Виправлення пошукових запитів за допомогою Denoising Autoencoder

Вступ

Розв'язання задачі корекції пошукових запитів є критично важливим аспектом для підвищення точності інформаційно-пошукових систем і вдосконалення користувацького досвіду. Запити користувачів часто містять орфографічні, семантичні або синтаксичні помилки, такі як перестановка літер, пропуски або дублювання символів. У цьому контексті розробка моделі Denoising Autoencoder (DAE) спрямована на автоматичне виправлення таких помилок із збереженням оригінального змісту запиту.

Цільова функціональність моделі полягає у забезпеченні виправлення помилок у пошукових запитах у режимі реального часу з урахуванням обмежень продуктивності. Виправлення вважається успішним, якщо воно відповідає еталонному списку назв пристроїв і має Levenshtein відстань не більше 2.

Процес роботи

Збір і підготовка даних

У якості основи для тренувального датасету був обраний датасет Smartphones_Dataset https://www.kaggle.com/datasets/informrohit1/smartphones-dataset який був доповненний даними про мобільні телефони з іншого датасету Massive Product Text Classification Dataset https://www.kaggle.com/datasets/asaniczka/product-titles-text-classification/data Дані були очищені, уніфіковані і підготовлені для створення тренувального датасету (усі скрипти додані в проєкт).

За відсутності реальних помилок при пошуку було використано штучне зашумлення тексту. Реалізація зашумлення здійснена за допомогою скрипта `./src/generate_dae_dataset.py`, який включає наступні типи помилок:

- Легкі помилки:

    - Випадкова заміна одного символу.
    - Видалення одного символу.
    - Вставка зайвого символу.
- Середні помилки:
    - Перестановка двох сусідніх символів.
    - Подвоєння літер.
    - Усічення закінчень слів.
    - Помилки у типовому закінченні слів.
    - Використання поширених реальних помилок із логів.
- Складні помилки:
    - Помилки введення свайпом.
    - Комбіновані помилки з кількома типами спотворень.
    - Імітація помилок автокорекції.

Також був реалізований механізм додавання в датасет даних з пошукових логів (`./src/real_words_errors.py`), що дозволяє враховувати реальні сценарії помилок користувачів.

Навчання моделі

Для реалізації DAE було обрано архітектуру на базі трансформерів із такими параметрами:

- Вбудовування розмірністю 256
- 8 голів самоуваги (attention heads)
- 4 шари трансформера
- Розмірність feed-forward шару — 1024

Ключові особливості моделі:

- Механізм самоуваги з урахуванням позиційної інформації (Position-aware attention mechanism)
- Нормалізація по шарах (Layer normalization) для забезпечення стабільності навчання
- Dropout для регуляризації моделі та запобігання перенавчанню

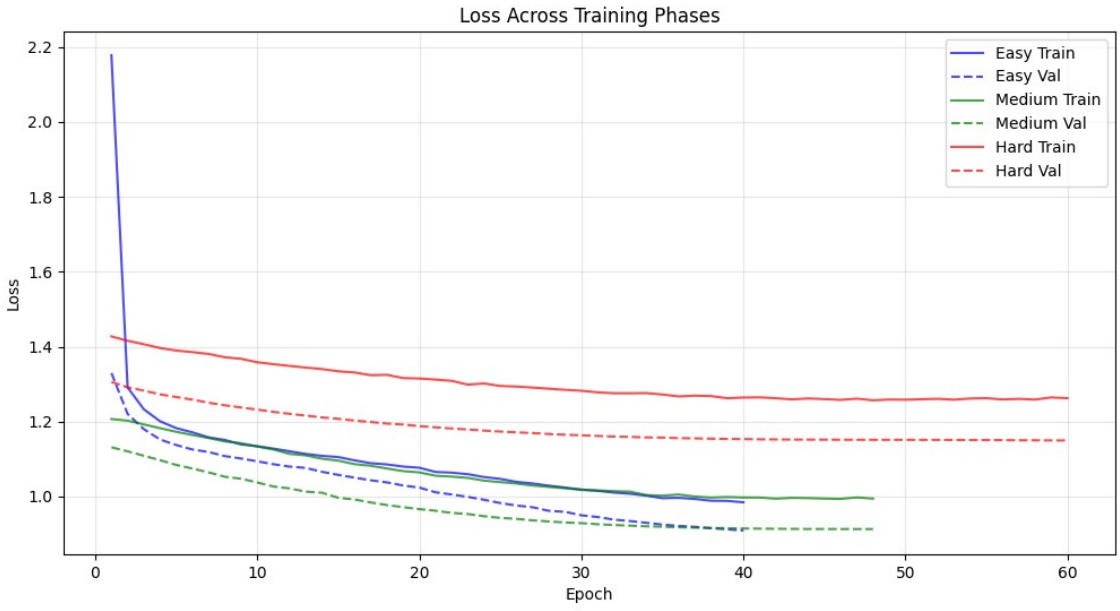
- Ініціалізація ваг, оптимізована для задач виправлення тексту

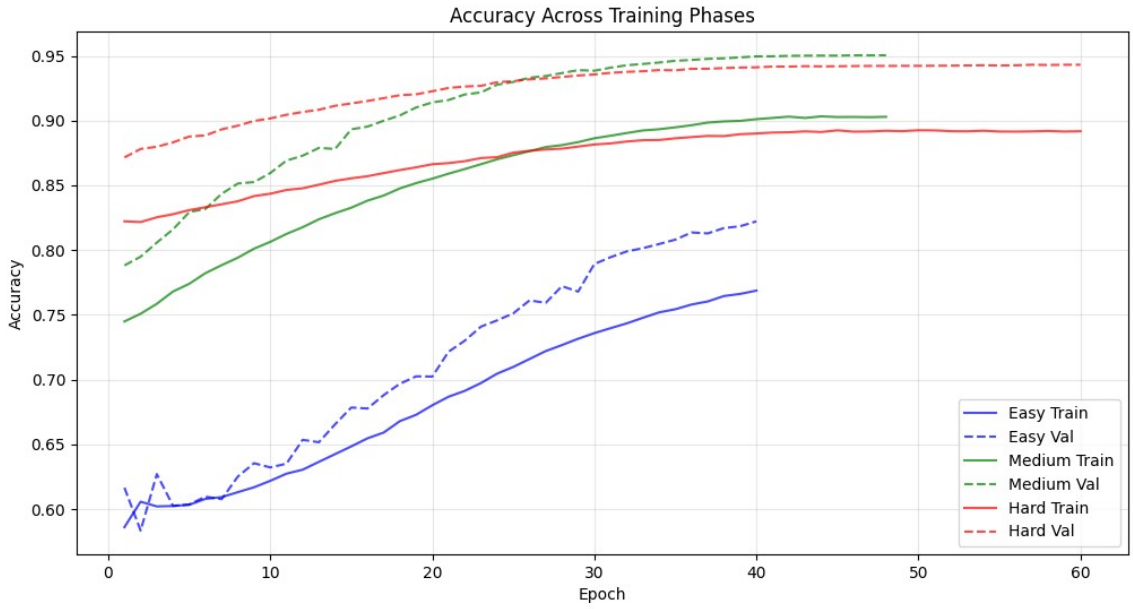
Особлива увага була приділена функції втрат, яка реалізована у файлі `./src/dae_loss.py`. Функція втрат використовує комбінацію крос-ентропійної втрати з додатковими механізмами для покращення навчання моделі. Основні особливості реалізації включають:

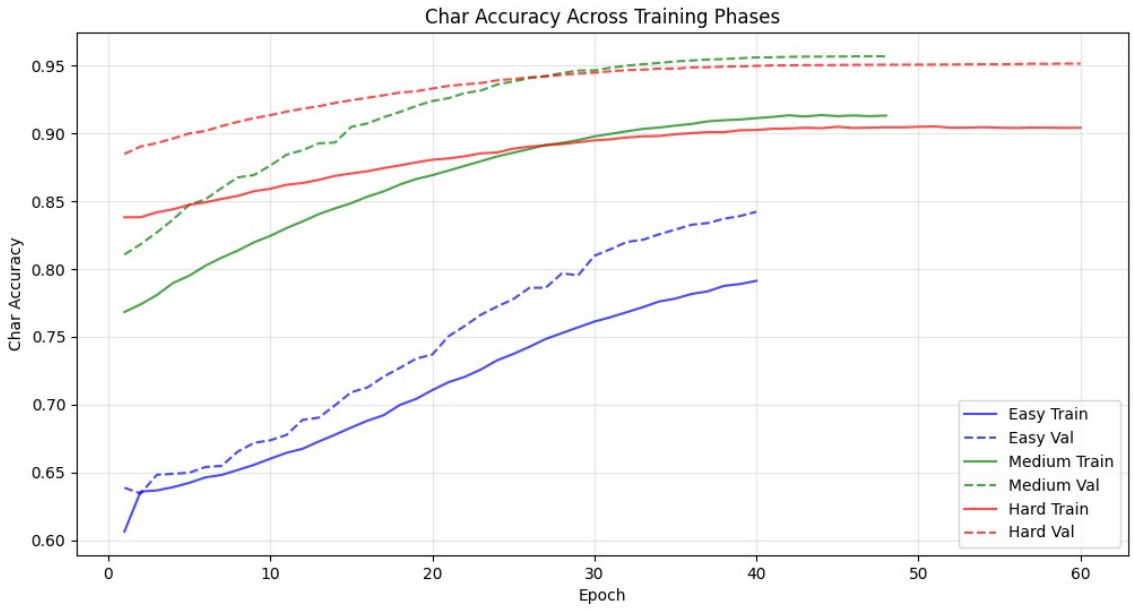
- Label smoothing для пом'якшення жорстких кордонів між класами, що сприяє кращій генералізації.
- Вагування позицій символів з акцентом на кінцеві символи, які часто містять помилки у пошукових запитах.
- Пеналізація довжини для зменшення помилок, пов'язаних із зайвими або пропущеними символами.
- Втрати на основі n-грам (біграми та триграми) для забезпечення коректності послідовностей символів у контексті.

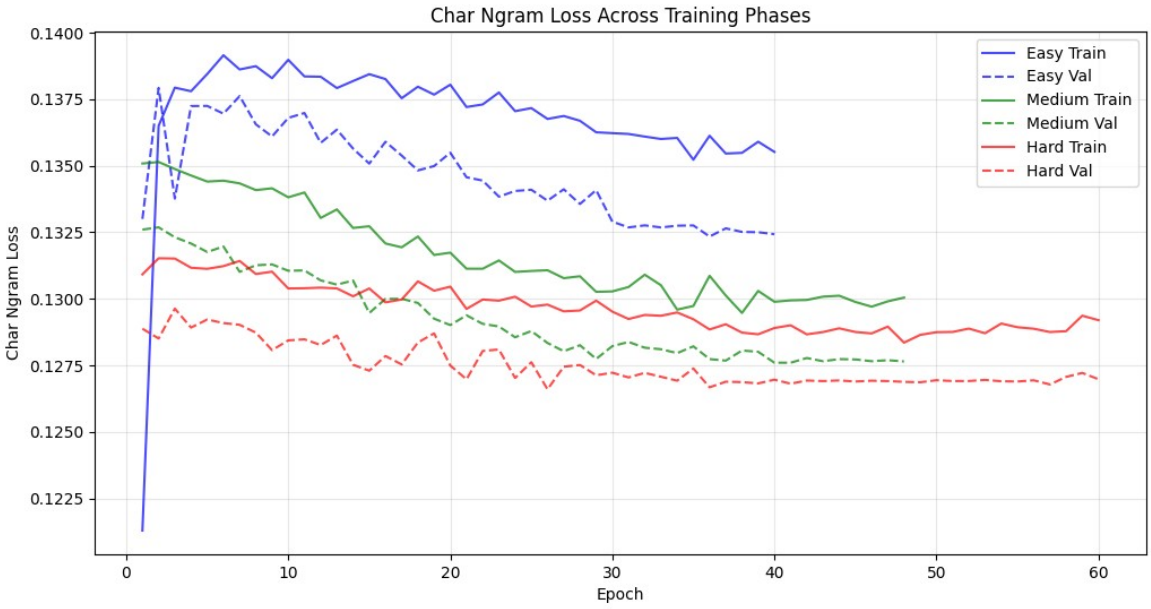
Завдяки цим механізмам модель досягає більшої точності у виправленні помилок у запитах, знижуючи частоту хибнопозитивних відповідей і покращуючи загальну продуктивність.
Для покращення результактів в тренувальний процес був доданий curriculum learning - поступове ускладнення типів помилок, спочатку легкі, потім середні, потім важкі.

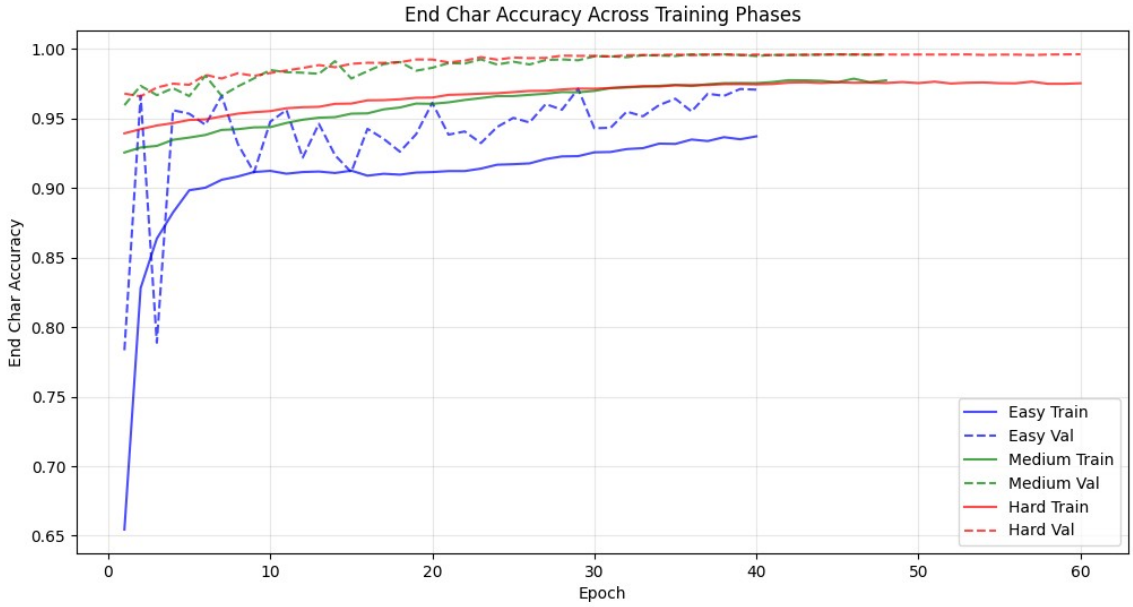
Графіки процесу навчання наведені нижче:

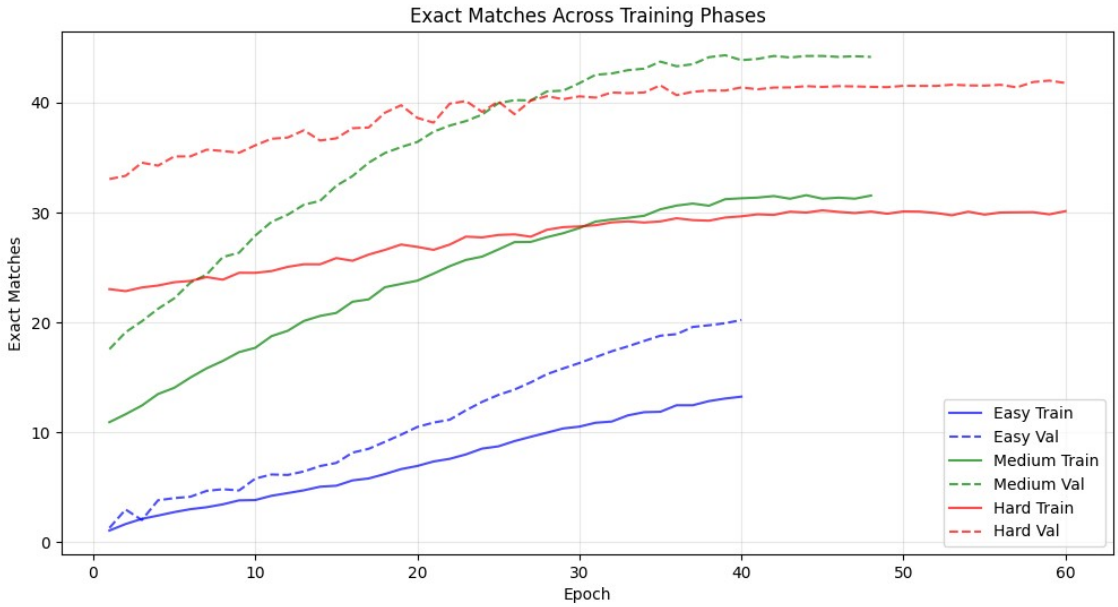


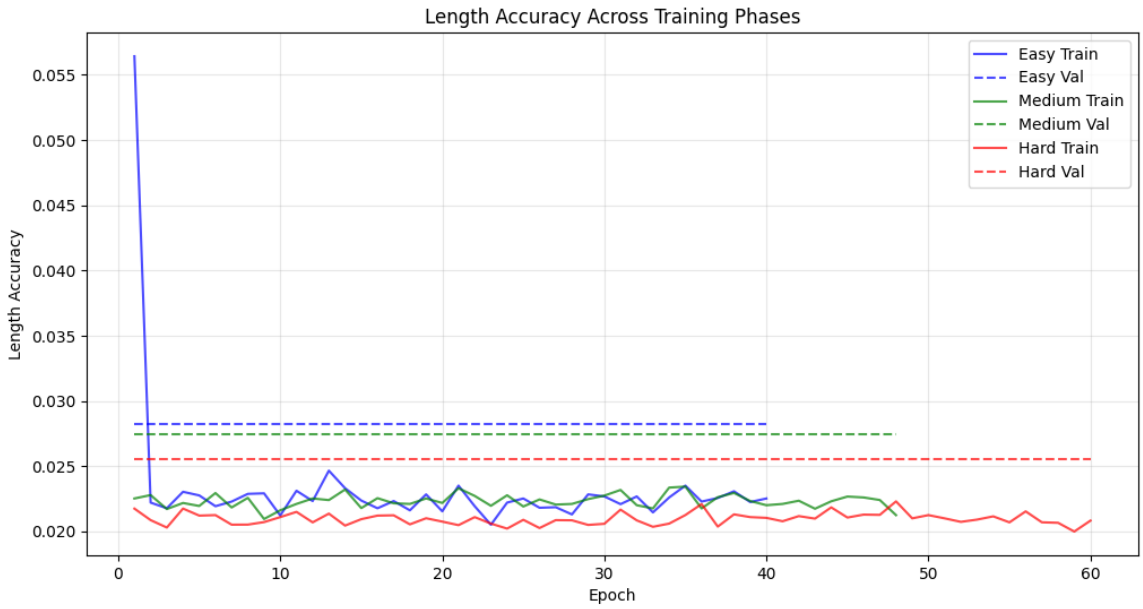

Accuracy Across Training Phases
Accuracy
Epoch
Easy Train
Easy Val
Medium Train
Medium Val
Hard Train
Hard Val

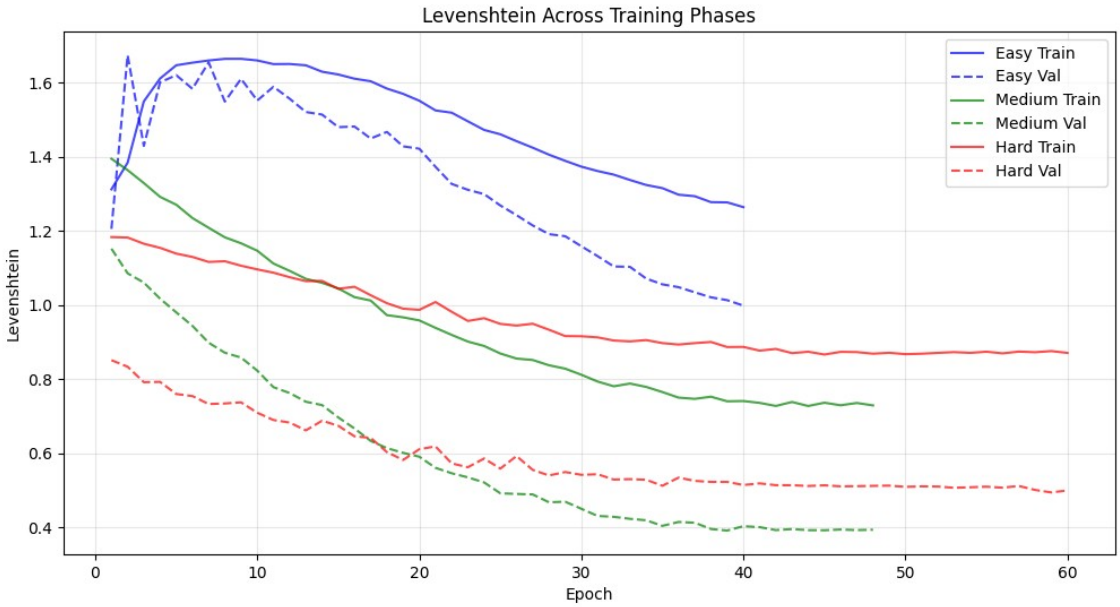

Char Accuracy Across Training Phases
Char Accuracy
Epoch
Easy Train
Easy Val
Medium Train
Medium Val
Hard Train
Hard Val


Char Ngram Loss Across Training Phases
Char Ngram Loss
Epoch
Easy Train
Easy Val
Medium Train
Medium Val
Hard Train
Hard Val


End Char Accuracy Across Training Phases
End Char Accuracy
Epoch
Easy Train
Easy Val
Medium Train
Medium Val
Hard Train
Hard Val


Exact Matches Across Training Phases
Exact Matches
Epoch
Easy Train
Easy Val
Medium Train
Medium Val
Hard Train
Hard Val


Length Accuracy Across Training Phases
Length Accuracy
Epoch
Easy Train
Easy Val
Medium Train
Medium Val
Hard Train
Hard Val



Виклики та їх вирішення

Однією з основних проблем у розробці моделі було обмеження доступного датасету. Початковий набір містив менше ніж 100 унікальних слів, що значно обмежувало можливості моделі до узагальнення нових назв пристроїв. Такі короткі назви, як "ace", "max", "neo", створювали додаткові труднощі для коректного виправлення помилок, оскільки моделі складно визначити правильний контекст для коротких слів.

Проблеми перенавчання також стали викликом через невеликий обсяг даних. Модель демонструвала високу точність на тренувальних даних, але показувала гірші результати при роботі з новими запитами.

Для подолання цих проблем були застосовані наступні підходи:

1. **Розширення датасету:** Ми додали нові слова з іншого відкритого датасету, що включає більше назв пристроїв і брендів. Це дозволило моделі краще узагальнювати нові запити.
2. **Фільтрація коротких слів та слів з цифрами:** З тренувального датасету були виключені слова, що складаються з трьох або менше літер, а також слова, що містять цифри. Це дозволило зосередити навчання моделі на більш складних і реалістичних запитах, зменшуючи кількість потенційно двозначних виправлень.

3. **Генерація помилок на основі реальних сценаріїв:** Був додатково розроблений скрипт `real_words_errors.py`, щоб додати у датасет типові помилки з реальних логів пошуку. Це дозволило моделі навчатися на реалістичних прикладах помилок.
4. **Curriculum Learning:** Застосування поступового ускладнення завдань допомогло уникнути перенавчання. Спочатку модель тренувалася на простих помилках, після чого поступово переходила до більш складних типів помилок.
5. **Оптимізація функції втрат:** Ми адаптували функцію втрат (`dae_loss.py`) для кращого врахування помилок у коротких назвах, застосувавши вагування кінцевих символів і штрафи за некоректну довжину слів.
6. **Використання BART для додаткового уточнення:** Інтеграція моделі BART для перевірки складних семантичних помилок дозволила підвищити точність виправлень у випадках, коли DAE не давало задовільного результату.

Ці заходи дозволили значно покращити здатність моделі до виправлення помилок у пошукових запитах, зменшивши ризик перенавчання і підвищивши загальну ефективність.

Інтеграція рішення в продукт

Для інтеграції моделі в продукт було реалізовано RESTful API, що дозволяє взаємодіяти з пошуковим механізмом у реальному часі. API приймає вхідний пошуковий запит, передає його через DAE для виправлення локальних помилок, після чого виконує додаткову обробку через BART для уточнення семантики.

- Архітектура API: Реалізована на базі Flask із підтримкою масштабованості через Docker.
- Паралельна обробка: Запити обробляються асинхронно для забезпечення високої швидкодії.
- Фільтрація результатів: Після виправлення запиту модель перевіряє результат за еталонним списком назв телефонів, щоб виключити некоректні виправлення.

Інтеграція BART відбувається через попередньо треновану модель `facebook/bart-base`, адаптовану для виправлення семантичних помилок у

пошукових запитах. BART використовується лише у випадках, коли виправлення DAE не відповідає встановленим критеріям якості.

Переваги такого підходу:

- Висока точність виправлень: BART забезпечує глибокий контекстуальний аналіз, що дозволяє коригувати складніші семантичні помилки, які важко ідентифікувати на рівні символів.
- Гнучкість системи: Використання BART лише у випадках необхідності оптимізує продуктивність системи, зменшуючи затрати ресурсів.
- Покращення користувацького досвіду: Такий підхід забезпечує максимально коректні результати пошуку, що підвищує задоволеність користувачів і точність пошукової системи.
- Масштабованість: Архітектура дозволяє легко інтегрувати інші передтреновані трансформери для розширення функціональності або роботи з іншими мовами.

Результати

Отримані метрики

У результаті навчання моделі з використанням підходу curriculum learning були отримані наступні метрики для кожної фази:
Фаза 1 (легкі помилки):

- Точність символів (Char Accuracy): 84.22%
- Точність кінцевих символів (End Char Accuracy): 97.08%
- Точність довжини (Length Accuracy): 2.83%
- Levenshtein відстань: 0.9984
- Точні співпадіння (Exact Matches): 20.23%

Фаза 2 (середні помилки):

- Точність символів (Char Accuracy): 91.30%
- Точність кінцевих символів (End Char Accuracy): 97.75%
- Точність довжини (Length Accuracy): 2.12%
- Levenshtein відстань: 0.7289

- Точні співпадіння (Exact Matches): 31.54%
  Фаза 3 (складні помилки):
- Точність символів (Char Accuracy): 95.14%
- Точність кінцевих символів (End Char Accuracy): 99.63%
- Точність довжини (Length Accuracy): 2.08%
- Levenshtein відстань: 0.8706
- Точні співпадіння (Exact Matches): 30.12%

Порівняння з іншими підходами

Для порівняння ефективності моделі, було використано результати зі статті https://thesai.org/Downloads/Volume12No8/Paper_93-Grammatical_Error_Correction_with_Denoising_Autoencoder.pdf, де застосовувався схожий підхід для виправлення граматичних помилок у текстах англійською мовою.

- **F-score на тестовому наборі BEA 2019:** 73.90
- **F-score на валідаційному наборі BEA 2019:** 56.58

Хоча пряме порівняння F-score та Levenshtein відстані не є повністю коректним через різні задачі (граматичне виправлення проти виправлення пошукових запитів), наша модель демонструє високі показники точності і ефективності, зокрема у складних випадках. Це свідчить про високу узагальнюючу здатність DAE в різних доменах і підтверджує ефективність застосованих підходів для виправлення текстових помилок.

Альтернативні підходи до виправлення пошукових запитів

Окрім інтегрованого підходу DAE+BART, існують інші методи корекції пошукових запитів, що можуть бути застосовані в залежності від контексту використання:

- Нейромережеві підходи
  - Seq2Seq (LSTM/Transformer) – моделі перетворення послідовностей, які автоматично трансформують некоректні запити у правильні.

    - BERT-класифікація – використання попередньо натренованих моделей для визначення і корекції неправильних слів у контексті запиту.
- Лексикографічні та статистичні методи:
    - Алгоритми перевірки орфографії (Spell Checking), такі як Norvig's Spell Checker або SymSpell, що базуються на частотному аналізі словників.
    - Графові методи, що використовують зв'язки між запитами для виправлення помилок через аналіз структурних залежностей.
- Гібридні методи
    - Поєднання класичних алгоритмів перевірки орфографії з нейронними моделями, наприклад, попередня орфографічна корекція із наступним контекстуальним уточненням за допомогою трансформерів.

Висновки та подальші напрями розвитку

Розроблена модель Denoising Autoencoder (DAE) показала високу ефективність у виправленні локальних помилок у пошукових запитах, демонструючи точність понад 90% навіть для складних помилок. Застосування підходу curriculum learning дозволило моделі поступово адаптуватися до різних рівнів складності помилок, що позитивно вплинуло на загальні результати.

Однією з ключових проблем, з якою ми зіткнулися, було обмеження навчального датасету. Виключення коротких слів (до трьох літер) і слів із цифрами, а також розширення датасету реальними запитами дозволили уникнути перенавчання та підвищити узагальнюючу здатність моделі. Інтеграція реальних помилок із пошукових логів сприяла покращенню адаптації моделі до практичних сценаріїв.

Для виправлення складніших семантичних помилок було інтегровано модель BART, що дозволило досягти ще вищої точності завдяки глибокому контекстуальному аналізу. Порівняння з результатами моделей для граматичного коригування текстів, таких як DAE для BEA 2019, показало, що наша модель демонструє співставні або кращі результати у своїй доменній задачі.

У подальшому розвиток моделі буде зосереджений на впровадженні динамічного навчання на основі реальних користувацьких запитів і дослідженні інших трансформерних архітектур, таких як T5 чи BERT.

Крім того, передбачається впровадження наступних напрямів розвитку:

1. **Оптимізація продуктивності:** Зменшення часу відповіді моделі для роботи в режимі реального часу, використовуючи методи квантування моделей та оптимізації інференсу.
2. **Мультимовна підтримка:** Розширення функціональності моделі для роботи з іншими мовами, що дозволить застосовувати рішення в багатомовних середовищах.
3. **Адаптивне навчання:** Використання механізмів зворотного зв’язку від користувачів для автоматичного покращення моделі на основі реальних сценаріїв використання. Це дозволить моделі адаптуватися до змінних умов пошуку та забезпечувати ще кращу точність і ефективність виправлень.

Інструкції з запуску

Клонування репозиторію:

```
git clone https://github.com/kolenchuk/dae
cd dae
```

Налаштування віртуального середовища та встановлення залежностей:

```
python -m venv
venv source venv/bin/activate  # Для Linux/Mac
venv\Scripts\activate # Для Windows
pip install -r requirements.txt
```

Завантаження моделей та підготовка даних:

- Виконайте скрипт для генерації датасету:

```
python ./src/generate_dae_dataset.py
```

- Завантажте та збережіть модель BART:

```
python ./src/bart_loader.py
```

Навчання моделі:

```
python ./src/dae_train.py
```

Запуск API:

Варіант 1: Використання Docker.

```
docker-compose up --build
```

Варіант 2: Запуск без Docker.

Переконайтеся, що всі залежності встановлені:

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

Запустіть сервер API вручну:

```
uvicorn app:app --reload --port 8000
```

API буде доступне за адресою: http://127.0.0.1:8000

Перевірка роботи API:
Використовуйте інструменти типу Postman або CURL для тестування:

```
curl -X 'GET' 'http://127.0.0.1:8001/search?query=iphnoe%2013%20pro'
-H 'accept: application/json'
```