Implementacja transformera oraz eksperymenty z wariantami mechanizmu uwagi (Performer, Reformer) w zadaniach klasyfikacji tekstu

Dokumentacja projektu – część V

Bartłomiej Borycki, Michał Iwaniuk

16 stycznia 2026

Spis treści

1 Instrukcja instalacji

Poniżej przedstawiono kroki niezbędne do uruchomienia systemu w środowisku produkcyjnym.

1.1 Wymagania systemowe

- Python 3.12 lub nowszy
- CUDA (opcjonalnie, do treningu na GPU). Aby wykorzystać akcelerację GPU, upewnij się, że masz zainstalowane odpowiednie sterowniki CUDA oraz bibliotekę PyTorch z obsługą CUDA.

1.2 Instalacja środowiska

1. **Utworzenie wirtualnego środowiska Python:**

```
python -m venv .venv
source .venv/bin/activate   # Linux/macOS
# lub na Windows:
# .\.venv\Scripts\Activate.ps1
```

2. **Aktualizacja pip i instalacja zależności:**

```
pip install --upgrade pip
pip install -r requirements.txt
```

1.3 Zależności projektu

Plik `requirements.txt` zawiera następujące pakiety:

- `torch==2.8.0`
- `transformers==4.56.2`
- `tokenizers==0.22.1`
- `pandas==2.2.3`
- `numpy==1.26.4`
- `scikit-learn==1.6.1`
- `wandb==0.22.1`
- `PyYAML==6.0.2`
- `pytest==8.3.4`
- `datasets==4.3.0`
- `pydantic>=2.12.0`

2 Testy akceptacyjne

Tabela **??** prezentuje ocenę spełnienia wymagań niefunkcjonalnych określonych w sekcji 4 Dokumentu 1 (Laboratorium 1 – Uzasadnienie biznesowe).

Tabela **??** przedstawia ocenę spełnienia wymagań funkcjonalnych zdefiniowanych w sekcji 3 Dokumentu 1 (Laboratorium 1 – Uzasadnienie biznesowe).

Tabela 1: Wymagania niefunkcjonalne

Wymaganie	**Status**	**Komentarz**
WNF-1 — Wydajność i efektywność zasobowa		
Środowisko GPU (Colab)	Spełnione	Wykorzystano GPU *A100 40GB*
Wymóg kosztowy (Performer)	Częściowo spełnione *	Zgodnie z tabelami **??** i **??**
Wymóg kosztowy (Reformer)	Częściowo spełnione *	Zgodnie z tabelami **??** i **??**
Techniki optymalizacji	Spełnione	Zgodnie z sek. 4 Dok. 3
WNF-2 — Jakość, niezawodność i testowalność		
Jakość (SDPA)	Spełnione	Zgodnie z tabelą **??**
Jakość (Performer)	Częściowo spełnione	Zgodnie z tabelą **??**
Jakość (Reformer)	Spełnione	Zgodnie z tabelą **??**
Stabilność	Spełnione	Zgodnie z tabelą **??**
Testy komponentów	Spełnione	Zgodnie z sek. 1 Dok. 4
WNF-3 — Użyteczność i utrzymanie		
Dokumentacja	Spełnione	Zgodnie z sek. 3 Dok. 5
Struktura katalogów	Spełnione	Zgodnie z sek. 3 Dok. 5
Zgodność z PEP-8	Spełnione	-
Wersjonowanie	Spełnione	-
Rozszerzalność mechanizmów uwagi	Spełnione	Zgodnie z sek. 3.2 Dok. 3
Konfiguracja YAML	Spełnione	Zgodnie z sek. **??** Dok. 5
WNF-4 — Przenośność i kompatybilność		
Kompatybilność Python	Spełnione	Zgodnie z sek. 1 Dok. 5
Biblioteki	Spełnione	Zgodnie z sek. 1 Dok. 5
WNF-5 — Monitorowanie i obserwowalność		
Monitorowanie W&B	Spełnione	Zgodnie z sek. 4.3 Dok.3
Monitorowanie CSV	Spełnione	Zgodnie z sek. 4.3 Dok.3
Wznowienia treningu	Spełnione	Zgodnie z sek. 4.2.5 Dok.3

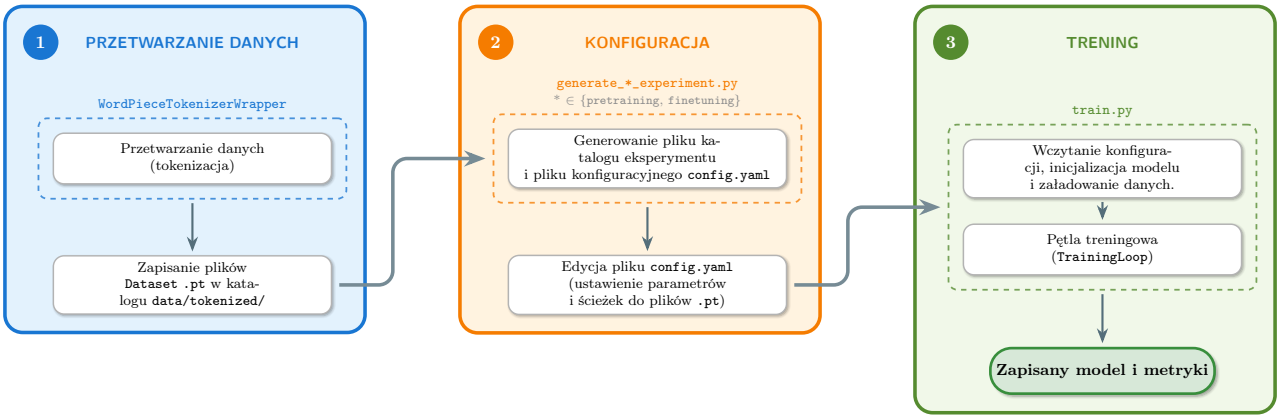
* Wymagania początkowo określono względem naszej implementacji SDPA. Końcowa weryfikacja została wykonana w porównaniu z natywną implementacją Flash Attention (zgodnie z sek. 2.4 Dok. 4), charakteryzującą się wyższą wydajnością.

Tabela 2: Weryfikacja wymagań funkcjonalnych

Wymaganie	**Status**	**Komentarz**
WF-1 — Pretrening i dostrajanie		
Pełny cykl uczenia (MLM + CLS)	Spełnione	Zgodnie z sek. 4.2 Dok.3
Zapis i wznawianie stanu	Spełnione	Zgodnie z sek. 4.2.5 Dok.3
Logowanie metryk (CSV, W&B)	Spełnione	Zgodnie z sek. 4.3 Dok.3
WF-2 — Wymienne mechanizmy uwagi		
Deklaratywny wybór w YAML	Spełnione	Zgodnie z sek. 5 Dok. 3
Obsługa SDPA, LSH, FAVOR+	Spełnione	Zgodnie z sek. 3.2 Dok. 3
Kompatybilność interfejsów	Spełnione	Zgodnie z sek. 3.2 Dok. 3
WF-3 — Obsługa długich sekwencji		
Kodowanie pozycyjne	Spełnione	Zgodnie z sek. 3.3.4 Dok. 3
Obsługa sekwencji > 512	Spełnione	Parametr `max_length` przyjmuje dowolną wartość (tab.1 Dok. 3).
WF-4 — Pipeline danych		
Tokenizacja WordPiece	Spełnione	Zgodnie z sek. **??** Dok. 5
Dynamiczny padding	Spełnione	Zgodnie z sek. 4.2.3 Dok. 3
Maskowanie MLM (BERT)	Spełnione	Zgodnie z sek. **??** Dok. 5
WF-5 — Konfiguracja		
Generator eksperymentów	Spełnione	Zgodnie z sek. **??** Dok. 5
Separacja pretrening/finetuning	Spełnione	Zgodnie z sek. **??** Dok. 5

3 Podręcznik użytkownika

Niniejszy podręcznik opisuje, jak korzystać z systemu. Uproszczony schemat użytkowania systemu przedstawiony jest na rys. ??.



Rysunek 1: Schemat użytkowania systemu

3.1 Przechowywanie danych

Katalog `data/` służy do przechowywania surowych oraz stokenizowanych danych z następującą strukturą:

- `data/raw/` — surowe pliki (np. CSV, TXT, Parquet).
- `data/tokenized/` — gotowe do użycia zestawy danych w formacie `.pt` (zserializowane przez `torch.save`), kompatybilne z oczekiwanym formatem `DataLoader`.

Oczekiwany format zestawu danych (.pt):

- **Pretrening (MLM):** `TensorDataset` zawierający tensory: `input_ids, attention_mask`
- **Dostrajanie (CLS):** `TensorDataset` zawierający tensory: `input_ids, attention_mask, labels`

3.2 Tokenizacja danych

Do tokenizacji wykorzystywany jest wrapper `WordPieceTokenizerWrapper`.

3.2.1 Trenowanie tokenizera

Jeśli nie dysponujemy gotowym tokenizerem (plik `vocab.txt`), możemy go wytrenować na dowolnym korpusie tekstowym:

```
from textclf_transformer.tokenizer.wordpiece_tokenizer_wrapper \
    import WordPieceTokenizerWrapper

tokenizer = WordPieceTokenizerWrapper()
tokenizer.train(tokenizer_dir="my_tokenizer_dir", input="data/raw/input.txt")
```

3.2.2 Tokenizacja z plików lub listy tekstów

```
from textclf_transformer.tokenizer.wordpiece_tokenizer_wrapper \
    import WordPieceTokenizerWrapper
import torch
import pathlib

tokenizer = WordPieceTokenizerWrapper()
tokenizer.load("src/textclf_transformer/tokenizer/my_tokenizer_dir")

# Użycie encode z plikiem tekstowym
ds = tokenizer.encode(
    input="data/raw/text_input.txt",
```

```
    # labels=labels_tensor, # opcjonalne
    max_length=512,
)

out = pathlib.Path("data/tokenized/train_dataset.pt")
out.parent.mkdir(parents=True, exist_ok=True)
torch.save(ds, out)
```

3.2.3 Tokenizacja z Pandas

```
import pandas as pd

df = pd.read_csv("data/raw/data.csv")
ds = tokenizer.encode_pandas(
    df=df,
    text_col="text",
    label_col="label", # opcjonalne
    max_length=512
)
```

3.3 Konfiguracja eksperymentów

Katalog `experiments/` służy do definiowania eksperymentów i ich konfiguracji. Każdy podkatalog w `pretraining/` lub `finetuning/` reprezentuje pojedynczy, powtarzalny przebieg eksperymentu. Szablony plików konfiguracyjnych `pretraining.yaml` i `finetuning.yaml` przechowywane są w `experiments/config_templates/`, na ich podstawie generowane są pliki `config.yaml` w odpowiednich katalogach eksperymentów.

3.3.1 Generowanie eksperymentów

Pretrening (MLM):

```
python experiments/generate_pretraining_experiment.py -p <pre_name>
```

Wynik: `experiments/pretraining/<pre_name>/config.yaml`

Dostrajanie (CLS):

```
python experiments/generate_finetuning_experiment.py \
    -f <fin_name> -p <pre_name>
```

Wynik: `experiments/finetuning/<fin_name>/config.yaml`

3.3.2 Wznawianie pretreningu

```
python experiments/generate_pretraining_experiment.py \
    -p <pre_name> -rp <resume_from_name>
```

Skrypt automatycznie kopiuje plik konfiguracyjny `config.yaml` i aktualizuje sekcję `training.resume`:

- Ustawia flagę `is_resume` na `true`.
- Przypisuje nazwę wznawianego eksperymentu do `resume_pretraining_name`.
- Ustawia ścieżkę `checkpoint_path` na ostatni zapisany model (`model.ckpt`).

W zależności od celu wznawiania, należy zweryfikować i ewentualnie dostosować parametr `load_only_model_state`:

- **Kontynuacja przerwanego treningu:** Ustaw `false`, aby wczytać pełny stan (model, optymalizator, scheduler, skaler).
- **Transfer learning / TAPT:** Ustaw `true`, aby wczytać wyłącznie wagi modelu.

Dodatkowo, w razie potrzeby można ręcznie zmienić ścieżkę do punktu kontrolnego, np. na `best-model.ckpt`.

3.4 Trening

Głównym interfejsem do uruchamiania treningu jest skrypt `train.py`.

3.4.1 Uruchomienie pretreningu

`python train.py -n <pre_name> -m pretraining`

3.4.2 Uruchomienie dostrajania

`python train.py -n <fin_name> -m finetuning`

3.4.3 Artefakty generowane w folderze eksperymentu

- **Checkpointy:** `checkpoints/`
  - `best-model.ckpt` – najlepszy model (pełny stan z optymalizatorem/schedulerem/skalerem)
  - `model.ckpt` – model końcowy (tylko wagi)
- **Metryki CSV:** `metrics/train/metrics.csv`, `metrics/eval/metrics.csv` (gdy `logging.log_metrics_csv=True`)
- **Logowanie W&B:** metadane i artefakty przechowywane w katalogu `wandb/` (gdy `logging.use_wandb=True`)

4 Testy integracyjne

W niniejszej sekcji przedstawiono scenariusze testów integracyjnych, weryfikujące poprawność współdziałania poszczególnych komponentów systemu w pełnym cyklu uczenia.

4.1 Środowisko testowe

Testy przeprowadzono w następującym środowisku:

- System operacyjny: Windows 11
- GPU: NVIDIA GeForce RTX 4050
- Python: 3.12
- PyTorch: 2.8.0

4.2 Scenariusz 1: TAPT + finetuning z mechanizmem SDPA (IMDB)

4.2.1 Cel testu

Weryfikacja poprawności integracji wszystkich komponentów systemu: tokenizacji danych, pretreningu MLM oraz dostrajania klasyfikatora na zbiorze IMDB z wykorzystaniem mechanizmu uwagi SDPA (Scaled Dot-Product Attention).

4.2.2 Kroki wykonania

1. **Tokenizacja danych:**

```
ds = load_dataset("imdb")
merged = concatenate_datasets([ds["test"], ds["train"]])
df = pd.DataFrame(merged)

tokenizer = WordPieceTokenizerWrapper()
tokenizer.load("src/textclf_transformer/tokenizer/BERT_original")

train_df, test_df = train_test_split(df, test_size=0.2, random_state=42,
                                     stratify=df['label'])
test_df, val_df = train_test_split(test_df, test_size=0.5, random_state=42,
                                   stratify=test_df['label'])

for name, data in [("train", train_df), ("val", val_df), ("test", test_df)]:
    ds = tokenizer.encode_pandas(data, text_col="text", label_col="label",
                                 max_length=512)
    torch.save(ds, f"data/tokenized/imdb_{name}.pt")
```

2. **Generowanie eksperymentu pretreningu:**

   ```
   python experiments/generate_pretraining_experiment.py -p sdpa_imdb_integration
   ```

3. **Uruchomienie pretreningu:**

   ```
   python train.py -n sdpa_imdb_integration -m pretraining
   ```

4. **Generowanie eksperymentu dostrajania:**

   ```
   python experiments/generate_finetuning_experiment.py -f sdpa_imdb_ft -p sdpa_imdb_integration
   ```

5. **Uruchomienie dostrajania:**

   ```
   python train.py -n sdpa_imdb_ft -m finetuning
   ```

4.2.3 Konfiguracja eksperymentu

Kluczowe parametry konfiguracji (plik `config.yaml`):

Parametr	**Wartość**
Pretrening	
Mechanizm uwagi	SDPA (mha)
Liczba epok	2
Batch size	16
Learning rate	2e-5
Max sequence length	512
Dostrajanie	
Liczba epok	2
Batch size	16
Learning rate	2e-4

4.2.4 Przykładowe logi

Pretraining (fragment):

```
$ python train.py -n sdpa_imdb_integration -m pretraining

wandb: Syncing run sdpa_imdb_integration
wandb: View run at https://wandb.ai/praca-inzynierska/demo/runs/cnr5rft3

Epoch: 0
Epoch: 1

wandb: Run summary:
wandb:            eval/epoch  2
wandb:             eval/loss  5.21974
wandb:       eval/perplexity  184.88689
wandb:  train/avg_epoch_loss  5.36163
wandb: train/gpu_mem_peak_mb  3996.75342

[OK] Zapisano checkpoint: .../checkpoints/model.ckpt
```

Finetuning (fragment):

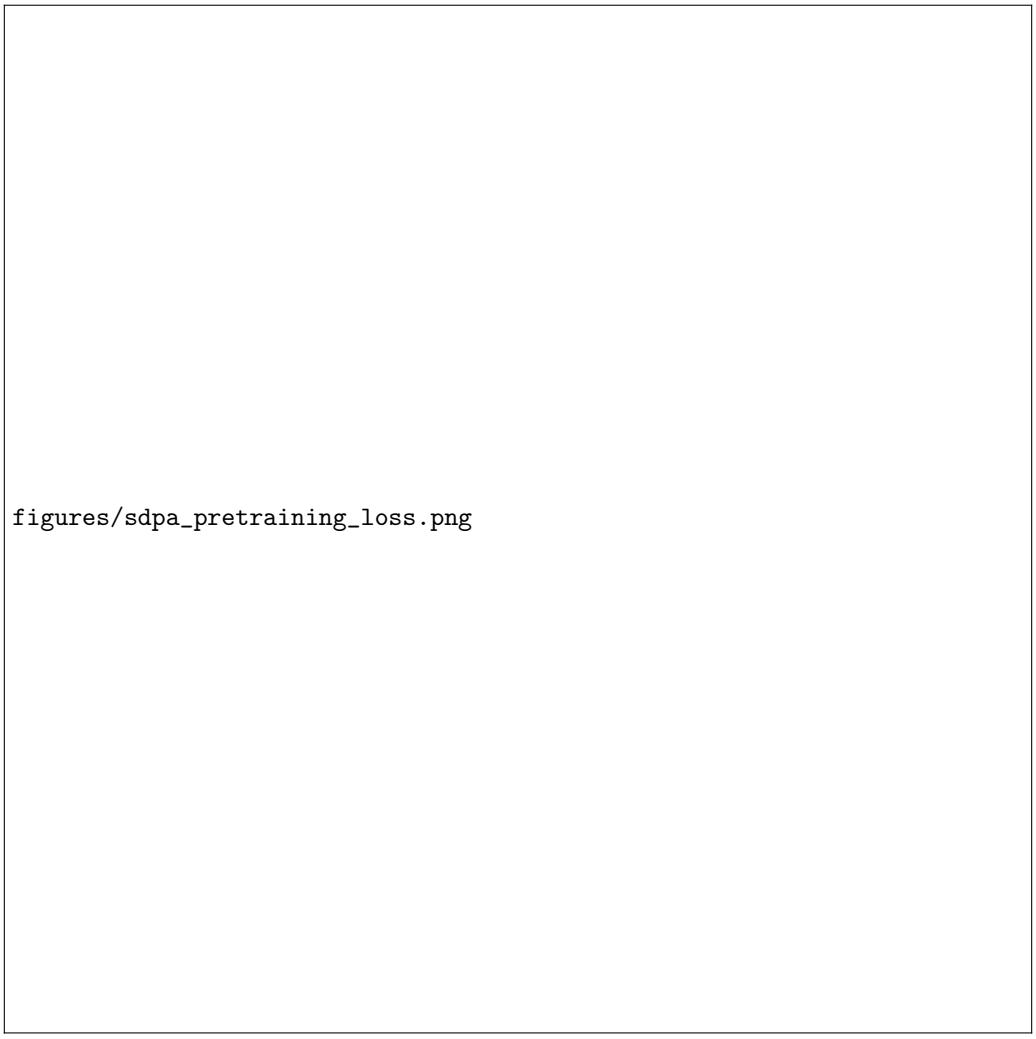
```
$ python train.py -n sdpa_imdb_ft -m finetuning

[WARN] Brakujące klucze: ['classifier.pooler.0.weight', ...]
[WARN] Nieoczekiwane klucze: ['mlm.transform.0.weight', ...]

wandb: Syncing run sdpa_imdb_ft
wandb: View run at https://wandb.ai/praca-inzynierska/demo/runs/u601225i
```

```
Epoch: 0
Epoch: 1

wandb: Run summary:
wandb:           eval/accuracy  0.8908
wandb:  eval/balanced_accuracy  0.8908
wandb:         eval/class_0_f1  0.89188
wandb:         eval/class_1_f1  0.8897
wandb:           eval/f1_macro  0.89079
wandb: train/gpu_mem_peak_mb    3996.75

[OK] Zapisano checkpoint: .../checkpoints/model.ckpt
```

4.2.5 Wyniki

Metryka	**Wartość**
Accuracy (eval)	0.8908
F1-macro (eval)	0.8908
Pretraining loss (final)	5.220
Pretraining perplexity (final)	184.89
Max VRAM (pretraining)	3996.75 MB
Max VRAM (finetuning)	3996.75 MB

4.2.6 Wizualizacja procesu uczenia



Rysunek 2: Krzywa straty podczas pretreningu MLM (SDPA)

4.2.7 Status testu

PASSED — System poprawnie wykonał pełny cykl pretrening → dostrajanie z mechanizmem SDPA.

4.3 Scenariusz 2: TAPT + finetuning z mechanizmem FAVOR+ (IMDB)

4.3.1 Cel testu

Weryfikacja poprawności integracji komponentów systemu z alternatywnym mechanizmem uwagi FAVOR+ (Performer). Test potwierdza możliwość wymiany mechanizmu uwagi bez modyfikacji pozostałych komponentów pipeline'u.

4.3.2 Kroki wykonania

1. **Tokenizacja danych:**

   Wykorzystano dane stokenizowane w Scenariuszu 1.

2. **Generowanie eksperymentu pretreningu:**

   ```
   python experiments/generate_pretraining_experiment.py -p favor_imdb_integration
   ```

3. **Konfiguracja mechanizmu FAVOR+:**

   W pliku `config.yaml` ustawiono:

```
attention:
  kind: favor
  favor:
    nb_features: 256
```

4. **Uruchomienie pretreningu:**

   ```
   python train.py -n favor_imdb_integration -m pretraining
   ```

5. **Generowanie eksperymentu dostrajania:**

   ```
   python experiments/generate_finetuning_experiment.py -f favor_imdb_ft -p favor_imdb_integration
   ```

6. **Uruchomienie dostrajania:**

   ```
   python train.py -n favor_imdb_ft -m finetuning
   ```

4.3.3 Konfiguracja eksperymentu

Kluczowe parametry konfiguracji (plik `config.yaml`):

Parametr	**Wartość**
Pretrening	
Mechanizm uwagi	FAVOR+
nb_features	256
Liczba epok	2
Batch size	16
Learning rate	2e-5
Max sequence length	512
Dostrajanie	
Liczba epok	2
Batch size	16
Learning rate	2e-4

4.3.4 Przykładowe logi

Pretraining (fragment):

```
$ python train.py -n favor_imdb_integration -m pretraining

wandb: Syncing run favor_imdb_integration
wandb: View run at https://wandb.ai/praca-inzynierska/demo/runs/id0t8lnc

Epoch: 0
Epoch: 1

wandb: Run summary:
wandb:            eval/epoch  2
wandb:             eval/loss  6.11616
wandb:       eval/perplexity  453.12034
wandb:  train/avg_epoch_loss  6.2152
wandb: train/gpu_mem_peak_mb  5924.81982

[OK] Zapisano checkpoint: .../checkpoints/model.ckpt
```

Finetuning (fragment):

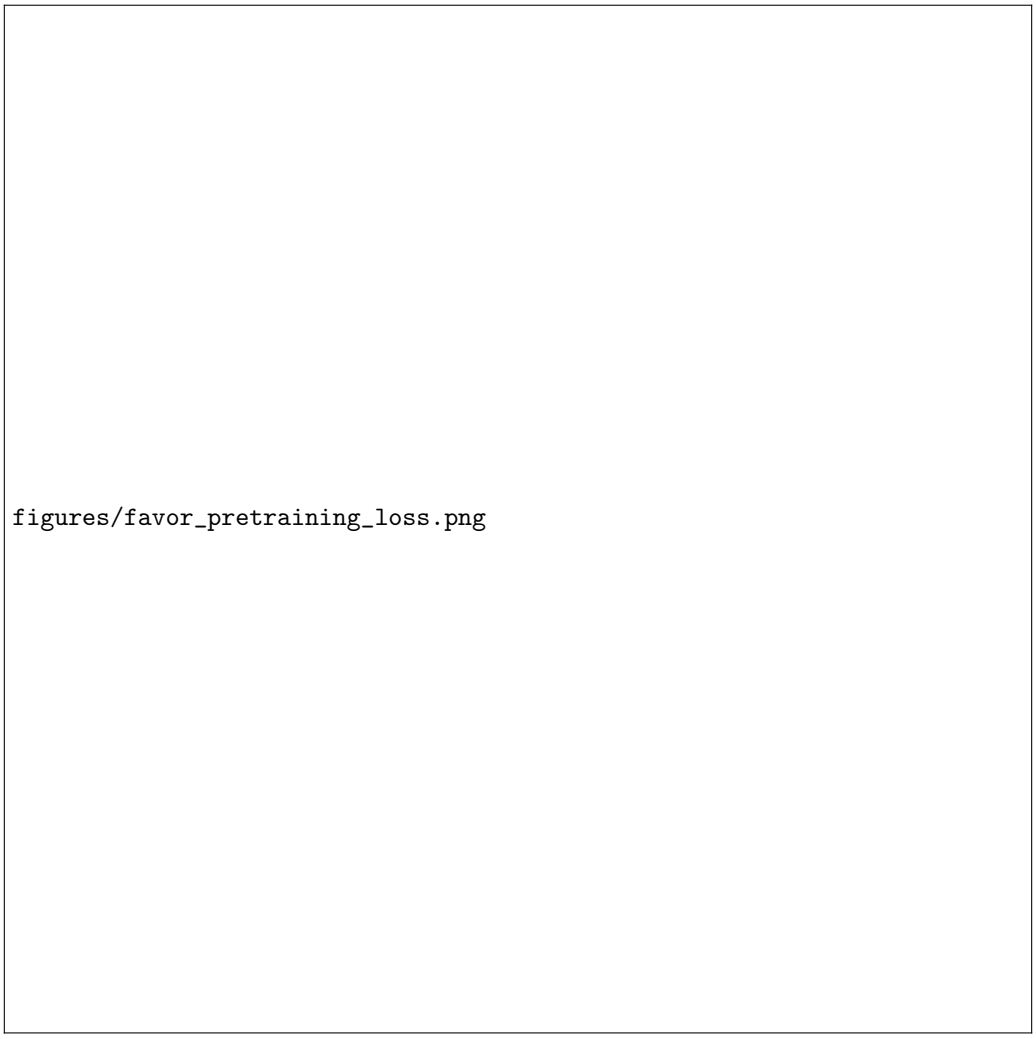
```
$ python train.py -n favor_imdb_ft -m finetuning

[WARN] Brakujące klucze: ['classifier.pooler.0.weight', ...]
[WARN] Nieoczekiwane klucze: ['mlm.transform.0.weight', ...]
```

```
wandb: Syncing run favor_imdb_ft
wandb: View run at https://wandb.ai/praca-inzynierska/demo/runs/ygtyw0og

Epoch: 0
Epoch: 1

wandb: Run summary:
wandb:           eval/accuracy  0.8958
wandb:  eval/balanced_accuracy  0.8958
wandb:         eval/class_0_f1  0.89549
wandb:         eval/class_1_f1  0.89611
wandb:           eval/f1_macro  0.8958
wandb: train/gpu_mem_peak_mb    5924.82

[OK] Zapisano checkpoint: .../checkpoints/model.ckpt
```

4.3.5 Wyniki

Metryka	**Wartość**
Accuracy (eval)	0.8958
F1-macro (eval)	0.8958
Pretraining loss (final)	6.116
Pretraining perplexity (final)	453.12
Max VRAM (pretraining)	5924.82 MB
Max VRAM (finetuning)	5924.82 MB

4.3.6 Wizualizacja procesu uczenia



Rysunek 3: Krzywa straty podczas pretreningu MLM (FAVOR+)

4.3.7 Status testu

PASSED — System poprawnie wykonał pełny cykl pretrening → dostrajanie z mechanizmem FAVOR+. Potwierdza to modularność architektury i możliwość wymiany mechanizmu uwagi poprzez konfigurację YAML.

4.4 Podsumowanie testów integracyjnych

Nr	Scenariusz	Status
1	Pełny pipeline SDPA na IMDB (TAPT + finetuning)	PASSED
2	Pełny pipeline FAVOR+ na IMDB (TAPT + finetuning)	PASSED

Przeprowadzone testy integracyjne potwierdzają:

- Poprawność współdziałania wszystkich komponentów systemu (tokenizacja → pretrening → dostrajanie).
- Modularność architektury — możliwość wymiany mechanizmu uwagi (SDPA, FAVOR+) bez modyfikacji kodu, wyłącznie poprzez zmianę konfiguracji YAML.
- Zgodność z wymaganiami funkcjonalnymi określonymi w specyfikacji projektu.

5 Dokumentacja doświadczenia projektowego

Realizacja niniejszego projektu stanowiła kompleksowe przedsięwzięcie inżynierskie, łączące teorię głębokiego uczenia ze standardami wytwarzania oprogramowania. Poniżej przedstawiono kluczowe obszary kompetencji oraz wnioski płynące z realizacji prac.

5.1 Głębokie Uczenie i NLP

Najistotniejszym elementem projektu była implementacja architektury Transformer *from scratch*, bez polegania na gotowych abstrakcjach modelowych z biblioteki `transformers`. Pozwoliło to na:

- **Zrozumienie mechanizmu uwagi:** Implementacja klasycznego *Scaled Dot-Product Attention* oraz jego wariantów : *LSH Attention* (Reformer) i *FAVOR+* (Performer). Analiza ta uwypukliła kompromisy między precyzją aproksymacji a zyskiem obliczeniowym.
- **Pełny cykl treningowy:** Praktyczne opanowanie wieloetapowego procesu uczenia modeli NLP, obejmującego pretrening na korpusie ogólnym (MLM), adaptację do domeny (TAPT) oraz końcowe dostrajanie (finetuning) na zadaniu klasyfikacji.

5.2 Inżynieria Oprogramowania

Projekt został zrealizowany zgodnie z zasadami *Clean Code* i nowoczesnymi praktykami Python:

- **Modularność i wzorce projektowe:** Zastosowanie wzorca Strategii do implementacji różnych wariantów mechanizmu uwagi, co umożliwia łatwą wymianę komponentów.
- **Typowanie statyczne:** Konsekwentne stosowanie `type hints` oraz walidacja konfiguracji za pomocą biblioteki `Pydantic` zapewniła bezpieczeństwo typów i czytelność kodu.
- **Testowanie:** Implementacja testów jednostkowych z użyciem `pytest`, obejmujących kluczowe komponenty (tokenizery, warstwy uwagi), co gwarantuje poprawność implementacji.

5.3 MLOps i Zarządzanie Eksperymentami

Istotnym aspektem pracy było stworzenie powtarzalnego i skalowalnego środowiska badawczego:

- **Reprodukowalność:** Pełna kontrola nad ziarnem losowości (*seed*) oraz wersjonowanie konfiguracji eksperymentów.
- **Zarządzanie konfiguracją:** Wykorzystanie szablonów YAML i skryptów generujących (`generate_*_experiment.py`) do automatyzacji tworzenia struktury eksperymentów, co eliminuje błędy ludzkie przy ręcznej edycji parametrów.
- **Śledzenie eksperymentów:** Integracja z platformą *Weights & Biases* (W&B) umożliwiająca monitorowanie metryk eksperymentów, zużycia zasobów systemowych oraz porównywanie wielu przebiegów.

Podsumowując, projekt ten pozwolił na zdobycie praktycznego doświadczenia w pełnym cyklu badania modelu uczenia maszynowego: od implementacji niskopoziomowych algorytmów, przez inżynierię oprogramowania, aż po zarządzanie procesem badawczym i ewaluację wyników.

A Załączniki - wyniki eksperymentów

W niniejszym załączniku przedstawiono szczegółowe wyniki eksperymentów, do których odwołano się w sekcji testów akceptacyjnych oraz w tabelach wymagań funkcjonalnych i niefunkcjonalnych.

A.1 Szczegółowe wyniki eksperymentów

Tabela 3: Wyniki testowanych konfiguracji w pretreningu na zbiorze Wikipedia

Konfiguracja	**Max VRAM [MB]**	**Czas/epokę [s]**	**Min. loss**
SDPA	14786	472.86	2.157
LSH			
N_h=2, C=64	19544	764.79	2.345
N_h=2, C=128	21848	1088.88	2.278
N_h=4, C=64	24964	1076.57	2.286
N_h=4, C=128	29574	1703.65	2.248
FAVOR+			
N_f=0.125	17018	659.34	3.069
N_f=0.25	18843	727.16	2.694
N_f=0.5	22494	917.32	2.666
N_f=1.0	29795	1303.79	2.579

Tabela 4: Wyniki testowanych konfiguracji w zadaniu TAPT

	Max VRAM [MB]			**Czas/epokę [s]**			**Min avg loss**		
Model	**IMDB**	**Hyper.**	**Arxiv**	**IMDB**	**Hyper.**	**Arxiv**	**IMDB**	**Hyper.**	**Arxiv**
SDPA	14784	14786	14793	74.21	999.06	4205.80	2.388	1.887	1.672
LSH									
N_h=2, C=64	19544	19545	19552	118.27	1116.21	2289.29	2.603	2.557	2.300
N_h=2, C=128	21848	21850	21856	136.91	1294.44	2615.22	2.502	2.440	2.224
N_h=4, C=64	24966	24967	24974	168.14	1609.29	3300.44	2.535	2.351	2.072
N_h=4, C=128	29574	29575	29580	204.73	1967.90	3950.40	2.478	2.263	2.024
FAVOR+									
N_f=0.125	17016	16990	16992	91.05	930.79	2070.34	3.200	4.505	5.033
N_f=0.25	18841	18787	18784	102.68	1055.93	2324.77	2.866	3.914	4.797
N_f=0.5	22490	22377	22372	129.63	1357.21	2843.42	2.834	3.665	4.632
N_f=1.0	29793	29564	29547	202.15	1922.34	3895.12	2.762	3.268	4.316

Tabela 5: Wyniki testowanych konfiguracji w zadaniu klasyfikacji

	Max VRAM [MB]			**Czas/epokę [s]**			**F1 macro**		
Model	**IMDB**	**Hyper.**	**Arxiv**	**IMDB**	**Hyper.**	**Arxiv**	**IMDB**	**Hyper.**	**Arxiv**
TF-IDF	-	-	-	-	-	-	0.8950	0.4223	0.8362
SDPA	3399.7	3518.3	3524.0	38.65	619.76	3308.77	0.929	0.646	0.884
LSH									
N_h=2, C=64	9258.3	5297.5	6224.1	80.85	756.79	1556.02	0.928	0.654	0.869
N_h=2, C=128	12042.3	6567.8	12174.6	98.85	938.26	1881.28	0.927	0.613	0.873
N_h=4, C=64	16270.9	8770.4	16404.6	127.57	1252.92	2583.98	0.925	0.625	0.867
N_h=4, C=128	21841.3	11314.8	21973.8	162.27	1616.40	3236.11	0.930	0.557	0.874
FAVOR+									
N_f=0.125	5633.7	5727.9	5725.7	56.58	562.32	1328.55	0.912	0.534	0.860
N_f=0.25	7456.0	7519.0	7517.1	66.99	688.69	1572.99	0.910	0.570	0.865
N_f=0.5	11106.0	11114.3	11104.4	91.07	984.21	2079.88	0.916	0.593	0.865
N_f=1.0	18846.4	18797.1	18783.6	154.00	1537.12	3096.50	0.917	0.550	0.856

A.2 Porównanie kosztów i metryki F1-macro względem SDPA na poszczególnych zbiorach danych

Tabela 6: Porównanie maksymalnego zużycia pamięci VRAM: pretrening na Wikipedii oraz TAPT+dostrajanie na zbiorach dowcelowych. Wartości pokazują różnicę procentową względem SDPA.

	Pretrening	**TAPT + Finetune**		
Model	**Wikipedia**	**IMDB**	**Hyper.**	**Arxiv**
LSH				
N_h=2, C=64	+32.2%	+32.2%	+32.2%	+32.2%
N_h=2, C=128	+47.8%	+47.8%	+47.8%	+47.7%
N_h=4, C=64	+68.8%	+68.9%	+68.9%	+68.8%
N_h=4, C=128	+100.0%	+100.0%	+100.0%	+100.0%
FAVOR+				
N_f=0.125	+15.1%	+15.1%	+14.9%	+14.9%
N_f=0.25	+27.4%	+27.4%	+27.1%	+27.0%
N_f=0.5	+52.1%	+52.1%	+51.3%	+51.2%
N_f=1.0	+101.5%	+101.5%	+99.9%	+99.7%

Tabela 7: Porównanie czasu treningu: pretrening na Wikipedii oraz TAPT+dostrajanie na zbiorach docelowych. Wartości pokazują różnicę procentową względem SDPA.

	Pretrening	**TAPT + Finetune**		
Model	**Wikipedia**	**IMDB**	**Hyper.**	**Arxiv**
LSH				
N_h=2, C=64	+61.7%	+70.2%	+16.1%	−50.1%
N_h=2, C=128	+130.3%	+100.0%	+38.7%	−41.1%
N_h=4, C=64	+127.7%	+149.1%	+78.3%	−21.8%
N_h=4, C=128	+260.3%	+207.2%	+123.8%	−3.7%
FAVOR+				
N_f=0.125	+39.4%	+27.8%	−7.9%	−56.3%
N_f=0.25	+53.8%	+46.0%	+8.0%	−49.5%
N_f=0.5	+94.0%	+87.9%	+45.5%	−35.3%
N_f=1.0	+175.7%	+199.8%	+115.8%	−6.8%

Tabela 8: Porównanie wyników testowanych mechanizmów atencji z baseline TF-IDF+LR i SDPA. Dla TF-IDF+LR i SDPA podano wartości bezwzględne metryki F1-macro (×100), dla LSH i FAVOR+ podano różnicę w punktach procentowych (pp) względem SDPA i względem TF-IDF+LR.

Model	**IMDB**	**Hyperpartisan**	**Arxiv**
TF-IDF + LR (baseline)	89.50%	42.23%	83.62%
SDPA	92.90% (+3.4 pp)	64.6% (+22.3 pp)	88.4% (+4.8 pp)
LSH (vs SDPA / vs TF-IDF)			
N_h=2, C=64	−0.1 / +3.3	+0.8 / +23.1	−1.5 / +3.3
N_h=2, C=128	−0.2 / +3.2	−3.3 / +19.0	−1.1 / +3.7
N_h=4, C=64	−0.4 / +3.0	−2.1 / +20.2	−1.7 / +3.1
N_h=4, C=128	+0.1 / +3.5	−8.9 / +13.4	−1.0 / +3.8
FAVOR+ (vs SDPA / vs TF-IDF)			
N_f=0.125	−1.7 / +1.7	−11.1 / +11.2	−2.4 / +2.4
N_f=0.25	−1.9 / +1.5	−7.6 / +14.8	−1.9 / +2.9
N_f=0.5	−1.2 / +2.1	−5.2 / +17.1	−1.9 / +2.9
N_f=1.0	−1.1 / +2.2	−9.5 / +12.8	−2.8 / +1.9

A.3 Stabilność uczenia

Tabela 9: Stabilność wyników F1-macro (×100) na zbiorze testowym dla modelu SDPA na zbiorze IMDB (3 uruchomienia z różnymi seedami)

Konfiguracja	**Agregacja**	**Max F1**	**Min F1**	**Różnica [p.p.]**
f=0, d=0.1	CLS	93.58	93.46	0.12
f=0, d=0.1	Mean	93.88	93.80	0.08
f=0, d=0.2	CLS	93.52	93.44	0.08
f=0, d=0.2	Mean	93.86	93.72	0.14
f=1, d=0.1	CLS	93.68	93.54	0.14
f=1, d=0.1	Mean	94.12	93.80	0.32
f=1, d=0.2	CLS	93.70	93.52	0.18
f=1, d=0.2	Mean	93.98	93.80	0.18
f=2, d=0.1	CLS	93.66	93.44	0.22
f=2, d=0.1	Mean	93.86	93.64	0.22
f=2, d=0.2	CLS	93.64	93.46	0.18
f=2, d=0.2	Mean	93.88	93.62	0.26