# Rekomendacje dla portali informacyjnych

Systemy Rekomendacyjne 2024/2025

# Charakterystyka problemu

- Elementy rekomendowane (artykuły) pojawiają się często i żyją krótko (zazwyczaj max. 100 godzin)
- Niewielki odsetek użytkowników pozostawia wystarczająco dużo informacji, by otrzymać "zindywidualizowane" rekomendacje
- System powinien działać niemal w czasie rzeczywistym i reagować na zmienne trendy zainteresowań użytkowników

# Dane

- Historia użytkowników
- Strumień zdarzeń od użytkowników
- Teksty artykułów

# Metody oparte o algorytmy bandyty

# Po prostu bandyta

- Każdy artykuł traktujemy jako ramię bandyty
- Na podstawie zebranych zdarzeń aktualizujemy payout każdego ramienia
  - Najlepiej sprawdza się optymalizacja payoutów znormalizowanych, np. CTR

# Po prostu bandyta

- Zalety:
  - Bardzo prosta implementacja, zarówno samego algorytmu jak i infrastruktury gromadzącej dane
  - Minimalny stopień skomplikowania z punktu widzenia aplikacji frontendowej
- Wady:
  - Nie uwzględniamy zmienności zainteresowań użytkowników w czasie (ograniczony wpływ, ponieważ czas życia artykułów jest krótki)
  - Nie uwzględniamy zainteresowań użytkowników (każdemu rekomendujemy te same treści)
  - Zagładzamy większość artykułów, prezentując tylko te najbardziej interesujące dla ogółu (szczególnie dotkliwe w przypadku dużej puli różnorodnych artykułów)

# Bandyta z wygasającą wiedzą

- Zamiast przechować dane o payoucie materiału z całego okresu jego istnienia (np. średni CTR od momentu opublikowania artykułu), implementujemy mechanizm wygasania wiedzy z czasem
- Przykłady:
  - Okno czasowe, czyli przechowywanie danych z $N$ ostatnich godzin
  - Wykładnicze wygasanie, czyli regularne (np. co kilka minut) przemnażanie przechowywanych wartości przez $\alpha < 1$

# Bandyta z wygasającą wiedzą

- Zalety:
  - Stosunkowo prosta implementacja (choć implementacja podejścia z oknem czasowym wymaga wykorzystania odpowiednich narzędzi, np. *Apache Druid*)
  - Minimalny stopień skomplikowania z punktu widzenia aplikacji frontendowej
  - Uwzględniamy zmienność zainteresowań użytkowników w czasie
- Wady:
  - Nie uwzględniamy zainteresowań użytkowników (każdemu rekomendujemy te same treści)
  - Zagładzamy większość artykułów, prezentując tylko te najbardziej interesujące dla ogółu (szczególnie dotkliwe w przypadku dużej puli różnorodnych artykułów)

# Bandyci w segmentacji

- Dzielimy użytkowników na segmenty zgodnie z ich zainteresowaniami
- Dane o payoucie każdego materiału przechowujemy w rozbiciu względem segmentu
- Każde przychodzące zapytanie o rekomendację musi zawierać ID użytkownika
- W ramach pojedynczego segmentu wykorzystujemy algorytm bandyty (najlepiej z wygasającą wiedzą)

# Bandyci w segmentacji

- Zalety:
  - Uwzględniamy zmienność zainteresowań użytkowników w czasie
  - W ograniczonym stopniu uwzględniamy różnorodne zainteresowania użytkowników
  - Zwiększamy różnorodność rekomendacji, w większym stopniu wykorzystujemy niszowe treści
  - Stosunkowo niewielki stopień skomplikowania z punktu widzenia aplikacji frontendowej (choć wymaga przechowywania stanu w postaci ID użytkownika)
- Wady:
  - Istotnie zwiększyliśmy stopień skomplikowania systemu (zarówno pod kątem przechowywania danych o payoucie jak i całego podsystemu obliczania i serwowania danych o segmentach)
  - Nadal nie uwzględniamy preferencji pojedynczych użytkowników

# Metody oparte o embedding

# Problem

- Część najaktywniejszych użytkowników jest dla nas na tyle cenna, że chcemy zaprezentować im spersonalizowane rekomendacje
- Nasza wiedza o użytkownikach ogranicza się do historii aktywności (historia kliknięć, wagi kliknięć)
- Chcemy uwzględnić preferencje pojedynczego użytkownika, ale także możliwość zmian tych preferencji w czasie

# Recykling pomysłów

- Embeddingi użytkowników możemy wykorzystać nie tylko do wyznaczenia segmentów zainteresowań
- Obliczone embeddingi treści i użytkowników mogą posłużyć do wykrywania treści podobnych do gustu konkretnego użytkownika
- Do tego celu możemy wykorzystać dowolny algorytm obliczania embeddingów, zarówno *Item2Vec* jak i bardziej zaawansowane modele ML (np. *NewsBERT*, o którym zaraz), a także modele grafowe (np. *Node2Vec* czy *RippleNet*)

# Problemy i rozwiązania

- Proste embeddingi uwzględniają tylko sam fakt kliknięcia użytkownika w artykuł - chcemy wykorzystać wiedzę o tym, jak bardzo dany tekst się spodobał (np. głębokość scrolla)
  - Embedding użytkownika możemy obliczyć jako średnią ważoną embeddingów artykułów, gdzie waga jest proporcjonalna do stopnia zainteresowania
- Musimy na bieżąco obliczać embeddingi nowopowstałych artykułów
  - Konieczna jest zmiana podejścia, z czysto offline'owego wytrenowania i odpytania modelu (co robiliśmy w przypadku segmentacji) na serwowanie modelu w sposób możliwy do odpytywania przez np. REST API
- Model zwracający listę artykułów "najbliższych" użytkownikowi prawdopodobnie uzyska słabe rezultaty
  - Musimy wykluczyć z rekomendacji te artykuły, które użytkownik już przeczytał bądź odrzucił

# NewsBERT

# NewsBERT

- https://aclanthology.org/2021.findings-emnlp.280.pdf
- Opublikowany 11.2021

# NewsBERT

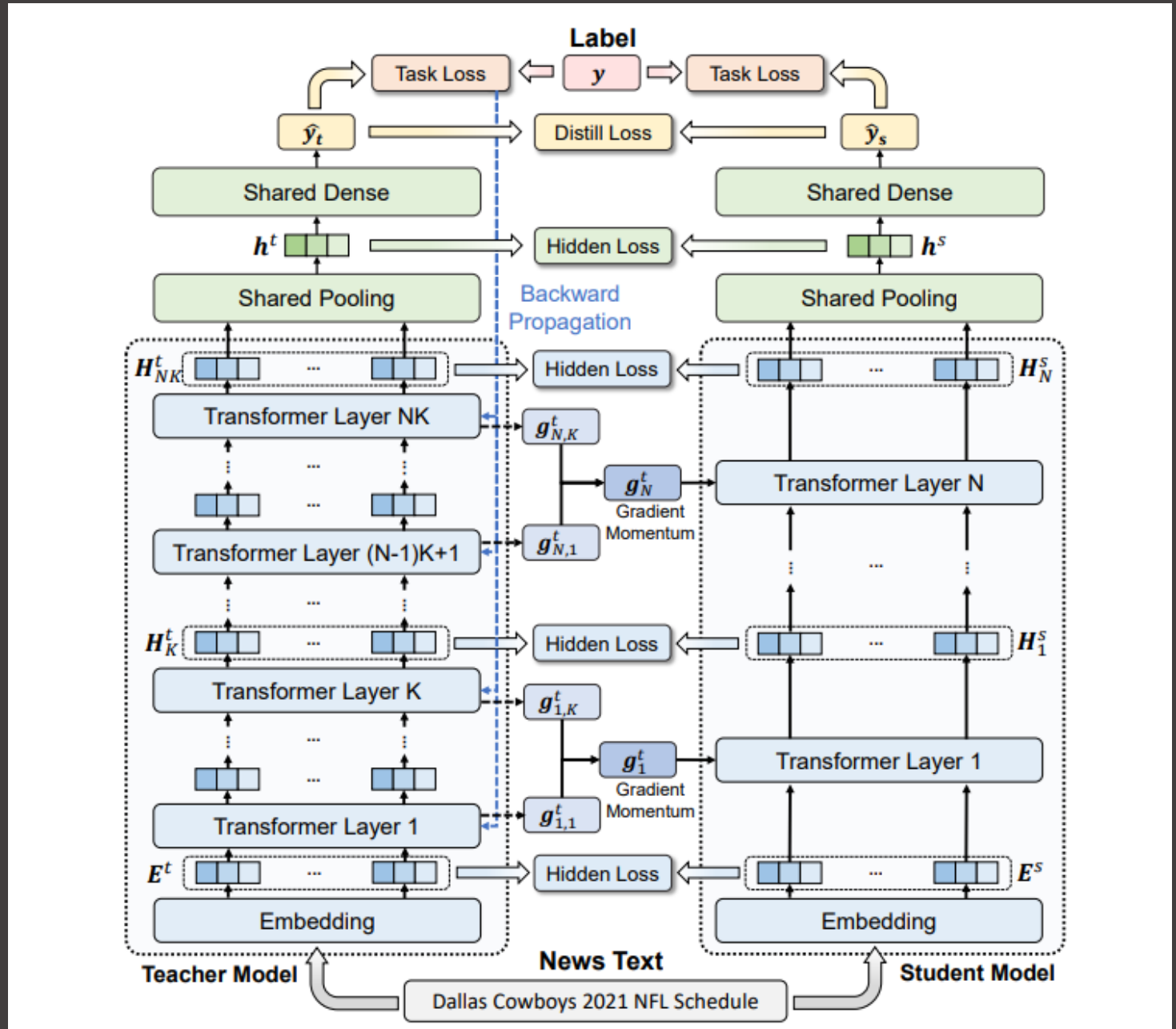



Figure 1: The framework of *NewsBERT* in an example task, i.e., news classification.

# NewsBERT

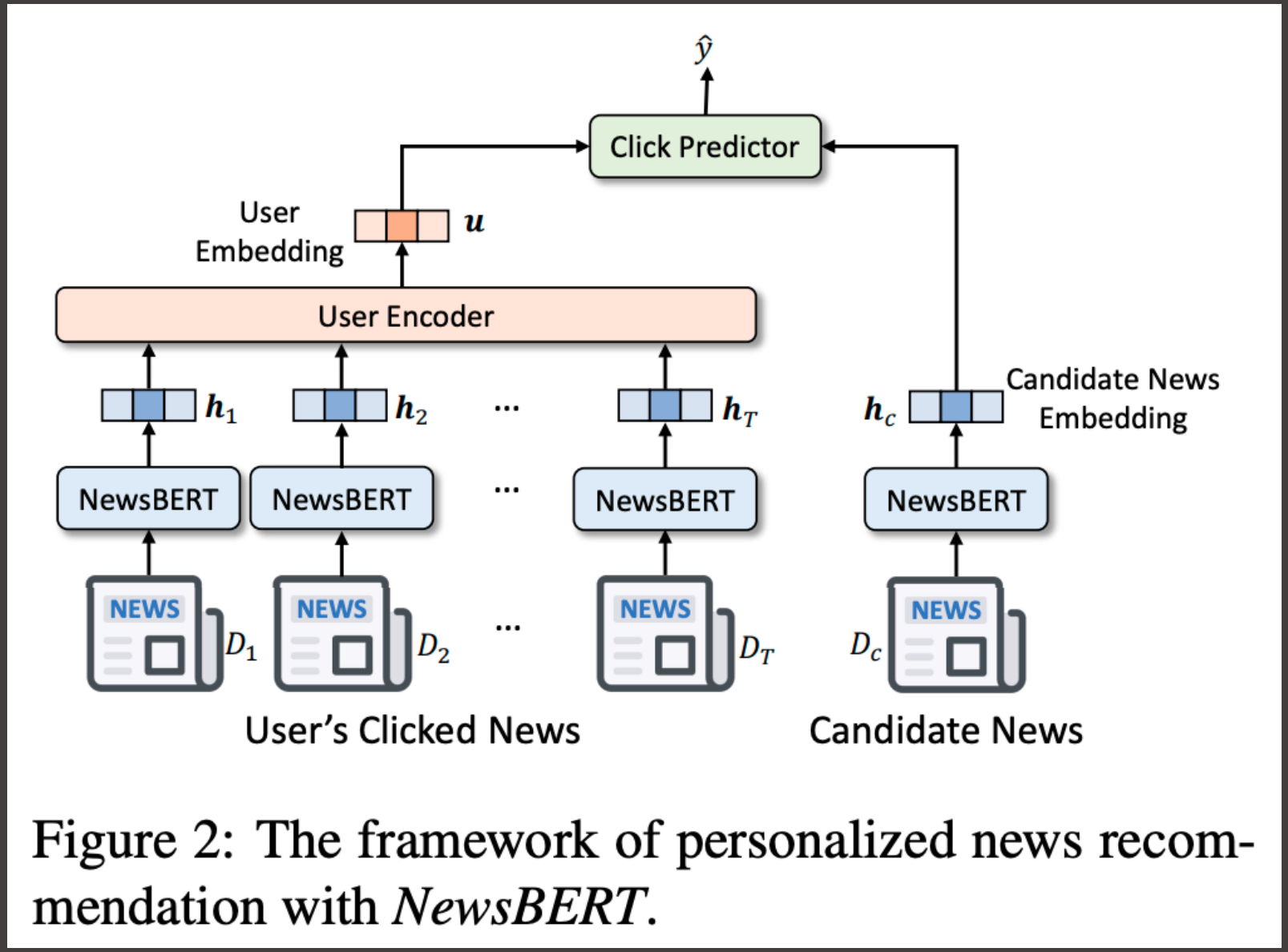



Figure 2: The framework of personalized news recommendation with *NewsBERT*.

# NewsBERT

- Zalety:
  - Bierze pod uwagę indywidualne gusta użytkowników
  - Brak "cold startu" - embeddingi tekstów liczone na podstawie treści, nie aktywności użytkowników
- Wady:
  - Bardzo wysoki koszt treningu i utrzymania modelu

# Zmiana paradygmatu

- W niektórych sytuacjach nie zależy nam na jak najlepszym dopasowaniu do gustu użytkowników
- Chcemy dopasować rekomendacje nie do użytkowników, a do kontekstu

# Podobieństwo artykułów

# Rekomendacje oparte na podobieństwie tekstów

- Definiujemy miarę podobieństwa tekstów (np. *TD-IDF*) tak, by móc wytypować *N* artykułów najbardziej podobnych do danego
- Rekomendacją może być po prostu *K* najpodobniejszych treści albo *K* najlepszych treści (wytypowanych np. algorytmem bandyty) spośród *N* najbardziej podobnych

# TF-IDF

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

**TF-IDF**

Term $x$ within document $y$

$tf_{x,y}$ = frequency of $x$ in $y$

$df_x$ = number of documents containing $x$

$N$ = total number of documents

Źródło: https://ted-mei.medium.com/demystify-tf-idf-in-indexing-and-ranking-5c3ae88c3fa0

# *TF-IDF* - implementacja

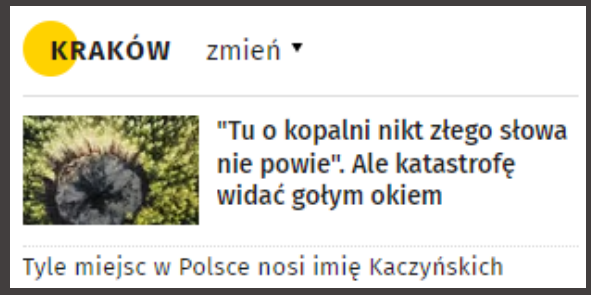
1. Preprocesing – sprowadzenie słów w całym korpusie do form podstawowych, usunięcie stop-words itp.
2. Utworzenie listy unikalnych tokenów w korpusie
3. *Bag of words* – reprezentacja każdego tekstu w korpusie jako wektora częstości tokenów
4. Obliczenie macierzy wartości *TF-IDF* na podstawie macierzy *bag of words*
5. Obliczenie macierzy podobieństwa (np. metryka cosinusowa)

# Automatyczne tagowanie, sekcje tematyczne

# Kategoryzacja treści

- Zakładamy, że każdy tekst w naszym systemie może mieć przypisaną jedną lub kilka kategorii tematycznych
  - Kategorie mogą posiadać swoje podkategorie, np. Sport -> Curling
- Ręczna kategoryzacja jest podatna na błędy, nawet jeśli dostarczymy zamknięty katalog kategorii
- Poprawna kategoryzacja umożliwi tworzenie sekcji tematycznych, np. Biznes, Sport, Technologie, Kultura





# Proste metody nadzorowane

- Zakładamy, że mamy dostępny sensownych rozmiarów korpus tekstów z przypisanymi kategoriami
- Chcemy wytrenować prosty model przypisujący (albo chociaż proponujący) kategorie, do jakich należeć powinien nowopowstały tekst

# TF-IDF

- Dla każdej kategorii wyznaczamy słowa, które w tekstach należących do tej kategorii miały najwyższy score *TD-IDF*
- Obliczamy wektor wartości *TF-IDF* nowopowstałego materiału
- Na podstawie słów o najwyższym scorze zgadujemy kategorie, do których należy nowy tekst

# NER (Named Entities Recognition)

- Dla każdej kategorii wyznaczamy najpopularniejsze *Named Entities*
- Wyszukujemy *NE* w nowopowstałym tekście i na tej podstawie przypisujemy tekst do odpowiednich kategorii

# NewsBERT po raz drugi

- Możemy wykorzystać model ze wszystkimi warstwami – jest trenowany tak, by przewidywać tagi

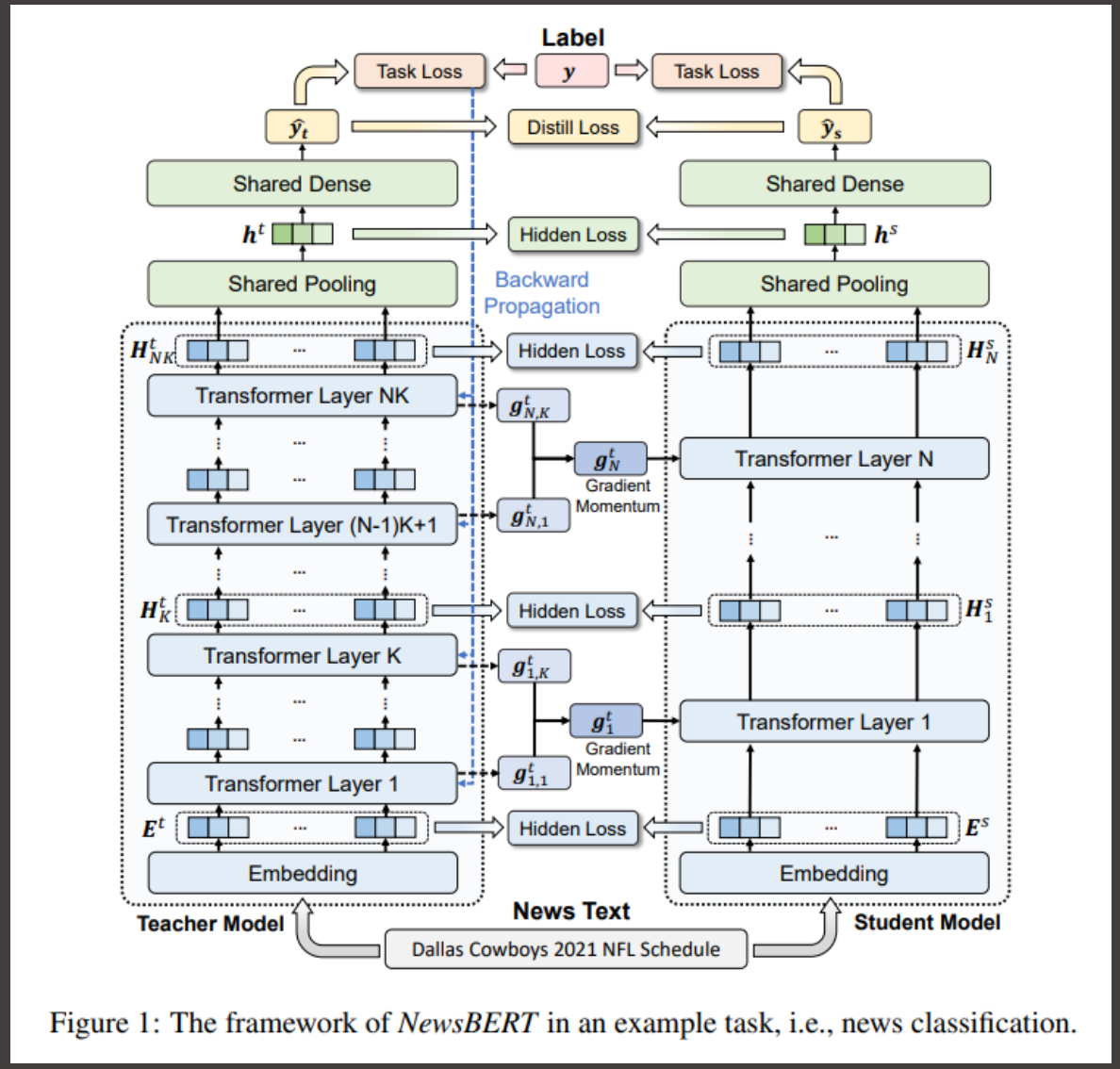



Figure 1: The framework of *NewsBERT* in an example task, i.e., news classification.

# Bonus – modele generatywne

# Co sprawi, że klikniesz?

- Użytkownik widzi tylko tytuł artykułu i w niektórych sytuacjach jedno zdjęcie
- Na tej podstawie musi podjąć decyzję, czy chce czytać dalej
- Być może warto dostosować tytuł do preferencji użytkownika?

# Metody generowania streszczeń tekstów

- Ekstrakcyjna - wybór krótkiego fragmentu tekstu, który najlepiej podsumowuje całość (np. jest najpodobniejszy według pewnej miary)
- Abstrakcyjna – wytrenowanie modelu, który na podstawie tekstu jest w stanie wygenerować nową sekwencję słów
  - GPT i podobne modele

# Mamy kilka tytułów - co dalej?

- Najlepszy tytuł do danego artykułu możemy dobrać przy użyciu znanych już metod - np. algorytmu bandyty
- Statystyki uzyskane w wyniku kliknięcia w poszczególne tytuły przechowujemy osobno – do celów analitycznych

# Podsumowanie

- Rekomendacje oparte o segmentacje i bandytów
- Rekomendacje oparte o osadzenia
- Metody bazujące na podobieństwie tekstów
- Automatyczna kategoryzacja
- Generowanie tytułów i podsumowań artykułów

# Więcej pomysłów

- https://bbcnewslabs.co.uk/projects/