МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА БИБЛИОТЕКИ ДЛЯ СУММАРИЗАЦИИ ТЕКСТОВ МЕТОДАМИ ГЛУБОКОГО ОБУЧЕНИЯ

АВТОРЕФЕРАТ МАГИСТЕРСКОЙ РАБОТЫ

студента 2 курса 273 группы
направления 02.04.03 — Математическое обеспечение и администрирование
информационных систем
факультета КНиИТ
Лезгян Артема Саркисовича

Научный руководитель
к. ф.-м. н., доцент _____ А. С. Иванов

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире неуклонно увеличивается объем информации, которую так или иначе ежедневно обрабатывает каждый человек. Поэтому все более актуальной становится задача автоматической суммаризации текстовых данных. Краткое содержание длинных документов, новостных статей или переписок может помочь существенно сократить время, затрачиваемое на изучение или поиск той или иной информации.

Задача суммаризации обычно заключается в том, чтобы сжать некоторый текст до более короткого фрагмента, который при этом должен содержать основную информацию из оригинала. Данная задача сложна тем, что состоит не только в выделении наиболее значимых смысловых моментов исходного текста, но также обычно требует, чтобы получаемый на выходе текст был связным, построенным по правилам естественного языка. Суммаризация является одной из основных задач NLP и может применяться, например, для:

- непосредственно сжатия текста, что существенно снижает время, затрачиваемое человеком на чтение;
- решения задач информационного поиска, где краткое резюме может упростить процесс выбора документа или увеличить эффективность индексации;
- персонализированные резюме могут использоваться в вопросно-ответных системах, так как позволяют обобщать запросы пользователя и строить по ним соответствующий ответ.

Целью магистерской работы является создание библиотеки для построения суммаризации текстов.

Поставленная задача подразумевает выполнение следующих этапов:

- создание или поиск подходящего датасета русских текстов;
- реализация и обучение нескольких нейросетевых архитектур;
- написание программного интерфейса для работы с обученными моделями;
- создание архитектуры, позволяющей масштабировать полученное решение для практического применения;
- оптимизация производительности системы;
- анализ эффективности полученного решения.

1 Теоретические основы системы суммаризации

1.1 Подходы к задаче суммаризации

Существует несколько классификаций методов построения суммаризаций [1]. В данной работе применяются подходы, основанные на типе выходных данных. *Экстрактивный* метод проще, так как копирование текста исходного документа обеспечивает высокие показатели грамотности и точности. С другой стороны, этот метод лишен таких преимуществ, как возможность перефразировать исходный текст или обобщать информацию. Развитие языковых моделей, в особенности основанных на архитектуре Transformer, открыло возможность создания *абстрактивных* моделей, способных с высокой точностью обобщать информацию. Однако такие модели довольно часто оказываются чрезвычайно требовательными к вычислительным ресурсам.

На практике, в зависимости от поставленной задачи, данные подходы могут использоваться как по отдельности, так и как дополняющие друг друга шаги конвейера обработки данных.

1.2 Используемые архитектуры

1.2.1 GPT

GPT является по сути авторегрессионным декодировщиком из стандартной Transformer архитектуры [2] с некоторыми дополнительными модификациями. Все модели на основе архитектуры GPT обучаются языковому моделированию, то есть минимизации кроссэнтропии между исходным текстом и генерируемым на каждом шаге декодера.

1.2.2 BERT

Еще одной модификацией исходной Transformer является модель BERT [3]. Архитектура модели BERT представляет собой многослойный двунаправленный кодировщик, основанный на стандартном кодировщике Transformer. Главным отличием BERT от GPT является реализация механизма внимания. BERT использует двунаправленное самовнимание, позволяющее анализировать данные в контекстном окне в обоих направлениях, в то время как GPT использует ограниченное самовнимание, при котором каждый токен может обращаться только к контексту слева от него.

1.2.3 BART

BART — это автоэнкодер с шумоподавлением [4], построенный на основе модели seq2seq, которая применима к очень широкому кругу конечных задач. BART использует гибридную архитектуру, которая сочетает наработки из моделей BERT (из-за двунаправленного кодировщика) и GPT (из-за устройства декодера). BERT обучается восстанавливать случайным образом удаленные из текста символы. При этом текст сканируется в обоих направлениях, благодаря чему BERT представляет собой хорошую языковую модель, однако для генерации текста он подходит плохо. GPT обучается авторегрессионно генерировать некоторую последовательность данных. Благодаря этому данная модель хорошо подходит для генерации текстов, однако из-за того, что при генерации извлекается информация только из контекста слева от последнего слова, ухудшается качество извлечения контекстной информации. BART же объединяет преимущества обоих подходов.

1.3 Суммаризация длинных текстов

При резюмировании длинного текста человек обычно сначала пытается понять текст, потом выделяет наиболее важную информацию и наконец перефразирует ее в сжатом виде.

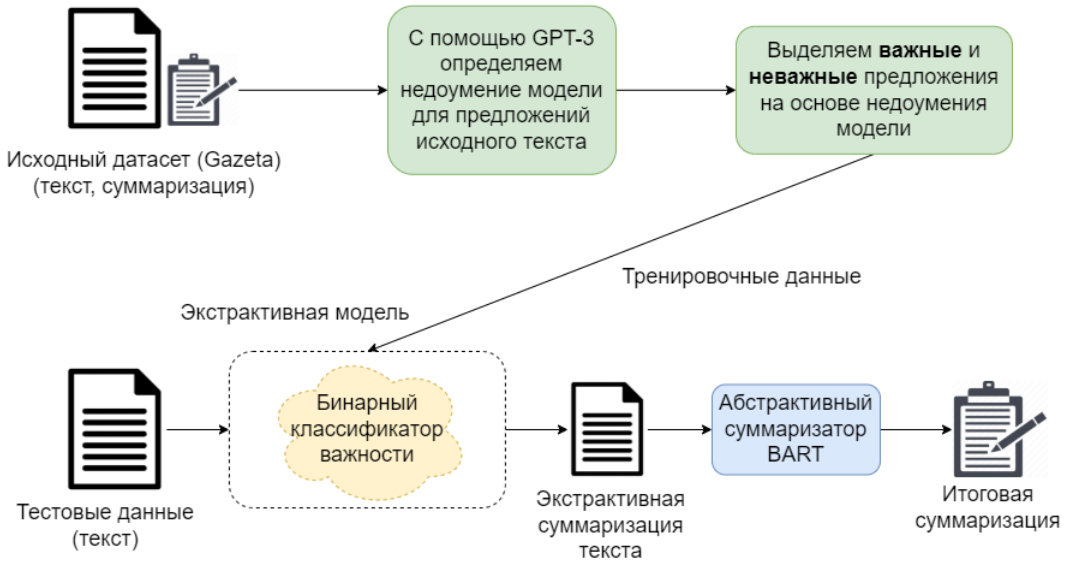

Рисунок 1 – Архитектура конвейера суммаризации длинных текстов

Используя подобную логику, можно сформулировать алгоритм суммаризации длинных текстов с использованием небольшого обучающего набора [5]:

— выделить для каждого текста наиболее важные предложения. Например, путем сравнения предложений из текста с предложениями из резюме и выделения тех, информация из которых была включена в итоговую сводку;
— обучить на полученных данных бинарный классификатор, с помощью которого можно будет извлекать из текста только предложения, несущие наиболее важную информацию, тем самым существенно сокращая его;
— сжатый документ суммаризировать предварительно обученной на схожих данных моделью, например BART или GPT-3.

1.4 Микросервисная архитектура

Архитектура разработанной системы суммаризации текстов представлена на рисунке 2. Система включает в себя микросервисы экстрактивного сжатия текста на основе бинарного классификатора BERT, микросервисы абстрактивной суммаризации на основе BART и сервер пользовательского интерфейса. Благодаря микросервисной архитектуре при возрастании нагрузки система может быть расширена путем увеличения числа реплик каждого сервиса. При необходимости для балансировки нагрузки между одинаковыми сервисами используется nginx прокси. Пользователь может взаимодействовать с системой при помощи WEB-интерфейса, разработанной клиентской библиотеки или непосредственно через REST-API.

1.5 Оптимизация логического вывода

1.5.1 Экспортирование обученной модели

Чаще всего перед этапом логического вывода требуется экспортировать обученную модель в некоторый универсальный формат, который позволит перенести ее в производственную вычислительную среду. Для этого может использоваться универсальный формат ONNX. Он поддерживается большей частью фреймворков машинного обучения и вычислительных сред, что в большинстве случаев позволяет с минимумом усилий переносить обученную модель.

Помимо универсального формата сериализации нейронных сетей, экосистема ONNX предлагает еще и кроссплатформенный ускоритель машинного обучения для логического вывода — ONNX Runtime. ONNX Runtime совместим с различным оборудованием, драйверами и операционными системами,

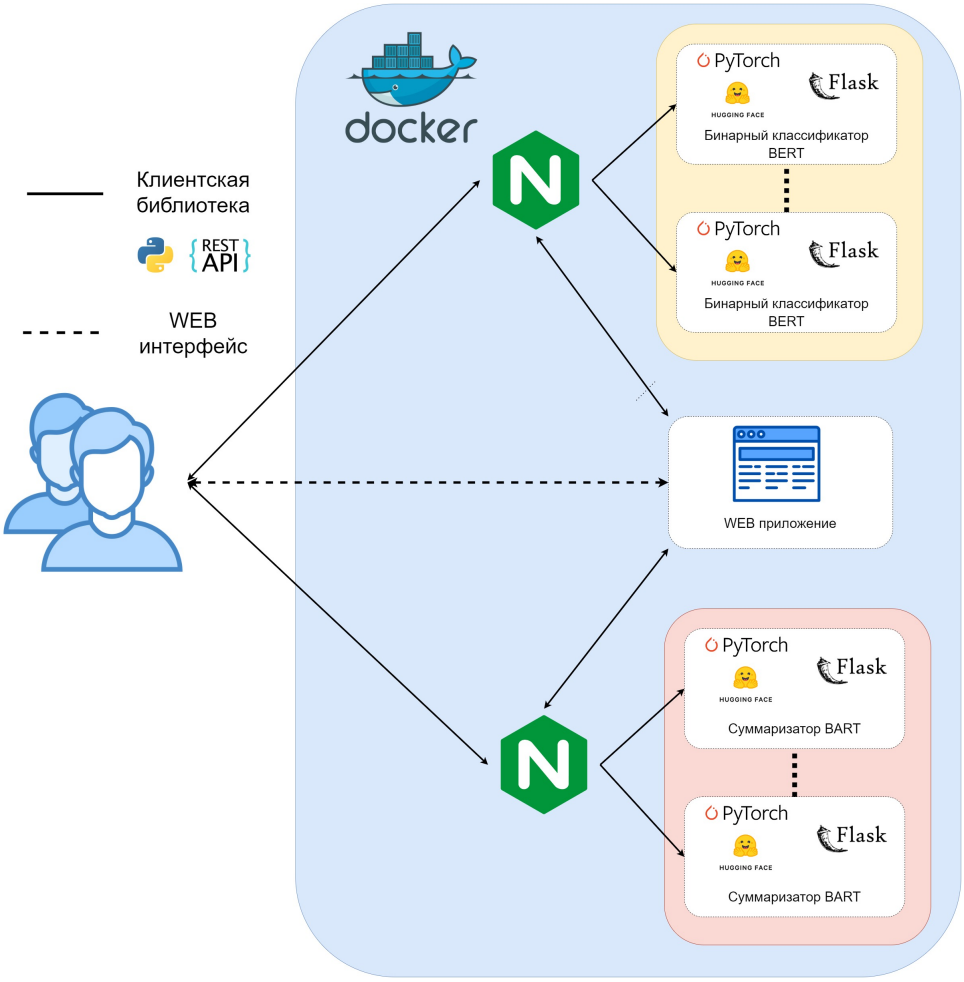

Рисунок 2 – Архитектура системы автоматической суммаризации текстов

и обеспечивает оптимальную производительность за счет использования аппаратных ускорителей, наряду с преобразованием и оптимизацией графа модели. Одним из главных преимуществ ONNX Runtime является широкий набор графовых оптимизаций, позволяющих без потери точности существенно ускорять логический вывод.

1.5.2 Сервер Triton

Использование высокопроизводительной вычислительной среды решает проблему быстродействия нейросетевой модели, однако еще одним фактором, снижающим общую производительность системы, является скорость работы сервера, предоставляющего доступ к обученной модели посредством REST-API. В качестве решения данной проблемы может использоваться NVIDIA Triton Inference Server — продукт с открытым исходным кодом, предназначенный для стандартизации, развертывания и масштабирования систем на основе машинного обучения.

2 Программная реализация

2.1 Особенности реализации нейросетевых моделей

2.1.1 Особенности обучения GPT-3

При обучении GPT-3 модели подаются на вход сконкатенированные через символ-разделитель исходный текст и его резюме. Генератор батчей обрезает исходный текст таким образом, чтобы итоговая длина текста и резюме не превышала максимальной для модели длины. Дополнительно сохраняется информация о индексе токена-разделителя. Это нужно для того, чтобы вычислять функцию потерь только для сгенерированной и целевой суммаризаций, игнорируя разделитель и исходный текст.

Чрезвычайно большой размер модели GPT-3 даже в минимальной ее версии (с 125 млн. параметров) не позволяет эффективно применять обучение на основе мини-батчей. 16 Гб видеопамяти хватает только на обработку одного обучающего примера за раз. Однако в большинстве случаев алгоритм градиентного спуска на основе мини-батчей намного эффективнее стохастического градиентного спуска. Решением может быть распараллеливание модели для обучения на нескольких GPU, однако чаще всего вычислительные ресурсы достаточно сильно ограниченны. В качестве решения данной проблемы может быть применено **накопление градиентов** gradient accumulation.

2.1.2 Особенности обучения BART

Обучение модели BART, по сути, ничем не отличается от обучения любой другой seq2seq модели. Сначала кодировщик считывает исходный текст, после чего декодировщик, опираясь на данные с предыдущего шага декодирования и информацию из кодировщика генерирует итоговую последовательность. Особенность обучения проявляется в том, что так как используемая в данной работе модель BART является мультиязычной, и в первую очередь предназначена для перевода, необходимо помимо специального токена `</s>` конца предложения добавлять в конец исходной текстовой последовательности специальный языковой токен, на основе которого модель устанавливает язык текстовой последовательности. Так же языковой токен добавляется в начало генерируемой последовательности, что позволяет модели определить целевой язык.

Помимо этого, для обучения моделей такого размера (за основу была

выбрана mbart-large-cc25 с 610 млн. параметров) недостаточно даже использования накопления градиента. Однако существует способ в два раза уменьшить вес модели, при этом ускорив ее обучение в 2.5–3 раза. Для этого используется замена 32-битных чисел с плавающей точкой на 16-битные числа половинной точности. Оптимальным решением является использование смешанного варианта, при котором определяются шаги, требующие полной точности и для них все значения приводятся к 32-битным числам. При этом во всех других случаях используются 16-битные.

2.2 Процесс обучения

Во время обучения каждые 50 батчей производилась валидация модели на тестовых данных. Для борьбы с переобучением использовалось сохранение чекпойнтов модели раз в 150 батчей, после чего была выбрана лучшая относительно метрик модель. Обучение производилось на видеокарте NVIDIA TESLA V100 с 16 Гб видеопамяти. При обучении использовался оптимизатор Adam [6].

Лучшей оказалась модель BART обученная на датасете Gazeta. GPT-3 продемонстрировала не самые лучшие результаты, однако это можно объяснить тем, что использовалась модель GPT-3 small с 125 млн. параметров, тогда как для тонкой настройки BART за основу была взята мультиязычная модель mbart-large-cc25 с 610 млн. параметров. Возможно, использование более крупных моделей GPT-3 улучшит результаты, но маловероятно, что они превзойдут показатели BART. Сама архитектура GPT-3 создана для генерации текста и испытывает некоторые проблемы с анализом исходного документа. Ансамбль из экстрактивного бинарного классификатора и абстрактивного суммаризатора BART показал себя немного хуже, чем BART, примененный к текстам без сжатия, однако это можно объяснить тем, что использовался новостной датасет, в котором основная информация статьи располагается в самом начале.

2.3 Реализация микросервисной архитектуры

Для создания образов обученных моделей использовался базовый образ, предоставляемый Nvidia nvcr.io/nvidia/pytorch:22.04-py3. Он содержит предустановленный фреймворк Pytorch, а также драйверы CUDA, необходимые для запуска моделей на GPU.

Запуск контейнеров, имеющих доступ к графическим ядрам, невозможен

с использованием базового функционала Docker, поэтому необходимо предварительно установить набор инструментов NVIDIA Container Toolkit. Набор инструментов включает в себя библиотеку контейнерной среды выполнения, а также утилиты для автоматической настройки контейнеров под использование графических процессоров NVIDIA.

Для настройки взаимодействия контейнеров используется конфигурация Docker Compose. Это инструмент, поставляемый с дистрибутивом Docker и позволяющий настраивать совместный запуск нескольких связанных друг с другом контейнеров и конфигурировать их.

2.4 Оптимизация системы

Для увеличения быстродействия самих моделей в данной работе используется экспорт в формат ONNX с последующей оптимизацией при помощи высокопроизводительной вычислительной среды ONNX Runtime. Для ускорения работы модели в кластере используется специализированный для задач машинного обучения сервер Triton.

2.4.1 Конвертация в формат ONNX

Для сериализации был выбран метод трассировки, поскольку в качестве основы для бинарного классификатора BERT и абстрактивного суммаризатора BART используются предобученные модели Hugging Face [7] с довольно сложной внутренней архитектурой, что делает достаточно трудоемкой задачу ручного переписывания модели на TorchScript.

При трассировке мы изначально подготавливаем некоторый набор данных, который будет передан на вход модели. Важно отметить, что так как размерности входных данных после трассировки станут неизменяемыми, нужно заранее выбрать оптимальные параметры. На втором шаге в память загружается предобученная модель и ее веса переносятся на необходимое устройство (CPU или GPU). Так же чрезвычайно важно перевести модель в режим валидации `model.eval()`, поскольку в противном случае трассировка сохранит в том числе и операции, необходимые только для обучения модели. Третьим шагом является непосредственно экспорт в формат ONNX.

2.4.2 Настройка сервера Triton

Для запуска экспортированной в формат ONNX модели на сервере Triton достаточно:

- сохранить модель в определенную директорию;
- описать конфигурацию каждой из используемых моделей и при необходимости правила их взаимодействия;
- создать Dockerfile с описанием образа сервера и указанием на директории, откуда должны быть скопированы конфигурация и обученные модели.

При необходимости можно добавлять собственные расширения, позволяющие строить сложные конвейеры обработки данных. Например, для BERT необходимо было добавить токенизатор, позволяющий полностью выполнять обработку текста на сервере.

2.5 Анализ результатов оптимизации

Во время применения всех вышеописанных шагов был проведен ряд экспериментов, призванных оценить влияние той или иной оптимизации на итоговую производительность каждой модели. Результаты экспериментов представлены в таблице 1.

Таблица 1 – Сравнение быстродействия при оптимизации ONNX

Модель	Формат	CPU, с	CUDA, с
Extractor BERT	PyTorch	3.77	0.311
	без оптимизации	3.21	0.25
	базовые оптимизации	3.15	0.23
	расширенные оптимизации	2.78	0.19
	оптимизация архитектуры	2.2	0.18
BART	PyTorch	29.77	2.26
	без оптимизации	25.15	1.9
	базовые оптимизации	23.12	1.67
	расширенные оптимизации	19.24	1.14
	оптимизация архитектуры	13.78	1.11

Можно заметить, что даже без применения каких-либо оптимизаций модели логический вывод с применением вычислительной среды ONNX Runtime примерно на 15% более производителен, чем при использовании PyTorch.

Наиболее эффективным для логического вывода на GPU оказался режим расширенных оптимизаций. Для BERT он позволил добиться 27% ускорения при использовании CPU и 40% ускорения на GPU. А для BART — 36% ускорения на CPU и двукратного увеличения производительности на GPU.

Так же было проведено сравнение производительности серверов для развертывания моделей в кластере (таблица 2).

Таблица 2 – Производительность серверных решений

Модель	Формат	CPU, с	CUDA, с
Extractor BERT	PyTorch Flask	5.98	0.446
	ONNX Flask	3.3	0.37
	ONNX Triton	2.5	0.21
BART	PyTorch Flask	32.77	3.30
	ONNX Flask	23.28	1.68
	ONNX Triton	18.57	1.29

Для сравнения в таблице в том числе приведены и результаты запуска в кластере исходных не оптимизированных моделей PyTorch. Хорошо заметно, что использование оптимизаций и вычислительной среды ONNX Runtime позволяет двукратно увеличить быстродействие полученных микросервисов, однако используемый сервер все еще остается узким местом системы. Применение NVIDIA Triton Inference Server позволяет решить и эту проблему, в среднем увеличивая общее быстродействие микросервиса на 20%. При этом, помимо увеличения производительности Triton предоставляет еще и удобный способ организации конвейеров обработки данных.

ЗАКЛЮЧЕНИЕ

В рамках данной работы были выполнены следующие поставленные задачи:

- выбраны подходящие для обучения суммаризации датасеты русских текстов;
- обучены нейросетевые модели BERT, GPT-3 и BART;
- реализована клиентская библиотека и пользовательский интерфейс для взаимодействия с обученными моделями;
- разработана и реализована микросервисная архитектура системы суммаризации текстов;
- произведена комплексная оптимизация нейросетевых моделей и серверной части системы;
- произведен анализ производительности полученного решения.

По материалам данной работы была опубликована статья «Разработка системы автоматического реферирования текстов методами глубокого обучения», материалы которой опубликованы в сборнике статей по итогам конференции «Молодой исследователь: вызовы и перспективы» [8].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Widyassari, A. P.* Review of automatic text summarization techniques & methods / A. P. Widyassari, S. Rustad, G. F. Shidik, E. Noersasongko, A. Syukur, A. Affandy, D. R. I. M. Setiadi // *Journal of King Saud University - Computer and Information Sciences*. — 2022. — Vol. 34, no. 4. — Pp. 1029–1046. https://www.sciencedirect.com/science/article/pii/S1319157820303712.

2 Attention Is All You Need [Электронный ресурс]. — URL: https://arxiv.org/pdf/1706.03762.pdf (Дата обращения 19.05.2022). Загл. с экр. Яз. англ.

3 BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [Электронный ресурс]. — URL: https://arxiv.org/pdf/1810.04805.pdf (Дата обращения 08.05.2022). Загл. с экр. Яз. англ.

4 *Lewis, M.* BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension // Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. — Virtual Event: Association for Computational Linguistics, 2020. — Pp. 7871–7880. https://aclanthology.org/2020.acl-main.703.

5 Long Document Summarization in a Low Resource Setting using Pretrained Language Models [Электронный ресурс]. — URL: https://arxiv.org/pdf/2103.00751.pdf (Дата обращения 03.05.2022). Загл. с экр. Яз. англ.

6 *Loshchilov, I.* Decoupled weight decay regularization // 7th International Conference on Learning Representations, (ICLR). — New Orleans, LA, USA: OpenReview, 2019. — Pp. 1–25.

7 Hugging Face Transformers [Электронный ресурс]. — URL: https://huggingface.co/docs/transformers/index (Дата обращения 05.05.2022). Загл. с экр. Яз. англ.

8 *Лезгян, А. С.* Разработка системы автоматического реферирования текстов методами глубокого обучения // Молодой исследователь: вызовы и перспективы: сб. ст. по материалам CCLVIII междунар. науч.-практ. конф. — № 16(258). — М.: «Интернаука», 2022. — С. 261–265.