EDN THKFAV DOI 10.5281/zenodo.14514613
УДК 004.853

МЕТОДЫ ПРОЕКТИРОВАНИЯ АРХИТЕКТУРЫ МОДЕЛИ ИНТЕЛЛЕКТУАЛЬНОГО АССИСТЕНТА ДЛЯ ОБРАЗОВАТЕЛЬНОЙ ОРГАНИЗАЦИИ

© 2024 ***Елисеев В.О., Бондаренко В.И.***

Большинство наиболее успешных больших генеративных языковых моделей (LLM) состоят из трех составных частей: кодировщика, декодера и механизма внимания. В статье были рассмотрены основные подходы к проектированию каждой из этих частей, а также наиболее популярные архитектуры LLM. Кроме того, был проведен сравнительный анализ методов обучения и применения больших генеративных языковых моделей на предмет оптимальности в использовании для построения интеллектуального ассистента образовательной организации. Результаты исследования показали, что при имеющихся ресурсах наиболее подходящим способом является точная настройка существующей LLM на текстовом корпусе данных об образовательной организации с последующим ее использованием напрямую либо посредством механизма RAG.

Ключевые слова: генерация текста, LLM, GPT, точная настройка, fine-tuning, чат-бот.

Введение. В настоящее время в университетах РФ наблюдается большой рост числа бюджетных мест, а соответственно и абитуриентов. Следствием этого является затратность и сложность коммуникации с абитуриентами, своевременного информирования студентов по интересующим их вопросам, приводящие к простоям и негативно сказывающиеся на качестве процесса набора и обучения студентов.

Ведущие вузы РФ решают эту проблему путем внедрения на свои платформы различных по назначению и сложности чат-ботов – от обычных коммуникаторов пользователя с администрацией вуза до более продвинутых автоматизированных версий, оперирующих знаниями о вузе или интересующих конечного пользователя его аспектах.

В связи с этим появляется задача создания инструмента, позволяющего сократить до минимума время ожидания пользователями ответа от представителей вуза и упростить работу администрации. Наилучшим решением этой задачи является создание собственного чат-бота, работающего с помощью большой языковой модели (LLM) и имеющего доступ к релевантным и актуальным данным о конкретной образовательной организации, что позволит ему давать качественные ответы на вопросы пользователей.

Для решения подобной задачи необходимо выполнить следующие шаги:

- провести исследование существующих методов разработки / обучения больших генеративных языковых моделей;
- собрать текстовый корпус данных об образовательной организации (в данном конкретном случае будет рассматриваться Донецкий государственный университет), подготовить его для тренировки LLM;
- выбрать архитектуру генеративной модели, которая будет обучена на подготовленном текстовом корпусе;

Работа выполнена при финансовой поддержке Минобрнауки России в рамках научной темы «Разработка и совершенствование интеллектуальных методов классификации и прогнозирования для задач распознавания образов и моделирования информационных процессов» FREM-2024-0001 (Регистрационный номер 1023111000141-9-1.2.1)

- обучить LLM на текстовом корпусе;
- реализовать сервис интеллектуального ассистента университета с веб-интерфейсом, либо в виде бота на одной из платформ (Telegram, VK и т. д.), интегрировать обученную модель в этот сервис, проверить ее работоспособность.

В данной статье выполним первые два шага плана: рассмотрим наиболее популярные и качественные методы построения больших генеративных моделей и возможные подходы к процессу тренировки модели на данных о конкретной образовательной организации, выберем наиболее подходящий в рамках решения задачи и соберем корпус данных, но основе которого будет производиться обучение.

Целью работы является рассмотрение и анализ методов построения и способов обучения больших генеративных языковых моделей с целью выбора их наиболее подходящей комбинации для решения задачи построения генеративной LLM, управляющей интеллектуальным ассистентом образовательной организации и строящей свои ответы на основе данных о нем.

Описание входных данных. В качестве текстового корпуса для решения поставленной задачи будут использоваться данные, полученные из официального сайта образовательной организации. Согласно методическим рекомендациям [1], сайт образовательной организации обязательно должен содержать такие разделы, как: «Основные сведения», «Структура и органы управления образовательной организацией», «Документы», «Образование», «Образовательные стандарты», «Руководство. Педагогический (научнопедагогический) состав», «Материально-техническое обеспечение и оснащённость образовательного процесса», «Стипендии и иные виды материальной поддержки», «Платные образовательные услуги», «Финансово-хозяйственная деятельность», «1 Вакантные места для приема (перевода)», «Доступная среда», «Международное сотрудничество», «Абитуриенту», «Уровни образования «Бакалавриат», «Специалитет», «Магистратура»», «Уровень образования «Аспирантура»», «Уровень образования «Ординатура»». Кроме того, допускается создание и собственных разделов, отражающих специфику конкретного образовательного учреждения.

Данные в разделах сайта образовательного учреждения могут храниться как в текстовом формате на HTML-страницах, так и в различных файлах. Согласно рекомендациям [1], допускается предоставление файлов в форматах Portable Document Files (.pdf), Microsoft Word/Microsoft Excel (.doc, .docx, .xls, .xlsx), Open Document Files (.odt, .ods), также рекомендуется делать файлы машинночитаемыми.

Исходя из регламентированности предоставления данных образовательными организациями о себе и преобладанием текстовых документов на сайтах образовательных организаций можно сделать вывод, что текстовый корпус, полученный на основе данных с сайта конкретной образовательной организации, будет обладать достаточно хорошим качеством для целей обучения LLM.

Поскольку нам важно максимальное погружение в специфику работы конкретной образовательной организации, мы будем использовать не строго унифицированные разделы целевого сайта, а полную его информацию, которую можно получить путем обхода его страниц. С этой целью был создан набор скриптов, позволяющих распарсить контент всех HTML-страниц сайта, а также загрузить все файлы, на которые будут найдены ссылки при обходе страниц. Подробнее о скриптах рассказано в тезисах [2].

Общее описание генеративных моделей. Генеративные модели искусственного интеллекта – алгоритмы, которые способны генерировать новые данные, имитируя статистические свойства исходных данных. В контексте процедурной генерации текстов, генеративные модели используются для создания текстов, которые имеют определенную структуру и логику [3].

Во время обучения, такие модели (при решении задач генерации текста) получают на вход текстовый корпус, в зависимости от архитектуры модели предобработанный или нет, изучают его структуру и особенности и пытается посимвольно создать новый текст на основе полученных знаний. Глобально такие модели состоят из кодировщика, декодера и иногда слоев внимания. Выбор конкретной реализации каждой из частей влияет на качество генерируемого текста.

Роль кодировщика – преобразовать входные текстовые данные в векторное представление, построить так называемые эмбеддинги [4]. Цель такого векторного преобразования – передать компьютеру такое числовое представление слов (формат, понимаемый им) чтобы эти числа отражали суть и семантику переданного слова. Таким образом на основе векторного расстояния между закодированными словами можно определить их взаимосвязь, что является первым шагом к успешной процедуре генерации. Стоит отметить, что построение эмбеддингов не обязательно привязывать к словам, вполне возможно строить их на основе целых предложений или частей слов. Еще одним из полезных свойств эмбеддингов является тот факт, что мы можем поставить в конец генеративной модели обычный линейный слой, обучить веса этого слоя на своем датасете (причем он может быть сравнительно небольшим) под собственную задачу (например, классификация текста) и такая модель также будет хорошо выполнять поставленную задачу. Подобный процесс называется тонкой настройкой модели.

Декодер преимущественно представляет собой RNN (рекуррентную нейронную сеть), получающую от кодировщика векторные представления и непосредственно генерирующую символьную последовательность. Благодаря своей архитектуре и учету предыдущих символов RNN является хорошим базовым решением для подобных задач.

Возможным улучшением архитектуры генеративной нейронной сети является добавление слоев внимания. Данный механизм позволяет модели фокусироваться на определенных частях входных данных, что делает процесс обучения более гибким и в целом увеличить качество генерации.

Для оценки качества генерируемого текста можно использовать семантическое сравнение сгенерированных предложений с эталонными, языковые модели, обученные для оценки правдоподобия сгенерированного текста, использовать такие метрики как BLEU [5], ROUGE [6], perplexity [7], а также использовать человеческую оценку результатов генерации, что лежит в основе подхода RLHF [8].

Далее рассмотрим основные подходы к построению кодировщиков и декодеров, механизм внимания, а также затронем вопрос токенизации текста и успешные существующие архитектуры генеративных текстовых моделей.

Использование word2vec для генерации эмбеддингов. Word2vec – метод эффективного создания вложений, разработанный в 2013 году. Кроме работы со словами, некоторые его концепции оказались эффективны в разработке рекомендательных механизмов и придании смысла данным в коммерческих, неязыковых задачах. Эту технологию применили в своих движках рекомендаций такие компании, как Airbnb, Alibaba, Spotify и Anghami.

Подробно реализация алгоритма word2vec описана в оригинальной статье [9]. Также в статье описывается один из возможных вариантов улучшения процесса обучения word2vec – архитектура skip-gram. Она основана на угадывании слова в середине сэмпла по предшествующим и последующим, то есть модель учится смотреть в 2 стороны. Еще одним способом совершенствования алгоритма является отрицательное сэмплирование [10].

В результате обучения word2vec мы получаем векторы для каждого слова, обладающие рядом интересных свойств. Во-первых, это свойство аналогии. Если из векторного представления слова «Париж» вычесть векторное представление слова «Франция» и добавить векторное представление слова «Россия», то ближайшим вектором к результату окажется векторное представление слова «Москва», что говорит о семантической наполненности полученного векторного представления для каждого слова, построенного моделью word2vec.

Еще одним свойством получаемых векторов является их близость, что означает сходство концептов слов, которые они кодируют. Это еще раз подтверждает семантическую наполненность векторных представлений слов и уже может использоваться для простейших задач классификации.

На базовом уровне алгоритм word2vec предоставляет весьма качественные эмбеддинги слов и может выступать в качестве кодировщика генеративной сети для самых простых задач. Для анализа предложений используются их векторные представления, полученные по формуле:

$$v(sentence) = \sum_{w} P(word)v(word),$$

где v – векторное представление, P – вероятность появления конкретного слова, w – слово в предложении.

Таким образом появляется возможность векторизации предложений/текстов и их дальнейшей обработки.

FastText от Facebook. FastText – библиотека, содержащая предобученные готовые векторные представления слов и классификатор, который можно настроить под собственную задачу.

Векторные представления слов создавались с помощью word2vec с отрицательным сэмплированием. Существенным улучшением подхода к тренировке являлось использование subword-модели наряду с основной. Данное улучшение решает проблему OOV (out of vocabulary), когда на вход модели поступает новое слово, ни разу не появившееся в датасете, его эмбеддинг не построен и соответственно обработка такого слова может привести к ошибкам в работе модели. Subword-модель является представлением слова через цепочки символов (n-граммы) с n от 3 до 6 символов (число n преимущественно зависит от языка – для языков с достаточно короткими словами и простой морфологией можно использовать небольшие n, иначе это число должно быть больше) от начала до конца слова плюс само слово целиком. Например, если взять n = 3 и слово «шахматы», то в модели оно будет представлено как набор n-грамм: <ша, шах, ахм, хма, мат, аты, ты> и последовательностью <шахматы>, причем модель понимает разницу между последовательностью <шах> из слова шах и n-граммой шах из слова шахматы. Результатом будет являться сумма усредненных эмбеддингов по n-граммам и эмбеддинга всего слова-последовательности. Такой подход позволяет работать с новыми словами – если последовательность не встречалась, ее эмбеддингом будет усреднение эмбеддингов n-грамм, что тоже несет семантический смысл и позволяет достигать результатов лучших, чем результаты word2vec. Более подробно процесс обучения fastText описан в статье [11].

Стоит отметить, что данный подход масштабируем и при наличии достаточного корпуса текста можно обучить модель на любом языке с символами любого вида. Таким образом были получены векторы слов для 157 языков, которые находятся в открытом доступе [12]. Исходный код библиотеки также находится в открытом доступе [13].

Рекуррентные нейронные сети в роли декодеров. В отличие от большинства архитектур нейронных сетей, поддерживающих фиксированные размеры ввода и вывода (линейные, сверточные нейронные сети), рекуррентные позволяют работать с входом и выходом разных размеров.

Работа с последовательностями обеспечивается следующим внутренним механизмом RNN: каждому последующему слою рекуррентной нейронной сети в качестве входа передается не только входной вектор с данными, но и выходной вектор предыдущего слоя, которые вместе (посредством конкатенации) образуют матрицу весов для конкретного слоя RNN. Таким образом нейронная сеть помнит свое состояние на предыдущем шаге и способна учитывать его при своей дальнейшей работе, что, например, полезно для задач генерации текста, когда мы по набору предыдущих символов-состояний предсказываем следующий символ. В качестве функции активации внутри блока RNN в основном используется гиперболический тангенс, поскольку он может работать и с отрицательными значениями (в отличие от RELU), а также растет менее быстро, что позволяет адекватно обработать большее количество контекста.

RNN является базовой архитектурой построения декодера, далее рассмотрим варианты ее улучшения и другие подходы к генерации текста.

Сети LSTM как более продвинутая вариация декодеров. Одной из серьезных проблем классических RNN является их неспособность удерживать в памяти слишком большие последовательности. Это происходит потому, что когда градиент проходит через большое количество ячеек, то либо он уменьшается настолько, что теряется связь между ошибками в начале и в конце последовательности, либо разрастается до такой степени, что процесс обучения расходится (даже при использовании более подходящей и плавно растущей функции активации).

Следующим шагом в развитии декодеров стала замена ячеек RNN на ячейки LSTM [14] (long short-term memory). К обычной ячейке RNN добавляется внутреннее состояние S_t, которое преобразовывается способом, основанном на использовании skip connections, при меньшем количестве преобразований, чем у идущего через все ячейки и градиенты состояния H_t, что помогает решить проблему исчезающих или взрывающихся градиентов. Также были добавлены input и output gate, влияющие на то, что из входных данных окажет влияние на внутренние состояние ячейки и что из внутреннего состояния должно отправится на выход.

Одной из последующих модификаций ячейки LSTM являлось добавление forget gate, задачей которого является зануление некоторых компонент S_{t-1} перед дальнейшей передачей. Это может быть особенно полезно, например при анализе длинной последовательности-отзыва на товар на предмет тональности. Так, в самом начале отзыва могут проступить негативные слова-маркеры, но дальше он приобретет смысл близкий к нейтральному или позитивному (возможна и обратная ситуация), поэтому иногда полезно «забыть» слишком длинный предварительный контекст.

Еще одним дополнением к базовой ячейке LSTM является peephole connection, которое учитывает состояние S_{t-1} для вычисления всех ворот, что делает поведение ячейки концептуально более осмысленным.

Таким образом LSTM является более продвинутым и качественно превосходящим вариантом декодера, в связке с word2vec или fastText (на вход передаются не символы, а векторные представления слов) такая архитектура может выдавать достаточно качественные результаты для задачи генерации текста и ряда других задач NLP.

Механизм внимания (attention). Несмотря на отличия LSTM от RNN, проблемы последней все равно имеют место быть. В частности, это заметно при решении задачи машинного перевода, при которой последовательности символов одной длины соответствует последовательность символов совершенно другой, и при попытке перевести текст слишком большого размера с помощью обычной сети LSTM люди сталкивались с проблемой удержания контекста. Улучшения в архитектуре, связанные с разделением на кодировщик и декодер, имеющие разные веса, а также ряд трюков, связанных с особенностями языка (попытка научить декодер предсказывать слова в обратном порядке и др.), механизмами дополнительной передачи контекста к блокам декодера привели к созданию архитектуры seq2seq [15], которая была SOTA-решением для домена машинного перевода в 2014 году. Однако такой подход не решал всех проблем, и вопрос обработки длинных последовательностей оставался открытым.

Одним из серьезных улучшений архитектуры является использование механизма внимания (attention), призванного искать взаимосвязи между различными частями входных и выходных данных. Впервые данный механизм был описан в статье [16] и был успешно применен для улучшения архитектуры seq2seq.

При помощи механизма внимания достигается «фокусирование» декодера на определенных скрытых состояниях. В задаче машинного перевода эта возможность помогает декодеру предсказывать на какие скрытые состояния при исходных определенных словах на языке A необходимо обратить больше внимания при переводе данного слова на язык B, то есть учесть другие слова в предложении при переводе, что «оживит» результат. Стоит отметить, что вся система внимания дифференцируема, а значит мы можем полностью ее оптимизировать в процессе градиентного спуска.

Результат работы такого механизма визуализирован на рисунке 3. На нем представлено переведенное с английского на французский язык предложение с помощью seq2seq с добавленным механизмом внимания. Чем ярче закрашенный прямоугольник, тем больше вес соответствующего слова в механизме внимания.

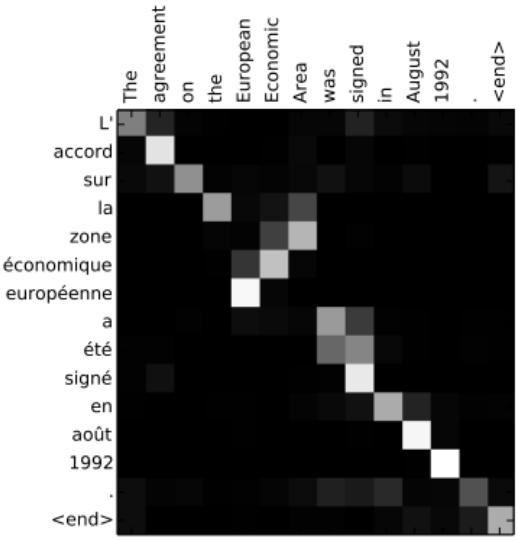

Рис. 1. Визуализация механизма внимания при переводе предложения с английского языка на французский

Архитектура Transformer. Новым шагом в развитии NLP стала архитектура Transformer, представленная Google в 2017 году в статье «Attention is all you need» [17]. Ее новизна связана с тем, что у модели отсутствуют блоки RNN, CNN и LSTM, а используется лишь механизм self-attention – внимание ко внутреннему состоянию (что и послужило причиной такого названия статьи), благодаря чему модель может сосредоточиться не на всех сразу, а на наиболее важных элементах входной последовательности, так как self-attention определяет веса важности элементов входной последовательности и не требует сохранять состояния, что решает упомянутые выше проблемы RNN. Схематически архитектура Transformer представлена на рисунке 4.

Сеть с одинаковыми весами параллельно запускается на каждом слове входной последовательности. На вход дается слово Inputs, проходящее векторизацию в блоке Input Embedding, далее результирующий вектор складывается с информацией о расположении слова в последовательности – Positional Encoding. Когда эти данные достигают слоя Multi-head attention, каждый входной вектор, посредством механизма внимания, может взаимодействовать с другими параллельно обрабатываемыми словами. Использование механизма внимания в таком подходе позволяет нам отойти от передачи внутреннего состояния как это было у RNN и LSTM.

Об устройстве блока Multi-head attention подробно можно прочитать в статье [17], можно отметить, что на входе ему поступает эмбеддинг обрабатываемого в данный момент слова (query), а также ряд пар key и value (которые всегда равны), и представляющие собой эмбеддинги слов, на которые смотрит конкретный блок посредством механизма внимания. Далее каждый из этих векторов проходит через обучаемый линейный слой, после чего вычисляется скалярное произведение query со всеми key, результат проходит через softmax и полученные веса суммируются с векторами value, после чего на выходе получается единый вектор. Таким образом мы можем использовать механизм внимания не только при декодинге, но и во время работы кодировщика, сеть может видеть, что происходит при кодировке других слов что делает кодировку конкретного слова более контекстуально осмысленной.

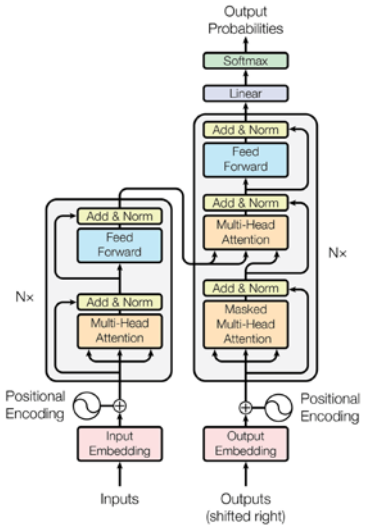

Рис. 2. Архитектура модели Transformer

Декодер работает также по одному слову, на вход он получает прошлое сгенерированное слово (или токен <start>, если процесс находится на первой итерации). В отличие от кодировщика, у декодера есть 2 блока Multi-head attention – первый способен обращаться к векторам уже декодированных слов, а второй – к выходу

кодировщика, где query представляет собой вектор входа в декодере, а пары Key/Value – финальные эмбеддинги кодировщика, где key равен value, но линейные преобразования внутри модуля внимания для них разные. В конце сети находится softmax, выдающий вероятности появления слов на выходе сети. По его результату выдается следующее слово в предложении, процесс повторяется пока не будет выдан токен <end of sentence>.

Данная архитектура полностью дифференцируема, что позволяет ее оптимизировать во время градиентного спуска. В предложенной в статье архитектуре глубина кодировщика составляет 8 блоков внимания, а декодера – 6. Основное различие механизмов работы кодировщика и декодера заключается в том, что во время кодирования каждый вектор взаимодействует со всеми другими, а во время декодинга каждое следующее слово взаимодействует с предыдущими и с векторами кодировщика.

Данный подход стал SOTA решением в домене машинного перевода, на рисунке 5 можно увидеть зависимость метрики BLEU [5] от архитектуры нейронной сети-переводчика (тесты проводились для перевода с английского языка на немецкий).

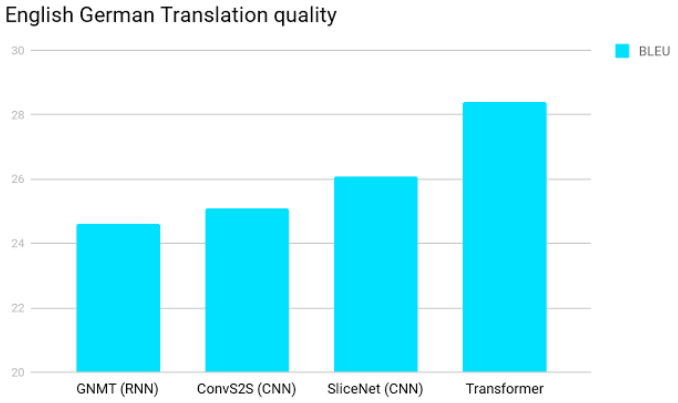

Рис. 3. Превосходство архитектуры Transformer над другими в задаче машинного перевода

Архитектура BERT. Архитектура BERT (Bidirectional Encoder Representations from Transformers – «двунаправленные презентации кодировщика для трансформеров») была представлена компанией Google в статье [18]. Она стала SOTA решением на момент выхода для ряда задач NLP, включая генерацию ответов на вопросы, формирование рассуждений и др. Модель была натренирована на большом количестве текста и была представлена в ряде конфигураций – максимальная из которых состояла из 340 млн. параметров и 24 блоков трансформеров.

Подход к тренировке выделялся тем, что если все предыдущие исследования, связанные с тренировкой языковых моделей, рассматривали текст как однонаправленную последовательность либо только слева направо, либо сочетали обучение слева направо и справа налево, то в данном случае было предложено двунаправленное обучение способное достичь более глубокого понимания языкового контекста и потока, чем модели, натренированные однонаправленно. В статье [18] была описана техника под названием MLM (Masked Language), позволяющая проводить двунаправленное обучение в моделях, для которых ранее это было невозможно.

Еще одним подходом к тренировке BERT является NSP (прогнозирование следующего предложения). При данном подходе в качестве входа модель получает пары фраз и учится предсказывать, является ли вторая фраза продолжением первой. Входные данные разбиты на пары таким образом, что половина пар действительно связана между собой, а другая половина заполняется случайным образом.

Одной из особенностей архитектуры BERT является ее способность к тонкой и точной настройке. Так, заменив последний линейный слой, можно поставить перед моделью задачу классификации, генерации текста и другие. Таким образом были натренированы вариации BERT, способные нативно работать с русским языком [19]. Также возможно натренировать BERT под задачу классификации, на официальной странице BERT на Hugging Face [20] существует специально заготовленный набор классов для гибкой настройки последнего линейного классифицирующего слоя.

Архитектура BERT была прорывом в области машинного обучения в задачах NLP, особенно благодаря возможности проводить с ней transfer learning и универсальности. Модель демонстрировала SOTA результаты во многих доменах NLP до появления архитектуры GPT, о которой пойдет речь дальше.

Generative Pretrained Transformer (GPT). Архитектура GPT была впервые представлена компанией OpenAI в 2018 году в статье [21]. Эта архитектура была также основана на трансформерах, но если до этого все модели использовали подход «обучение с учителем», то для GPT проводили обучение с частичным привлечением учителя. Обучение GPT проходило в 2 этапа: сначала проводилось генеративное обучение без учителя на большом неразмеченном текстовом корпусе, а после – тонкая настройка под конкретную задачу на сравнительно небольшом размеченном наборе данных.

OpenAI решили, что от трансформера нужно оставить только декодер, потому архитектура модели состояла из 12-слойного декодера трансформера с 12-ю слоями masked head self-attention (то есть при обучении использовался подход предсказания замаскированного слова в последовательности, или MLM). Модель имела 117 млн. параметров, еще одним из улучшений в процессе тренировки стало использование оптимизатора Adam [22] вместо применяемого ранее стохастического градиентного спуска. Также стоит отметить, что на протяжении всей тренировки (как генеративного обучения без учителя, так и при тонкой настройке с учителем) модель работала не напрямую с текстом, а с его числовым представлением в виде токенов. О процедуре токенизации речь пойдет в следующем подразделе. В качестве данных для обучения использовался набора данных BookCorpus [23], содержащий в себе текст из 7000 книг.

Первая версия модели показала себя плохо в задачах генерации текста, однако показала ряд SOTA результатов на задачах NLP, под которые была проведена процедура тонкой настройки модели, что показало высокое качество usupervised language modeling pretraining в области построения эмбеддингов, способных хорошо работать после донастройки под конкретные задачи.

В 2019 году OpenAI выпустили обновленную модель GPT-2 [24], которая имела уже 48 слоев и 1.5 млрд. параметров. Также помимо книг в качестве данных для тренировки использовались данные с 8 млн. сайтов, суммарно вышло около 40 Гб текста. Подобное количественное улучшение принесло свои плоды – модель улучшила свои результаты на задачах, под которые была произведена тонкая настройка и значительно лучше стала справляться с задачами генерации текста: модель смогла писать длинные связные тексты и решать ряд новых задач при помощи prompt engineering. Также увеличился размер запоминаемого моделью контекста – если GPT-1 запоминала только 512 токенов, то для GPT-2 это значение выросло до 1024.

Архитектура GPT показывает хорошие результаты при использовании после тонкой и точной настройки. Так, исследователи из компании Сбер провели процедуру точной настройки модели GPT-2, натренировав ее на текстовом корпусе на русском языке [25], и полученная модель продемонстрировала высокие результаты, наибольшая по размеру вариация модели легла в основу сервиса GigaChat.

Также настроенная под русский язык модель GPT показала превосходство над русскоязычной моделью BERT в задаче анализа тональности текстов разных размеров (покупательские отзывы и новостные статьи). Модель RuGPT_small показала значение усредненной F1-меры для отзывов, равное 0.721, в то время как RuBERT – 0.647 и 0.726 для новостей в то время, как RuBERT показала 0.582 [26]. Это говорит о более качественной работе GPT на больших текстовых последовательностях и более качественных эмбеддингах, получаемых при обработке входных последовательностей. Процесс тонкой настройки в проводимом исследовании заключался в замене линейного классифицирующего слоя новым, имеющим 3 выхода (по числу классов), заморозке предыдущих слоев модели и тренировке нового слоя, что дает возможность судить о качестве эмбеддингов слов, которыми оперирует модель при принятии решения.

Токенизация текста. Токенизация – процедура преобразования текста в последовательность чисел. Она необходима ввиду того, что модели машинного обучения справляются лучше с числовыми последовательностями, нежели текстовыми.

Существует множество методов токенизации – от присваивания каждому слову конкретного номера, до построения специальных токенизирующих моделей машинного обучения. Коротко рассмотрим наиболее применяемые методы токенизации на подслова: WordPiece, ULM, BPE, BBPE, BPE-Dropout. Стоит отметить, что процесс токенизации не является ресурсозатратным и может выполняться на CPU. Токенизация на подслова – попытка усреднения токенизации на уровне слов (имеющей недостаток в виде огромного размера словаря) и токенизации на уровне символов, ведущей к потере семантического значения слова. Также мы решаем проблему OOV, поскольку модель будет работать не с представлениями слов, а с представлениями их частей.

BPE-токенизация (byte pair encoding) – восходящий алгоритм токенизации подслов, который изучает словарь подслов определенного размера (размер словаря является гиперпараметром). Идея BPE-токенизации заключается в том, что более частым словам следует присваивать уникальные идентификаторы, тогда как менее частые слова следует разбивать на единицы подслов, которые лучше всего сохраняют свое значение [27]. Подробно алгоритм описан в оригинальной статье [28]. Данный вид токенизации используется в моделях GPT от OpenAI.

BBPE-токенизация (Byte-level Byte Pair Encoding) представляет собой модификацию BPE-токенизации, работающую не с текстом напрямую, а с его байтовым представлением. Подробно метод был описан в статье [29]. Данный метод используется в модели GPT-2 от OpenAI и в семействе натренированных моделей на русском языке RuGPT. Размер словаря для большой и средней модели GPT-2 составляет 50257 токенов.

WordPiece представляет собой метод, близкий к BPE-токенизации, к котором при слиянии токенов оптимизируется не частота совместного появления, а правдоподобие подобных появлений. Был предложен в статье [30] и использовался в архитектуре BERT. Популярность метода сошла после того, как авторы RoBERT показали отсутствие существенных преимуществ данного метода перед BPE.

ULM представляет собой подход к задаче токенизации, в котором предполагается независимость подслов и то что вероятность p(x) слова x равна произведению вероятностей входящих слов p(xi). Далее ищется подходящая сегментация и на ее основе составляется словарь токенов. Создается большой словарь любым алгоритмом (от простого перебора символов и часто встречающихся подслов со словами до BPE словаря). После этого, до тех пор, пока размер словаря не достигнет порога, словарь фиксируется, с помощью алгоритма EM [31] максимизируется p(x), для каждого

подслова вычисляется функция потерь (уменьшение правдоподобия при удалении из словаря), удаляется 20% слов с наименьшим значением функции потерь (наименее важные слова). Символы из словаря не удаляются, так как полное удаление может привести к проблеме OOV. Подробно алгоритм описан в оригинальной статье [32].

BPE-Dropout является методом-комбинацией классического BPE и ULM. Метод был направлен на устранение таких проблем BPE, как однозначная сегментация каждого слова и слабая интерпретируемость подслов редких слов. Предлагается использовать словарь и таблицу слияний BPE и на каждом шаге слияния случайно его пропускать. В таком случае вероятность пропуска слияния p является настраиваемым гиперпараметром: если p = 0, то получаем классический алгоритм BPE, если p = 1, то получаем сегментацию по буквам. Подробнее алгоритм описан в оригинальной статье команды Яндекса [33]. По мнению авторов, оптимальным значением вероятности p является 0.1. В практических задачах метод продемонстрировал результаты, превосходящие результаты классических BPE и ULM.

Подходы к обучению генеративных LLM. Для построения генеративных LLM обычно используется одни из следующих методов: построение LLM с нуля (подход использовался компанией Bloomberg для построения модели BloombergGPT [34]), точная настройка существующей LLM на собственных данных (метод был использован компанией Google для создания модели, обрабатывающей медицинские данные, Med-PaLM2 [35]), настройка модели через расширение контекста prompt-запросы (модель остается неизменной, собственные данные подгружаются через prompt). Далее рассмотрим эти методы в целом и в контексте решаемой задачи.

Вариант с созданием LLM с нуля подходит только для тех, у кого есть огромные вычислительные ресурсы и текстовые корпусы высокого качества. Необходимы миллиарды слов и кластеры GPU чтобы качественно реализовать данный подход. Так, при создании BloombergGPT использовалась финансовая документация за 40 лет, а также большой текстовый корпус из сети Интернет. В общей сложности было задействовано 350 млрд. слов, полученная модель имела 50 млрд. параметров, а на процесс тренировки модели было потрачено 1.3 млн. часов обработки графических единиц [34]. Большой проблемой подхода является и тот факт, что необходимо проводить не только предметно-ориентированную тренировку, но и обучение модели основам языка, что также занимает множество часов обработки графических единиц. Поэтому в рамках решаемой задачи подобный подход не является оптимальным.

Подход с расширением контекста модели путем prompt-engineering не требует вычислительных мощностей, предполагая неизменность весов и архитектуры модели, что полностью исключает необходимость использования GPU. Также этот подход не требует наличия больших текстовых корпусов. Он был применен компанией Morgan Stanley, добавившей к модели GPT-4 данные со 100 000 документов на тему инвестирования. Проблема ограниченности размера контекстного окна решалась путем внедрения векторного сложения (с эмбеддингами имеющихся данных, полученными от другой модели). При вводе пользователем запроса в систему, алгоритм сходства определяет, какие векторы должны быть отправлены модели, тем самым экономя место в контекстном окне модели, оставляя в нем только необходимые данные. Данный подход называется RAG (Retrieval-Augmented Generation) [36]. И хотя он не является ресурсозатратным, качество ответов модели может быть менее предметно-ориентированным чем ожидается, возможны проблемы галлюцинаций модели (особенно при передаче кардинально не знакомого ей контекста, который к тому же может быть неверно представлен внешней векторизующей моделью) и в целом

качество генераций может быть довольно низким. Поэтому данный подход также не будет использоваться в качестве основного при решении поставленной задачи, однако все еще может быть полезен для улучшения качества работы генеративной модели.

Оптимумом между двумя рассмотренными выше подходами является точная настройка существующей LLM на имеющихся данных. Данный метод не требует огромных вычислительных ресурсов и текстовых корпусов данных (достаточно иметь от сотен тысяч до миллионов слов вместо миллиардов, как это необходимо для полноценного создания LLM с нуля). Так, модель Med-PaLM2 от Google, обученная подобным образом, смогла ответить на 85% вопросов для получения медицинской лицензии США. Схожий метод был использован и при адаптации моделей BERT и GPT под русский язык, модели семейства RuGPT показывают хорошие результаты в практическом применении. Также удобство использования данного метода заключается в том факте, что ряд крупных LLM (таких как GPT-2, LLAMA и другие) являются OpenSource решениями, что позволяет оптимизировать процесс дополнительного обучения модели на собственных данных.

Последний рассмотренный подход является наиболее подходящим для решения поставленной задачи, поскольку его реализация сочетает в себе как практическую значимость – ответы полученной модели будут предметно-ориентированными, так и исследовательскую, поскольку в реализации подхода будет задействован полный цикл обучения LLM. При этом для реализации точной настройки модели не нужно располагать огромными вычислительными ресурсами и текстовыми корпусами данных, что делает процесс реализуемым без привлечения сверхмощных ПК и кластеров GPU. Также возможно совершенствование работы модели путем комбинации данного подхода и механизма RAG, что позволит актуализировать базу знаний модели без необходимости в постоянном дообучении, а также позволит улучшить качество генерации.

Выводы. По результатам рассмотрения и анализа методов построения и способов обучения больших генеративных языковых моделей можно сказать, что в качестве основы архитектуры будущей LLM наиболее подходящим выбором будет GPT, поскольку данная архитектура является SOTA решением во многих задачах как NLP в целом, так и генерации текста в частности. Кроме того, данная архитектура показывает хорошую способность к дообучению на собственных данных, что было продемонстрировано в соответствующей секции.

В качестве метода обучения LLM, оптимальным выбором является процедура точной настройки существующей LLM (в конкретном случае – основанной на архитектуре GPT), поскольку данный подход сочетает в себе как высокие понимание моделью предметной области и качество генерации ответов, так и более низкую ресурсозатратность процесса обучения, относительно полной тренировки LLM с нуля. Также возможно улучшение качества генераций полученной в результате точной настройки модели с помощью механизма RAG.

Поэтому для создания LLM, управляющей интеллектуальным ассистентом образовательной организации, будет использовано точное обучение генеративной LLM, основанной на архитектуре GPT. В качестве тренировочного набора данных будет использоваться внутренняя документация образовательного учреждения, а также данные о нем как с официальных, так и с ряда открытых источников в сети Интернет. Далее, полученная модель будет работать с обновляемой базой знаний, дополняя свои ответы ее содержимым с помощью RAG.

СПИСОК ИСТОЧНИКОВ

1. Методические рекомендации представления информации об образовательной организации высшего образования в открытых источниках с учетом соблюдения требований законодательства в сфере образования (вступают в силу с 01.09.2024). – Федеральная служба по надзору в сфере образования и науки. – 2024. – 101 с.
2. Бондаренко, В. И. Подготовка данных и выбор способа обучения модели интеллектуального ассистента для образовательной организации / В. И. Бондаренко, В. О. Елисеев // Донецкие чтения 2024: образование, наука, инновации, культура и вызовы современности: Материалы IX Международной научной конференции (Донецк, 15–17 октября 2024 г.). – Том 2: Физические, химические, технические и компьютерные науки. Часть 2. – Донецк: Изд-во ДонГУ, 2024. – С. 127-130.
3. Deep Learning и генеративные модели для создания процедурных текстов [Электронный ресурс]. – URL: https://aiplayzone.ru/tehnologii-i-metody/deep_learning_i_generativnye_modeli_dlja_sozdanija_protsedurnyh_tekstov/ (дата обращения 08.02.2024).
4. Wang, L [et al.] Text embeddings by weakly-supervised contrastive pre-training. – 2022. – URL: https://arxiv.org/abs/2212.03533.
5. Reiter, E. A structured review of the validity of BLEU / E. A. Reiter // Computational Linguistics. – 2018. – V. 44. – №. 3. – P. 393-401.
6. Lin, C. Y. Rouge: A package for automatic evaluation of summaries / C. Y. Lin // Text summarization branches out. – 2004. – P. 74-81.
7. Miaschi, A. [et al.] Is Neural Language Model Perplexity Related to Readability? / Computational Linguistics CLiC-it 2020. – 2020. – P. 303.
8. Casper, S [et al.] Open problems and fundamental limitations of reinforcement learning from human feedback. – 2023. – URL: https://arxiv.org/abs/2307.15217.
9. Mikolov, T. [et al.] Efficient estimation of word representations in vector space. – 2013. – URL: https://arxiv.org/abs/1301.3781.
10. Goldberg, Y. [et al.] word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. – 2014. – URL: https://arxiv.org/abs/1402.3722.
11. Joulin, A. [et al.] Bag of tricks for efficient text classification. – 2016. – URL: https://arxiv.org/abs/1607.01759.
12. Word vectors for 157 languages [Электронный ресурс]. – URL: https://fasttext.cc/docs/en/crawl-vectors.html (дата обращения 11.02.2024).
13. FacebookResearch. FastText [Электронный ресурс]. – URL: https://github.com/facebookresearch/fastText (дата обращения 11.02.2024).
14. Staudemeyer, R. C. [et al.] Understanding LSTM – a tutorial into long short-term memory recurrent neural networks. – 2019. – URL: https://arxiv.org/abs/1607.01759.
15. Sutskever, I. [et al.] Sequence to sequence learning with neural networks. – 2014. – URL: https://arxiv.org/abs/1409.3215.
16. Bahdanau, D. [et al.] Neural machine translation by jointly learning to align and translate. – 2014. – URL: https://arxiv.org/abs/1409.0473.
17. Vaswani, A. [et al.] Attention is all you need. – 2017. – URL: https://arxiv.org/abs/1706.03762.
18. Devlin, J. [et al.] Bert: Pre-training of deep bidirectional transformers for language understanding. – 2018. – URL: https://arxiv.org/abs/1810.04805.
19. Kuratov, Y. [et al.] Adaptation of deep bidirectional multilingual transformers for Russian language. – 2019. – URL: https://arxiv.org/abs/1905.07213.
20. BERT [Электронный ресурс]. – URL: https://huggingface.co/docs/transformers/model_doc/bert (дата обращения 14.02.2024).
21. Radford, A. [et al.] Improving language understanding by generative pre-training / Computer Science, Linguistics. – 2018. – 12 p.
22. Kingma, D. P. [et al.] Adam: A method for stochastic optimization. – 2014. – URL: https://arxiv.org/abs/1409.0473.
23. Bookcorpus [Электронный ресурс]. – URL: https://huggingface.co/datasets/bookcorpus (дата обращения 15.02.2024).
24. Radford, A. [et al.] Language models are unsupervised multitask learners // OpenAI blog. – 2019. – V. 1. – №. 8. – P. 9.
25. Zmitrovich, D. [et al.] A family of pretrained transformer language models for Russian. – 2023. – URL: https://arxiv.org/abs/2309.10931.

26. Бондаренко, В. И. Анализ эффективности глубоких языковых моделей для задачи определения тональности русскоязычных текстов / В. И. Бондаренко, В. О. Елисеев, Т. В. Ермоленко // Проблемы искусственного интеллекта. – 2024. – № 1(32). – С. 51-62. – EDN KJRRQW.
27. Пикалев, Я. С. Адаптация нейросетевой модели ALBERT для задачи языкового моделирования / Я. С. Пикалев, Т. В. Ермоленко // Проблемы искусственного интеллекта. – 2020. – № 3(18). – С. 111-122. – EDN INNNNX.
28. Sennrich, R. [et al.] Neural machine translation of rare words with subword units. – 2015. – URL: https://arxiv.org/abs/1508.07909.
29. Wang, C. [et al.] Neural machine translation with byte-level subwords. – 2020. – URL: https://arxiv.org/abs/1909.03341.
30. Schuster, M. Japanese and korean voice search / M. Schuster, K. Nakajima // 2012 IEEE international conference on acoustics, speech and signal processing (ICASSP). – IEEE, 2012. – P. 5149-5152.
31. Dellaert, F. The expectation maximization algorithm: Technical Report number GIT-GVU-02-20 / F. Dellaert // College of Computing, Georgia Institute of Technology, 2002. – 7 p.
32. Kudo, T. Subword regularization: Improving neural network translation models with multiple subword candidates. – 2018. – URL: https://arxiv.org/abs/1804.10959.
33. Provilkov, I. [et al.] BPE-dropout: Simple and effective subword regularization. – 2019 – URL: https://arxiv.org/abs/1910.13267.
34. Wu, S. [et al.] Bloomberggpt: A large language model for finance. – 2023. – URL: https://arxiv.org/abs/2303.17564.
35. Singhal, K. [et al.] Towards expert-level medical question answering with large language models. – 2023. – URL: https://arxiv.org/abs/2305.09617.
36. Lewis, P. [et al.] Retrieval-augmented generation for knowledge-intensive nlp tasks / Advances in Neural Information Processing Systems. – 2020. – V. 33. – P. 9459-9474.

Поступила в редакцию 13.09.2024 г., рекомендована к печати 04.10.2024 г.

METHODS OF DESIGNING THE ARCHITECTURE OF AN INTELLIGENT ASSISTANT MODEL FOR AN EDUCATIONAL ORGANIZATION

Eliseev V.O., Bondarenko V.I.

The vast majority of powerful modern LLM's consist of encoder, decoder and attention mechanism. This paper reviews the main approaches to designing each of these components, as well as the most popular LLM architectures. Additionally, a comparative analysis of training methods and applications of large generative language models was conducted to evaluate their optimal use in developing an intelligent assistant for an educational organization. The results of the study indicate that, given the available resources, the most suitable approach is to fine-tune an existing LLM on a textual corpus of data related to the educational organization and subsequently use it either directly or through the RAG mechanism.

Keywords: text generation, LLM, GPT, fine-tuning, chatbot.

Елисеев Вадим Олегович
стажер-исследователь лаборатории интеллектуальных систем ФГБНУ «Институт прикладной математики и механики»,
Российская Федерация, ДНР, Донецк
E-mail: eliseevv02@mail.ru

Eliseev Vadim Olegovich
Intern Researcher at AI Systems Laboratory of Institute of Applied Mathematics and Mechanics,
Russian Federation, DPR, Donetsk.

Бондаренко Виталий Иванович
кандидат технических наук, доцент, доцент кафедры компьютерных технологий ФГБОУ ВО «Донецкий государственный университет»,
Российская Федерация, ДНР, Донецк
E-mail: bondarenko@donnu.ru

Bondarenko Vitalii Ivanovich
Candidate of Technical Sciences, Associate Professor, Associate Professor at Department of Computer Technologies of Donetsk State University,
Russian Federation, DPR, Donetsk.