Лекция 1. Что такое эмбеддинг и логика этой идеи

Подробное, последовательное изложение: определения, интуиция, формулы с «человеческой» расшифровкой, связь подходов, геометрия, типы эмбеддингов, практические рекомендации и типичные ошибки.

1.1. Интуиция «на пальцах»

Мы хотим превратить символические объекты (слова, токены, предложения, документы, пользователи, товары) в векторы чисел фиксированной длины так, чтобы близкие по смыслу объекты были геометрически близки. Это и есть эмбеддинг: функция f: X → R^d, сопоставляющая каждому объекту x ∈ X вектор f(x).

Почему это нужно:

- Моделям нужны численные признаки; «один-хот» огромен и не несёт семантики.
- Непрерывное пространство позволяет использовать геометрию: косинусная близость, усреднение, аналогии.
- Переносимость: предобученный словарь векторов можно использовать в новых задачах.

Мини-схема расположения эмбеддинга в пайплайне

```
[Текст] → [Токенизация] → [Индексы] → [Матрица эмбеддингов E ∈ R^{|V|×d}]
                                          │  (берём строку E по индексу)
                                          └→ [вектор x ∈ R^d] → [Модель/Лосс]
```

1.2. Распределительная гипотеза: фундаментальная логика

Смысл слова проявляется через его окружение. Если два слова появляются в похожих контекстах, их векторы должны быть похожи. Формализация: считаем совместные встречаемости слов с соседями в окне и превращаем эти статистики в векторы.

1.3. От «счётных» представлений к плотным эмбеддингам

Счётные (разреженные) представления: Bag-of-Words (TF), tf-idf и BM25. Они быстрые и объяснимые, но игнорируют порядок и синонимию, векторы огромны и разрежены. Плотные эмбеддинги учатся так, чтобы предсказывать или объяснять наблюдаемые контексты.

Формулы, чтобы видеть контраст:

$$idf(t) = \log(N / (df(t)+1))$$

$$score\_BM25(d,q) = \Sigma_{t \in q} IDF(t) * ((k1+1)*tf_{t,d}) / (tf_{t,d} + k1*(1 - b + b*|d|/avg|d|))$$

По-человечески: BoW/tf-idf/BM25 — это «веса» слов в документе. BM25 добавляет нормализацию по длине и насыщение TF, чтобы длинные тексты не выигрывали «за счёт болтовни».

1.4. Два главных пути построения плотных эмбеддингов

A) Count-based → факторизация (LSA/PMI/GloVe)

1) Строим матрицу кооккурентностей X (кто с кем рядом). 2) Преобразуем её в информативную форму, напр. PMI(w,c)=log(P(w,c)/(P(w)P(c))). 3) Делаем факторизацию (SVD и родственные). В GloVe мы напрямую аппроксимируем log X через скалярные произведения:

$$\min \Sigma_{i,j} f(X_{ij}) \cdot (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

Перевод: хотим, чтобы скалярное произведение векторов w_i и $\tilde{w}_j$ объясняло логарифм частоты совместных появлений. $f(\cdot)$ задаёт веса для стабильности.

B) Predictive → предсказание контекста (Word2Vec/fastText)

Word2Vec: CBOW — по словам контекста предсказываем центральное слово; Skip-gram — по центральному слову предсказываем соседей. Чтобы избежать дорогого softmax, применяют Negative Sampling:

$$L \approx \log \sigma(v_c^T u_w) + \Sigma_{n\sim P_n} \log \sigma(- v_n^T u_w)$$

Перевод: хотим, чтобы «правильные» пары (слово, контекст) давали высокий отклик ($\sigma\approx1$), а случайные — низкий ($\sigma\approx0$). Это стягивает близкие по смыслу слова.

fastText: как Word2Vec, но вектор слова — сумма символьных n-грамм. Это помогает с редкими/новыми словами и учитывает морфологию (важно для русского).

1.5. Важная связь: предсказательные ≈ факторизационные

Интуитивно: Skip-gram с Negative Sampling учится так, что его скалярные произведения приближённо соответствуют сдвинутому PMI. Обе ветки (count/factorization и predictive) используют одну и ту же распределительную информацию, но разными алгоритмами.

1.6. Геометрия и меры близости

$$\cos(x,y) = (x\cdot y) / (||x|| \cdot ||y||)$$

Перевод: измеряем «угол» между векторами. Если направлены примерно одинаково — косинус близок к 1 → объекты похожи. Часто также используют скалярное произведение (при нормировке) или евклидово расстояние. L2-нормировка делает сравнение устойчивее.

1.7. Статические vs контекстные эмбеддинги

Статические (Word2Vec/GloVe/fastText): один вектор на слово для всех контекстов — полисемия неразличима. Контекстные (BERT/GPT): вектор зависит от предложения; одно и то же слово получает разные представления в разных контекстах. В трансформерах начальный эмбеддинг — сумма токенного+позиционного(+сегментного), затем самовнимание формирует контекстные вектора.

1.8. Эмбеддинги выше уровня слова

- Предложение/документ: усреднение, взвешивание (tf-idf), [CLS], attention-пулинг.
- Doc2Vec: обучаемый «вектор документа» как часть задачи предсказания слов.
- Dense Retrieval: би-энкодеры для запроса и документа в одном пространстве.
- Графовые: node2vec/DeepWalk (эмбеддинги вершин по «прогулкам»).

• Мультимодальные: CLIP (изображение/текст) — общее векторное пространство.

1.9. Как понять, что эмбеддинг «хороший»

Внутренние проверки (intrinsic): сходство, аналогии, кластеризация.

Внешние (extrinsic): качество на целевой задаче (классификация, поиск и т.п.).

Признаки «здорового» пространства: кластеры синонимов; развод полисемии у контекстных; разумные линейные отношения.

Визуализация (t-SNE/UMAP) полезна как иллюстрация, но не как «доказательство».

1.10. Практические вопросы

Размерность d: 100–300 (классика статических), 512–1024 (современные энкодеры). Большее d — выше ёмкость и риск шума.

Нормализация: L2-нормировать перед сравнением; полезно в ANN-поиске (FAISS/HNSW).

Редкие/новые слова: subword (fastText, BPE) решают OOV и помогают морфологии.

Домен/перенос: эмбеддинг чувствителен к корпусу; делайте адаптацию на профильных данных.

Смещения/этика: эмбеддинги наследуют предвзятости корпуса; оценивать и при необходимости снижать их влияние.

1.11. Небольшой «разбор формул» человеческим языком

1) Косинусная близость: cos(x,y) = (x·y)/(||x||·||y||)
Перевод: «угол» между смыслами — чем меньше, тем ближе.

2) IDF: idf(t)=log(N/(df(t)+1))
Перевод: слово, встречающееся почти везде, малоинформативно → понижаем его вес.

3) BM25: score(d,q)=Σ IDF(t) * ((k1+1)tf)/(tf + k1(1 − b + b|d|/avg|d|))
Перевод: вклад термина растёт с tf, но с насыщением; учитываем длину документа.

4) PMI: PMI(w,c)=log(P(w,c)/(P(w)P(c)))
Перевод: «насколько чаще вместе, чем ожидалось по независимости».

5) Negative Sampling:
L ≈ log σ(v_c^T u_w) + Σ_{n} log σ(− v_n^T u_w)
Перевод: «правильные пары» близко, «случайные» далеко.

1.12. Частые ошибки и как их избежать

• Путать «эмбеддинг слова» и «контекстное представление токена» в трансформере.

• Слепо верить t-SNE/UMAP (красивая картинка ≠ доказательство).

• Игнорировать токенизацию и морфологию (для русского критично subword).

• Не учитывать домен и предвзятости корпуса.

1.13. Чек-лист для устного ответа

1) Определение эмбеддинга. 2) Распределительная гипотеза. 3) Два пути (count/factorization и predictive). 4) Формулы + перевод (косинус, IDF, BM25, PMI, NS). 5) Статические vs контекстные. 6) Практика (d, нормализация, subword, домен, bias). 7) Оценка (intrinsic/extrinsic).

Конец лекции. Можно добавить практику: посчитать PMI/tf-idf на маленьком корпусе и найти соседей по косинусу.

Лекция 2. Счётные способы построения эмбеддинга: Bag-of-Words, tf-idf, BM25 (расширенная версия)

Полноценная учебная лекция: от мотивации и математики до разборов с числами, инженерных деталей индексации, тонкостей препроцессинга для русского и практических сценариев (классификация, поиск).

2.1. Мотивация и место «счётных эмбеддингов» в NLP

Счётные представления — это отображение текста в высокоразмерные разреженные векторы, где ось ≡ термин (слово, n-грамма). В промышленности они служат: (1) быстрыми признаками для линейных моделей; (2) «кандидатным» этапом в поиске перед нейронным реранкером; (3) диагностическим инструментом — веса легко интерпретировать и проверять.
Ключевая идея: мы кодируем «важность» терма для документа и/или запроса. Важность складывается из локальной частоты и глобальной дискриминативности.

2.2. Пайплайн препроцессинга для русского (и не только)

- Нормализация регистра: casefolding (лучше, чем просто lower) — уменьшает размер словаря.
- Очистка: удаление HTML, URL, эмодзи/пунктуации — по задаче; числа иногда оставляют (цены, даты).
- Токенизация: для русского учитываем дефисы/частицы («как-то», «по-русски»), апострофы.
- Морфология: лемматизация (pymorphy/mystem) vs стемминг (SnowballRu). Леммы дают компактный словарь и объединяют формы.
- Стоп-слова: для классификации часто удаляют; для поиска — аккуратно (могут важны в фразах).
- n-граммы: биграммы/триграммы ловят устойчивые словосочетания («машинное обучение», «Красная площадь»).
- Hashing trick: заменяет явный словарь хеш-функцией → меньше памяти, но возможны коллизии.
- Ограничение словаря: min_df/max_df, top-k по частоте/информативности — для стабильности и памяти.

2.3. Словарь и пространство признаков

Пусть словарь V имеет |V| термов. Документ d представляется как вектор $x_d \in R^{|V|}$. Разреженность — доли процента.
Инвертированный индекс (для поиска): для каждого терма храним список (doc_id, tf), плюс длину документа |d| и df. Это позволяет быстро считать BM25/tf-idf по запросу.

2.4. Bag-of-Words (BoW): определение, схемы TF и нормировки

BoW игнорирует порядок слов. Важен выбор схемы TF и нормировок.
SMART-обозначение (Salton): ltc, lnc, ntc и т.д.

```
Частые TF-схемы:
  raw: tf = #(t в d)
  binary: tf ∈ {0,1}
  log: tf' = 1 + log(tf)        # «сжатие» больших частот
  double-norm K: tf' = K + (1−K) * tf / max_tf(d), K∈(0,1]

Нормировки по документу:
  L2: x := x / ||x||_2          # косинусное сравнение = скалярное произведение
  L1 / max-norm — реже, по задаче
```

Интуиция: log-тф уменьшает дисбаланс между «очень частыми» и «просто частыми» термами в документе; L2-нормировка делает вектора сопоставимыми по длине.

Ограничения BoW: нет порядка, нет неоднозначности значений (полисемии) и синонимии.

2.5. tf-idf: локальная частота × глобальная редкость

```
idf(t) = log( N / (df(t)+1) )              # «классика»
idf_smooth(t) = log( (N+1)/(df(t)+1) ) + 1   # вариант в sklearn

tf-idf(t,d) = tf'(t,d) · idf(t)            # tf' часто берут лог-нормированным
Нормировка документа: L2 (или cosine normalization)
```

Почему работает: слова, встречающиеся почти в каждом документе, малоинформативны (низкий IDF). Редкие — сильнее характеризуют содержание.

Связь с косинусом: если все вектора L2-нормированы, скалярное произведение равно косинусу угла между ними, что удобно для поиска схожих документов.

Тонкости: база логарифма не важна (монотонная трансформация). Важны сглаживания на малых корпусах (idf_smooth).

2.6. BM25: от вероятностной модели релевантности к практической формуле

BM25 (Okapi) происходит из вероятностной модели релевантности (Robertson–Sparck Jones). Важнейшие изменения: насыщение TF и нормализация по длине.

```
score(d,q) = Σ_{t∈q} IDF_BM25(t) *
          ((k1+1) * tf_{t,d}) / (tf_{t,d} + k1 * (1 − b + b * |d|/avg|d|))

IDF_BM25(t) = log( (N − df_t + 0.5) / (df_t + 0.5) )
BM25^+: IDF^+(t) = log( 1 + (N − df_t + 0.5) / (df_t + 0.5) )   # всегда ≥0

Параметры: k1≈1.2–2.0  регулирует степень насыщения TF
        b≈0.75      сила нормализации по длине (0 — не учитывать, 1 — полностью)
```

Интуиция: первые вхождения терма сильно повышают релевантность, но после нескольких появлений прирост замедляется. Длинные документы нормализуются, чтобы не выигрывать только из-за длины.

Замечание: в исторической формуле был параметр k3 — вес частоты терма в запросе. В современных приложениях часто используют бинарные веса запроса и k3 не применяют.

2.7. Варианты и родственные идеи

- BM25+: добавляет константу δ к TF-части, чтобы длинные документы с tf=1 не оказывались слишком низко.

• BM25F: поля/атрибуты (title, body): длины и веса считаются по полям, затем агрегируются.
• Pivoted Length Normalization: альтернативная нормализация длины (Singhal, 1996).
• Divergence from Randomness (DFR): семейство альтернативных весов (PL2 и др.).
• Языковые модели для IR (QLM, Dirichlet/Jelinek–Mercer): не BoW-веса, но часто конкурирующий класс базовых ранжировщиков.

2.8. Подробный пример (tf-idf и BM25 «на числах»)

Корпус (N=3) и длины:

D1: "молоко корова трава" |D1|=3
D2: "трава зелёная корова корова трава зелёная" |D2|=6
D3: "молоко белое молоко" |D3|=3
Средняя длина: avg|d| = (3+6+3)/3 = 4
df: df(молоко)=2, df(трава)=2, df(корова)=2, df(зелёная)=1, df(белое)=1

tf-idf (с idf_smooth, log-тф и L2-нормой)

idf_smooth(df=2)=log((N+1)/(2+1))+1=log(4/3)+1≈1.2877
idf_smooth(df=1)=log((N+1)/(1+1))+1=log(4/2)+1=log2+1≈1.6931

Лог-тф: tf' = 1 + log(tf)
D3: tf'(молоко)=1+log2≈1.6931; tf'(белое)=1
веса до нормировки: молоко: 1.6931*1.2877≈2.18; белое: 1*1.6931≈1.6931
далее L2-нормируем вектор D3.
Аналогично считаются веса для D1/D2.
Косинусная близость запроса к документу — скалярное произведение нормированных векторо

BM25 (k1=1.5, b=0.75, IDF^+)

Запрос q="корова зелёная"

IDF^+(корова) = log(1 + (3−2+0.5)/(2+0.5)) = log(1.6) = 0.4700
IDF^+(зелёная)= log(1 + (3−1+0.5)/(1+0.5)) = log(2.6667) = 0.9808

Нормализация по длине: k1*(1 − b + b*|d|/avg|d|)
D1: 1.5*(0.25 + 0.5625)=1.21875
D2: 1.5*(0.25 + 1.125) =2.0625
D3: 1.21875

TF: D2: tf(корова)=2, tf(зелёная)=2
Вклад(корова, D2) = 0.4700 * ((2.5*2)/(2 + 2.0625)) = 0.4700 * (5/4.0625) ≈ 0.578
Вклад(зелёная,D2) = 0.9808 * (5/4.0625) ≈ 1.206
Итого score_BM25(D2,q) ≈ 1.784; для D1/D3 вклад нулевой по «зелёная», поэтому итог ниже.

Сравнение: tf-idf склонен «равнять» по L2-норме; BM25 явно учитывает длину и насыщение TF, что часто точнее в веб-поиске.

2.9. Инженерия: как это бегает быстро

Индекс: инвертированный список для каждого терма с (doc_id, tf) + doclen. Списки хранятся сжатым образом (varint, PForDelta).
Онлайн-скоring по запросу: читаем списки для термов запроса, считаем вклад по формулам tf-idf/BM25, агрегируем по doc_id.

Top-k поиск: алгоритмы WAND/BMW/Block-Max-Index позволяют перепрыгивать по спискам и быстро находить top-k документов.

2.10. Классификация текстов с BoW/tf-idf

Линейные модели (LogReg, Linear SVM) отлично работают с разреженными векторами. Важные нюансы:

- Баланс классов: используйте class_weight='balanced' или выборки; метрика — macro-F1 при дисбалансе.
- Регуляризация: C (LogReg/SVM). Слишком маленький — недообучение, слишком большой — переобучение на шуме.
- Признаки: словоформы vs леммы; добавляйте биграммы для коротких текстов; ограничивайте словарь.
- Кросс-валидация: стратифицированные сплиты, контроль утечек (не строить словарь по train+test).

2.11. Типичные ошибки

- «Переочистили» текст и потеряли важные токены (нормативные сокращения, числа).
- Забыли нормировку/сглаживание IDF на малых корпусах — веса скачут.
- Не учли длину документа: tf-idf без нормировки любит длинные тексты.
- Неверная токенизация для русского — раздутый словарь и шум.
- Не добавили n-граммы, хотя задача «коротких заголовков» их требует.

2.12. Расширения и связь с обучаемыми эмбеддингами

Гибридные системы: BM25 → top-K кандидатов → нейронный bi-encoder/кросс-энкодер для rerank. Это даёт лучшее из двух миров: скорость и качество.
tf-idf-признаки можно скрестить с нейронными (stacking/feature union). Для мультиязычных кейсов используйте лемматизацию и char n-граммы.

2.13. Краткое резюме (что сказать на экзамене)

- BoW — сырые/лог-нормированные частоты + нормировка. Простой, быстрый, интерпретируемый.
- tf-idf — TF×IDF: редкое важнее. После L2-нормы косинус=скалярное произведение.
- BM25 — «умный tf-idf»: насыщение TF и нормализация по длине через b; IDF с 0.5-смещениями; k1≈1.2–2.0, b≈0.75.
- В проде: инвертированный индекс, ускорители top-k (WAND/BMW). Часто — гибрид BM25+нейрореранкер.

Конец расширенной лекции 2. При желании добавлю приложение с кодом (sklearn/pyserini/rank_bm25) и чек-листом настройки индекса.

Лекция 3. Счётные vs обучаемые эмбеддинги: различия, преимущества и недостатки

Полная учебная версия: системное сравнение двух семейств представлений текста; математика, инженерия, кейсы применения, гибридные пайплайны и чек-листы выбора.

3.1. Что именно сравниваем

Счётные (разреженные) представления — BoW, tf-idf, BM25: не требуют обучения параметров; размерность равна размеру словаря |V|.
Обучаемые (плотные) эмбеддинги — статические (Word2Vec, GloVe, fastText) и контекстные (BERT/GPT): требуют обучения/предобучения; размерность d ≪ |V|; геометрия отражает семантику.

```
[Текст] → [токенизация] →
  ├─ (count)   → разреженный x ∈ R^{|V|}  (BoW/tf-idf/BM25)
  └─ (learned) → плотный     x ∈ R^{d}    (W2V/GloVe/fastText/BERT/GPT)
```

3.2. Оси сравнения (с объяснениями)

1) Размерность и геометрия

- Count: разреженно и высокоразмерно (|V| тысячи–миллионы). Косинус/скалярное произведение применимы, но нужны индексы/сжатие.
- Learned: компактное d (100–1024). Геометрия смысла: близость ≈ семантика; быстрое сравнение и кластеризация.

Вывод: плотные вектора удобнее для ANN и композиционных операций (усреднение, PCA/UMAP).

2) Порядок и контекст

- Count: «мешок слов» — порядок/синтаксис отсутствует (частично помогают n-граммы).
- Learned статические: учитывают частотный контекст в среднем по корпусу, но у слова один вектор.
- Learned контекстные: самовнимание даёт вектор токена, зависящий от всего предложения/окна.

Вывод: задачи, где важен порядок и неоднозначность значений, требуют контекстных представлений.

3) Семантика: синонимы и полисемия

- Count: не «склеивает» синонимы; «банк» и «берег» пересекаются только если фразы совпадают.
- Learned: синонимы ближе; контекстные разводят разные значения омонимов.

Вывод: при лексическом разрыве (query–doc mismatch) dense-подходы существенно лучше.

4) Морфология и OOV

• Count: словарь раздувается из-за словоформ (русский!), OOV — пробелы. Спасают лемматизация и n-граммы.

• Learned: fastText/subword устойчивы к редким и новым словам; BPE/WordPiece почти устраняют OOV.

5) Данные и обучение

• Count: обучения нет; быстро стартовать; легко обновлять.

• Learned: нужно обучение или предобученная модель; статические дёшево обучить на своём корпусе; контекстные — берём предобученные и дообучаем (PEFT).

6) Интерпретируемость и отладка

• Count: высокая — видно вклад каждого терма (вес tf-idf, вклад BM25).

• Learned: ниже; нужны соседи/атрибуция/attention-карты; понятность снижается.

7) Память, индексация, скорость

• Count: инвертированные индексы, быстрый first-stage (BM25).

• Learned: ANN-индексы (HNSW/IVF-PQ/ScaNN); дороже поддержка, но сравнения в d-мерном пространстве очень быстры.

8) Устойчивость к шуму/опечаткам/SEO

• Count: чувствительны к набивке термов и опечаткам; но детерминированы и предсказуемы.

• Learned: subword сглаживает опечатки; контекст помогает «понять», но возможны переобобщения.

9) Доменная адаптация

• Count: легко пересчитать статистики на новом корпусе.

• Learned: перенос через предобучение+дообучение; без адаптации возможны сдвиги.

10) Типичные задачи и качество

• Классификация: tf-idf + логрег/SVM — сильный baseline при малых данных; трансформеры — лучше при достаточном объёме.

• Поиск: BM25 силён на точных совпадениях; dense — на парафразах/семантике; лучшее — гибрид.

• Семантический поиск/кластеризация: плотные эмбеддинги выигрывают.

• NER/QA/NLI: контекстные энкодеры доминируют.

3.3. Сводная таблица сравнения

Критерий	Счётные (BoW/tf-idf/BM25)	Обучаемые статич. (W2V/GloVe/fastText)	Контекстные (BERT/GPT-класс)
Пространство	Разреж., |V|	Плотное, d≪|V|	Плотное, контекст-завис.
Контекст/порядок	Нет (кроме n-грамм)	Средний по корпусу	Полный (self-attention)
Синонимы/полисемия	Слабо	Синонимы ближе, полисемия нет	Синонимы/полисемия — лучше все
OOV/морфология	Проблема; леммы/n-gram	fastText/subword — ок	BPE/WordPiece — ок
Обучение	Не нужно	Нужно (дёшево)	Нужно (дорого; берём предобуч.)
Интерпретируемость	Высокая	Средняя	Ниже

Индексация/скорость	Инверт. индекс; быстро	ANN; быстрое сравнение	ANN + тяжёлый энкодер
Лучшие кейсы	Бейзлайны, точные совпадения, короткие тексты	Семантика без контекста	SOTA там, где важен контекст

3.4. Ключевые формулы (с «переводом»)

tf-idf:
  idf(t) = log(N / (df(t)+1))
  tf-idf(t,d) = tf'(t,d) · idf(t)

BM25:
  score(d,q) = Σ_{t∈q} IDF(t) * ((k1+1) tf_{t,d}) / (tf_{t,d} + k1(1 − b + b|d|/avg|d|))

Word2Vec (Negative Sampling, идея):
  L ≈ log σ(v_c^T u_w) + Σ_{n~P_n} log σ(− v_n^T u_w)

GloVe:
  min Σ_{i,j} f(X_ij) · (w_i^T w̃_j + b_i + b̃_j − log X_ij)^2

Перевод: tf-idf/BM25 взвешивают лексическое совпадение; Word2Vec/GloVe стягивают часто совместные слова; контекстные энкодеры строят представления, зависящие от окружения предложения.

3.5. Примеры задач и типичные пайплайны

- Классификация отзывов (1–10 тыс. доков): tf-idf + логрег/SVM; дальше — энкодер (BERT) с лёгким дообучением/LoRA.
- FAQ/поиск: BM25 (кандидаты 200–1000) → bi-encoder rerank; топ-50 — cross-encoder.
- Семантический поиск/кластеризация: bi-encoder + ANN (HNSW/FAISS); для точности — cross-encoder на top-k.
- Мультиязычные/морфологически богатые корпуса: лемматизация+n-gram (count) и subword-токенизация (learned).

3.6. Ловушки и анти-паттерны

- tf-idf без нормировки по документу: переоценивает длинные тексты.
- Отсутствие n-грамм в коротких текстах/заголовках.
- Плохая токенизация/лемматизация для русского → раздутый словарь/шум.
- Оценка learned-эмбеддингов только по t-SNE/UMAP вместо метрик задачи.
- Переобучение трансформера на малом датасете — предпочитать PEFT/заморозку и стратифицированную валидацию.

3.7. Гибридный пайплайн (IR/поиск): схема

[Запрос] ⟶ [BM25 (инверт. индекс)] ⟶ top-K кандидатов
  └⟶ [Dense bi-encoder + ANN] (опц.) → merge (RR/Fusion)
  ⟶ [Cross-encoder rerank top-50] ⟶ [Итоговый список]

Примечание: часто применяют reciprocal rank fusion (RRF) для объединения кандидатов от BM25 и dense-ретривера.

3.8. Чек-лист выбора подхода

- Быстро/объяснимо/мало данных → tf-idf/BM25 + линейные модели.
- Нужна семантика/синонимы, умеренные ресурсы → fastText/статические эмбеддинги + простая модель.

• Максимум качества, важен контекст → BERT/контекстные; в поиске — гибрид с BM25.

• Прод: инвертированный индекс + ANN; каскады: кандидаты → нейрореранк → (опц.) rerank.

Конец лекции 3. Контрольные вопросы: 1) Почему BM25 и bi-encoder комплементарны? 2) Что такое лексический разрыв? 3) Как subword снижает OOV? 4) Когда интерпретируемость важнее потенциального прироста качества? 5) Какие риски у t-SNE-интерпретаций?

Лекция 4. Построение эмбеддинга с помощью глубинного обучения: Word2Vec, GloVe, fastText

Подробная учебная версия: интуиция и формальные цели, архитектуры и трики обучения, параметры и их влияние, разборы формул «человеческим языком», морфология и OOV, практические рецепты и типичные ошибки.

4.1. Зачем нам обучаемые эмбеддинги

Счётные представления (BoW/tf-idf/BM25) быстры и прозрачны, но игнорируют порядок и синонимию. Обучаемые плотные эмбеддинги используют распределительную гипотезу («скажи мне контексты — скажу значение») и создают компактное пространство, где геометрия отражает семантику.

```
[Корпус] → [окна контекста] → (оптимизируем цель)
  ├─ Word2Vec: предсказываем слово по контексту (CBOW) или контекст по слову (Skip-gram)
  ├─ GloVe: аппроксимируем log-кооккурентности скалярными произведениями
  └─ fastText: как Word2Vec, но слово = сумма символьных n-грамм
```

4.2. Word2Vec: CBOW и Skip-gram

Идея: обучаем две таблицы векторов (входные u_w и выходные v_c). Окно контекста ширины W, центр — целевое слово.

- CBOW: по контексту C предсказываем центральное слово w.
- Skip-gram: по центральному слову w предсказываем каждое слово контекста $c \in C$.

Полный softmax (теория):

$$P(c|w) = \exp(v_c^T u_w) / \Sigma_{j\in V} \exp(v_j^T u_w)$$

$$\text{Лосс (SG)}: -\Sigma_{(w,c)\in D} \log P(c|w) \qquad \# D \text{ — наблюдаемые } (w,c) \text{ пары}$$

Перевод: хотим, чтобы «правильный» контекст имел высокую вероятность. Но считать softmax по всему словарю дорого.

Negative Sampling (NS): быстрая аппроксимация

Для каждой истинной пары (w,c) берём k «негативов» $n \sim P_n$ (обычно unigram$^{3/4}$).
Оптимизируем бинарную классификацию «настоящая пара» vs «случайная»:

$$L_{NS}(w,c) = \log \sigma(v_c^T u_w) + \Sigma_{i=1..k} \log \sigma(-v_{n_i}^T u_w)$$

Человечески: подтянуть ближе эмбеддинги «правильных» пар и раздвинуть «случайные». Выбор распределения негативов $P_n \propto f(w)^{3/4}$ делает обучение стабильнее (частые слова не доминируют).

Важные трюки Word2Vec

- Субсэмплинг частых слов: $p_{discard}(w) = 1 - \sqrt{t / f(w)}$, $t \approx 1e{-}5$
  → уменьшает шум от стоп-слов, ускоряет и улучшает качество.
- Динамическое окно: фактический размер контекста равномерно выбирается из $[1..W]$.
- Phrase detection (word2phrase): «машинное_обучение», «Нью-Йорк» → лучше контексты.
- Выбор CBOW vs Skip-gram: CBOW быстрее для больших корпусов; Skip-gram лучше на редких с
- Hierarchical softmax: альтернатива NS для малых корпусов/словаря.

Связь с PMI (важно для теории)

Результат (Levy & Goldberg, 2014): SGNS ≈ факторизация матрицы SPPMI.

$$\text{То есть } v_c^T u_w \approx PMI(w,c) - \log k, \quad \text{где } PMI = \log\left(P(w,c) / (P(w)P(c)) \right)$$

Перевод: несмотря на «предсказательность», SGNS по сути учится на той же распределительной статистике, что и метод факторизации.

Гиперпараметры (типичные ориентиры)

dim d: 100–300 классика; 300–600 для богатых корпусов
window W: 2–10 (симметричное)
negatives k: 5–15 (зависит от размера словаря)
min_count: 5–50 (фильтрация редких)
epochs: 5–15; lr: 0.025→0.0001 (linearly decayed)

Плюсы/Минусы Word2Vec

• + Быстро, просто, качественно на больших данных; переносимо между задачами.

• − Один вектор на слово (нет контекста); OOV-проблема без subword; чувствителен к качеству корпуса.

4.3. GloVe: Global Vectors

Идея: вместо предсказания оптимизируем аппроксимацию логарифма совместных встречаемостей слов.

Имеем матрицу кооккурентностей X, где X_ij = #(слово i рядом со словом j в окне).
Оптимизируем по w_i, w̃_j, b_i, b̃_j:
J = Σ_{i,j} f(X_ij) · (w_i^T w̃_j + b_i + b̃_j − log X_ij)^2

Весовая функция:
f(x) = (x/x_max)^α, если x < x_max; иначе 1   (α≈3/4, x_max≈100)

Человечески: хотим, чтобы скалярное произведение векторов «объясняло» наблюдаемую частоту совместных появлений. f(·) уменьшает влияние очень редких и очень частых пар, делая обучение устойчивым.

Практические моменты

• Окно контекста обычно взвешивают по расстоянию (1/|Δ|).
• Строим только ненулевые (i,j) — экономия памяти/времени.
• Итоговый вектор слова = w_i + w̃_i (или один из них).
• Гиперпараметры: d=100–300; α≈0.75; x_max≈100; epochs 20–50.

Сравнение с Word2Vec: GloVe использует глобальную статистику (вся матрица), тогда как SGNS — локальные пары. На практике качество близкое; GloVe устойчив на больших статичных корпусах, но требует этапа построения X.

4.4. fastText: моделируем морфологию через subword

Идея: вектор слова — сумма векторов его символьных n-грамм (с границами слова). Это решает OOV и помогает морфологии.

Для слова w добавляем «якоря» < и > и строим все n-граммы, n∈[min_n, max_n], напр. [3,6].
Вектор слова: z_w = Σ_{g ∈ G(w)} z_g
Оптимизация: та же, что в Word2Vec (CBOW/Skip-gram) с Negative Sampling, но таблица эмбедди

Практические параметры:
min_n=3, max_n=6; bucket=1e6..10e6 (хеш-таблица n-грамм);
d=100–300; lr=0.05; epoch=5–15; ws (окно)=5–10.

Человечески: если встречается новая форма «читателей», модель всё равно строит её вектор по знакомым кусочкам «чит», «тате», «елей». Это особенно полезно для русского, немецкого, турецкого и других морфологически богатых языков.

Плюсы/Минусы fastText

• + Отличная устойчивость к редким/новым словам; лучше кодирует морфологию; прост в использовании (официальный CLI).

• − Больше память из-за n-грамм; возможны коллизии в хеш-таблице; не решает полисемию (как и Word2Vec/GloVe).

4.5. От слов к предложениям и документам

• Усреднение (mean pooling): просто и часто достаточно для классификации/кластеризации.

• Взвешенное усреднение: веса ∝ tf-idf; снижает влияние общеязыковых слов.

• SIF (Smooth Inverse Frequency): a/(a+p(w)) как вес + удаление главной компоненты (Arora et al.) — часто улучшает качество.

• Doc2Vec: отдельный обучаемый «тег» документа, который участвует в предсказании слов — даёт вектор документа.

4.6. Практические рецепты обучения

```
Корпус → очистка/токенизация → (опц.) лемматизация/фразовые бигираммы → фильтр редких т

Word2Vec (gensim пример):
  Word2Vec(sentences, vector_size=300, window=5, min_count=5, sg=1, negative=10,
           sample=1e-5, workers=cores)

GloVe (glove-python/torchglove):
  Строим X (взвеш. окно), затем оптимизируем по J; эмбеддинг = w + w̃

fastText (официальный CLI):
  ./fasttext skipgram -input corpus.txt -output model -dim 300 -minn 3 -maxn 6 -ws 5 -neg 10 -epoch 1
```

Советы: включайте phrase detection для устойчивых словосочетаний; следите за балансом классов/домена; для русского почти всегда используйте subword (fastText или BPE в трансформерах).

4.7. Как оценивать качество эмбеддингов

Внутренние (intrinsic): сходство слов (WordSim), аналогии (king−man+woman≈queen), кластеризация тематических групп.
Внешние (extrinsic): качество на целевой задаче (классификация, поиск, NER, QA).
В проде ориентируйтесь на extrinsic.
Визуализация (t-SNE/UMAP) — только как иллюстрация: красиво, но искажает расстояния.

4.8. Типичные ошибки и как их избежать

• Переобучение на узком корпусе → переносимость падает; храните «чистую» предобученную версию.

• Отсутствие subword в морфологических языках → много OOV и слабые редкие слова.

• Игнорирование phrase detection — теряется смысл устойчивых выражений.

• Неправильные гиперпараметры (слишком маленькое окно или d) — бедная семантика.

• Оценка только по аналогиям — не гарантирует качество в реальной задаче.

4.9. Чек-лист для устного ответа

1) Word2Vec: CBOW/Skip-gram, Negative Sampling, субсэмплинг частых слов, связь с PMI.

2) GloVe: регрессия на log X, весовая функция f(x), глобальная статистика.

3) fastText: слово как сумма n-грамм, параметры min_n/max_n, выгоды для морфологии/OOV.

4) Практика: подготовка корпуса, типичные гиперпараметры, усреднение для предложений/доков.

5) Оценка: intrinsic vs extrinsic; т-SNE — только иллюстрация.

4.10. Формулы — и что они значат «по-русски»

(1) Softmax Skip-gram:
$P(c|w) = \exp(v_c^T u_w) / \Sigma_j \exp(v_j^T u_w)$
→ максимизируем вероятность «правильных» контекстов

(2) Negative Sampling:
$L_{NS} = \log \sigma(v_c^T u_w) + \Sigma_i \log \sigma(- v_{n_i}^T u_w)$
→ хотим: «правильные» пары → 1, «случайные» → 0

(3) Субсэмплинг частых слов:
$p_{discard}(w) = 1 - \sqrt{t / f(w)}$, $t \approx 1e{-}5$
→ уменьшаем шум от «и», «в», «на»

(4) GloVe:
$J = \Sigma f(X_{ij}) \cdot (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$
→ скалярные произведения объясняют log-кооккурентности

(5) fastText (слово как сумма n-грамм):
$z_w = \Sigma_{g \in G(w)} z_g$
→ новая форма строится из знакомых кусочков

Конец лекции 4. При необходимости добавлю приложение: таблицу гиперпараметров и примеры запуска для gensim/fastText.

Экзамен: Классификация текстов — что отвечать

Каркас ответа по формуле преподавателя: «Достали эмбеддинг → отдали его в классификатор / или обучили end-to-end». Далее: единый конвейер, 5 архитектур и краткий экзаменный спич.

1) Единый каркас

[Текст] → [Токенизация/препроцессинг] → [Эмбеддинг/Энкодер]
→ [Пулинг: mean / max / attention / [CLS]]
→ [Классификатор: Linear / MLP]
→ [Потеря: CE (multiclass) / BCE-with-logits (multilabel)]

Multiclass — softmax+CE; Multilabel — сигмоиды+BCE-with-logits. Дисбаланс — class_weights/focal.

2) Пять канонических архитектур

2.1. Счётные + линейный: tf-idf → LogisticReg/SVM (быстро, объяснимо).
Текст → токены/леммы → tf-idf ∈ R^{|V|} → Linear → Softmax/Sigmoid

2.2. Статические эмбеддинги: Word2Vec/GloVe/fastText → mean/SIF → Linear/MLP.
Токены → lookup(E) → mean → z → Linear/MLP

2.3. CNN / BiLSTM+attention (end-to-end).
CNN: Emb → Conv+ReLU → Global MaxPool → FC; RNN: Emb → BiLSTM → Attn-pool → FC

2.4. BERT-энкодер: h_[CLS]/mean → Linear; обучаем end-to-end.
[CLS]+subwords → BERT → h_[CLS]/mean → Dropout+Linear → Softmax/Sigmoid

2.5. T5/mT5: генерация метки или encoder-head.
A: “classify: <текст>” → T5 → “спорт” | B: T5-encoder → mean → Linear

3) Инженерные детали (кратко)

Длины: Longformer/BigBird или чанки + агрегирующий пулинг. Дисбаланс: class_weights/focal; метрики: F1-macro, PR-AUC. Без утечек: словарь/IDF только по train.

4) Экзаменный спич (60–90 секунд)

Любая классификация — эмбеддинг → head. Фиксированный эмбеддинг: tf-idf или mean fastText/Word2Vec → логрег/SVM. End-to-end энкодер: CNN/biLSTM+attention или чаще BERT: берём h[CLS]/mean, сверху Linear и дообучаем. Для T5 можно генерировать метку или брать encoder-выход с head’ом. Multiclass — softmax+CE; Multilabel — сигмоиды+BCE; при дисбалансе — веса классов или focal, метрики — F1-macro/PR-AUC; длинные тексты — чанкуем и агрегируем.

Лекция. Механизм внимания (Attention): от RNN-seq2seq до Трансформеров

Зачем нужно внимание, формулы Bahdanau/Luong, scaled dot-product и multi-head в трансформерах, self- vs cross-attention, маски (padding/causal), позиционные кодировки, coverage/pointer, сложность и длинные последовательности, чек-лист для ответа.

1) Интуиция: почему вообще нужно внимание

Без внимания энкодер должен упаковать всё предложение в один вектор — «бутылочное горлышко». Внимание позволяет декодеру динамически смотреть на разные части входа и собирать контекст под текущий шаг генерации.

Идея: посчитать «важность» каждого h_i для шага t, нормировать softmax и взять взвешенную

2) Внимание в RNN-seq2seq: Bahdanau (additive) и Luong (dot/general)

Bahdanau (additive) attention:

e_{t,i} = v^T tanh(W_s s_{t−1} + W_h h_i)
α_{t,i} = exp(e_{t,i}) / Σ_j exp(e_{t,j})
c_t = Σ_i α_{t,i} h_i

Luong (dot/general) attention:

score_dot(s_t, h_i) = s_t^T h_i
score_general(s_t, h_i) = s_t^T W h_i
α_{t,i} = softmax_i(score(s_t, h_i))
c_t = Σ_i α_{t,i} h_i

Additive использует маленький MLP; dot/general опирается на скалярные произведения (быстрее). Оба возвращают α и контекст c_t.

3) Scaled Dot-Product Attention и Multi-Head (трансформеры)

Q=XW_Q, K=XW_K, V=XW_V
Attn(Q,K,V)=softmax(QK^T/√d_k) V
MultiHead=Concat(head_1..head_h)W_O, где head_j=Attn(QW_Q^j,KW_K^j,VW_V^j)

Токен формирует запрос и «спрашивает» других токенов; несколько голов смотрят под разными углами.

4) Self-attention, Cross-attention и маски

Self: Q,K,V из одной X; в декодере — causal маска. Cross: Q из декодера, K,V из энкодера. Маски

Энкодер: self-attention смешивает информацию по входу; декодер: masked self-attention + cross-attention к энкодеру.

5) Позиционные кодировки

PE(pos,2i)=sin(pos/10000^{2i/d}); PE(pos,2i+1)=cos(pos/10000^{2i/d}); альтернативы: learned/re

Они дают self-attention чувство порядка.

6) Coverage и Pointer-Generator

Coverage: накапливаем прошлые α и штрафуем повторы; Pointer: p_gen смешивает генерацию и

Помогает избегать повторов и копировать имена/числа/OOV.

7) Сложность и длинные последовательности

Стандартный self-attention O(n^2); удешевление: Longformer/BigBird/Linformer/Performer/Reforme

8) Обучение и декодирование

MLE с teacher forcing (+label smoothing). Декодирование: greedy/beam + length/coverage penaltie

9) Схемы (ASCII)

(A) Additive: s_{t−1}→tanh→v^T + h_i → softmax → α → c_t=Σαh
(B) Enc-block: X→Multi-Head SA→+res→FFN→+res
(C) Dec-step: S→Masked SA→+res→Cross(H_enc)→+res→FFN→+res→logits

10) Чек-лист для устного ответа

Мотивация; Bahdanau/Luong; scaled dot-product и multi-head; self vs cross; маски; позиции; covera

Лекция. Архитектура Encoder–Decoder и используемые модели (RNN / LSTM / GRU / Transformer)

Подробная учебная версия: зачем нужен Seq2Seq, базовая математика RNN/LSTM/GRU, внимание (Bahdanau/Luong), обучение и декодирование, улучшения (coverage/pointer), варианты энкодеров/декодеров и примеры применения.

1) Зачем нужен Encoder–Decoder (Seq2Seq)

Вход: последовательность $x_1 \ldots x_T$. Выход: другая последовательность $y_1 \ldots y_{T'}$ (часто другой длины). Encoder превращает вход в скрытое представление; Decoder авторегрессивно генерирует выход, опираясь на это представление (обычно через внимание).

[ВХОД x1..xT] --(Encoder)--> [скрытое представление входа] --(Context/Attention)--> (Decoder) -->

2) Базовая математика Seq2Seq-RNN

2.1. Простой RNN

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$$

По-русски: смешиваем текущий вход с прошлой «памятью» и получаем новое состояние.

2.2. LSTM — долгая память

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$
$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$
$$\hat{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t$$
$$h_t = o_t \odot \tanh(c_t)$$

Перевод: три «крана» решают, что забыть, что добавить и что показать из памяти c_t.

2.3. GRU — упрощённый LSTM

$$z_t = \sigma(W_z x_t + U_z h_{t-1})$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$
$$\hat{h}_t = \tanh(W_h x_t + U_h (r_t \odot h_{t-1}))$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$$

Перевод: две ручки смешивают старую и новую информацию; проще и быстрее LSTM.

3) Классическая архитектура: RNN-Encoder → RNN-Decoder

Encoder: принимает токены $x_1 \ldots x_T$ (эмбеддинги), часто двунаправленный (bi-RNN): $h_t=[\rightarrow h_t; \leftarrow h_t]$. Выдаёт последовательность состояний $h_1 \ldots h_T$.

Decoder: на шаге t берёт предыдущий токен y_{t-1} (на обучении — teacher forcing), обновляет состояние s_t и выдаёт распределение по словарю.

$$p(y_t \mid y_{<t}, x) = \text{softmax}(W_o \cdot g(s_t, c_t) + b)$$

4) Механизм внимания (Attention) — «куда смотреть на входе»

Bahdanau (additive) attention:

$e_{t,i} = v^T \tanh(W_s s_{t-1} + W_h h_i)$
$\alpha_{t,i} = \exp(e_{t,i}) / \Sigma_j \exp(e_{t,j})$
$c_t = \Sigma_i \alpha_{t,i} h_i$

Luong (dot/general) attention — быстрее: $score(s_t, h_i) = s_t^T h_i$ или $s_t^T W h_i$.
Перевод: для каждого шага декодера считаем «важность» каждого входного состояния и берём их взвешенную сумму как контекст.

5) Обучение и декодирование

Обучение (MLE): оптимизируем сумму лог-вероятностей правильных токенов с teacher forcing.

$L = - \Sigma_{t=1..T'} \log p(y_t^{gold} | y_{<t}^{gold}, x)$

Декодирование: greedy или beam search (top-B) + length/coverage penalties.
Exposure bias смягчают scheduled sampling, label smoothing.

6) Улучшения для RNN-Seq2Seq

- Input feeding: подавать $[s_t; c_t]$ на следующий шаг
- Coverage attention/loss: учитывать «сколько уже посмотрели» → меньше повторов
- Pointer-generator / Copy: уметь копировать слова из входа (имена, числа, OOV)
- Subword-токенизация (BPE/WordPiece) или char/byte уровни

7) Какие модели могут быть внутри Encoder/Decoder

RNN-семейство (LSTM/GRU), CNN-варианты, и Transformer как современный encoder–decoder (self-attn/cross-attn).

[Encoder]*L: Multi-Head Self-Attn → FFN (+residual+LayerNorm)
[Decoder]*L: Masked Self-Attn → Cross-Attn(на выход энкодера) → FFN

8) Применения

- Машинный перевод, суммаризация, диалог, ASR, перефразирование, грамкоррекция

9) Чек-лист для устного ответа (2–3 минуты)

1) Encoder–Decoder + внимание; 2) RNN (LSTM/GRU) и Transformer; 3) формулы внимания; 4) MLE + teacher forcing, beam search; 5) улучшалки; 6) примеры задач.

Готово: Encoder–Decoder. При необходимости добавлю приложение с псевдокодом beam search.

Лекция: Механизм внимания (Attention) — интуиция, формулы, варианты и примеры

5-минутный учебный конспект: от интуиции и математики до практических вариантов (Bahdanau/Luong, Scaled Dot-Product, Multi-Head, Self/Cross-Attention) и примеров применения.

1) Интуиция: зачем нужно внимание

В seq2seq-задачах модель должна на каждом шаге «посмотреть» на релевантные части входа. Внимание даёт дифференцируемый способ взвесить элементы источника и собрать из них контекст для текущего шага.

Идея: «Что из входа важно сейчас?»
↓
[Вход: h1 h2 ... hT] → веса α_{t,·} = softmax(score(текущий_запрос, ключи))
↓
Контекст c_t = Σ_i α_{t,i} · h_i → помогает предсказать следующий токен/класс

2) Базовая математика: Q, K, V и Scaled Dot-Product

Матрицы запросов (Q), ключей (K) и значений (V):
Q = X W_Q, K = X W_K, V = X W_V

Scaled Dot-Product Attention:
Attn(Q,K,V) = softmax(Q K^T / √d_k) V

По-русски: считаем сходства «запрос ↔ ключ», делим на √d_k для устойчивости, нормируем softmax’ом и берём взвешенную сумму значений.

3) Варианты внимания: additive, dot/general, scaled, multi-head

3.1. Additive (Bahdanau):

e_{t,i} = v^T tanh(W_q q_t + W_k k_i) → α_{t,i} = softmax(e_{t,·})
c_t = Σ_i α_{t,i} v_i

Перевод: «оценка важности» через маленькую сеть над (q_t, k_i).

3.2. Luong (dot/general):

score_dot(q,k) = q^T k
score_general(q,k) = q^T W k

Быстрее, чем additive; классика для RNN-декодеров.

3.3. Scaled Dot-Product (transformers):

softmax(Q K^T / √d_k) V

Масштабирование 1/√d_k предотвращает «пересатурацию» softmax при высоких размерностях.

3.4. Multi-Head Attention:

Для каждой головы h:
Q_h = X W_Q^h, K_h = X W_K^h, V_h = X W_V^h
H_h = softmax(Q_h K_h^T / √d_k) V_h
Concat(H_1,...,H_H) W_O

Несколько «ракурсов» внимания одновременно; разные головы фокусируются на разных зависимостях.

4) Self-Attention vs Cross-Attention

Self-Attn: Q=K=V из одной последовательности (пример: энкодер).
Cross-Attn: Q из декодера, K,V из энкодера (пример: MT).
Masked Self-Attn (декодер): запрещаем смотреть вперёд (каузальная маска).

В трансформере энкодер использует self-attention, декодер — masked self-attention + cross-attention.

5) Примеры применения

5.1. Машинный перевод (RNN-seq2seq с вниманием):

[Encoder RNN (bi-LSTM)] → h_1..h_T
На шаге t декодера: α_{t,·} по h_i → c_t = Σ α_{t,i} h_i → предсказываем y_t

Выигрыш: убираем узкое горлышко и получаем выравнивание вход↔выход.

5.2. Классификация с attention-пулингом:

e_i = tanh(W h_i), α_i = softmax(q^T e_i), z = Σ α_i h_i → Linear → классы

Модель учится выделять важные слова/фразы для метки, что повышает интерпретируемость.

5.3. Pointer-Generator (копирование):

P(w) = p_gen · P_vocab(w) + (1−p_gen) · Σ_{i: x_i = w} α_{t,i}

Можно сгенерировать слово или скопировать его с позиций источника по весам внимания — полезно для имён/чисел/OOV.

5.4. ViT/Трансформеры:

ViT: изображение → патчи → эмбеддинги → [Self-Attn блоки] → [CLS] для классификации

6) Вычислительная сторона и практика

- Стоимость: O(n^2·d). Для длинных последовательностей: Longformer/BigBird/Performer, FlashA
- Маски: padding-mask и causal-mask — обязательны для корректности.
- Dropout на матрицах внимания; контроль «остроты» через масштаб и температуру softmax.

7) Подводные камни

- Веса внимания ≠ доказанная причинная важность (не путать с объяснимостью).
- Без масок языковая модель «подсмотрит» будущее.
- n^2 сложность по длине — планируйте архитектуру и длины контекстов.

8) Чек-лист для устного ответа

1) Определение и формула: Attn(Q,K,V)=softmax(QK^T/√d_k)V.
2) Разновидности: Bahdanau/Luong, multi-head; self- vs cross-attention; маски.
3) Примеры: MT с выравниванием; attention-пулинг; pointer-generator; ViT.
4) Практика: сложность O(n^2), маски и регуляризация.

Готово: короткая, но полная лекция по вниманию. Могу расширить слайдами-схемами и упражнениями.

Лекция: Архитектура трансформера — за 5 минут

Интуиция, блоки Encoder/Decoder, формулы внимания (QKV), позиционные коды, FFN, резидуалы+LayerNorm, маски, варианты (BERT/GPT/T5), а также вычислительная сложность и практические заметки.

1) Зачем трансформер

RNNы сложны для параллелизации и плохо переносят очень длинные зависимости. Трансформер убирает рекурсию и строит представления через self-attention с параллельной обработкой всех позиций.

Высокоуровневая схема (encoder–decoder):
[Входные токены] → [Encoder блоки]*L → H_enc
↘
[Decoder блоки]*L: Masked Self-Attn + Cross-Attn(H_enc) → [токен за токеном]

2) Базовая математика: Q, K, V и Multi-Head Attention

Q = X W_Q, K = X W_K, V = X W_V # X — матрица скрытых состояний (последовательно
Scaled Dot-Product:
Attn(Q,K,V) = softmax(Q K^T / √d_k) V

Multi-Head (H голов):
H_h = softmax(Q W_Q^h (K W_K^h)^T / √d_k) (V W_V^h)
concat(H_1,...,H_H) W_O

Перевод: считаем сходства «запрос–ключ», нормируем, берём взвешенную сумму значений. Несколько голов — это несколько «ракурсов» внимания одновременно.

3) Блоки энкодера и декодера

3.1. Encoder-блок (слой):

x → [LayerNorm] → [Multi-Head Self-Attn] → + x (residual) → y
y → [LayerNorm] → [FFN (позиционно-обособленная MLP)] → + y (residual)
FFN обычно: FFN(h) = W2 · GELU(W1 · h) (или ReLU/SiLU)

Примечание: в современных реализациях часто используется Pre-LN (LayerNorm до блока), улучшая стабильность обучения.

3.2. Decoder-блок (слой):

x → [LayerNorm] → [Masked Multi-Head Self-Attn] → + x
→ [LayerNorm] → [Cross-Attn(на выход энкодера H_enc)] → + ...
→ [LayerNorm] → [FFN] → + ...

«Masked» означает каузальную маску (токен t не видит t+1..n). Cross-Attn смотрит ключи/значения из энкодера.

4) Позиционные представления (куда девать порядок?)

Синусоидальные (Vaswani et al.):
PE(pos,2i) = sin(pos / 10000^{2i/d_model})
PE(pos,2i+1) = cos(pos / 10000^{2i/d_model})

Альтернативы: обучаемые позиционные эмбеддинги; Rotary Positional Embeddings (RoPE); ALiBi.

По-русски: добавляем позиционный «сигнал» к эмбеддингам токенов, чтобы self-attention различал порядок.

5) Маски в внимании

- Padding-mask: игнорировать «пустые» токены (паддинги) в батче.
- Causal-mask (декодер/LM): запрещаем смотреть вперёд → строго авторегрессия.

6) Семейства на базе трансформера

- Encoder-only: BERT/roBERTa — классификация, токен-классификация, поиск (bi-encoder), MLM-
- Decoder-only: GPT — авторегрессия/генерация; causal self-attention.
- Encoder–Decoder: T5/BART — seq2seq (перевод, суммаризация), cross-attn к энкодеру.

Различие — не в базовой механике внимания, а в том, какие блоки включены и как ставятся маски/задачи обучения.

7) Обучение и выходные головы

- BERT: Masked LM (закрываем часть токенов), NSP (исторически), задачи классификации → [CLS
- GPT: next-token prediction (NTP), head — линейный слой на словарь.
- T5: text-to-text; head — лингвистический словарь декодера.

В классификации часто берут [CLS] или mean-пулинг скрытых состояний энкодера и подают в линейный классификатор.

8) Вычислительная сложность и оптимизации

Self-Attn: O(n^2 · d). Для длинных последовательностей: Longformer/BigBird/Performer, Linformer, Reformer, а также FlashAttention (эффективнее по памяти/скорости).
Практика: mixed precision (bf16/fp16), grad checkpointing, KV-кэш в декодере.

9) Подводные камни

- Неправильные маски → утечки будущего или «съедание» паддингов.
- Слишком высокая LR при fine-tuning → «забывание» предобученного.
- Контекст обрезан слишком коротко → потеря дальних зависимостей.
- Отсутствие токенизации под язык → OOV/дробление на символы без надобности.

10) Чек-лист для устного ответа (≈1–2 минуты)

1) Блоки: (a) Multi-Head Self-Attn, (b) FFN (позиционный MLP), (c) резидуалы + LayerNorm.
2) Формула: Attn(Q,K,V)=softmax(QK^T/√d_k)V; multi-head = параллельные «ракурсы».
3) Позиционные коды (sin/learned/RoPE), маски (padding/causal).
4) Семейства: BERT (encoder-only), GPT (decoder-only), T5 (enc-dec).
5) Сложность O(n^2); варианты для длинных последовательностей.

Готово: краткая, но полноценная лекция по трансформеру. По желанию добавлю страницу с различиями Post-LN vs Pre-LN и схемой параметров матриц W_Q/W_K/W_V.

Лекция: Стратегии генерации текста — Beam search, температура, top-k, top-p и др. (≈5 минут)

Зачем нужны стратегии, как они работают математически, плюсы/минусы и когда что выбирать. Плюс: современные варианты (typical, contrastive, mirostat, speculative).

1) Постановка: авторегрессивная генерация

Модель предсказывает следующий токен по уже сгенерированным: p(y_t | y_{<t}, x). Наша задача — выбрать стратегию, как извлекать последовательность из распределения модели.

Две семьи методов:
• Поиск (deterministic): greedy / beam — максимизируем лог-вероятность, меньше рандома.
• Семплинг (stochastic): температура, top-k, nucleus (top-p) — контролируем разнообразие.

2) Greedy и Beam search

Greedy: берём argmax на каждом шаге — быстро, но «жадно» и может застрять в повторениях.

Beam (ширина B): поддерживаем top-B гипотез по сумме лог-вероятностей.
Цель: y* = argmax_y (Σ_t log p(y_t | y_{<t}, x) / len(y)^α) # часто с length-penalty α∈[0,1]
Улучшения:
• coverage penalty (MT/суммаризация) • diversity beam (разнообразие между лучами)
• no-repeat-n-gram (запрет повторов n-грамм) • repetition penalty

Когда хорош: перевод, краткое суммирование — где важна высокая вероятность и детерминизм. Минусы: может быть «скучным», усиливает повторяемость.

3) Температура (Temperature Sampling)

Пере-нормируем логиты: z' = z / τ
p(y_t=k) = softmax(z'_k)
• τ < 1 → острее распределение (более детерминировано)
• τ > 1 → более ровное (больше разнообразия/шума)

Интуитивно: температура управляет «жадностью» vs креативностью. Хороша в паре с top-k/top-p.

4) Top-k и Nucleus (Top-p) sampling

Top-k: оставить k самых вероятных токенов, пере-нормировать и сэмплировать из них.
Nucleus (Top-p): выбрать минимальный набор токенов с суммарной массой ≥ p, сэмплировать из
Рецепт: k=40…200, p=0.8…0.95 (подбирать под задачу)

Отличие: top-k фиксирует размер «окна»; top-p — фиксирует «надежность» (массу вероятности). Top-p адаптивен: при пике — узко, при плоском — шире.

5) Современные и гибридные стратегии

5.1. Typical sampling

Идея: выбирать токены с «типичной» сюрпризностью близкой к энтропии распределения.
Эвристика: отбирать токены, где −log p(k) ≈ H(p), отсекая «слишком предсказуемые/редкие».

Даёт баланс между банальностью и галлюцинациями; часто стабилен при диалогах.

5.2. Contrastive search

Баланс «правдоподобия и не-повторяемости»:
argmax_y [λ·log p(y_t | y_{<t}) − (1−λ)·max_{j < t} cos(h_t, h_j)]

Снижает «тупое» повторение, сохраняя связность. Полезно для длинных ответов/статей.

5.3. Mirostat (1.0/2.0)

Онлайн-контроль целевой перплексии τ̂: подстраивает «эффективный» порог во время генера
стремясь удерживать заданный уровень «неопределённости». Хорош для устойчивых диалого

5.4. Спекулятивное декодирование (Speculative decoding)

Быстрение: маленькая «черновая» модель предлагает несколько шагов;
большая проверяет/принимает/отклоняет. Ускоряет без изменения распределения таргет-моде

6) Контроль повторов и форматирования

- no-repeat-n-gram (запрет n-грамм)
- repetition penalty (штраф логитам встречавшихся токен
- presence/frequency penalties
- бан листы токенов (stop-words/разметка)
- контроль длины: min/max length, length penalty, EOS threshold

Для задач фактов — лучше ниже температура и аккуратный top-p; для креатива — выше температура/шире p, но сильнее анти-повторные ограничения.

7) Как выбирать стратегию (правило большого пальца)

- Перевод/резюме/QA с точностью → Beam (B=4..8) + length/coverage penalties; no-repeat-n-gram
- Диалог/креатив → top-p (0.9) + τ≈0.7 (или typical). Иногда top-k=50..100.
- Длинный текст → contrastive / mirostat; включить анти-повторы.
- Продуктив → ограничить температуру (≤0.7), строгие стоп-токены, калиброванные пороги.

8) Чек-лист для устного ответа (≈1–2 минуты)

1) Поиск vs семплинг: детерминизм vs разнообразие.
2) Beam: сумма лог-вероятностей + length penalty; проблемы повторов — coverage/diversity/no-r
3) Температура, top-k, top-p: что изменяют и как настраивать.
4) Современные: typical, contrastive, mirostat, speculative — когда полезны.
5) Практика: анти-повторы, контроль длины, выбор по задаче (точность vs креатив).

Готово: краткая лекция со справочными формулами и рецептами. Могу добавить разворот с графиками влияния τ/p/k на распределение.

Лекция: Токенизация — BPE/BBPE, WordPiece, SentencePiece (Unigram/BPE) — ≈5 минут

Зачем субсловные токенизаторы, базовый пайплайн, алгоритмы обучения/кодирования, различия маркеров (##, ▁, Ġ), плюсы/минусы и когда что выбирать.

1) Зачем субсловные токены

Открытый словарь: редкие слова, формы, имена, эмодзи. Субсловный подход уменьшает OOV и размер словаря, сохраняя способность моделировать морфологию.

Пайплайн:
[Нормализация] → [Пре-токенизация (опц.)] → [Модель токенизации] → [Пост-обработка: спецто

2) BPE (Byte-Pair Encoding в NLP, Sennrich et al.)

Идея: итеративно склеиваем самые частые соседние символы/сабтокены, формируя словарь фиксированного размера K. Кодирование — жадное применение правил слияния.

\
Обучение (эскиз):
1) Инициализация: разбейте слова на символы + маркер конца слова </w> (или пробельный ма
2) Повторять, пока |V| < K:
- Посчитать частоты пар (a,b) во всех словах корпуса.
- Выбрать самую частую пару (A,B) и слить её → новый символ A_B.
- Обновить представление слов (заменить все вхождения A B на A_B).

Кодирование:
- Разбить слово на символы (+</w>), затем применить merge-правила в порядке обучения, по

Пример: "lower" → l o w e r </w> → low er </w> → low er</w>
Плюсы: простой/быстрый, открытый словарь от символов.
Минусы: чувствителен к предразбиению/регистру/пробелам, не оптимален по правдоподобию.

3) Byte-Level BPE (BBPE, как в GPT-2)

\
Идея: работать на уровне байтов (UTF-8), а не символов Unicode/слов.
• Начальный алфавит = 256 байтов → никакого OOV (всегда разложим на байты).
• Пробел часто кодируется отдельным признаком (например, символ 'Ġ' перед токеном у GPT-2
• Нормализация минимальна; эмодзи/диакритика/редкие символы обрабатываются устойчиво.
Плюсы: «нулевой» OOV, единый механизм для любых языков/символов.
Минусы: токены иногда выглядят непривычно; больше шагов для обычных слов (до слияний).

4) WordPiece (Google, используется в BERT)

Похоже на BPE, но критерий выбора слияний связан с максимизацией правдоподобия корпуса (через оценку вероятностей субслов). На практике тренируется жадно: добавляем слияния, которые улучшают лог-правдоподобие.

\
Особенности:
• Маркер продолжения слова: префикс '##' у продолжений (например, "play", "##ing").
• Разбиение обычно после пробельной претокенизации (слова → субслова).
• Словарь фиксированного размера (например, 30k).
Плюсы: хорошо работает в энкодерах (BERT), устойчив к морфологии.
Минусы: зависит от претокенизации/регистра и нормализации (нужна аккуратность).

5) SentencePiece (Kudo): BPE и Unigram LM без претокенизации

Работает по «сырому» тексту (без разбиения на слова). Специальный символ '▁' означает границу слова (замена пробела). Поддерживает два режима: BPE и Unigram LM.

\
5.1. Unigram LM (основной режим):
• Старт: большой «начальный» словарь сабтокенов.
• Обучение: итеративно удаляем токены, чьё удаление минимально ухудшает правдоподобие
• Регуляризация: можно семплировать альтернативные сегментации при обучении/инференсе
Плюсы: вероятностная модель сегментаций, гибкость, хорошая перплексия.
Минусы: обучение сложнее/медленнее, чем у BPE.

5.2. SentencePiece-BPE:
• Реализация BPE внутри SentencePiece, но без претокенизации; границы слов — через '▁'.
Плюсы: стабильность и единый пайплайн; меньше зависимостей от языка.

6) Маркеры и пост-обработка

\
• BPE (классический): </w> (или 'Ġ' как «пробел-метка» у BBPE).
• WordPiece: '##' префикс продолжения (начало слова — без '##').
• SentencePiece: '▁' — явная граница слова в самом токене.
• Спецтокены: <pad>, <bos>/<eos>, <unk>, <cls>, <sep> и др.

Замечание: явные метки границ делают обратную детокенизацию однозначной и устойчивой к пробелам.

7) Практика: как выбирать

\
• GPT-подобные: Byte-Level BPE (open-vocab, устойчив к любым байтам).
• BERT-подобные: WordPiece (стабильно в энкодерах, '##' маркеры).
• Универсальный/многоязычный пайплайн: SentencePiece (Unigram) — без претокенизации, сим
• Гиперпараметры: размер словаря (e.g., 30k–100k), нормализация (NFKC/нижний регистр), char

Правило большого пальца: если важна универсальность и простота пайплайна — SentencePiece (Unigram). Если нужна совместимость с BERT — WordPiece. Для автогенных моделей и любых символов — BBPE.

8) Чек-лист для устного ответа (≈1–2 минуты)

\
1) Зачем субслова: открытый словарь, морфология, компактность.
2) BPE: частотные слияния символов → словарь; кодирование жадное по merge-правилам.
3) Byte-Level BPE: на байтах UTF-8 → нулевой OOV; спецмаркеры пробела (например, 'Ġ').
4) WordPiece: слияния под правдоподобие; '##' у продолжений; good для BERT.
5) SentencePiece: без претокенизации; '▁' как пробел; режимы BPE и Unigram (EM, удаление ток
6) Что выбрать: BBPE для генерации; WordPiece для энкодеров; Unigram SP для универсальност

Готово: краткая лекция с алгоритмами и практическими рецептами. Могу добавить слайд с пошаговым BPE на слове ('banana' → ba na na → ...).

Лекция: Метрики BLEU, ROUGE, Perplexity — и как ещё мерить качество (≈5 минут)

Зачем нужны метрики, формулы и «человеческие» переводы, ограничения и практические рекомендации по выбору набора метрик под задачу.

1) Зачем метрики вообще

Оцениваем: правдоподобие, смысл/покрытие фактов, связность/стиль, калибровку вероятностей, разнообразие и безопасность. Одна метрика не покрывает всё → используем набор метрик + человеческую оценку.

2) BLEU — n-граммная точность с штрафом за длину (перевод/пересказ)

Формула (корпусная):
BLEU = BP · exp(Σ_{n=1..N} w_n · log p_n)
p_n — модифицированная точность n-грамм (clipped precision).
BP (brevity penalty) = 1, если c > r; иначе exp(1 − r/c).
c — суммарная длина гипотез; r — «эффективная» длина референсов.

По-русски: доля совпавших 1/2/3/4-грамм среди сгенерированных × штраф за слишком коротки
Плюсы: быстро, исторический стандарт в MT. Минусы: плохо ловит перефразы/синонимы; надёж

3) ROUGE — recall-ориентировано (суммаризация)

ROUGE-N: recall по n-граммам (сколько из референса покрыли).
ROUGE-L: через LCS (наибольшая общая подпоследовательность), часто F-мера:
R = LCS/m, P = LCS/n, F_L = ((1+β^2)·P·R)/(R + β^2·P)

По-русски: «насколько хорошо покрыли то, что важно в эталоне?». Хорошо для экстрактивной с

4) Perplexity (PPL) — внутренняя метрика языковых моделей

Средняя отрицательная лог-вероятность:
H = −(1/T) Σ_t log p(y_t | y_{<t})
PPL = exp(H)

По-русски: «насколько модель удивляется тестовому тексту». Низкая PPL ≠ хороший перевод/о

5) Улучшенные автоматические метрики генерации

- METEOR — учитывает стемминг/синонимы; лучше BLEU на уровне предложений.
- chrF / chrF++ — F-мера по символьным n-граммам (робастно к морфологии).
- BERTScore — сходство гипотезы и референса в пространстве контекстных эмбеддингов (cosin
- BLEURT / COMET — обучаемые метрики, хорошо коррелируют с людьми.
- MAUVE — дивергенция распределений «человек vs модель» (естественность/диверсификация

6) QA, извлечение и RAG

- QA (короткие ответы): Exact Match (EM), F1 по токенам (SQuAD-стиль).
- Factuality: QAGS, FactCC, SummaC — фактическая согласованность с источником (ограниченна
- RAG: качество извлечения — Recall@k, MRR, NDCG@k, Hit@k; качество ответа — EM/F1.

7) Классификация/ранжирование и калибровка

- Accuracy при балансе; Precision/Recall/F1 (macro/micro/weighted) — при дисбалансе.
- ROC-AUC и PR-AUC (при дисбалансе информативнее PR-AUC).
- Калибровка вероятностей: ECE (Expected Calibration Error), Brier score.

- Ранжирование: MRR, NDCG, MAP.

8) Диверсификация и качество формы

- distinct-n (доля уникальных n-грамм) • repetition rate / no-repeat violations
- Перплексия внешней LM на выходах (гладкость) • читаемостные индексы (ограниченно пол

9) Надёжность/безопасность/этика

- Токсичность, bias/fairness (целевые наборы). • Утечки PII.
- Hallucination rate — доля фактических ошибок (часто ручная разметка).
- Робастность: устойчивость к перефразам/атакам (стресс-тесты).

10) Человеческая оценка — золотой стандарт

- Pairwise A/B (предпочтение), Likert (1–5: адекватность, беглость, фактичность, стиль).
- Слепая разметка + межсогл. (Cohen’s κ, Krippendorff’s α).
- LLM-as-a-Judge — только с калибровкой/перекрёстными проверками и разными judge-моделям

11) Практический playbook: «задача → метрики»

- MT: COMET/BLEURT + chrF/BLEU; обязательна выборка для human eval.
- Суммаризация: ROUGE-(1/2/L) + QAGS/FactCC; human factuality.
- Диалог/креатив: BERTScore/BLEURT/MAUVE + distinct-n; human (coherence/helpfulness).
- Языковые модели: PPL (внутр.) + задачные метрики (QA EM/F1, code pass@k, RAG EM/F1+Recal
- Классификация: F1-macro/PR-AUC; ECE/Brier для калибровки.
- Отчётность: несколько метрик, доверительные интервалы/бутстрэп, significance tests (paired

12) Чек-лист для устного ответа (≈1–2 минуты)

1) BLEU = n-грамм precision + brevity penalty; ROUGE = recall (LCS как ROUGE-L).
2) Perplexity — «удивление» LM, не равна полезности в задачах.
3) Современные: BERTScore, COMET/BLEURT, chrF, MAUVE.
4) Под задачу выбираем набор метрик + human eval; для RAG учитываем и извлечение (Recall@
5) Калибровка (ECE/Brier), разнообразие (distinct-n), безопасность (токсичность/bias/hallucinatio

Готово: сжатая, но полная лекция по метрикам. Могу добавить страницу с примерами расчёта BLEU/ROUGE на игрушных строках.

Лекция: Модели семейства GPT (Generative Pre-trained Transformer) — ≈5 минут

Что такое GPT, как устроена архитектура decoder-only Transformer, как обучают (NTP), чем дополняют (SFT/RLHF/DPO), как генерируют (KV-кэш, стратегии), и как адаптируют (PEFT, RAG, инструменты).

1) Идея и постановка задачи

GPT — это decoder-only Transformer, обучаемый предсказывать следующий токен (NTP):
maximize_θ Σ_{t=1..T} log p_θ(y_t | y_{<t})

По-русски: учим модель «продолжать текст»; оптимизируем кросс-энтропию; внутренняя метр
Пайплайн:
[Текст] → [Токенизация (BPE/BBPE)] → [Эмбеддинги + позиционные коды]
→ [[Masked Self-Attn] + [FFN]] × L → [LM head] → p(next | контекст)

2) Архитектура (decoder-only Transformer)

Вход: сумма токенного эмбеддинга и позиционного (sin/learned/RoPE/ALiBi).
Блок декодера (Pre-LN часто):
x → LayerNorm → Multi-Head Masked Self-Attn → +x
→ LayerNorm → FFN (W2·GELU(W1··)) → +…
Masked Self-Attn (1 голова):
Attn(Q,K,V) = softmax((QK^T)/√d_k + mask) V
Голова LM: линейный слой на размер словаря (часто weight tying с входными эмбеддингами).
Интуиция: токен на позиции t «смотрит» только назад (каузальная маска), смешивая разные «ракурсы» через multi-head attention.

3) Предобучение и законы масштабирования

Данные: веб/книги/код/диалоги → чистка, дедуп, безопасность (PII).
Оптимизация: AdamW, warmup, большие батчи, mixed precision.
Scaling laws: качество ~ степенной функции от {данные, параметры, FLOPs}; нужно согласовы
Контекст: рост 2k→8k→32k→128k+; используют RoPE/ALiBi и KV-кэш.

4) От «болтливой» LM к ассистенту: инструкции и предпочтения

1) SFT (Supervised Fine-Tuning): (инструкция, хороший ответ).
2) Reward Model (RM): обучаем на парах предпочтений (лучший/хуже).
3) RLHF (PPO) или DPO/IPO: дообучаем, чтобы модель предпочитала «хорошие» ответы.
4) Политики/безопасность: системные подсказки, правила, фильтры, конституционное обучени
Итог: модель становится послушной, полезной и безопасной, не теряя базовое языковое мастерство.

5) Инференс и скорость

KV-кэш: сохраняем K/V для уже сгенерированных токенов → следующий шаг O(L).
Спекулятивное декодирование: маленькая модель предлагает черновик, большая проверяет →
Стратегии генерации: greedy/beam (детерминизм) vs температура/top-k/top-p/typical/mirostat/co

6) Адаптация и инструменты

PEFT: LoRA/QLoRA, Adapters, Prefix/Prompt/Soft prompts — дешёвая настройка под домен.
RAG: извлекаем знания («контекст из базы/поиска») → добавляем в промпт → генерация.
Инструменты (tool use): function calling, код/таблицы/веб; агентные циклы (план→действие→кон

7) Сильные стороны и ограничения

Сильные: универсальность, перенос, масштабируемость качеством, zero/few-shot, гибкость дек
Ограничения: галлюцинации без RAG/инструментов; чувствительность к промптам; O(n^2) по д

8) Эволюция семейства (вехи)

GPT (2018): decoder-only + NTP → сильный перенос.
GPT-2 (2019): масштабирование, первые zero-shot промпты.
GPT-3 (2020): few-shot через in-context learning.
InstructGPT / 3.5: SFT + RLHF → ассистенты.
GPT-4-класс: лучше рассуждение, длинный контекст, мультимодальность, инструментальность
(Точные детали закрытых моделей раскрыты частично; принципы — как выше.)

9) Как решаем задачи GPT на практике

Классификация/извлечение: инструкция + формат (JSON), низкая температура.
Суммаризация/перефраз: чанки (map-reduce), контроль фактов (ссылки/проверка).
Код/агенты: план → инструменты → верификация → итерации.
Диалог: системное сообщение + persona + память; умеренные параметры семплинга.

10) Экзаменный чек-лист (≈90 секунд)

1) Определение: GPT — decoder-only Transformer, NTP: maximize Σ log p(y_t | y_{<t}).
2) Архитектура: токенизация → эмбеддинг+позиции → [Masked Self-Attn + FFN]×L → LM head.
3) От LM к ассистенту: SFT → RM → RLHF/DPO; правила/безопасность.
4) Инференс: KV-кэш, speculative; генерация — greedy/beam vs temperature/top-p и анти-повторы
5) Адаптация: LoRA/QLoRA, prompt-tuning; RAG и tool use для фактов/действий.
6) Плюсы/минусы: масштабируется и универсальна, но возможны галлюцинации и ограничение

Готово: компактная лекция для ответа на экзамене. Могу добавить сравнение GPT vs BERT/T5 и схемку размеров (d_model, d_k, d_ff, H).

Лекция: Сравнение архитектур BERT, GPT и T5 — ≈5 минут

Высокоуровневые различия: какие блоки включены (encoder/decoder), какие цели предобучения (MLM/NTP/denoising), как используется модель на инференсе, и когда какую семью стоит брать.

1) Коротко: зачем три разные архитектуры

- BERT — encoder-only (двунаправленное кодирование): лучше для понимания/классификации/и
- GPT — decoder-only (авторегрессия): лучше для генерации продолжений/диалогов/кода.
- T5 — encoder–decoder (seq2seq): гибкий «text-to-text» для преобразований (перевод/суммари

2) Архитектура и поток данных

BERT (encoder-only):
[Токены] → [Эмбеддинги + позиции] → [Self-Attn + FFN] × L → [Пулинг: [CLS]/mean] → [Задачная

GPT (decoder-only):
[Токены] → [Эмбеддинги + позиции] → [Masked Self-Attn + FFN] × L → [LM head] → генерация ток

T5 (encoder–decoder):
[Вход] → [Encoder: Self-Attn + FFN] × L → H_enc
[Выход] ← [Decoder: Masked Self-Attn + Cross-Attn(H_enc) + FFN] × L → генерация

3) Цели предобучения

- BERT: Masked Language Modeling (MLM) (+ историч. NSP). Двунаправленное «угадывание» зама
- GPT: Next-Token Prediction (NTP). Каузальная маска; учим «продолжать» текст.
- T5: Span-Corruption Denoising («text-to-text»): вырезаем спаны и просим декодер восстановить

Перевод: BERT учится «понимать контекст вокруг дырок»; GPT — «писать дальше»; T5 — «восстанавливать пропуски как задачу переформулирования текста».

4) Токенизация и позиционные представления

- BERT: WordPiece; как правило обучаемые (absolute) позиционные эмбеддинги.
- GPT: Byte-Level BPE; позиционные — обучаемые/альтернативно RoPE/ALiBi; каузальная маска с
- T5: SentencePiece (Unigram); относительные позиционные смещения/байасы в attention.

Важно: выбор токенизатора влияет на OOV/многоязычность; позиционные коды — на длину контекста и перенос между длинами.

5) Как используется на инференсе

- BERT: извлекаем эмбеддинг [CLS] или mean-пулинг → классификация/ретривер; токен-класси
- GPT: авторегрессия → стратегии генерации (greedy/beam/temperature/top-p...); диалоги, код, с
- T5: «text-to-text»: "task: <описание> input: <текст>" → декодер генерирует ответ; beam для

6) Сильные/слабые стороны и когда что брать

BERT — сильные:
- Быстрый и дешёвый на инференсе для классификации/извлечения (только энкодер).
- Отличные эмбеддинги предложений/токенов; устойчив к коротким контекстам.
- Минусы: не генерирует; контекст обычно короче; NSP исторически спорен.

GPT — сильные:
- Лучшая генерация, zero/few-shot через in-context; длинные контексты; гибкие стратегии выв
- Минусы: для «понимания без генерации» дороже на инференсе; склонен к галлюцинациям б

T5 — сильные:

- Универсальный seq2seq: перевод, суммаризация, QA, перефраз; естественно поддерживает
- Минусы: для генерации дороже, чем GPT (нужно гнать и encoder, и decoder); тонкая настрой

7) Практика выбора (правило большого пальца)

- Классификация/извлечение/поиск эмбеддингов → BERT/roBERTa/DeBERTa (encoder-only).
- Открытая генерация/диалог/код → GPT-класс (decoder-only) + анти-повторы/контроль фактов
- Перевод/суммаризация/табличные-to-текст → T5/BART (encoder–decoder), beam + coverage/len
- Многоязычие: mBERT/XLM-R (encoder-only) или mT5 (enc–dec); для генерации — mT5.

8) Дообучение и адаптеры

- BERT: добавляем головы (классификация, токен-классификация), fine-tune end-to-end; PEFT —
- GPT: SFT → RLHF/DPO; PEFT (LoRA/QLoRA, prefix/prompt/soft prompts); RAG/инструменты для фак
- T5: обучаем «инструкции» в стиле text-to-text; делим задачи на промпты; PEFT (LoRA/Adapters

9) Вычислительная сторона

- Encoder-only (BERT): O(n^2) по длине в self-attn, но нет дорогого декодера на инференсе → дё
- Decoder-only (GPT): O(n^2) в обучении; на инференсе с KV-кэшем шаг O(n); генерация последо
- Encoder–decoder (T5): два стека → дороже для генерации; зато мощный cross-attn для точных

10) Экзаменный чек-лист (≈90 секунд)

1) BERT = encoder-only + MLM (двунаправл. понимание); GPT = decoder-only + NTP (авторегресси
   T5 = encoder–decoder + span-denoising («text-to-text»).
2) Токенизаторы: WordPiece (BERT), BBPE (GPT), SentencePiece Unigram (T5); позиции: abs (BERT),
3) Инференс: BERT — эмбеддинг → head; GPT — генерация; T5 — seq2seq «task: ... input: ...» → де
4) Когда что: понимание/классификация → BERT; открытая генерация/диалог → GPT; структурны
5) Ограничения: длина контекста и O(n^2); галлюцинации у GPT; стоимость enc–dec у T5.

Готово: компактная лекция для сравнения BERT/GPT/T5. По запросу добавлю таблицу отличий по гиперпараметрам (d_model, d_ff, H, токенизация, позиции).

Лекция: In-Context Learning — zero-shot, few-shot, chain-of-thought (≈5 минут)

Как LLM учится «на лету» из подсказки без обновления весов: виды промптов, схемы, приёмы повышения качества и подводные камни.

1) Интуиция: что такое In-Context Learning (ICL)

ICL — это когда модель извлекает «правило»/паттерн прямо из примеров в контексте и применяет его к новой задаче без градиентного обучения. Контекст = инструкция + (опц.) демонстрации + вопрос. Модель использует self-attention, чтобы сопоставить текущий запрос с примерами.

Схема:
[Инструкция] + [Демонстрации (опц.)] + [Вопрос] → Модель → [Ответ]

2) Zero-shot prompting (без примеров)

Идея: описать задачу словами и попросить ответ.
Шаблон:
Инструкция: "Классифицируй отзыв как positive/negative. Объяснения не давай."
Вход: "Сервис ужасный, но еда отличная."
Выход: "mixed" (или "positive"/"negative" — зависит от инструкции и набора меток)
Трюки:
- Ясные метки и формат ("Ответ: <label>", "JSON с полем label").
- Роль/стиль ("Ты — строгий модератор тональности").
- Подсказки-рамки: "Подумай пошагово, затем выведи только метку."

3) Few-shot prompting (с примерами-демонстрациями)

Идея: показать K примеров «вход → правильный выход», затем дать новый вход.
Шаблон k-shot:
Task: sentiment (labels: positive, negative)
Ex1: "Отличная подача." → positive
Ex2: "Скучно и затянуто." → negative
Query: "Неплохо, но второй акт слабый." → ???
Практика:
- Порядок примеров влияет (order effects).
- Баланс по классам; близость домена к запросу.
- Формат единообразен (разделители, префиксы "Вход:", "Выход:").
- Явная спецификация пространства меток (чтобы не «придумывал» лишнее).

4) Chain-of-Thought (CoT): пошаговые рассуждения

Идея: заставить модель «расписать» промежуточные шаги для сложных задач (арифметика, л
Базовый прием:
"Давай подумаем шаг за шагом." (англ. "Let's think step by step.")
Few-shot CoT:
Примеры включают и ход рассуждений, и финальный ответ.
Self-Consistency:
Сэмплировать несколько CoT-траекторий (T раз) → выбрать наиболее частый финальный ответ
Scratchpad:
Хранить промежуточные переменные/соображения в явном формате (списки, таблицы).

Важно: CoT повышает точность на сложных задачах, но удлиняет ответы. В проде часто применяют «скрытый CoT» (не показывать пользователю рассуждение, а выводить только ответ).

5) Структурированные промпты и инструменты

- Формат ответа: JSON/таблица/ключи → снижает халлюцинации и повышает пригодность.
- ReAct: чередуем Reason + Act (вызов инструмента: поиск/код), сохраняем трассу.
- Program-Aided LM (PAL): просим писать маленькие функции/псевдокод для расчётов.
- RAG: добавляем релевантные документы в контекст перед ICL → фактичность выше.

6) Подводные камни и как их обходить

- Порядок и формулировки в примерах (position & framing bias).
- Спутывание меток → всегда явно объявляйте допустимые метки.
- Перегруз контекста: длинные примеры «съедают» окно → сжимайте/суммаризируйте.
- Доменный сдвиг: берите примеры ближе к запросу, обновляйте набор демо.
- Разглашение CoT: в некоторых задачах CoT не следует показывать пользователю → разделяй

7) Мини-шаблоны (за 30 секунд настроить промпт)

Zero-shot (классификация):
  Задача: Определи тональность текста (labels: positive, negative, neutral).
  Формат: выведи JSON {"label": "<...>"} без комментариев.
  Текст: "..."
Few-shot (извлечение сущностей):
  Задача: NER на метках PER, ORG.
  Примеры:
    B: "Илья Сегалович основал Яндекс." O: [{"text":"Илья Сегалович","label":"PER"},{"text":"Янде
    B: "Сатя Наделла возглавляет Microsoft." O: [{"text":"Сатя Наделла","label":"PER"},{"text":"Mic
  B: "<новый текст>" O: ???
CoT (арифметика):
  B: "У Кати было 5 яблок, подарили ещё 7, затем съела 3. Сколько осталось?"
  O: "Думаю шаг за шагом: ... Итог: 9"

8) Чек-лист для устного ответа (≈60–90 сек.)

1) ICL = обучение «на лету» по примерам в контексте (без обновления весов).
2) Zero-shot: одна инструкция; Few-shot: K демонстраций → правило; CoT: явные шаги рассужде
3) Усилители: self-consistency, структурный формат, ReAct/PAL, RAG.
4) Риски: порядок примеров, двусмысленные метки, длина контекста, разглашение рассуждени

Готово: компактная лекция с практическими шаблонами. Могу добавить таблицу «когда применять CoT и как выбрать K для few-shot».

Лекция: Parameter-Efficient Finetuning (PEFT) — Soft Prompts, (Q)LoRA, IA³ (≈5 минут)

Зачем нужны PEFT-методы, как они экономят параметры/память, как устроены Soft Prompts/Prompt-/Prefix-/P-tuning v2, LoRA/QLoRA и IA³, когда что выбирать.

1) Мотивация: почему не full fine-tune

Проблема: большие LM (B) → полное дообучение = много VRAM, времени и риск забывания.
Идея PEFT: «замораживаем» базовую модель и учим малую добавку/параметры (1e4–1e8 вместо
Цели: (а) дешевле хранить множество задач, (б) проще переключаться, (в) устойчивее к катаст

2) Soft Prompts / Prompt-/Prefix-/P-tuning v2

2.1. Soft Prompt (Prompt Tuning):
- Вводим trainable векторы P ∈ R^{m×d_model} и конкатенируем с эмбеддингами входа: [P ||
- Учим только P; веса модели фиксированы. Хорошо для одной задачи/домена.

Плюсы: минимализм (очень мало параметров), быстро. Минусы: ограниченная мощность; чувст

2.2. Prefix Tuning:
- Для каждого слоя attention добавляем «ключи/значения» префикса (K_pref, V_pref), зависящи
- Модель «видит» эти виртуальные токены на каждом слое без изменения базовых весов.

Плюсы: сильнее, чем plain soft prompt; гибкость по слоям. Минусы: больше параметров, чем у pr

2.3. P-tuning v2:
- Обобщение/усовершенствование: вставляем обучаемые «префиксы» на всех слоях в стиле p
- Характерно для encoder/decoder и больших LM; хорошо масштабируется.

3) LoRA и QLoRA

3.1. LoRA (Low-Rank Adapters):
- Идея: не дообучать матрицу W ∈ R^{d_out×d_in}, а учить низкоранговую добавку ΔW = A B^
  где A ∈ R^{d_out×r}, B ∈ R^{d_in×r}, r ≪ min(d_in, d_out). Итог: W' = W + α/r · A B^T.
- Обычно вставляют в проекции внимания (q_proj, v_proj; иногда k,o) и/или в FFN (up/down).
- Учим только A,B; базовые W заморожены.

Плюсы: сильный компромисс «качество↔параметры», легко комбинировать несколько задач (сл
Минусы: требует аккуратного выбора r, α и целевых модулей.

Типичные гиперпараметры:
r: 4–64 (часто 8/16/32), α: 8–64, dropout_lora: 0–0.1, target_modules: ["q_proj","v_proj"].

3.2. QLoRA:
- Замораживаем базовую модель в 4-бит (NF4) + 8-бит оптимизаторы страницами (paged optim
- LoRA-адаптеры учим в 16-бит/32-бит, сохраняя качество при очень низкой VRAM.

Плюсы: дообучение 7–13B на одной 24–48GB GPU; почти как full FT по качеству на ряде задач.
Минусы: чувствительность к квантованию/скейлам; важны правильные инференс/сохранение ч

4) IA³ (Infused Adapters by Inhibiting and Amplifying Inner Activations)

Идея: вместо ΔW учим векторные «масштабы» (гейты), которые переумножают отдельные кана
Формально (эскиз):
- Внимание: g_q, g_k, g_v по размерности head/feature — масштабирование перед/после проек
- FFN: g_ffn по промежуточному размеру d_ff — масштабирование активаций.

Параметров ещё меньше, чем у LoRA; вставляется в многие слои.

Плюсы: крайне параметро-экономично; хорошая совместимость с большими LM.
Минусы: иногда уступает LoRA по потолку качества; требует подбора, куда именно встраивать

5) Что выбирать и когда (правило большого пальца)

- Быстрый доменный «почерк» на готовой LM, мало VRAM → Soft/Prefix/P-tuning v2.
- Общая высококачественная адаптация под задачу (классиф./генерация) → LoRA (r=8..32, α=1
- Очень мало VRAM, большие модели (7–70B) → QLoRA (NF4, paged Adam).
- Супер-экономия параметров (десятки тысяч) → IA³ (если качество ок на вашем датасете).

Комбинации:

- Soft Prompts + LoRA (часто работают лучше вместе).
- RAG + PEFT: снижает требования к дообучению, если факты извлекаем из базы.

6) Практика дообучения (tips)

- Заморозить базу, обучать только адаптеры; grad checkpointing, mixed precision (bf16/fp16).
- LR подбирайте отдельно для адаптеров (у LoRA часто 1e-4..3e-4); warmup 3–8% шагов.
- LoRA: проверьте какие модули целевые (q,v vs все проекции); r слишком мал → недообучение
- QLoRA: NF4 + double quant + paged AdamW; не забыть save_safetensors лора-весов и конфиг к
- Оценка: задачные метрики (F1/ROUGE/COMET…), а не только Perplexity; следите за калибровк

7) Подводные камни

- Catastrophic forgetting при слишком агрессивном FT даже с PEFT (например, если трогаете Lay
- Drift стиля/формата ответов; фиксируйте формат (JSON/регекс) в промпте.
- Несовместимость чекпоинтов (квантованные базовые веса vs не-квантованные); проверяйте в
- Утечка приватных данных при FT; применяйте фильтрацию, дифф. приватность при необходи

8) Чек-лист для устного ответа (≈60–90 сек.)

1) PEFT = дообучаем малую добавку, базу замораживаем (дешевле/быстрее/многозадачно).
2) Soft/Prefix/P-tuning v2: обучаемые префиксы/токены (на входе/на слоях).
3) LoRA: $W' = W + \alpha/r \cdot A\,B^T$, $r \ll d$; учим A,B; обычно на q,v (и/или FFN).
4) QLoRA: базу квант. до 4-бит (NF4), учим LoRA в 16-бит; paged оптимизаторы.
5) IA³: обучаемые масштабы каналов в attention/FFN; ещё меньше параметров.
6) Выбор: быстрый стиль → Soft/Prefix; качество/универсальность → LoRA; мало VRAM → QLoRA; с

Готово: компактная лекция по PEFT. По запросу добавлю страницу с псевдокодом LoRA/QLoRA и схемой размещения адаптеров по слоям.

Лекция: RLHF — PPO и DPO (≈5 минут)

Зачем нужен RLHF, как устроены этапы SFT → Reward Model → RL (PPO), и как DPO упрощает пайплайн, оптимизируя предпочтения напрямую.

1) Мотивация RLHF

Сырая языковая модель умеет «продолжать текст», но не всегда следует инструкциям, может быть небезопасной и галлюцинировать. RLHF добавляет человеческие предпочтения, чтобы сместить политику к ответам, которые люди считают полезными и безопасными.

Классический пайплайн:
(1) SFT: обучаем на парах (инструкция → хороший ответ).
(2) Reward Model (RM): учим оценивать «насколько ответ хорош» по человеческим сравнениям
(3) RL (чаще PPO): дообучаем политику максимизировать reward и сдерживаем отклонение от

2) Reward Model (RM): обучение по парам предпочтений

Дано: пары (x, y^+, y^-), где y^+ предпочтён человеку над y^- для запроса x.
Модель вознаграждения $r_\varphi(x, y)$ (обычно LM с головой-скалярой).
Обучение (Bradley–Terry):

$$\text{maximize}_\varphi \quad \Sigma \log \sigma(r_\varphi(x, y^+) - r_\varphi(x, y^-))$$

По-русски: RM учится предсказывать более высокий «скор» для ответов, которые предпочитают люди.

3) RL через PPO для дообучения политики

Состояние s: (инструкция x, уже сгенерированные токены); действие a: следующий токен.
Вознаграждение: $r = r_{RM}(x, y) - \beta \cdot KL(\pi_\theta(\cdot|x) \,||\, \pi_{ref}(\cdot|x))$ # штраф за сильный уход от рефе
Цель: максимизировать ожидаемую сумму вознаграждений (обычно вся награда в конце ответа

PPO (клипнутый суррогат):

$$L_{CLIP}(\theta) = E[\min(\rho_t(\theta) \hat{A}_t , \text{clip}(\rho_t(\theta), 1-\varepsilon, 1+\varepsilon) \hat{A}_t)] - \beta \cdot KL$$

где $\rho_t(\theta) = \pi_\theta(a_t|s_t) / \pi_{\theta\_old}(a_t|s_t)$, $\hat{A}_t$ — оценка преимущества (например, GAE).
Практика в LLM:
- Генерируем ответы текущей политикой, получаем reward от RM.
- Оцениваем преимущества (вознаграждение минус базлайн/ценность).
- Обновляем θ на несколько эпох с KL-контролем к π_{ref} (обычно SFT-модель).

Интуиция: поднимаем вероятность «хороших» ответов по RM, но держим модель близко к референсу, чтобы не сломать стиль/знания.

4) Трудности PPO-этапа

- Нестабильность: reward hacking, переподгон к RM, коллапс стиля.
- Дорого: онлайновая генерация, оценка RM, несколько эпох обновлений.
- Инженерия: баланс β (KL), длина/штрафы, значение-базлайн, нормировки.

5) DPO: оптимизация предпочтений напрямую (без RM/PPO)

Идея: обойти RM и RL, свернуть задачу в логистическую регрессию на логитах политики относи
Для пары (x, y^+, y^-):

$$J_{DPO}(\theta) = - \Sigma [\log \sigma(\beta (\log \pi_\theta(y^+|x) - \log \pi_{ref}(y^+|x)) - \beta (\log \pi_\theta(y^-|x) - \log \pi_{ref}(y^-|x)))]$$

Эквивалентно: хотим, чтобы политика θ давала выше «приведённый логит» выбранному ответу
Параметр β управляет «жёсткостью» предпочтения (температура).

По-русски: DPO — это обычное SGD-обучение над парами «хороший/плохой ответ», где сравниваются лог-вероятности политики и референса; никакой онлайновой

генерации и отдельной RM не нужно.

6) Сравнение PPO и DPO

PPO (через RM):
+ Гибкость: можно комбинировать разные источники наград/штрафов, дообучать онлайново.
+ Явный контроль KL к референсу; можно ввести длину/токсичность/прочие сигналы.
− Сложнее и дороже; риск reward hacking; чувствителен к качеству RM.

DPO (прямо по парам):
+ Просто и стабильно: одна цель; стандартный градиентный спуск; быстро.
+ Не нужен отдельный RM и онлайновая генерация.
− Ограничен статическим датасетом пар; сложнее учесть сложные/многофакторные награды

7) Практика: что важно в данных и гиперпараметрах

- Качество пар «предпочтённый/отвергнутый»: чистота, разметка по критериям (helpful/harmle
- Референс π_ref: обычно SFT-модель; фиксированная на протяжении обучения.
- DPO: подберите β (например, 0.05–0.5), баланс длины/стоп-слов, выбор «температуры» на ген
- PPO: β для KL; длина/coverage penalties; нормализация reward; регуляризация RM; early stoppi
- Оценка: человеческая (A/B, Likert), задачные метрики (например, полезность, фактичность), б

8) Безопасность и «конституции»

- Включайте «негативные» примеры (отвергнутые за токсичность/опасность).
- Constitutional-style: правила/принципы → само-критика и перегенерация.
- Red-teaming/стресс-тесты; мониторинг сдвига поведения после обучения.

9) Чек-лист для устного ответа (≈60–90 сек.)

1) RLHF = SFT → RM (по парам) → RL (обычно PPO с KL к референсу).
2) RM: обучаем по Bradley–Terry: maximize Σ log σ(r(x,y⁺) − r(x,y⁻)).
3) PPO: L_CLIP с преимуществом Â и KL-пенальти к π_ref; в LLM награда чаще в конце ответа.
4) DPO: без RM/PPO; логистическая цель на разности лог-вероятностей относительно π_ref, с те
5) Сравнение: PPO гибок, но дорог/нестабилен; DPO прост/стабилен, но ограничен статическим
6) Оценка: human A/B + задачные метрики + безопасность.

Готово: компактная лекция по RLHF с акцентом на PPO и DPO. По запросу добавлю схему с потоками данных и псевдокод обоих методов.

Лекция: Retrieval-Augmented Generation (RAG) — компоненты системы и Dense vs Sparse Retrieval (≈5 минут)

Зачем RAG, из чего он состоит, где стоит Dense/Sparse/Hybrid, как проектировать пайплайн и что мерить. В конце — чек-лист для устного ответа.

1) Что такое RAG и зачем он нужен

RAG — это схема, где LLM генерирует ответ, опираясь на извлечённые из внешнего хранилища документы. Цель: снизить галлюцинации, добавить свежие знания, повысить цитируемость/проверяемость.

Интуитивная схема (один проход):

```
[Запрос пользователя q]
  └─► [Retrieval: найти k релевантных кусочков D_k]
      (поиск в индексе: BM25/TF-IDF/ColBERT/DPR/Векторная БД)
      └─► [Rerank (опц.)]
          └─► [Сборка промпта: инструкция + цитаты]
              └─► [LLM-генерация ответа y с ссылками/экстрактами]
```

2) Компоненты RAG-системы

- Ингест (off-line):
  - Нормализация текста: чистка, дедупликация, устранение boilerplate.
  - Чанкинг: фикс. окна (e.g., 512–2048 токенов) с overlap; либо по заголовкам/абзацам.
  - Индексация:
    Sparse: inverted index (BM25/TF-IDF).
    Dense: эмбеддинги (bi-encoder) → ANN-индекс (HNSW, IVF-PQ, ScaNN, FAISS).
- Retrieval (on-line):
  - Формирование запроса (опц.: перефраз, расширение — HyDE, RAG-Fusion, query rewriting).
  - Первичный ретривер: Sparse / Dense / Hybrid.
  - Реранкер (cross-encoder/Bi-encoder + late interaction): сокращаем до k' лучших.
- Конструирование контекста:
  - Форматы: цитаты с источником, структурированный JSON, таблички.
  - Правила: лимит токенов, сортировка, дедуп, highlight снейппеты.
- Генерация и постобработка:
  - LLM с системным промптом («отвечай, ссылаясь на документы»).
  - Пост-валидация: проверки ссылок, answer grounding, фильтры безопасности.
  - Кэш/шардирование/логирование, обратная связь для дообучения ретривера.

3) Sparse Retrieval (BM25/TF-IDF) — суть и формулы

TF-IDF (эскиз): $w_{t,d} = tf_{t,d} \cdot \log(N / df_t)$

BM25 (Okapi):

$$score(q,d) = \Sigma_{t \in q} IDF(t) \cdot (tf \cdot (k1+1) / (tf + k1 \cdot (1 - b + b \cdot |d|/avgdl)))$$

Плюсы:

- Быстро и дёшево, прозрачные веса, устойчив к доменному сдвигу.
- Лёгкая интерпретация/дебаг (какие слова «сработали»).

Минусы:

- Точный лексический матч: плохо ловит синонимы/парафразы/межъязыковые запросы.
- Ограничения на короткие/длинные документы (настройка k1, b).

4) Dense Retrieval — идея, модели, индексы

Идея: обучить bi-encoder (E_q, E_d), чтобы $\cos(E_q(q), E_d(d))$ был высоким для релевантных пар.

Варианты:

- DPR (dual encoder), ANCE (негативы из ANN), Contriever (безразметочное), GTR/MPNet (универса
- Late interaction: ColBERT (MaxSim по токенам) — лучше точность, выше затраты на индекс.

Индексы:

- ANN: HNSW (граф), IVF-PQ/OPQ (квантование), ScaNN. Векторы нормируем (cos или dot).

Плюсы:

- Семантический матч, синонимы/перефразы, многоязычие; лучше на длинных/разреженных фо

Минусы:

- Дорогая индексация (эмбеддинги) и обновления; чувствителен к домену/языку → нужен FT.
- Память под вектора; сложнее дебажить; требуется тюнинг ANN (accuracy/latency).

5) Hybrid Retrieval и реранкинг

Hybrid:

score = α · score_sparse + (1−α) · score_dense (или конкатенация кандидатов с последующим ре

Реранкеры:

- Cross-encoder (BERT-класс): [q] [SEP] [chunk] → скор → топ-k'; лучшая точность, дороже.
- ColBERT-v2: компромисс через late interaction.

Практика:

- Возьмите top-K из Sparse+Dense, затем реранк до K' (например, 50 → 10).
- Балансируйте α; храните фичи для аудита и обучения следующей версии.

6) Мульти-хоп и планирование запроса

- Iterative RAG: сгенерируй гипотезу → доизвлеки уточняющие факты → обнови ответ.
- RAG-Fusion / Query decomposition: расщепляем сложный вопрос на под-вопросы.
- HyDE: LLM сперва пишет «гипотетический документ», его вектор → ретривер.
- Tool use: внешние API/БД (табличные, калькуляторы) как «retrievers» специального вида.

7) Как мерить качество RAG

Retrieval:

- Recall@k, MRR, NDCG@k, Hit@k, precision@k (на тестовом наборе q→релевантные доки).

End-to-end ответ:

- EM/F1 (QA), ROUGE/BERTScore (суммаризация), фатк-метрики (QAGS/SummaC), human A/B.

Grounding/citations:

- Доля ссылок, реально поддерживающих факты; penalize unsupported claims.

Latency/cost:

- ms на запрос, $ за 1k запросов; кэширование; шардирование/предварительная выборка.

8) Dense vs Sparse: плюсы/минусы и выбор

Sparse (BM25/TF-IDF):

+ Дёшево/просто, интерпретируемо, мало зависимостей, хорошо при узком домене и чётких кл
− Плохо синонимы/перефразы/мультиязычие; может «промахиваться» при длинных запросах.

Dense (DPR/ColBERT/Contriever):

+ Семантический матч, перенос между языками/стилями; лучше на свободных формулировках.
− Эмбеддинги стоят времени/денег; чувствительность к сдвигу домена; сложнее тюнинг ANN и

Hybrid:

+ Лучшее из двух миров; устойчивость к разным типам запросов.
− Сложнее система (двойной индекс, фьюжн, реранк); выше латентность/стоимость.

9) Инженерный чек-лист

- Чанкинг: 512–1024 токенов с overlap 10–20%; отдельные стратегии для таблиц/кода.
- Нормализация/дедуп: уменьшить шум и повторяемость.
- Ретривер: начните с Hybrid (BM25 + Contriever/DPR), K=50; затем реранк до K'=10 cross-encoder'
- Промпт: просите цитировать источники; ограничьте «галлюцинации» (если нет источника — ск
- Кэш: запросы, эмбеддинги, результаты реранка.
- Мониторинг: Recall@k и доля «unsupported» утверждений в проде.

10) Экзаменный чек-лист (≈90 секунд)

1) Определение: RAG = LLM + внешнее извлечение (retrieval) → снижение галлюцинаций и актуал
2) Компоненты: ингест (чанкинг, индекс), ретривер (sparse/dense/hybrid), реранк, сборка контекст
3) Sparse (BM25) vs Dense (DPR/ColBERT): лексика vs семантика; цена/скорость vs качество; гибри
4) Мульти-хоп и query planning (RAG-Fusion, HyDE). 5) Метрики: Recall@k, MRR, end-to-end EM/F1/RO

Готово: компактная лекция по RAG. По желанию добавлю страничку с сравнением ANN-индексов (HNSW vs IVF-PQ) и шаблоном промпта для цитирования.

Лекция: Что такое агент? Чем агент отличается от «классической» LLM? (≈5 минут)

Дадим строгое, но практичное определение агента, разберём его компоненты и петлю «восприятие→план→действие→наблюдение». Сравним с обычной LLM, которая генерирует ответ в один проход без действий во внешней среде.

1) Определение агента

Агент — это система, которая (1) получает цель/задачу и наблюдения из среды, (2) планирует
(3) выполняет действия через инструменты/интерфейсы внешней среды, (4) получает новые на
повторяя цикл до достижения цели или остановки.
Формула цикла:

```
while not done:
    o_t = observe(env)
    a_t = policy(o_{≤t}, goal, state)        # «подумать» (reason) и «выбрать» (act)
    (env, o_{t+1}, r_t) = act(env, a_t)      # действие через инструменты/API
    state = update(state, o_{t+1}, r_t)      # короткая/долгая память, план
```

2) Базовые компоненты агента

- Интерфейсы «восприятия»: чтение документов, веб-поиск, база знаний, датчики/файлы/БД.
- Политика/мозг: LLM или другая модель, которая планирует (plan) и выбирает действие (act).
- Инструменты (tools): функции/API — веб-поиск, калькулятор, код-исполнитель, базы данных, е
- Память/состояние:
  - Краткосрочная (scratchpad/context): текущие мысли, промежуточные результаты.
  - Долгосрочная (vector store/БД): опыт прошлых эпизодов, заметки.
- Планировщик/контроллер: ReAct/Plan-Act/Tree-of-Thoughts/Reflexion — управляет шагами.
- Правила остановки и бюджет: лимиты по шагам, времени, стоимости; условия успеха/неуспех

3) Варианты агентного цикла (паттерны)

- ReAct: чередуем Reason (пишем краткий план/обоснование) и Act (вызываем инструмент) → О
- Plan→Execute: сначала план из K шагов, затем по одному выполняем с проверками.
- Toolformer / Function Calling: модель сама решает, какую функцию и с какими аргументами вы
- Reflexion: после ошибки агент анализирует причины и корректирует стратегию.
- Multi-agent: роли (исследователь/критик/исполнитель) + координация.

4) Классическая LLM vs Агент — принципиальные различия

Классическая LLM (без агентности):

- Ввод → один проход генерации → Вывод. Нет внешних действий. «Знания» ограничены обуч
- Нет явного состояния между шагами (кроме токенов в текущем промпте).
- Контроль качества — только через настройки генерации (temperature/top-p и др.).

Агент:

- Многошаговый цикл с планом и действиями во внешней среде (инструменты/API).
- Явное состояние/память между шагами; может уточнять запрос, искать, проверять, исправл
- Целевая оптимизация: метрики успеха задачи (task success rate), а не просто правдоподобны
- Нужны механизмы безопасности/ограничений (policy, sandbox, бюджет шагов).

5) Примеры агентных сценариев

- Web-исследователь: формирует план поиска, открывает ссылки, извлекает факты, пишет отч
- Data-agent: подключается к SQL/пандас, строит графики, валидирует гипотезы, сохраняет ар
- Coding-agent: редактирует репозиторий, запускает тесты, читает логи, делает PR.

- Ops-agent: следит за метриками, перезапускает сервисы, оповещает, документирует.

6) Агент vs RAG и tool calling

- RAG ≠ агент: RAG — это стратегия подачи контекста (извлечение + генерация).
  Агент может использовать RAG как один из инструментов, но также делает другие действия
- Tool calling: интерфейс для вызова функций. Агент использует его в цикле, сверяя результат

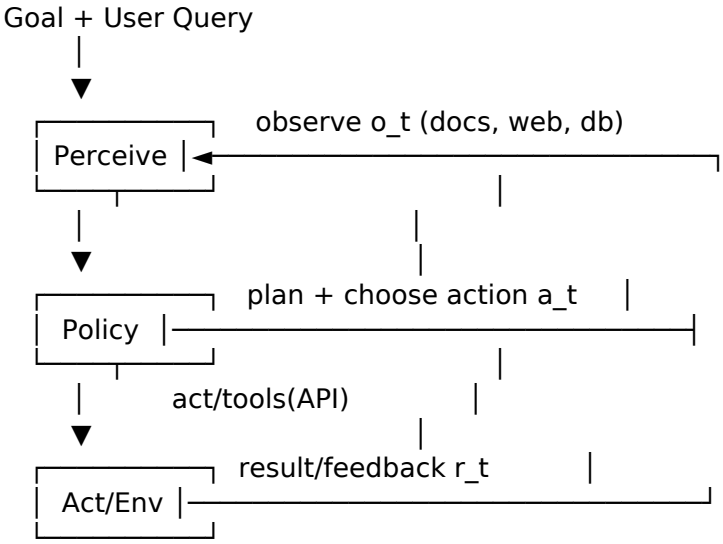
7) Как мерить качество агента

- Task Success Rate / Pass@k: доля успешно решённых задач/кейсов.
- Корректность результата (accuracy/EM/F1 или доменные метрики), наличие ссылок/обоснован
- Эффективность: шаги, латентность, стоимость ($/запрос), число вызовов инструментов.
- Безопасность: нарушения политик, рискованные действия, утечки PII.
- Надёжность: воспроизводимость (seed), робастность к шуму/ошибкам инструментов.

8) Риски и как их снижать

- Инструментальные галлюцинации: вызов несуществующих/неуместных функций → строгая сх
- Зацикливание/«тупёж»: лимиты шагов, watch-dog, эвристики остановки («если 2 раза подряд
- Выполнение кода/запросов: изоляция (sandbox), контроль разрешений, аудит логов.
- Надёжность источников: верификация фактов (grounding), цитаты, проверка ссылок.

9) Мини-схема петли агента (ASCII)



10) Экзаменный чек-лист (≈90 секунд)

1) Определение: агент = цикл восприятие→план→действие→наблюдение с инструментами и па
2) Компоненты: политика (LLM), инструменты, память (short/long), планировщик/правила остано
3) LLM vs Агент: один проход текста vs многошаговая цель-ориентированная интеракция с внеш
4) Паттерны: ReAct, Plan→Execute, Function Calling, Reflexion, multi-agent.
5) Метрики: успех задачи, корректность, стоимость/шаги, безопасность и воспроизводимость.

Готово: компактная лекция. По просьбе могу добавить страницу с примером промпта ReAct и шаблоном JSON-схемы для инструментов.