. Oferuje szeroką gamę modeli transformerów z kartami modeli, kodem źródłowym, zbiorami danych i dodatkowymi zasobami programistycznymi. Ponadto Hugging Face oferuje zaawansowane API oraz API zarządzane przez programistów. W kilku rozdziałach tej książki przyjrzymy się bliżej Hugging Face jako narzędziu edukacyjnemu i możliwemu rozwiązaniu wielu problemów.

Google Cloud, Microsoft Azure, AWS, Hugging Face i inni oferują fantastyczne usługi! Przyglądając się rosnącej górze modeli AI opartych na transformerach, musisz znaleźć swoje miejsce.

Podsumowanie

Powstanie transformerów zmusiło branżę AI do głębokiej ewolucji. Modele podstawowe, w tym ich zdolności w zakresie generatywnej sztucznej inteligencji, opierają się na rewolucji cyfrowej, która łączy różne systemy i dane z wykorzystaniem działających w tle rozproszonych procesów. Zautomatyzowane procesy zastępują ludzkie decyzje w kluczowych obszarach, w tym w zadaniach przetwarzania języka naturalnego.

Sieci neuronowe RNN spowolniły postęp zautomatyzowanych zadań przetwarzania języka naturalnego, niezbędnych w szybko zmieniającym się świecie. Transformery wypełniły tę lukę. Aby sprostać wyzwaniom związanym z rosnącą ilością napływających informacji, korporacje potrzebują podsumowań, tłumaczeń i szerokiej gamy narzędzi NLP.

Transformery zapoczątkowały erę industrializacji sztucznej inteligencji. Najpierw pokazałem, jak złożoność czasowa warstw uwagi wynosząca $O(1)$ i ich czasowa złożoność obliczeniowa $O(n^2 \cdot d)$ wstrząsnęły światem sztucznej inteligencji.

Jak można się przekonać, elastyczność modeli transformerów przeniknęła każdą dziedzinę codziennego życia.

Platformy takie jak Hugging Face, Google Cloud, OpenAI i Microsoft Azure udostępniają gotowe zadania przetwarzania języka naturalnego. Implementacja modelu transformera we własnych programach nie wymaga instalacji i dodatkowych zasobów. Na przykład OpenAI udostępnia API, dzięki któremu uruchomienie jednego z zaawansowanych modeli generacji GPT-4 wymaga zaledwie kilku linijek kodu. Google Trax udostępnia kompleksową bibliotekę, Google Vertex AI oferuje łatwe do wdrożenia narzędzia, a Hugging Face dostarcza różne modele transformerów i ich implementacje. W dalszej części książki będziemy badać te ekosystemy.

Następnie pokazałem, że zastosowanie architektury transformerów prowadzi do automatyzacji, która znacząco odbiega od dawnej sztucznej inteligencji. W rezultacie od

specjaliści AI wymagane są większe umiejętności. Na przykład kierownik projektu może podjąć decyzję o zaimplementowaniu architektury transformerów poprzez zlecenie projektantowi stron internetowych stworzenia API dla Google Cloud AI lub OpenAI za pośrednictwem inżynierii zapytań. W razie potrzeby kierownik projektu może poprosić specjalistę AI o pobranie zasobów Google Trax lub Hugging Face w celu opracowania pełnowymiarowego projektu ze spersonalizowanym modelem transformera.

Transformer to prawdziwa rewolucja dla programistów, których zakres obowiązków będzie się rozszerzał i będzie wymagał więcej projektowania niż programowania. Ponadto wbudowane transformery umożliwią wspomaganie tworzenia i użytkowania kodu. *Te nowe umiejętności stanowią wyzwanie, ale otwierają nowe, ekscytujące horyzonty.*

Rozdział 2. „Wprowadzenie do architektury modelu transformera” rozpocznie od wykozystania architektury *Original Transformer*.

Pytania

1. ChatGPT zmienił oblicze sztucznej inteligencji. (Prawda/Fałsz)
2. ChatGPT może zastąpić wszystkie algorytmy sztucznej inteligencji. (Prawda/Fałsz)
3. W niektórych projektach programiści AI nie będą musieli tworzyć żadnego kodu. (Prawda/Fałsz)
4. Programiści AI czasami będą zmuszeni do implementacji transformerów od podstaw. (Prawda/Fałsz)
5. Nie ma konieczności nauki obsługi więcej niż jednego ekosystemu transformerów, takiego jak Hugging Face. (Prawda/Fałsz)
6. Gotowe do użycia API transformera może spełnić wszystkie potrzeby. (Prawda/Fałsz)
7. Firma zaakceptuje ekosystem transformerów, który programista zna najlepiej. (Prawda/Fałsz)
8. Transformery chmurowe stały się powszechne. (Prawda/Fałsz)
9. Projekt transformera można uruchomić na laptopie. (Prawda/Fałsz)
10. Specjaliści AI będą musieli wykazać się większą elastycznością. (Prawda/Fałsz)

Odnosiniki

- Bommansani et al., 2021, *On the Opportunities and Risks of Foundation Models*: <https://arxiv.org/abs/2108.07258>.

- Rishi Bommasani, Dilara Soylu, Thomas I. Liao, Kathleen A. Creel, Percy Liang, 2023, *Ecosystem Graphs: The Social Footprint of Foundation Models*: <https://arxiv.org/abs/2303.15772>.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, 2017, *Attention is All You Need*: <https://arxiv.org/abs/1706.03762>.
- Chen et al., 2021, *Evaluating Large Language Models Trained on Code*: <https://arxiv.org/abs/2107.03374>.
- Microsoft AI: <https://innovation.microsoft.com/en-us/ai-at-scale>.
- OpenAI: <https://openai.com/>.
- Google AI: <https://ai.google/>.
- Google Trax: <https://github.com/google/trax>.
- AllenNLP: <https://docs.allennlp.org/models/main/>.
- Hugging Face: <https://huggingface.co/>.
- Google Cloud TPU: <https://cloud.google.com/tpu/docs/intro-to-tpu>.

Lektura uzupełniająca

- Tyna Eloundou, Sam Manning, Pamela Mishkin, Daniel Rock, 2023, *GPT to GPT: An Early Look at the Labor Market Impact Potential of Large Language Models*: <https://arxiv.org/abs/2303.10130>.
- Jussi Heikkilä, Julius Rissanen, Timo Ali-Vehmas, 2023, *Coopetition, standardization, and general purpose technologies: A framework and an application*: <https://www.sciencedirect.com/science/article/pii/S0308596122001902>.
- Blog firmy NVIDIA na temat modeli podstawowych: <https://blogs.nvidia.com/blog/2023/03/13/what-are-foundation-models/>.
- O łańcuchach Markowa: <https://mathshistory.st-andrews.ac.uk/Biographies/Markov/>.

Skorowidz |

A

agent czatu
 filtrowanie danych, 216
 inicjalizowanie obiektu trenera, 220
 instalowanie bibliotek, 216
 interfejs użytkownika, 222
 konfigurowanie modelu, 218
 ograniczenia, 224
 pobieranie zbioru danych, 216
 przetwarzanie zbioru danych, 219
 sprawdzanie ograniczeń zasobów, 218
 wstępne szkolenie modelu, 221
 zapisywanie modelu, 221
agenty AI, 503
AGI, Artificial General Intelligence, 490
algorytm
 BLEU, 97
 klasteryzacji k-średnich, 382
 klasyfikacji, 50
analiza
 części mowy, POS, 391
 emergencji, 107
 głównych składowych, PCA, 286
animacja Stability AI, 588
API, 55, 59, 61
 AJAX, 150
 Discorda, 682
 modeli DALL-E 2 i DALL-E 3, 555
 modelu GPT-4, 254
 modelu GPT-4V, 565
 OpenAI, 60
 PaLM 2 Vertex AI, 468
 Stable Diffusion, 584
architektura
 BERT, 158
 CUDA, 169
 DALL-E, 554
 kreowania pomysłów, 666
 modelu CLIP, 549

 PaLM 2, 453
 Pathways, 448
 transformera, 49, 66–69
 transformera GPT, 234, 239, 241
 ViT, 536
 wielogłowicowej uwagi, 79
asystent
 AI, 59, 232, 244, 454
 Google'a, 454
 tworzenia kodu, 245, 249
 Vertex AI PaLM 2, 465
Auto-BIG-bench, 495
automatyczne projektowanie promptów,
 668, 674, 675
automatyzacja zadań, 48
AutoTrain, 598
 klasyfikacja obrazów, 611
 konfiguracja przeszkolonych modeli, 613
 obrazy walidacyjne, 609
 przesyłanie zestawu danych, 602
 szkolenie modeli, 606
 tworzenie nowego projektu, 602
 uruchamianie modeli, 609
 wdrażanie modeli, 607

B

baza
 reguł, 521
 wiedzy, 525, 527
BERT, Bidirectional Encoder
 Representations from Transformers, 157
 aktywowanie tokenizera, 171
 architektura, 158
 dodawanie tokenów, 171
 dostrajanie modelu, 164, 166–186
 importowanie modułów, 168
 interfejs Pythona, 187
 konfigurowanie modelu, 174
 konwertowanie danych, 173

- listy etykiet, 171
 - ładowanie modelu bazowego, 176
 - ocena modelu, 184
 - ocena szkolenia, 181
 - określanie celu, 167
 - optymalizator parametrów modelu, 177
 - pętla szkolenia, 180
 - hiperparametry, 179
 - prognozowanie, 181
 - przetwarzanie danych, 172
 - stos kodera, 159
 - tworzenie
 - iteratora, 173
 - masek uwagi, 172
 - zdań, 171
 - współczynnik korelacji Matthews, 184, 185
 - wstępne szkolenie, 164
 - metoda MLM, 162
 - metoda NSP, 163
 - wybieranie rozmiaru partii, 173
 - zbiór
 - szkoleniowy, 173
 - walidacyjny, 173
 - zestaw danych CoLA, 169
 - BertViz, 287
 - dostęp do warstwy uwagi, 288
 - instalowanie, 288
 - przesyłanie strumieniowe wyników, 294
 - widok
 - głowic, 289
 - modelu transformera, 291
 - wizualizacja związków między słowami, 296
 - wyświetlanie
 - głowic uwagi, 289
 - prawdopodobieństw, 293
 - biblioteka
 - accelerator, 216
 - API OpenAI, 555
 - datasets, 216
 - Google Trax, 55
 - googletrans, 150
 - IPython, 609
 - pandas, 296
 - Pillow, 555
 - transformers, 216, 293, 611
 - Trax, 145
 - Vertex AI, 27, 63
 - BLEU, Bilingual Evaluation Understudy, 97, 131
 - ocena tłumaczeń maszynowych, 141
 - BPE, Byte-Pair Encoding, 194
- ## C
- ChatGPT, 50, 228
 - algorytm klasyfikacji, 50
 - baza wiedzy, 525
 - drzewo decyzyjne, 54
 - eksperymenty SRL, 393
 - funkcja obsługi dialogu, 528
 - generowanie
 - kodu, 107, 244
 - promptu, 694
 - ilustracja prezentacji, 671
 - Plus, 244
 - tworzenie obrazu, 563
 - prezentacja HTML, 669
 - rozpowszechnianie, 231
 - tworzenie
 - funkcji dialogu, 404
 - streszczeń tekstu, 438
 - udoskonalenia, 230
 - wszechobecność, 232
 - wykonywanie SRL, 403
 - z modelem GPT-4, 393, 669
 - ciągła kontrola jakości, 323
 - CLIP, Contrastive Language-Image Pre-Training, 549
 - architektura modelu, 549
 - w kodzie, 550
 - zamiana tekstu na wideo, 590
 - CNN, Convolutional Neural Network, 43
 - CoLA, Corpus of Linguistic Acceptability, 124, 169, 419
 - CPU, Central Processing Unit, 32
 - CUDA, Compute Unified Device Architecture, 169, 202, 218
 - CustomGPT, 649
 - czterofazowy proces, 650
 - czasowa złożoność obliczeniowa
 - przy użyciu CPU, 33
 - przy użyciu GPU, 36
 - przy użyciu TPU, 38

D

DAL-E
architektura modelu, 554
tworzenie obrazu, 563
DAL -E 2, 553
tworzenie obrazu, 555
tworzenie wariantu obrazu, 557
DAL -E 3, 553, 561
API, 555
interfejs użytkownika, 559
tworzenie obrazu, 559
dopasowywanie zbiorów danych
najlepsze praktyki, 321
tokenizacja Word2Vec, 325
dostrajanie modeli
BERT, 164, 166–186
GPT, 266–282
PaLM 2, 482
drzewo decyzyjne, 53
duże modele językowe, LLM, 27, 66
dyskryminatywna sztuczna inteligencja, 48
działanie sieci RNN, 43

E

ekspert dziedzinowy, SME, 55, 149
emergencja, 49, 101, 534
analizowanie, 107
exBERT, 298

F

F-AGI, Functional Artificial General
Intelligence, 630
definicja systemu, 632
FFN, Feedforward Network, 93
format
JSON, 270
JSONL, 272, 483
TSV, 363
framework BERT, 165
funkcja softmax, 295, 464
funkcjonalna sztuczna inteligencja ogólna,
F-AGI, 490, 630

G

Gemini, 152, 456
generatywna sztuczna inteligencja, 48,
215, 228

generatywne
kreowanie pomysłów, 664, 666, 690
automatyzacja projektowania
promptów, 668
framework, 666
zautomatyzowana architektura, 666
projektowanie obrazów, 668
wstępnie przeszkolone transformery,
GPT, 229, 263
generowanie
obrazów, 668, 680, 696
promptu, 668, 694
streszczeń, 415, 423, 431, 439
wspomagane wyszukiwaniem, RAG,
228, 257, 260, 267, 350, 520
Gensim, 356
GitHub Copilot
tworzenie oprogramowania, 245
Google Cloud Vision
klasyfikowanie obrazów
poziom bardzo trudny, 654
poziom łatwy, 651
poziom trudny, 653
z ChatGPT, 656
Google Colab Copilot, 460
okno wprowadzania promptu, 460
opcja generowania kodu, 462
Google Translate Ajax API, 150
Google Trax
implementacja transformera
tłumaczeniowego, 145
instalowanie biblioteki, 145
Google Workspace, 457
GPT, Generative Pretrained Transformer,
229, 263
architektura transformera, 234, 239, 241
baza wiedzy, 369, 371
dostrajanie modeli, 266–82
przygotowywanie danych, 270
ryzyko, 267
uruchamianie, 277
użycie gpt-3.5-turbo, 275
zarządzanie modelami, 280
ewolucja liczby parametrów, 236
Few-shot, FS, 239
inteligencja maszynowa, 240
maksymalna długość ścieżki, 237
One-shot, 1S, 239
precyzyjne dostrajanie, FT, 239
rozmiar kontekstu, 237
stos warstw dekodera, 240

- wydajność modelu, 236
- Zero-shot, ZS, 239
- GPT -4
 - API, 254
 - automatyczne instrukcje, 691
 - baza wiedzy, 525
 - pozyskiwanie informacji z dokumentów, 258, 259
 - replikacja, 503
 - technika RAG, 258, 260
 - uruchamianie
 - wielu zadań NLP, 257
 - zadania NLP, 254
 - zadań SRL, 405
 - w roli asystenta, 249
 - wyjaśnienie działania neuronów, 313
- GPT-4 explainer, 309
- GPT-4V, 561, 565
 - zadania DAT, 565
- GPU, Graphics Processing Unit, 32, 202, 218
- graf wnioskowania GPT, 45
- gry MMO, 699

H

- hiperparametry, 256
- homogenizacja, 49, 535
- Hugging Face, 168, 194, 197, *Patrz także*
 - AutoTrain
 - bert-base-uncased, 176
- HuggingGPT, 638
 - czterofazowy system AI, 639
 - proces walidacji, 638
 - uruchamianie, 640
 - zadanie klasyfikacji
 - poziom bardzo trudny, 645
 - poziom łatwy, 640
 - poziom trudny, 642

I

- instalacja OpenAI, 404, 492
- interfejs
 - API, 59
 - LIT, 309
 - Pythona dla modelu BERT, 186
 - typu Proof of Concept, 224
 - Vertex AI, 462
 - wizualizacji, 307
- inżynieria promptów, 352
- IR, Intermediate Representation, 450

K

- KantaiBERT, 195
 - inicjalizowanie
 - modelu, 203
 - trenera, 210
 - instalowanie transformerów Hugging Face, 197
 - konfigurowanie modelu, 203
 - ładowanie
 - tokenizera, 201, 203
 - zbioru danych, 196
 - mechanizm zbierania danych, 210
 - modelowanie języka, 213
 - sprawdzanie ograniczeń zasobów, 202
 - szkolenie tokenizera, 198
 - tworzenie zbioru danych, 209
 - wstępne szkolenie modelu, 212
 - zapisywanie
 - plików, 200
 - przeszkolonego modelu, 212
- klasa
 - BeitForImageClassification, 618
 - ConvNextForImageClassification, 622
 - ResNetForImageClassification, 624
 - SwinForImageClassification, 617, 620
 - ViTForImageClassification, 614
- klasteryzacja, 382
- klasyfikacja obrazów, 611, 613, 640–645, 651–654
- kodowanie
 - par bajtów, BPE, 194, 330, 342
 - pozycyjne, positional encoding, 73–78, 95
- kontrola jakości, 322, 325, 346
- konwolucyjna sieć neuronowa, CNN, 43
- kreowanie pomysłów, 664

L

- LIME, 306
 - interfejs wizualizacji, 307
 - wizualizacja transformera, 306
- LIT, Language Interpretability Tool, 286
 - uruchamianie, 311
 - wizualizacja PCA, 310
- Llama 2, 674
 - automatyzacja projektowania promptów, 674, 675
- LLM, Large Language Models, 27, 66

Ł

łańcuchy Markowa, 42

M

macierz pomyłek, 53

MCC, Matthews Correlation Coefficient,
111, 184, 185

miara

dokładności, 111

F1, 111

Microsoft Designer

tworzenie klipu wideo, 689

zamiana obrazu, 687

Midjourney, 658, 659

API Discorda, 682

tworzenie obrazów, 680, 682, 685

MLM, Masked Language Modeling, 159, 162

MMLU, Massive Multitasking Language
Understanding, 482, 490

MMO, Massively Multiplayer Online, 699

model

BEiT, 618

BERT, 157, 194

ChatGPT, 229, 244

CLIP, 549

ConvNext, 622

DALL-E, 554

GitHub Copilot, 245

GPT, 241

GPT-4, 249

KantaiBERT, 194, 195

Llama 2, 674

LLM, 446

LLM OpenAI, 313

NLP, 101

Original Transformer, 67

PaLM, 451

PaLM 2, 452

ResNet, 624

RoBERTa, 194

Stability AI, 588

Stable Diffusion, 573, 690

Swin, 616, 620

T5, 415, 421

TimeSformer, 592

ViT, 540, 614, 626

modele

generatywne, 44

łączenie, 658

obsługi klienta, 215

oparte na jednym tokenie, 47

OpenAI, 60, 229, 234, 241, 249, 313

asystenty AI, 244

prompty ogólnego przeznaczenia, 247

podstawowe, Foundation Models, 49

samoobsługowe, 391

specyficzne dla zadania, 390

tekst – tekst, 417

transdukcji, 71

ukierunkowane na konkretne zadania, 48

uniwersalne, 391

wymagające programowania, 391

modelowanie MLM, 159

MRPC, Microsoft Research Paraphrase
Corpus, 126

N

narzędzia do łagodzenia zagrożeń, 520

narzędzie, *Patrz także* pakiet, technika

AutoTrain, 598

BeautifulSoup, 271

BertViz, 287

exBERT, 298

LIT, 309

SHAP, 285, 299

TensorFlow Embedding Projector, 363

WandB, 501

neurony multimodalne, 535

NLP, Natural Language Processing, 27, 41

NLTK, Natural Language Toolkit, 141, 353

NLU, Natural Language Understanding, 98,
101, 417

normalizacja Post-LN, 92, 96

NSP, next-sentence prediction, 162

O

odpowiadanie na pytania, QA, 469

ograniczenia sprzętowe, 167

OOV, out of vocabulary, 139

Original Transformer, 67

stos dekodera, 93

kodowanie pozycyjne, 95

podwarstwa główna wielogłowicowej

maskowanej uwagi, 93

podwarstwa główna

wielogłowicowej uwagi, 95

- podwarstwa główna ze sprzężeniem w przód, FNN, 96
- podwarstwa osadzeń wejść, 95
- warstwa normalizacji, Post-LN, 96
- stos kodera, 70
 - kodowanie pozycyjne, 73
 - podwarstwa główna wielogłowicowej uwagi, 79
 - podwarstwa główna ze sprzężeniem w przód, FNN, 93
 - podwarstwa osadzeń wejść, 71
 - warstwa normalizacji, Post-LN, 92
- szkolenie, 96
- wydajność, 96
- osadzanie tekstu, 353
 - analiza przestrzeni wektorowej, 360
 - dostęp do słowa, 359
 - moduł Word2Vec, 356
 - narzędzie TensorFlow Projector, 363
 - opis modelu, 357
 - pakiet Gensim, 356
 - pakiet NLTK, 353
 - tokenizer Punkt, 354
 - wektor słów, 359
- osadzenia
 - implementacja wyszukiwania, 367
 - funkcja zadawania pytań, 375
 - model GPT, 368
 - pakiet tiktoken, 367
 - przygotowywanie danych, 372
 - żądanie wyszukania, 373
- Ada, 379
 - obliczanie, 381
 - zapisywanie, 381
- LLM, 350, 352
- osadzenie
 - wejść, 71
 - wyjść, 95
- oznaczanie ról semantycznych, SRL, 388

P

- pakiet
 - Gensim, 356
 - json, 271
 - moviepy, 635
 - NetworkX, 45
 - NLTK, 353
 - PyTorch, 550
 - re, 271
 - requests, 271

- T2T, 144
- tiktoken, 367
- PaLM, Pathways Language Model, 446, 451
- PaLM 2, 452
 - analiza tonu, 473
 - API, 468
 - asystent Vertex AI, 465
 - dostrajanie, 484
 - generowanie kodu, 477
 - interfejs Vertex AI, 462
 - odpowiadanie na pytania, 469
 - parametry próbkowania, 463
 - podsumowanie dialogu, 472
 - utworzenie kontenera, 483
 - zadania wielokrotnego wyboru, 475
 - zadanie pytanie – odpowiedź, 471
- parsowanie żądań użytkownika, 527
- PCA, Principal Component Analysis, 286, 310
- PE, positional encoding, 74
- platforma
 - BIG-bench, 490
 - Hugging Face, 598, 601
 - Vertex AI, 468
- POC, 690
- podejmowanie decyzji
 - wytyczne, 58
- polisemia, 325
- prefiks, 419
- proces decyzyjny Markowa, 42
- procesor
 - główny, CPU, 32
 - graficzny, GPU, 32
 - tensorowy, TPU, 32
- prognozowanie, 181, 595
- prompty, 231
 - automatyzacja projektowania, 668, 674
 - generowanie, 694
 - projektowanie, 352
 - ogólnego przeznaczenia, 247
- przepływ pracy, 703
- przetwarzanie
 - języka naturalnego, NLP, 27, 41
 - zbioru danych, 135

Q

- QA, question answering, 469

R

RAG, Retrieval Augmented Generation, 228, 257, 260, 267, 350, 520
 regurgitacja, 557
 rekurencyjna sieć neuronowa, RNN, 41
 replikacja, 503
 reprezentacja pośrednia, IR, 450
 ResNet, Residual Network, 624
 RLHF, Reinforcement Learning from Human Feedback, 520
 RNN, Recurrent Neural Networks, 41
 rozbieżne skojarzenia semantyczne, 561, 569
 rozumienie języka naturalnego, NLU, 98, 101, 417
 Runway Gen-2
 dane wyjściowe Midjourney, 660
 łączenie modeli, 658

S

SaaS, Software as a Service, 60
 samoocena, 491, 501
 samouwaga, 31
 SHAP, SHapley Additive exPlanations, 285, 299
 interpretacja transformerów Hugging Face, 302
 wykres, 303
 sieć
 neuronowa
 konwolucyjna, CNN, 43
 rekurencyjna, RNN, 41
 ze sprzężeniem w przód, FFN, 93
 słowa spoza słownika, OOV, 139
 słownik, 328
 SME, subject matter expert, 149
 specjalista AI, 55, 56
 SRL, Semantic Role Labeling, 388
 analiza orzeczeń, 400
 definicja, 392
 kwestionowanie zakresu, 399
 ponowna definicja, 401
 wizualizacja, 393
 SST-2, Stanford Sentiment TreeBank, 125
 Stable Diffusion
 generatywne kreowanie pomysłów, 690
 generowanie obrazów, 696
 zamiana tekstu na obraz, 576
 zamiana tekstu na wideo, 588
 zautomatyzowany proces tworzenia, 691

streszczenia tekstu, 415, 431
 z użyciem ChatGPT, 439
 z użyciem modelu T5, 423
 szkolenie tokenizera, 194, 198
 sztuczna inteligencja, 68
 ogólna, AGI, 490
 dyskryminatywna, 48
 funkcjonalna ogólna, F-AGI, 490, 630
 generatywna, 48, 215, 228

T

T5, 421
 architektura modelu, 421
 inicjalizowanie modelu, 425
 struktura, 420
 tworzenie streszczeń tekstu, 423, 431, 438
 ujednolicenie formatu wejściowego, 420
 wybór modelu transformera, 423
 technika
 MLM, 159, 162
 RAG, 228, 257, 260, 267, 350, 520
 RLHF, 520
 SHAP, 285, 299
 SRL, 388
 VR-AI, 699
 tensor torch, 173
 TensorFlow Projector, 363
 test porównawczy
 GLUE, 113, 114
 SuperGLUE, 113–117
 WMT, 113
 TimeSformer
 tworzenie prognoz, 595
 zamiana wideo na tekst, 592
 Tłumacz Google, 61, 149
 interfejs Ajax API, 150
 tłumaczenia
 maszynowe, 132, 134
 inicjalizowanie modelu, 147
 ocena, 135
 ocena według BLEU, 141
 tworzenie modelu transformera, 145
 z użyciem Gemini, 152
 z użyciem Google Trax, 144
 z użyciem Tłumacza Google, 149
 wykonywane przez ludzi, 133
 tokenizacja, 326
 oparta na białych spacjach, 337
 oparta na wyrażeniach regularnych, 337

- słów, 336, 354
 - Unigram modelu języka, 340
 - wielowyrazowa, 338
 - zdań, 336, 354
 - tokenizer, 71, 171
 - BERT, 171
 - Punkt, 338, 354
 - SentencePiece, 341
 - Treebank, 337
 - Word2Vec, 325
 - WordPiece, 334, 343
 - tokenizery
 - badanie, 343
 - oparte na podwyrazach, 339
 - podwyrazów, 320, 334
 - szkolenie, 198
 - wykrywanie typu, 343
 - wyrazów i zdań, 334, 335
 - tokeny, 44, 47, 103
 - wstępne przetwarzanie, 354
 - zarządzanie, 530
 - TPU, Tensor Processing Unit, 32
 - TPU-LLM, 40
 - transdukcje, 71, 131–134
 - transformer, 27, 42, 47, 161, *Patrz także*
 - Original Transformer, model, modele
 - multimodalny CLIP, 549
 - architektura modelu, 549
 - kod, 550
 - zamiana tekstu na wideo, 590
 - obsługi klienta RoBERTa, 215
 - tłumaczeniowy
 - dekodowanie wyjścia, 147
 - detokenizowanie, 148
 - Google Trax, 144
 - inicjalizowanie modelu, 147
 - Tłumacz Google, 149
 - tokenizowanie zdania, 147
 - tworzenie modelu, 145
 - wyświetlanie tłumaczenia, 148
 - wizyjny, ViT, 536
 - architektura, 536
 - działanie transformera, 536
 - kod, 539
 - koder, 538
 - konfiguracja i kształty, 544
 - logity, 543
 - symulator ekstraktora cech, 540
 - transformery
 - architektura modelu, 49, 69
 - badanie potencjału modelu, 110
 - dane, 49
 - dopasowywanie tokenizerów, 320
 - dopasowywanie zbiorów danych, 320
 - działanie neuronów, 313
 - GPT, 234
 - Hugging Face, 97, 168, 197
 - inżynieria zapytań, 50
 - LLM, 56
 - miliardy parametrów, 235
 - moc obliczeniowa, 50
 - moderowanie wejścia i wyjścia, 521
 - narzędzia do interpretacji działania, 285
 - ocenianie modelu, 111, 112
 - ograniczenia, 316
 - pomiar wydajności modelu, 113
 - T5, 415, 425
 - tekst – tekst, 418
 - testy porównawcze, 112
 - tłumaczenia maszynowe, 131
 - uruchamianie zadań końcowych, 124
 - wbudowane, 56
 - wizualizacja, 287, 304, 306, 309
 - wskaźniki, 111
 - współczynniki, 304
 - wstępne szkolenie, 194
 - wykorzystanie tokenizerów, 334
 - zadania końcowe, 110
 - złożoność zarządzania, 390
 - TSV, Tab-Separated Values, 363
- ## U
- uczenie
 - nadzorowane, 47
 - nienadzorowane, 47
 - przez wzmacnianie, 520
 - słownikowe, 304
 - transferowe, 379
- ## V
- Vertex AI PaLM 2, 462
 - ViT, Vision Transformer, 540
- ## W
- WandB, 501
 - warstwa
 - liniowa, 96
 - rekurencyjna, 31, 32

warstwa

- stosu dekodera, 94
- stosu kodera, 70, 160
- uwagi, 30, 32

wielogłowicowa uwaga, 79–92, 95, 103, 162

- architektura, 79
- prawdopodobieństwa dla par słów, 104–107

wizualizacja

- PCA, 310
- SRL, 393
- transformera
 - interfejs LIT, 309
 - poprzez uczenie słownikowe, 304
 - z użyciem BertViz, 287
 - z użyciem LIME, 306

UMAP, 310

związków między słowami, 296

WMT, Workshop on Machine Translation, 113, 132

WSC, Winograd Schema Challenge, 123, 127

współczynnik korelacji Matthews'a, MCC, 111, 184, 185

wstępne przetwarzanie, 321

tokenów, 354

wstępne szkolenie modelu

- agenta czatu, 221
- BERT, 161, 164
- KantaiBERT, 212
- obsługi klienta, 215
- transformera, 194

wybór

- API, 61
- modelu transformera, 62
- platformy chmurowej, 62

wyglądanie, 143

chencherry, 144

Z

zadania

- Auto-BIG-bench, 497
- DAT, 565
- generowania streszczeń, 415
- końcowe, downstream tasks, 110, 124
- SRL, 391
- wielokrotnego wyboru, 475

zadanie

- Bank zaangażowania, 118
- BoolQ, 117
- CoLA, 124
- COPA, 116
- MLM, 213
- MRPC, 127
- MultiRC, 118
- NLP, 102, 254
- ReCoRD, 120
- RTE, 122
- Schemat Winograda, 123, 127
- SST-2, 126
- typu pytanie – odpowiedź, 471
- WiC, 122

zagrożenia, 505

- cyberbezpieczeństwo, 519
- dezinformacja, 513
- łagodzenie, 520
- naruszenia prywatności, 518
- nieprawdziwe odpowiedzi, halucynacje, 506
- ryzykowne zachowania emergentne, 511
- treści szkodliwe, 516
- wywieranie wpływu na opinię publiczną, 514
- zapamiętywanie, 506, 511

zamiana

- tekstu na obraz
 - z użyciem Stable Diffusion, 576
 - za pomocą API Stable Diffusion, 584
- tekstu na wideo
 - z użyciem CLIP, 590
 - z użyciem Stability AI, 588
- wideo na tekst, 592

zapytanie, prompt, 231

zarządzanie tokenami, 530

zbiór danych

- Amazon Fine Food Reviews, 379
- CoLA, 169
- Kaggle Customer Support on Twitter, 216
- SST-2, 125
- walidacyjny, 609, 635
- WMT, 135

złożoność czasowa $O(1)$, 29

związki między słowami, 31

PROGRAM PARTNERSKI

— GRUPY HELION —



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion

Sztuczna inteligencja, która widzi i mówi — przekonaj się, jak to działa!

Transformery zrewolucjonizowały przetwarzanie języka naturalnego, analizę obrazów i komputerowe widzenie. Oparte na transformerach duże modele generatywne dostępne za pośrednictwem systemu ChatGPT z GPT-4V w zadaniach przetwarzania tekstu i obrazów przewyższają wydajność człowieka. Aby uczestniczyć w tej nowej erze technologicznej, musisz zrozumieć, jak działają transformery.

Tę książkę docenią praktycy: analitycy danych i inżynierowie uczenia maszynowego. Opisano w niej różne architektury transformerów — od pierwszych modeli podstawowych po najnowsze osiągnięcia w generatywnej sztucznej inteligencji. Dzięki lekturze nauczysz się wstępnego szkolenia i dostrajania modeli LLM, a także pracy nad różnymi przypadkami użycia. Poznasz takie problemy jak halucynacje i zagrożenia prywatności, a następnie dowiesz się, jak je łagodzić. W książce pokazano ponadto, jak poprawiać dokładność modeli LLM i uzyskiwać większą kontrolę nad generowanymi przez nie wynikami. Nie zabrakło ciekawych szczegółów dotyczących modeli generatywnych opartych na transformerach, modeli wizyjnych i architektur multimodalnych, jak również opisu najlepszych praktyk.

Najciekawsze tematy:

- wstępne szkolenie i dostrajanie modeli LLM
- platformy: Hugging Face, OpenAI i Google Vertex AI
- tokenizery i najlepsze praktyki wstępnego przetwarzania danych językowych
- techniki łagodzenia halucynacji
- wizualizacja aktywności modeli transformerów z użyciem systemów BertViz, LIME i SHAP
- modele wizyjne i multimodalne oparte na transformerach: CLIP, DALL-E 2, DALL-E 3 i GPT-4V

Denis Rothman jest autorem jednego z pierwszych systemów kodowania i osadzania, a także twórcą botów kognitywnych AI. Optymalizował systemy obrony taktycznej z wykorzystaniem sztucznej inteligencji dla Airbus. Opracował optymalizator zasobów AI dla IBM, dzięki czemu powstał dobrze znany Advanced Planning and Scheduling (APS).

Helion 	KOD KORZYŚCI Sięgnij po więcej! 	
 helion.pl	ISBN 978-83-289-2050-7	
 HELION S.A. ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	 9 788328 920507	
Cena: 149,00 zł		

<packt>