

Parametry i mechanizmy modeli językowych

Hasło	Strona
1. Temperature — Temperatura generowania	2
2. Max Tokens — Maksymalna liczba tokenów	3
3. Top-p (Nucleus Sampling)	4
4. Top-k Sampling	5
5. Frequency Penalty — Kara za częstotliwość	6
6. Presence Penalty — Kara za obecność	7
7. Repetition Penalty (HuggingFace)	8
8. Beam Search — Przeszukiwanie wiązką	9
9. Greedy Decoding	9
10. Tokenizacja — czym jest token?	10
11. Perplexity — Perpleksja	11
12. Embedding — Osadzanie wektorowe	12
13. Attention — Mechanizm uwagi	13
14. Fine-tuning vs In-context Learning	14
15. Prompt Engineering — Inżynieria promptów	15

01 Temperature

Temperatura generowania — kontrola losowości i kreatywności

□ Temperature

Parametr skalujący rozkład prawdopodobieństwa tokenów

Zakres wartości: 0.0 – 2.0 (typowo 0.1 – 1.5)

Domyślna wartość: 1.0 (bez modyfikacji rozkładu)

Parametr w HuggingFace: temperature= (wymaga do_sample=True)

Parametr w OpenAI API: temperature=

Jak działa temperatura?

Przed wyborem kolejnego tokenu model oblicza logity (surowe wartości dla każdego tokenu w słowniku). Następnie są one przekształcane na prawdopodobieństwa przez funkcję softmax. Temperatura T modyfikuje ten rozkład PRZED zastosowaniem softmax:

$$p(\text{token}_i) = \exp(\text{logit}_i / T) / \sum \exp(\text{logit}_j / T)$$

Intuicja:

- $T < 1.0$ → logity są dzielone przez małą liczbę → różnice się powiększają → rozkład staje się ostrzejszy → model wybiera bardziej przewidywalnie
- $T = 1.0$ → brak modyfikacji → domyślne zachowanie modelu
- $T > 1.0$ → logity są dzielone przez dużą liczbę → różnice maleją → rozkład staje się płaszcy → model dobiera tokeny bardziej losowo

Wartość T	Zachowanie modelu	Typowe zastosowanie	Ryzyko
0.0 – 0.2	Prawie deterministyczny, wybiera najwyżej prawdopodobny token	Fakty, pytania zamknięte, tłumaczenie	Powtarzalność, brak kreatywności
0.3 – 0.6	Spójny tekst z odrobiną zmienności	Streszczenia, opisy, dokumentacja	Może być za mało kreatywny dla historii
0.7 – 0.9	Balans kreatywności i spójności	Pisanie kreatywne, chatboty, storytelling	Rzadkie błędy semantyczne
1.0 – 1.2	Wysoka kreatywność, więcej niespodziewanych wyborów	Generowanie pomysłów, brainstorming	Obniżona spójność
1.3 – 2.0	Chaotyczny, często niespójny tekst	Eksperymentowanie, badania	Błędy gramatyczne, nonsens

► Przykład: ten sam prompt, różne temperatury

Prompt: 'The sky at night is'

T=0.1 → 'The sky at night is dark and full of stars.' (deterministyczne)

T=0.7 → 'The sky at night is a canvas of constellations and mystery.'
T=1.4 → 'The sky at night is burning crystalline with ancient forgotten murmurs.'

Temperatura a inne parametry

Temperatura NIE działa w trybie greedy decoding (`do_sample=False`). Musi być połączona z `do_sample=True`. Temperatura współdziała z top-p i top-k — najpierw zastosowana jest temperatura, a następnie filtracja top-p lub top-k.

```
# Temperatura w HuggingFace
output = model.generate(
    input_ids,
    do_sample=True,          # WYMAGANE dla temperatury
    temperature=0.8,        # wartość temperatury
    max_new_tokens=100,
)

# Temperatura w OpenAI API
response = client.chat.completions.create(
    model='gpt-4',
    messages=[{'role': 'user', 'content': 'Hello!'}],
    temperature=0.7,        # 0.0-2.0
)
```

02 Max Tokens

Maksymalna liczba tokenów — limit długości generowanego tekstu

□ Max Tokens

Górna granica liczby tokenów, które model może wygenerować w jednej odpowiedzi

Parametr w HuggingFace: max_new_tokens= (tylko nowe) lub max_length= (cała sekwencja)

Parametr w OpenAI API: max_tokens=

Zakres wartości: 1 – do limitu kontekstu modelu (GPT-4: 128 000, GPT-2: 1 024)

Uwaga: max_new_tokens NIE wlicza tokenów promptu; max_length tak!

Token — podstawowa jednostka

Token to NIE jest to samo co słowo. Modele językowe dzielą tekst na tokeny za pomocą BPE (Byte Pair Encoding) lub WordPiece. Jeden token to zazwyczaj 3/4 słowa (angielskiego). W praktyce:

Tekst	Tokeny	Liczba tokenów
Hello, world!	['Hello', ',', ' world', '!']	4
Artificial intelligence	['Art', 'ificial', ' intelligence']	3
unhappiness	['un', 'happiness']	2
100 słów angielskich	—	~75 tokenów
1 strona A4 (500 słów)	—	~375 tokenów

max_new_tokens vs max_length

► Ważna różnica!

Prompt: 'Tell me about Python' (5 tokenów)

max_length=50 → generuje ŁĄCZNIE 50 tokenów = 45 nowych

max_new_tokens=50 → generuje 50 NOWYCH tokenów (łącznie 55)

Zawsze używaj max_new_tokens aby mieć kontrolę nad długością odpowiedzi!

Wpływ na koszt i jakość

- Niski limit (20–50 tokenów): odpowiedzi urwane, niepełne zdania; dobre dla krótkich faktów
- Średni limit (100–300 tokenów): odpowiedzi kompletne; optimum dla chatbotów
- Wysoki limit (1000+ tokenów): długie dokumenty, artykuły; wyższy koszt API

- W API (OpenAI, Anthropic) koszt jest proporcjonalny do liczby tokenów wejściowych + wyjściowych

```
# Przykład kontroli dlugosci w HuggingFace
output = model.generate(
    input_ids,
    max_new_tokens=150,    # generuj max 150 nowych tokenow
    min_new_tokens=20,     # generuj min 20 tokenow (unikaj przedwczesnego konca)
    early_stopping=True,   # zatrzymaj przy tokenie EOS
)

# Policzenie tokenow w tekście
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained('gpt2')
text = 'Hello, this is a sample text for token counting.'
tokens = tokenizer.encode(text)
print(f'Liczba tokenow: {len(tokens)}') # ~11
```

Okno kontekstu (Context Window)

Każdy model ma maksymalny limit ŁĄCZNEJ liczby tokenów (prompt + odpowiedź). Jeśli prompt jest długi, należy zostawić miejsce na odpowiedź:

Model	Okno kontekstu	Max tokens odpowiedzi
GPT-2	1 024 tokenów	do 1 024 minus prompt
GPT-3.5-turbo	16 385 tokenów	4 096
GPT-4o	128 000 tokenów	16 384
Claude 3.5 Sonnet	200 000 tokenów	8 192
BERT (encoder)	512 tokenów	nie dotyczy (nie generuje)
FLAN-T5-base	512 tokenów	~200

03 Top-p

Nucleus Sampling — próbkowanie jądra rozkładu

□ Top-p (Nucleus Sampling)

Wybiera tokeny z minimalnego zestawu, który pokrywa prawdopodobieństwo p

Zakres wartości: 0.0 – 1.0

Typowe wartości: 0.7 – 0.95

Parametr w HuggingFace: top_p= (wymaga do_sample=True)

Parametr w OpenAI API: top_p=

Oryginalna nazwa: Nucleus Sampling (Holtzman et al., 2020)

Jak działa Top-p?

Algorytm top-p sortuje wszystkie tokeny od najwyższego do najniższego prawdopodobieństwa, a następnie wybiera najmniejszy zestaw tokenów, których łączne prawdopodobieństwo wynosi co najmniej p . Tylko z tego zestawu ('jądra') losowany jest następny token:

► Krok po kroku dla top_p=0.9

Rozkład: [token_A=0.45, token_B=0.25, token_C=0.15, token_D=0.10, token_E=0.05]

Sortowanie: A(0.45) → B(0.25) → C(0.15) → D(0.10) → E(0.05)

Kumulatywne: A(0.45) → A+B(0.70) → A+B+C(0.85) → A+B+C+D(0.95) ≥ 0.9

Jądro: {A, B, C, D} — cztery tokeny o łącznym $p=0.95 \geq 0.9$

Token_E jest WYKLUCONY (kumulatywne przekroczyło 0.9 przed nim)

Losowanie: wylosuj z {A,B,C,D} po renormalizacji prawdopodobieństw

Wartość top_p	Efekt	Kiedy stosować
0.1 – 0.5	Jądro bardzo małe, tylko najbardziej prawdopodobne tokeny	Fakty, tłumaczenie, ściśle odpowiedzi
0.7 – 0.85	Umiarkowane jądro, dobre balansowanie	Chatboty, opisy, summaryzacja
0.9 – 0.95	Szerokie jądro, dużo kreatywności	Pisanie kreatywne, storytelling
1.0	Wszystkie tokeny dostępne (brak filtracji)	Nie zalecane — może generować nonsens

Top-p vs Top-k — kluczowa różnica

Top-k wyklucza wszystko poza k najlepszymi tokenami. Top-p adaptuje rozmiar zestawu do rozkładu: gdy model jest pewny (jeden token dominuje), jądro jest małe; gdy model jest niepewny (wiele tokenów podobnych), jądro jest duże. To sprawia, że top-p jest bardziej elastyczny.

Cecha	Top-k	Top-p
Rozmiar zestawu	Stały (zawsze k tokenów)	Adaptacyjny (zależy od rozkładu)
Gdy model pewny	Może uwzględniać mało-prawdopodobne tokeny	Jądro automatycznie małe
Gdy model niepewny	Może wykluczać wiele równoprawdopodobnych opcji	Jądro automatycznie duże
Typowa wartość	top_k=50	top_p=0.9
Popularność	Starsza technika	Nowsza, obecnie preferowana

Top-p w HuggingFace
output = model.generate(input_ids, do_sample=True, top_p=0.92, # nucleus sampling temperature=0.8, # mozna laczyc z temperatura max_new_tokens=100,)
Czy laczyc top_p z temperatura?
Tak – temperatura modyfikuje rozkład PRZED wyznaczeniem jądra.
Typowe ustawienie: temperature=0.8, top_p=0.9

04 Top-k

Próbkowanie z k najlepszych tokenów

□ Top-k Sampling

Ogranicza wybór do k tokenów o najwyższym prawdopodobieństwie

Zakres wartości: 1 – rozmiar słownika (~50 000)

Typowe wartości: 20 – 100 (najpopularniejsze: 50)

Parametr w HuggingFace: `top_k=` (wymaga `do_sample=True`)

top_k=1: Odpowiednik greedy decoding

top_k=0: Wyłącza filtrację (bez limitu)

Jak działa Top-k?

► Algorytm Top-k dla $k=3$

Słownik: ['cat':0.40, 'dog':0.25, 'bird':0.15, 'fish':0.12, 'ant':0.08]

Top-3: {'cat':0.40, 'dog':0.25, 'bird':0.15}

Renormalizacja: {'cat':0.50, 'dog':0.31, 'bird':0.19}

Losowanie z renormalizowanych prawdopodobieństw.

'fish' i 'ant' są całkowicie wykluczone niezależnie od kontekstu.

Wartość k	Zachowanie	Uwagi
$k=1$	Identyczne z greedy — zawsze najwyższy token	Deterministyczne, brak losowości
$k=10$	Mały słownik, dość konserwatywne	Spójne ale mało kreatywne
$k=50$	Standardowe ustawienie, dobre balansowanie	Najpopularniejszy wybór w praktyce
$k=100$	Szerszy wybór, więcej różnorodności	Może wprowadzać nieoczekiwane tokeny
$k=0$ (wyłącz)	Wszystkie tokeny dostępne	Używaj razem z <code>top_p</code> jako zabezpieczenie

Słabość top-k: gdy rozkład jest bardzo skoncentrowany (model pewny), $k=50$ może mimo to włączać tokeny o minimalnym prawdopodobieństwie. Gdy rozkład jest płaski, $k=50$ może arbitralnie odcinać równoprawdopodobne tokeny. Dlatego top-p jest uważany za lepszą alternatywę.

```
# Top-k w HuggingFace
output = model.generate(
    input_ids,
    do_sample=True,
    top_k=50,           # standardowe ustawienie
```

```
temperature=0.9,  
max_new_tokens=80,  
)  
  
# Kombinacja top_k + top_p (podwójne zabezpieczenie):  
output = model.generate(  
    input_ids,  
    do_sample=True,  
    top_k=50,          # max 50 tokenow  
    top_p=0.9,         # z tych 50 wybierz jądro 90%  
    temperature=0.8,  
    max_new_tokens=80,  
)
```

05 Frequency Penalty

Kara za częstotliwość — redukcja powtórzeń proporcjonalna do liczby wystąpień

□ Frequency Penalty

Odejmuje od logitu tokenu wartość proporcjonalną do jego dotychczasowej liczby wystąpień

Zakres wartości: -2.0 – 2.0 (OpenAI API); zazwyczaj 0.0 – 1.5

Domyślna wartość: 0.0 (brak kary)

Parametr w OpenAI API: frequency_penalty=

Odpowiednik w HuggingFace: repetition_penalty= (nieco inne działanie)

Efekt ujemnej wartości: Zachęca do powtarzania tokenów

Mechanizm działania

Kara za częstotliwość modyfikuje logit tokenu w sposób kumulatywny — im częściej token pojawił się w dotychczas wygenerowanym tekście, tym większą karę otrzymuje:

$$\text{logit_adjusted}(\text{token}) = \text{logit}(\text{token}) - \frac{\text{frequency_penalty}}{\text{count}(\text{token})}$$

► Przykład działania frequency_penalty=0.5

Token 'the' pojawił się już 4 razy w wygenerowanym tekście.

Oryginalny logit 'the' = 3.2

Kara = $0.5 \times 4 = 2.0$

Zmodyfikowany logit = $3.2 - 2.0 = 1.2$

Efekt: prawdopodobieństwo wyboru 'the' znacznie spada z każdym użyciem.

Wartość	Efekt	Zastosowanie
0.0	Brak kary — model może swobodnie powtarzać słowa	Domyślne, poezja z refrenami
0.3 – 0.5	Lekka kara — redukuje nadmierne powtórzenia	Teksty techniczne, artykuły
0.7 – 1.0	Silna kara — wymuszona różnorodność słownictwa	Pisanie kreatywne, eseje
1.2 – 2.0	Bardzo silna kara — model unika nawet koniecznych powtórzeń	Niezalecane — może zaburzyć spójność
< 0	Model chętniej powtarza tokeny	Specjalne zastosowania (np. slogany)

Frequency Penalty vs Presence Penalty — kluczowa różnica

Cecha	Frequency Penalty	Presence Penalty
Formuła	$\text{logit} - \text{penalty} \times \text{count}$	$\text{logit} - \text{penalty}$ (jeśli $\text{count} > 0$)
Zależność od count	Rośnie z każdym użyciem (kumulatywna)	Stała — nie zależy od liczby użyc
Token użyty 1 raz	Mała kara	Pełna kara
Token użyty 10 razy	10× większa kara niż przy 1 użyciu	Ta sama kara co przy 1 użyciu
Cel	Redukcja nadmiernych powtórzeń	Zachęcanie do nowych tematów/słów
Pytanie	'Jak często użyto tego słowa?'	'Czy to słowo już się pojawiło?'

```
# Frequency Penalty w OpenAI API
response = client.chat.completions.create(
    model='gpt-3.5-turbo',
    messages=[{'role':'user','content':'Write a 200-word essay on AI.'}],
    frequency_penalty=0.5, # redukcja powtórzeń słów
    presence_penalty=0.3,  # zachęcanie do różnorodnych tematów
)

# W HuggingFace odpowiednikiem jest repetition_penalty:
# (mechanizm inny – mnoży logity przez  $\text{penalty}^{\text{count}}$ )
output = model.generate(
    input_ids,
    repetition_penalty=1.3, # > 1.0 = kara; < 1.0 = nagroda
    max_new_tokens=100,
)
```

06 Presence Penalty

Kara za obecność — zachęcanie do wprowadzania nowych tematów

□ Presence Penalty

Odejmuje stałą karę od logitu każdego tokenu, który już wcześniej wystąpił

Zakres wartości: -2.0 – 2.0 (OpenAI API)

Domyślna wartość: 0.0 (brak kary)

Parametr w OpenAI API: presence_penalty=

Charakter: Kara binarna: 0 jeśli token nie wystąpił, stała C jeśli wystąpił

Różnica od frequency_penalty: Nie rośnie z kolejnymi użyciami — po pierwszym wystąpieniu kara jest stała

Mechanizm działania

$$\text{logit_adjusted}(\text{token}) = \text{logit}(\text{token}) - \text{presence_penalty} \times I(\text{count} > 0)$$

gdzie $I(\text{count} > 0) = 1$ jeśli token wystąpił już w tekście, 0 w przeciwnym razie.

► Porównanie frequency vs presence penalty

Niech penalty = 0.6, token 'AI' wystąpił 3 razy:

frequency_penalty: kara = $0.6 \times 3 = 1.8$ (rośnie!)

presence_penalty: kara = $0.6 \times 1 = 0.6$ (stała po pierwszym użyciu)

Wniosek: presence_penalty skuteczniej motywuje do nowych TEMATÓW, ale nie karze za wielokrotne użycie kluczowych słów artykułu.

Wartość	Efekt	Przykład wyjścia
0.0	Brak efektu, model naturalnie wybiera tokeny	Standard, bez ingerencji
0.3 – 0.5	Lekkie rozszerzenie tematyczne	Artykuł porusza kilka powiązanych wątków
0.7 – 1.0	Silne zachęcanie do nowych słów	Dywersyfikacja słownictwa, nowe aspekty tematu
1.5 – 2.0	Bardzo agresywne unikanie powtórzeń	Tekst może stać się niekoheren-tny tematycznie

Kiedy używać każdej kary?

Zadanie	Frequency Penalty	Presence Penalty	Uzasadnienie
Artykuł informacyjny	0.4	0.1	Redukuj powtórzenia, ale zachowaj temat
Pisanie kreatywne	0.6	0.5	Wysoka różnorodność: słów i tematów
Chatbot	0.3	0.3	Unikaj monotonii w dłuższych dialogach
Fakty/Q&A	0.0	0.0	Dokładność ważniejsza niż różnorodność
Podsumowanie	0.2	0.0	Redukuj powtórzenia, zachowaj słowa kluczowe
Kod programowania	0.0	0.0	Słowa kluczowe MUSZĄ się powtarzać

```
# Typowe kombinacje dla OpenAI API
```

```
# Do pisania kreatywnego:
```

```
params_creative = {  
    'temperature': 0.9,  
    'top_p': 0.95,  
    'frequency_penalty': 0.6,  
    'presence_penalty': 0.5,  
    'max_tokens': 500  
}
```

```
# Do odpowiedzi faktycznych:
```

```
params_factual = {  
    'temperature': 0.2,  
    'top_p': 0.8,  
    'frequency_penalty': 0.0,  
    'presence_penalty': 0.0,  
    'max_tokens': 200  
}
```

```
# Dla chatbota (balans):
```

```
params_chat = {  
    'temperature': 0.7,  
    'top_p': 0.9,  
    'frequency_penalty': 0.3,  
    'presence_penalty': 0.2,  
    'max_tokens': 300  
}
```

07 Repetition Penalty

Kara za powtórzenia w bibliotece HuggingFace Transformers

□ Repetition Penalty

Mnożnik logitów tokenów już wygenerowanych — mechanizm inny niż w OpenAI

Zakres wartości: 0.5 – 2.0 (1.0 = brak kary)

Typowa wartość: 1.1 – 1.5 (lekkie do silnego tłumienia powtórzeń)

Parametr w HuggingFace: repetition_penalty=

Ważne: Wartości > 1.0 KARZA; < 1.0 NAGRADZAJĄ powtórzenia

Mechanizm — inny niż w OpenAI!

HuggingFace używa INNEGO mechanizmu niż frequency/presence penalty w OpenAI. Zamiast odejmować, dzieli (lub mnoży) logit przez wartość kary:

```
logit_adjusted(token) = logit(token) / repetition_penalty    [jeśli  
                                                                logit > 0]  
  
logit_adjusted(token) = logit(token) × repetition_penalty    [jeśli  
                                                                logit < 0]
```

Intuicja: dla logitu dodatniego (token prawdopodobny) dzielenie przez liczbę > 1 zmniejsza logit. Dla logitu ujemnego (token mało prawdopodobny) mnożenie przez > 1 sprawia, że staje się jeszcze mniej prawdopodobny.

Wartość	Efekt	Kiedy stosować
< 1.0 (np. 0.7)	Wzmacnia powtórzenia	Slogany, refreny w piosenkach
= 1.0	Brak modyfikacji — standardowe zachowanie	Domyślne
1.1 – 1.3	Lekka kara — delikatne tłumienie powtórzeń	Chatboty, ogólne teksty
1.3 – 1.5	Średnia kara — wyraźna różnorodność	Artykuły, opisy produktów
1.5 – 2.0	Silna kara — model unika wszelkich powtórzeń	Pisanie kreatywne; ryzyko niespójności

```
# Repetition penalty w HuggingFace  
output = model.generate(  
    input_ids,  
    do_sample=True,  
    temperature=0.8,  
    repetition_penalty=1.3,    # 1.0 = brak kary  
    max_new_tokens=100,  
)
```

```
# No repeat ngram size – alternatywna metoda:
```

```
# Blokuje powtórzenie dowolnego n-gramu z dotychczasowego tekstu
```

```
output = model.generate(
```

```
    input_ids,
```

```
    no_repeat_ngram_size=3,    # żaden 3-gram nie może się powtórzyć
```

```
    max_new_tokens=100,
```

```
)
```

```
# Kombinacja obu:
```

```
output = model.generate(
```

```
    input_ids,
```

```
    repetition_penalty=1.2,
```

```
    no_repeat_ngram_size=3,
```

```
    max_new_tokens=100,
```

```
)
```

08-09 Beam Search & Greedy Decoding

Deterministyczne strategie dekodowania

Greedy Decoding

□ Greedy Decoding

W każdym kroku wybiera TOKEN o najwyższym prawdopodobieństwie

Cecha: Deterministyczne, szybkie, reproducibile

HuggingFace: `do_sample=False` (domyślne)

Wada: Może pomijać globalnie lepsze sekwencje; podatne na repetycje

Kiedy używać: Zadania z jednoznaczną odpowiedzią: tłumaczenie, klasyfikacja

Beam Search

□ Beam Search

Śledzi B równoległych 'wiązek' sekwencji i wybiera najlepszą

Parametr: `num_beams=B` (B=1 to greedy)

Typowe wartości: 4 – 8 beams

Zaleta: Lepsze sekwencje niż greedy; deterministyczne

Wada: B× wolniejszy od greedy; może generować powtarzające się frazy

Kiedy używać: Tłumaczenie maszynowe, summaryzacja, NMT

```
# Greedy (domyslnie)
out = model.generate(inputs, max_new_tokens=50) # do_sample=False

# Beam search z 4 wiązками
out = model.generate(inputs, max_new_tokens=50,
                    num_beams=4, early_stopping=True)

# Diverse beam search (rozne wiązki eksplorują rozne ścieżki)
out = model.generate(inputs, max_new_tokens=50,
                    num_beams=6,
                    num_beam_groups=2,
                    diversity_penalty=0.5)
```

10 Tokenizacja

Jak tekst jest zamieniany na liczby

Modele językowe nie widzą tekstu — widzą liczby całkowite (ID tokenów). Tokenizacja przekształca tekst na sekwencje liczb i z powrotem. Popularne algorytmy:

- BPE (Byte Pair Encoding) — używany przez GPT-2, GPT-4, RoBERTa; iteracyjnie łączy najczęstsze pary znaków
- WordPiece — używany przez BERT; podobny do BPE, ale z probabilistycznym kryterium
- SentencePiece — używany przez T5, FLAN-T5, BLOOM; działa na znakach unicode, nie słowach

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained('gpt2')

# Kodowanie: tekst -> ID tokenów
text = 'Artificial intelligence is fascinating!'
ids = tokenizer.encode(text)          # [3826, 7843, 4430, 318, 23061, 0]

# Dekodowanie: ID tokenów -> tekst
back = tokenizer.decode(ids)          # 'Artificial intelligence is fascinating!'

# Wyświetlenie tokenów
toks = tokenizer.tokenize(text)       # ['Art', 'ificial', ' intelligence', ...]

# Specjalne tokeny
print(tokenizer.bos_token)            # <|endoftext|> w GPT-2
print(tokenizer.eos_token)            # <|endoftext|> w GPT-2
```

11 Perplexity

Miara pewności i jakości modelu językowego

Perplexity (PPL) mierzy, jak dobrze model 'zna' dany tekst — jak zaskoczony jest modelem kolejnymi tokenami. Niska perplexity = model dobrze przewiduje tekst.

$$\text{PPL}(\text{tekst}) = \exp\left(-\left(1/N\right) \times \sum \log P(\text{token}_i \mid \text{kontekst})\right)$$

Tekst	Oczekiwana PPL GPT-2	Interpretacja
Naturalne angielskie zdanie	~50 – 100	Model dobrze przewiduje
Losowe słowa w złej kolejności	~500 – 5000	Model jest zaskoczony
Tekst spoza domeny treningowej	Wyższa PPL	Model mniej pewny
Tekst z treningu (wyuczony na pamięć)	< 10	Możliwe memoryzowanie

12–15 Embedding, Attention, Fine-tuning, Prompt Engineering

Pozostałe kluczowe koncepcje

Embedding — Osadzanie wektorowe

Embedding to gęsty wektor liczb rzeczywistych reprezentujący token, słowo lub cały tekst w przestrzeni wielowymiarowej (zazwyczaj 768 – 4096 wymiarów). Semantycznie podobne słowa mają bliskie wektory:

```
from transformers import AutoTokenizer, AutoModel
import torch

tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')
model = AutoModel.from_pretrained('bert-base-uncased')

def get_embedding(text):
    inputs = tokenizer(text, return_tensors='pt')
    with torch.no_grad():
        outputs = model(**inputs)
    # Srednia pooling po tokenach (CLS token lub srednia warstwy ostatniej)
    return outputs.last_hidden_state.mean(dim=1).squeeze()

e1 = get_embedding('king')    # wektor 768D
e2 = get_embedding('queen')
e3 = get_embedding('car')
sim_kq = torch.cosine_similarity(e1,e2,dim=0).item()
sim_kc = torch.cosine_similarity(e1,e3,dim=0).item()
print(f'king-queen: {sim_kq:.3f} | king-car: {sim_kc:.3f}')
```

Attention — Mechanizm uwagi

Mechanizm uwagi (Vaswani et al. 2017) pozwala każdemu tokenowi w sekwencji 'przyglądać się' wszystkim innym tokenom i ustalić, które są dla niego istotne. Wynik to ważona suma wartości:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \times V$$

- Q (Query) — 'co szukam?' — reprezentacja bieżącego tokenu
- K (Key) — 'co oferuję?' — reprezentacja każdego tokenu do porównania
- V (Value) — 'co przekazuję?' — informacja, którą przekazuję jeśli jestem wybrany
- $\sqrt{d_k}$ — skalowanie, aby zapobiec nasyceniu softmax dla dużych wymiarów

Fine-tuning vs In-context Learning

Cecha	Fine-tuning	In-context Learning (Prompting)
Zmiana wag modelu	Tak — wagi są aktualizowane	Nie — wagi zamrożone
Wymagane dane	100+ przykładów labeled	0–10 przykładów w prompcie
Koszt	Wysoki (GPU, czas)	Minimalny
Jakość	Wyższa dla specjalistycznych zadań	Niższa, ale zaskakująco dobra
Deployment	Osobny model do hostowania	Ten sam model bazowy
Kiedy stosować	Specjalizacja, poufne dane, produkcja	Prototypowanie, ograniczone zasoby

Prompt Engineering — Inżynieria promptów

Prompt engineering to sztuka projektowania instrukcji wejściowych, które skłaniają model do generowania pożądaných odpowiedzi bez zmiany jego wag. Kluczowe techniki:

- Zero-shot: 'Translate to French: Hello' — brak przykładów
- Few-shot: 2–5 przykładów w prompcie przed właściwym pytaniem
- Chain-of-thought (CoT): 'Let's think step by step...' — wymusza rozumowanie
- Role prompting: 'You are an expert Python developer...'
- Negative prompting: 'Do not include any opinions...' — wykluczanie niechcianych treści

```
# Few-shot prompt dla klasyfikacji sentymentu
few_shot_prompt = '''
Classify the sentiment as POSITIVE or NEGATIVE.
```

```
Text: 'This product is amazing!' → POSITIVE
Text: 'Worst purchase ever.'      → NEGATIVE
Text: 'It works as expected.'     → POSITIVE
```

```
Text: 'I am very disappointed.' →
'''
```

```
# Model powinien kontynuować: ' NEGATIVE'
```