Слайд 1 [Вступление]:

Здравствуйте, Меня зовут \_\_ и сегодня я расскажу про языковые модели, про то, что это такое, как это работает и почему сейчас этот термин так популярен.

Слайд 2 [Содержание]:

В ходе доклада затронем такие темы как векторное представление слов, в т.ч. эмбеддинги, рассмотрим задачу языкового моделирования и разные подходы к ее решению и разберем их применение в популярных гпт и ллама, также затронем проблему оценки качества.

Слайд 3 [НЛП и языковые модели]:

Но для начала давайте оговорим понятия.

**NLP –** это общее направление ИИ и матлингвистики. Изучает проблемы понимания языка и генерацию грамматически корректного текста.

Языковая модель – распределение вероятностей по последовательностям слов.

Стоит отметить, что отдельно выделяют подтермин **Большие языковые модели (**[**языковая модель**](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C)**, состоящая из**[**нейронной сети**](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C)**со множеством параметров (обычно миллиарды весовых коэффициентов и более), обученной на большом количестве неразмеченного текста с использованием**[**обучения без учителя**](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B1%D0%B5%D0%B7_%D1%83%D1%87%D0%B8%D1%82%D0%B5%D0%BB%D1%8F)**.),** но об этом позднее.

Слайд 4 [задачи NLP]:

Мы уже разграничили NLP и ЯМ, но т.к. история ЯМ неразрывно связана с NLP, то немало важно будет охарактеризовать задачи nlp.

- Классификация текста.

Обычная задача классификации в области НЛП, в которой нужно классифицировать тексты по каким-то параметрам.

Часто отдельно выделяют подзадачу

- анализ настроения

Часто встречается для оценки обновления приложения в маркете/карточек товара на маркетплейсе/политического события политтехнологами. Разумеется, что такой анализ не дает избыточной оценки, но дает дополнительные критерии для анализа.

- генерация текста

Это генеративные модели по типу известного вам GPT/Deepseek etc. Их задача -генерация правдоподобного текста.

- Машинный перевод

Одна из ключевых задач НЛП с большой историей. В настоящее время в этом направлении добились колоссальных успехов благодаря трансформерам. Очень близка к задаче генерации текста

- Вопросно-ответные системы

Примером может служить известная многим система Вольфрам. Он может решать не только интегралы, но и отвечать на вопросы, сформулированные на естественном языке, о [математике](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0), [физике](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%B7%D0%B8%D0%BA%D0%B0), [астрономии](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%82%D1%80%D0%BE%D0%BD%D0%BE%D0%BC%D0%B8%D1%8F), [химии](https://ru.wikipedia.org/wiki/%D0%A5%D0%B8%D0%BC%D0%B8%D1%8F), [биологии](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F), [медицине](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B4%D0%B8%D1%86%D0%B8%D0%BD%D0%B0), [истории](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%82%D0%BE%D1%80%D0%B8%D1%8F), [географии](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F), [политике](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B8%D1%82%D0%B8%D0%BA%D0%B0), [музыке](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%B7%D1%8B%D0%BA%D0%B0), [кинематографии](https://ru.wikipedia.org/wiki/%D0%9A%D0%B8%D0%BD%D0%B5%D0%BC%D0%B0%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84), а также информацию об известных людях и интернет-сайтах.

- Суммаризация текста

Нужно кратко аннотировать какой-то документ. Год назад реализована в гугл доксах.

- Диалоговые системы

Например, чат-боты в тг, иммитация собеседника

- Распознование именованных объектов.

Допустим, вам нужно реализовать модель, которая бы каждый день создавала дайджест самых важных новостей. Вы бы связали ее с сотней источников и ей нужно было бы из текста вычленять различные сущности (финансовое значение, личность, дата и тп) и придавать им разное значение. Для валют могуть быть какие-либо колебания, упоминание личностей может помогать отделять одни новости от других. Также эти сущности должны остаться после суммирования.

- Мультимодальность

На самом деле, нынешний ChatGPT уже является не LLM, а Large Multi Modal. Это означает, что он может работать не только в системе текст-текст, а еще и с обработкой аудио, картинок и тп.

Слайд 5 [Эмбеддинг]:

— общее название для различных подходов к моделированию языка и обучению представлений в обработке естественного языка, направленных на сопоставление словам (и, возможно, фразам) из некоторого словаря векторов из Rn для *n*, значительно меньшего количества слов в словаре. (source [Векторное представление слов — Википедия](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81%D0%BB%D0%BE%D0%B2))

hint:В широком смысле, эмбеддинг - это процесс преобразования каких-либо данных (чаще всего текста, но могут быть и изображения, звуки и т.д.) в набор чисел, векторы, которые машина может не только хранить, но и с которыми она может работать.

Проще говоря ***эмбеддинг*** *- это способ преобразования чего-то абстрактного, например слов или изображений в набор чисел и векторов*. Эти числа не случайны; они стараются отражать суть или семантику нашего исходного объекта ([Эмбеддинги для начинающих / Хабр](https://habr.com/ru/companies/otus/articles/787116/))

Эмбеддинг – учитывает семантику, векторное представление – не обязательно?

Т.к. одна из задач эмбеддингов – сохранение некоторого смысла, то можно подразумевать под эмбеддингов разновидность векторного представления слов, отражающую семантику

На слайде можете увидеть на фото образный пример того, как смысловая компонента должна работать (близкие слова лежат рядом, можно получать какие-то слова осмысленной комбинацией слов форм)

Слайд 6

Разумеется, учитывать смысл сложнее, чем не учитывать, поэтому давайте кратко рассмотрим векторное представление.

Самое простое, что может прийти в голову, если нам нужно как-то представить слова в числовом виде – это каждое слово соотнести с каким-то числом. В случае вектора мы просто каждое слово кодируем в виде 1 единицы и n-1 нулей, где n – размер нашего словаря.

Из плюсов такого кодирования можно перечислить лишь простоту, из минусов:

○Векторы слов не отражают смысл слова;

○Нельзя измерить “похожесть” двух слов по смыслу;

○Векторы довольно разрежены, требуют много лишней памяти;

○Размер словаря ограничен;

○Слова, не попавшие в словарь, не могут быть обработаны;

○При изменении размера словаря нужно пересчитывать векторы заново.

Слайд 7

Мешок слов – это способ кодирования предложений, в котором каждому столбцу привязано слово, а строке – предложение. На их пересечении – количество слова Х в предложении У.

Мешок слов (Bag of Words, BoW):

○На основе one-hot кодирования слов можно построить кодирование предложений.

○Вектор предложения — сумма векторов его слов.

Слайд 8

●Наследуются все недостатки one-hot encoding:

○Векторы предложений не очень хорошо отражают смысл предложения. Порядок слов не учитывается;

○Векторы довольно разрежены, требуют много лишней памяти;

○Фиксированный размер словаря. Слова, не попавшие в словарь, не могут быть обработаны.

○При изменении размера словаря нужно пересчитывать векторы заново

○

Еще один недостаток BoW — различные слова могут иметь разную важность для текста. Например, в английском языке часто в предложениях фигурируют служебные частицы. Артикли a/an, the etc. В предложении I ate an apple каждое слово встречается по одному разу, но это не значит, что они равнозначны.

Слайд 9.

(Получается, что важность не просто пропорциональна количеству упоминаний. )

Допустим, у нас есть 10 документов и мы хотим суммаризировать важную информацию по ним. Мы можем выписать самые частые слова в теги или по скелету самых частых составить текст. Но мы уже пришли к тому, что важность слова не определяется его частотой напрямую.

Мы не можем не учитывать какие-то грамматически служебные слова, т.к. их служебность определяется контекстом. Попробуем использовать контекст следующим образом: мы посчитаем частоту слов в документе и умножим на обратную частоту встречаемости каждого слова во всех документах.

На слайде написана формула, tf понятно, как считается, в idf – log от отношения количества всех документов к количеству документов, в которых встречается рассматриваемое слово хотя бы раз. Тогда для для всяких частиц, которые есть во всех документах, получим важность log 1 = 0.

Слайд 10

1. Можем использовать это и для 1 текста для предложений
2. Активно используется для ранжирования поисковой выдачи. Когда мы посылаем какой-то запрос, то упрощая можем сказать, что для каждого слова из нашего запроса составляется tf-idf к уже обработанной выборке сайтов и происходит их ранжировка.

Слайд 11 Плюсы и минусы тф-айдиф:

 Плюсы:

-Векторы имеют больший смысл, чем при BoW;

-Возможность решать такие задачи, как ранжирование документов и выделение ключевых слов;

Недостатки:

-Векторы все еще остаются довольно разрежеными;

-Фиксированный размер словаря (при добавлении нового слова придется все пересчитывать)

-При изменении коллекции документов векторы нужно пересчитывать

Слайд 12

Есть и еще один алгоритм, но на нем останавливаться не будем. Latent semantic analysis. Представляем нашу матрицу в виду свд/усв разложения. Особенностью будет являться то, что в матрицу У будут векторы документов, а в В – векторы слов. И тогда мы сможем сравнивать документы и слова по некоторой метрике (напр, чем больше скалярное произведение, тем ближе они будут).

Слайд 13 Эмбеддинги

Перейдем к эмбеддингам. Пусть у нас есть какой-то набор документов. Составим таблицу, в которой подсчитаем количество каждого слова в контексте всех остальных. Т.е. как часто слово Х идет вместе со словами А1, …Аn. Слова А1…An и будут называться контекстом. Как правило, под контекстом подразумевают два слова до и два после рассматриваемого. Можно и 1, но берется 2 обычно чтобы выходить за предлоги и артикли. Ну а брать больше – больше вычислений и избыточное усложнение.

Слайд 14 + и –

 Плюсы:

* Векторы начинают отражать смысл слов! Их можно сравнивать на схожесть по расстоянию (cosine distance/MSE)

Недостатки:

* Векторы все еще довольно разрежены, требуют много лишней памяти;
* Размер словаря ограничен. Слова, не попавшие в словарь, не могут быть обработаны;
* При изменении размера словаря нужно пересчитывать векторы заново;
* Векторы редких слов не очень информативны

С разреженностью можно справиться. Например, можно через метод понижения размерности PCA урезать размер с 50к до 500. Либо воспользоваться СВД-разложением.

Но главное отличие эмбедингов от простого векторного представления заключается в том, что мы можем провести некий процесс обучения.

Слайд 15 Ворд2век

Мы будем учить нейросеть по слову предсказывать слова, которые могут находиться в контексте (стоять вокруг этого слова).

Наш датасет — набор текстов. Мы будем идти по датасету скользящим окном размера 5, и в каждом положении окна по центральному слову учить нейросеть предсказывать слова, находящиеся в текущем окне.

Формализуем задачу:

-Ставится задача классификации. Количество классов — размер словаря n.

-На вход нейросеть принимает слово, выдает n значений — распределение на слова в словаре.

-Лосс-функция — кросс-энтропия между распределением, выданным сетью, и верным распределением (one-hot вектором)

Слайд 16

Будем применять на вход one-hot код слова, далее умножать на матрицу А. После слоя А не будет функции активации. После умножения на выходе получаем i-тую строку м-цы А. Потом полученный вектор умножается на матрицу B. Получаем вектор. I-тая строка – эмбеддинг слова А как центрального, а i-тый столбец Б дает эмбеддинг этого же слова для контекста. Процесс обучения заключается в подборе весов, т.е. значений матрицы А и Б таким образом, чтобы скалярное произведение слов было наибольшим для самых часто встречаемых между собою слов и меньше для редко встречаемых.

Слайд 17

Под близостью подразумевается косинусное расстояние. Тогда наиболее близкие по смыслу слова будут лежать на одном луче. Наглядный пример того, как работает ворд2век можно увидеть на уже виденной картинке.

Слайд 18

**Преимущества**:

-Векторы отражают смысл слов;

-Размерность векторов не зависит от размера словаря;

-При добавлении документов векторы можно дообучить.

Недостатки:

-Фиксированный размер словаря. При изменении размера словаря документов векторы нужно пересчитывать;

-Для редких слов эмбеддинги получаются неоптимальными;

-Слова, имеющие один корень, обрабатываются нейросетью по-разному. eat, eater, eatin

Слайд 19:

Разумеется, ворд2век не единственный эмбеддинг. У него есть две разновидности:

Skip-Gram – предсказание слов контекста по центральному слову

CBOW – предсказание центрального слова по контексту.

Существует еще отдельные подходы как GloVe и FastText.

Идея Fasttext построена на разрешении проблемы Ворд2века, что различные словоформы обрабатываются по-разному.

Суть подхода заключается в том, что мы строим векторы не для целых слов, а для их частей. Для этого мы делим слова на n-граммы (пример можно увидеть на слайде. Это совокупность всех n идущих друг за другом букв слова). Настраиваем векторы для н-грамм и получаем вектор слова как сумму его н-грамм

Плюсы:

-Можно получить более адекватные эмбеддинги для редких и неизвестных слов; Недостатки:

-n-грамм может быть очень много. Требуется больше вычислительных ресурсов.

Слайд 20

Глов же использует просто статистику следования слов друг за другом, полученную с корпуса текстов и близка к реализации СТАТИСТИЧЕСКИХ моделей.

Идея модели лежит на поверхности, много где применяется в самых разных вариациях даже в ХХ веке, поэтому сложно назвать авторов или точную дату создания. Однако этот метод популярен до сих пор — используется в клавиатурах смартфонов для исправления опечаток и быстрого набора текстов через Т9.

Реализуется с помощью цепей Маркова. Ну и тут видно, что мы преобразуем Р от н элементов, можем дальше продолжить и получим произведение Р от в1 на Р от в2 при условии в1, и так до Р от вн при условии в1…в н-1, что видно на примере. На практике редко берется контекст больше трех, т.к. объем хранимых данных для N слов контекста длины К равен как N^K, да и связанный контекст больше трех слов редко повторяется.

Достоинства статистических моделей:

* Простота имплементации.
* Высокая скорость работы алгоритма.
* Низкая вычислительная стоимость обучения и инференса.

Недостатки статистических моделей:

* Не сможет сгенерировать слова, которые не шли подряд в обучающем корпусе.
* Очень маленький контекст.
* Длинные последовательности равновероятны ≈ нулю (в цепях Маркова для длинных последовательностей много множителей меньше нуля, поэтому их произведение уже практически равно нулю для любых множителей). Отсюда алгоритм не может выдавать разумные продолжения большой длины.

Слайд 21 Токены

Следует упомянуть о том, что вообще такое токены. Повсеместно встречающийся термин в нлп. Токен – это наименьшая единица текста, которую различает алгоритм. Для ворд2века мы использовали токены как слова. Но токенами могут быть и н-граммы и любые морфемы слова.

Раньше предложение разбивалось на слова по пробелам, знакам препинания, исключались стоп-слова и так далее (назовем это [CountVectorizer](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html" \t "_blank)). Но у этого подхода возникали две проблемы с разными словоформами. Они:

* Либо обозначались разными токенами, что не совсем верно, ведь слово-то одно и то же. И получалось, что похожим смыслом обладало сразу несколько токенов.
* Либо приводились к начальной форме — и в итоге терялся падеж, время, число.

Современные токенизаторы построены на алгоритме BPE (Byte Pair Encoding) Мы задаем фиксированное число токенов. И в словарь добавляются все символы из текста, ищутся самые частые их сочетания и снова добавляются. Этот процесс продолжается до тех пор, пока число токенов не станет равно заданному значению.

Слайд 22 RNN

Посмотрим на полносвязную нейронную сеть. Если мы ей даем какой-то токен/эмбеддинг, то он проходит насквозь, мы получаем какое-то распределение следующих слов, после чего прогоняем самое вероятное, опять получаем распределение. По сути, результат будет близок тому, что получим, если будем в Т9 постоянно жать на слово по середине. Осмысленного текста, как мы выше уже оговорили, мы не получим. Проблема в том, что мы никак не учитываем контекст.

Слайд 23 RNN

Но как мы можем его учитывать? Нам нужно как-то передавать информацию об уже обработанных словах в процесс обработки текущего слова.

Рекуррентная нейросеть пошагово обрабатывает все слова. К слою добавляется связь “из себя в себя” — “память” слоя.

Слайд 24. Память РНН

Как это реализуется? Во время прогонки через скрытый слой мы не только умножаем веса на фичи + байес, но и прибавляем произведение вектора прошлого состояния на веса U. Так получаем вектор памяти t-того состояние и уже его умножаем на веса V и получаем через функцию активации выходной вектор игрек

Слайд 25

Для того чтобы было понятнее, сравним со слоем обычной нейросети.

Из-за рекуррентной основы алгоритм обучения процесс подбора весов называется обратное распространение ошибки сквозь время (backpropagation through time)

У RNN есть недостатки:

В зависимости от функции ошибки градиенты могут сильно возрастать и доходить до необрабатываемых компьютером значений и обращаться в NaNы, это называется взрыв градиента. Обычно они исправляются установкой верхних пределов. Т.е. при преодолении числа 100 мы просто дальше не можем растить значени.

Может быть и противоположная проблема – быстрое затухание. Тогда наша нн не будет учитывать слова, которые были 3-4 слова назад, т.е. будет быстро забывать контекст. Для этого используются оптимизаторы LSTM, GRU, но это отдельная тема и тут мы не затронем ее.

Слайд 26.

РННки вполне хорошо справлялись с задачами перевода по сравнению с аналогами, но вероятно вы помните, что в годах 15-16 интернет переводчики работали все еще неидеально. Очень часто перевод получался чопорным, не связным. И в 2017 году все поменялось. Потому что инженеры из гугла опубликовали статью на 10 страничек и представили миру архитектуру трансформер. Как можете видеть на слайде, она называлась Внимание – все что вам нужно.

Слайд 27

Как работает трансформер? В рекурентных сетях мы могли обрабатывать токены поэтапно. Это мешало распараллеливанию и не позволяло в разы ускорить процесс обучения за счет железа. Но когда мы хотим перевести какое-то предложение с одного языка на другой, то часто возвращаемся к предыдущим или впереди стоящим словам для уточнения их связи/формы/рода. Трансформер фокусируется сразу на нескольких словах и не завязан на реккурентности. Он состоит из двух частей (представлены в архитектуре) – энкодер и декодер.

Таким образом, энкодеры обладают следующими особенностями:

* Анализируют входной текст и связи между токенами.
* Выделяют важные токены для определённой задачи.
* Ничего не генерируют.

Энкодер, БЕРТ, читал текст с двух сторон.

На базе декодеров сделаны GPT-модели. Они обучаются предсказывать следующий токен на основе предыдущих. На инференсе, когда очередной токен сгенерирован, он добавляется в контекст, и уже на основе него выбирается новый токен. Таким образом модель:

* генерирует токен за токеном.
* смотрит на весь контекст, архитектурно, нет забывания токенов.
* имеет возможность (как и BERT-модели) обучаться параллельно.
* обладает достаточно высокой вычислительной стоимостью инференса.

**Контекст в случае трансформеров определяется числом токенов, которые они могут обработать за раз.** Поэтому пару лет назад много внимания уделялось именно количеству токенов, которые БЯМ может обработать за раз. Архитектурно за понимание контекста отвечает блок Attention, и размеры матриц в нём как раз определяют размер контекста.

Слайд 28

GPT-1 & GPT-2

Начнём немного издалека, с моделей GPT-1 и GPT-2.

Первая была обучена в 2018 году на 7000 книг и имела размер контекста в 512 токенов. И она сразу получилась довольно сильной: после дообучения на специализированные задачи (бенчмарки) показывала на них лучшее на то время качество.

Вторая модель была обучена в 2019 году. Она состояла из рекордных для того времени 1,5 млрд параметров (то есть была в ~10 раз больше первой), имела контекст в 1024 токена и была обучена на 40 ГБ текстовых данных. GPT-2 снова побеждала предыдущие подходы, включая GPT-1, на многих бенчмарках.

По сравнению с первой версией модели у второй произошел качественный рост: теперь она могла генерировать разумные тексты — а не только предложения. Правда, не всегда и не с первой попытки.

GPT-3

GPT-3 стала революцией с точки зрения качества и размеров. В 2020 году была получена модель размером в 175 млрд параметров, она обучалась на 570 ГБ текстовых данных с контекстом в 2048 токенов. Модель [могла](https://arxiv.org/abs/2005.14165) решать целый спектр задач, включая перевод, суммаризацию и ответы на вопросы, с качеством, близким к человеческому уровню, а также отличалась высокой способностью генерировать креативный контент.

Однако для применения решений остаётся проблема со стоимостью их обучения. Для обучения GPT-2 авторы использовали 16 GPU (иначе говоря — графических процессоров, видеокарт), а для GPT-3 уже 3200. Для дообучения модели под определенную задачу, конечно, понадобится меньше ресурсов, но всё равно достаточно много.

Что с этим делать? Использовать подводки.

Слайд 29:

Чтобы улучшить качество выдаваемых ответов нейросети можно использовать примеры. Например, нам нужно перевести слова с одного языка на другой. Если мы приведем n-примеров, а потом выпишем, какие нужно доделать, то это будет называться n-shot, а в общем случае few-shot подводка.

Слайд 30:

Подобно человеку, которому дают пример того, что от него требуют, БЯМы тоже повышают качество своих ответов при few-shot подводке

Вообще promt-tuning сильно повышает качество ответов. Все дело в том, что нейросеть обучается на текстах разных типов по типу сайтов/книг и тп. Она имеет большие знания в разных областях. Но формат заданий, которые она получает, отличается от ее датасета. Можно привести в пример студента, который хорошо знает лекционный материал по всем предметам за курс, но абсолютно не знаком с практикой. Если ему дать какое-то практическое задание, то решение в лучшем случае займет больше времени, чем у студентов, соблюдающих баланс лекций и практик. В силу ограниченной вычислительной мощности языковой модели, она скорее всего даст просто плохой результат. Но если привести пример студенту того, как его знания применяются в решаемой им задаче, то скорее всего он тоже сможет ее решить за приемлемое время. Так и нейросеть в первую очередь обучается на лекционной информации, без переложения ее на практику.

Слайд 31.

Одним из способов повысить качество для сложных задач является цепочка рассуждений. Суть заключается в том, чтобы дать не только ответ, но и привести подробное решение задачи. Пример можете видеть на слайде. Такой подход сильно улучшает качество для задач по типу поиска производной сложной функции, решение алгоритмической задачи на языке программирования.

Слайд 32

Реализовать дообучение нейросети на основе таких дозапросов получилось в инструктГПТ. Эта модель смогла хорошо отвечать на запросы даже при зеро-шоте. Успех заключался в том, что ее дообучали на наборе инструкций (типо СоТ) и добавили такую ревард-модель, что все ответы должны будут получать максимальное вознаграждение относительно следования инструкциям. ИнструктГПТ стал прародителем ЧатаГПТ.

К сожалению, детали чата гпт не были выложены в открытый доступ, но в общих чертах успех был достигнут за счет наследования логики инструкта гпт и изменению трейна обучения: было добавлено обучение на диалогах, улучшено качество и объем датасета, разметки, а также улучшена ревард-модель и этап с реинфорсемент лернинг

Слайд

Как обучить свою ЛЛМ?

Нам нужен будет сотни ГПУ и в целом большие вычислительные мощности (можно арендовать)

Терабайты данных (можно взять с таких сайтов как COmmonCrawl, C4)

Архитектура

Данные должны быть разнообразными и полными (чтобы не только стихи могла писать, но и физику решать)

Нужно будет очистить от дубликатов и хтмл-тегов и тп регулярками.

Существуют эмпирические законы обученности модели, но можно использовать упрощенный подход – количество обработанных за обучение токенов.

Если взять одну из самых популярных последних архитектур – Ллама, то это примерно 1-2 триллиона токенов. Архитектура Ллама представляет собой видоизмененный трансформер.

Дальше дообучить инструкциям, настроить ревард-систем и обучение с подкреплением.

Но очевидно, что с такой задачей вы вряд ли столкнетесь, потому что она требует больших финансовых издержек. Поэтому на практике большая часть исследований в НЛП занимается промт/файн и тп тюнингом моделей и пытается улучшить качество ответов.