ИТМО

ВВЕДЕНИЕ В МОДЕЛИ ТРАНСФОРМЕРОВ: НЕЙРОСЕТЕВЫЕ ИМИТАЦИИ МЕХАНИЗМОВ ВНИМАНИЯ

САНКТ-ПЕТЕРБУРГ
2026

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

Введение в модели трансформеров: нейросетевые имитации механизмов внимания

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлению подготовки 09.03.02 Информационные системы и технологии
в качестве учебно-методического пособия для реализации основных профессиональных образовательных программ высшего образования бакалавриата

ИТМО

Санкт-Петербург
2026

Введение в модели трансформеров: нейросетевые имитации механизмов внимания / Али А., Осман В., Кашевник А. М. [и др.] – СПб: Университет ИТМО, 2026. - 84 с.

Рецензент(ы):

Кугаевских Александр Владимирович, кандидат технических наук, доцент (квалификационная категория "ординарный доцент") факультета программной инженерии и компьютерной техники, Университета ИТМО.

Учебное пособие включает базовый материал, вопросы для самопроверки и задания для практических занятий, в которых рассматриваются основные темы в рамках дисциплины «Практики машинного обучения: методы и инструменты для соревновательного анализа данных». Представленные материалы могут быть использованы для проведения практических занятий с целью систематизации и закрепления теоретических знаний и для осуществления контроля уровня освоения компетенций дисциплины.

Содержание

1. Введение

За последние несколько лет архитектура трансформеров стала основой современных моделей глубокого обучения. Она была представлена в 2017 году [24] как основной конкурент, превосходящий свёрточные и рекуррентные нейронные сети благодаря использованию исключительно механизма внимания, обеспечив масштабируемость, параллелизм и высокую производительность в различных модальностях. Сегодня архитектура трансформеров лежит в основе почти каждого значимого достижения в области глубокого обучения. К таким достижениям относятся большие языковые модели (такие как GPT и LLaMA), трансформеры для обработки изображений (ViT), мультимодальные системы (например, CLIP и LLaVA), а также модели в других предметных областях.

Однако, несмотря на повсеместное распространение, поверхностного знакомства с архитектурой трансформеров уже недостаточно. По мере роста размеров моделей, объёмов данных и сложности приложений способность *понимать, анализировать, модифицировать и оптимизировать* механизм внимания (как ключевой механизм в архитектуре трансформеров) становится критически важным навыком для исследователей, инженеров и практиков. Многие рассматривают эту архитектуру как «чёрный ящик», вызывая библиотечную функцию и полагаясь на результаты, однако такой подход серьёзно ограничивает возможности для работы с данными. Для более эффективной обработки данных требуется понимание основ функционирования механизма внимания: как именно вычисляются зависимости? Почему квадратичная сложность механизма становится узким местом? Можно ли аппроксимировать softmax (нелинейную функцию активации, которая преобразует входной вектор в распределение вероятностей), чтобы достичь линейного масштабирования без потери выразительности? Как различаются паттерны внимания в задачах компьютерного зрения и обработки естественного языка? Когда следует использовать cross-attention, где запросы формируются из одного сигнала, а ключи и значения, из другого, для обеспечения эффективного поиска или выравнивания между модальностями?

Данное учебное пособие призвано ответить на эти вопросы не только посредством теории, но также и **практических упражнений**. Каждая глава последовательно описывает материал: от математических основ механизма внимания до его самых передовых реализаций в современных ИИ-системах. Обучающийся самостоятельно реализует ключевые компоненты, визуализирует карты внимания, оценивает время выполнения и потребление памяти, сравнивает архитектурные решения и, в конечном счёте, разработает собственные эффективные варианты механизма внимания.

В то время как во многих учебных курсах по машинному обучению механизм внимания лишь упоминается в рамках лекций по распознаванию естественного языка (NLP) или компьютерному зрению (CV), в дан-

ном пособии механизм внимания становится центральным объектом изучения. В начале рассматривается классический подход к самовниманию (self-attention) и перекресному вниманию (cross-attention), рассматриваются их основные свойства и ограничения. Затем рассматриваем современные решения: линейное внимание на основе ядерных методов, разреженные аппроксимации и архитектурные приёмы, сохраняющие производительность при увеличении длины контекста. Последующие главы применяют эти идеи к конкретным областям: кодировка изображений, генерация связного текста и объединение зрения и языка с помощью кросс-модального внимания. Помимо этого в пособии говорится о следующих современных областях применения трансформеров: большие языковые модели, генерация с внешней памятью и агентные системы, где внимание выступает в роли динамического контроллера над инструментами и памятью.

Практическая ориентированность и доступность для студента

Одним из ключевых принципов данного пособия является **доступность** для изучения студентом без необходимости организации доступа к современному оборудованию с высокими вычислительными возможностями, поэтому все задания разработаны так, чтобы запускаться на бесплатных облачных платформах, предоставляющих доступ к GPU:

- **Google Colab**: даёт бесплатный доступ к GPU (включая T4 и иногда лучше через Colab Pro) и обеспечивает простую интеграцию с GitHub и Google Drive. Блокноты можно сохранять, делиться ими и управлять ими.
- **Kaggle Notebooks**: предоставляет бесплатные часы работы с GPU, предустановленные библиотеки (PyTorch, TensorFlow) глубокого обучения и прямой доступ к наборам данных. Среда Kaggle обладает высокой стабильностью и идеально подходит для воспроизводимых экспериментов.

Локальная установка для выполнения упражнеий в рамках данного пособия не требуется. Достаточно стандартного веб-браузера и учётной записи Google или Kaggle. Все примеры кода используют PyTorch фреймворк, сочетающий гибкость и ясность, однако концепции легко переносятся и на другие библиотеки. Предполагается базовое знакомство с Python и основами глубокого обучения (тензоры, градиенты, обратное распространение ошибки), но предварительный опыт работы с трансформерами не обязателен.

Каждая лабораторная работа включает готовый блокнот с инструкциями, загрузчиками данных и базовыми функциями. Студенту предстоит

расширять эти шаблоны: реализовывать новые механизмы внимания, оптимизировать существующие или применять их к новым задачам. Акцент делается на чистом, модульном и эффективном коде, где важна не только корректность, но и *мастерство*.

Система оценки

В отличие от традиционных курсов, опирающихся на экзамены или теоретические доказательства, в рамках данного пособия предлагается использовать **практико-ориентированную, инженерную систему оценки**. Работа оценивается не только по факту работоспособности, но и по тому, насколько хорошо она работает на практике. Конкретно, каждое лабораторное задание оценивается по четырём критериям:

1. **Решение задачи (корректность и функциональность):** Корректно ли предложенная реализация решает поставленную задачу? Производит ли она точные выходные данные, сходится ли при обучении (если применимо) и соответствует ли техническому заданию?

2. **Качество кода (читаемость, структура и документирование).** Является ли предложенный код чистым, модульным и легко понятным? Имеют ли функции осмысленные имена, документированы ли параметры, и свободна ли логика от излишней сложности? Рекомендуется использовать аннотации типов, docstring'и и единый стиль оформления.

3. **Задержка и эффективность (скорость и потребление памяти).** Внимание – это не только точность, но и масштабируемость. Предложенная реализация должна быть протестирована по времени вывода и объёму видеопамяти при различных длинах последовательностей. Решение со сложностью $O(n^2)$ может подойти на коротких последовательностях, но провалится на длинных. От оптимизированных вариантов (например, линейного внимания) ожидается измеримый выигрыш, а инструменты профилирования являются неотъемлемой частью рабочего процесса.

4. **Результаты на соревновательном бенчмарке (минимальный порог на лидерборде).** Для повышения вовлечённости и реалистичности каждая лабораторная работа или группа работ включает лёгкое автоматически оцениваемое соревнование на частном лидерборде (симулируемом через Kaggle). Например, студенту может быть предложено классифицировать изображения с помощью ViT при ограничении памяти в 500 МБ или сгенерировать связный текст с декодером при задержке менее 50 мс на токен. Для зачёта требуется достичь *минимального порога производительности*, что отражает

реальные инженерные ограничения. Такой подход также стимулирует итеративное улучшение и творческую оптимизацию.

Такая многокритериальная система оценки отражает ожидания индустрии и научного сообщества: модель должна быть корректной, поддерживаемой, эффективной и конкурентоспособной.

План курса

Пособие состоит из шести основных глав (лабораторных работ), каждая из которых посвящена ключевой концепции внимания:

1. Self-Attention: основы, маскирование и реализация.
2. Cross-Attention: выравнивание, условная генерация и мультимодальное слияние.
3. Бутылочное горлышко механизма внимания: квадратичная сложность, коллапс внимания и ограничения контекста.
4. Линейное внимание: ядерные методы и эффективные аппроксимации.
5. Трансформеры для изображений: патч-эмбеддинги, позиционные кодировки и архитектура энкодера.
6. Трансформеры для NLP: каузальное маскирование, RoPE и динамика энкодер-декодер.
7. Заключение и перспективы развития.

Каждая глава включает теорию, пошаговый разбор кода, визуализации и оцениваемую лабораторную работу.

В эпоху, когда прогресс в искусственном интеллекте всё больше определяется архитектурной проницательностью и системным мышлением, освоение механизма внимания перестаёт быть опциональным, оно становится необходимым. Эта книга – ваша лаборатория. Начинайте экспериментировать.

2. Основные термины и обозначения

Для обеспечения единообразия изложения и корректного выполнения практических заданий в данном пособии используется унифицированный терминологический аппарат. Ниже приведены определения ключевых понятий, встречающихся в тексте лабораторных работ, а также таблица математических обозначений. Термины сгруппированы по областям применения для удобства восприятия информации. Понимание данных определений является необходимым условием для успешного освоения материалов курса.

2.1. Представление данных

В контексте архитектур трансформеров данные подвергаются специфической предобработке для приведения их к виду последовательностей векторов.

Токен (Token) – минимальная единица обработки текста или данных в модели трансформера. В задачах обработки естественного языка (NLP), рассматриваемых в лабораторных работах №6 и №7 (разделы 8 и 9), токеном может являться слово, часть слова (subword) или отдельный символ. В задачах компьютерного зрения (лабораторная работа №5, раздел 7) токеном выступает векторное представление патча изображения. *Контекст использования:* при подготовке данных входная последовательность разбивается на токены, каждому из которых присваивается уникальный индекс в словаре (vocabulary). Размер словаря определяет размерность выходного слоя модели.

Эмбеддинг (Embedding) – векторное представление токена в пространстве высокой размерности. В отличие от разреженного one-hot кодирования, эмбеддинги являются плотными векторами, несущими семантическую информацию о сходстве объектов. *Контекст использования:* в лабораторной работе №1 (раздел 3) токены преобразуются в эмбеддинги через слой `nn.Embedding` перед подачей в механизм внимания. В Vision Transformer (лабораторная работа №5, раздел 7) патчи изображений проецируются в эмбеддинги через линейный слой.

Патч (Patch) – прямоугольный фрагмент изображения фиксированного размера (например, 16×16 пикселей), используемый в архитектурах Vision Transformer. Изображение разбивается на сетку патчей, которые затем линейно проецируются в последовательность векторов для обработки трансформером, что позволяет применять архитектуру последовательностей к двумерным данным.

Позиционное кодирование (Positional Encoding) – вектор, добавляемый к эмбеддингу токена для передачи информации о его порядке в последовательности. Поскольку механизм внимания инвариантен к перестановкам входных данных, без позиционных кодирований модель не сможет различать порядок слов или патчей. *Контекст использования:* в пособии

рассматриваются как обучаемые позиционные эмбеддинги, так и детерминированные (синусоидальные, RoPE). В лабораторной работе №7 (раздел 9) используется RoPE для улучшения экстраполяции на длинные контексты.

2.2. Механизмы внимания

Механизм внимания является вычислительным ядром архитектуры трансформеров, обеспечивающим контекстную агрегацию информации.

Самовнимание (Self-Attention) – механизм, позволяющий каждому элементу последовательности взаимодействовать со всеми остальными элементами той же последовательности. Вычисляет контекстно-зависимые представления на основе сходства запросов и ключей. *Контекст использования:* базовый строительный блок трансформера, реализуемый в лабораторной работе №1, который обеспечивает моделирование глобальных зависимостей внутри входных данных.

Кросс-внимание (Cross-Attention) – механизм внимания, в котором запросы (Q) формируются из одной последовательности (например, декодера), а ключи (K) и значения (V) – из другой (например, энкодера или изображения). *Контекст использования:* используется для выравнивания модальностей в задачах машинного перевода и генерации подписей (лабораторные работы №2, №6, разделы 4 и 8). Позволяет декодеру «запрашивать» информацию у энкодера.

Запрос, Ключ, Значение (Query, Key, Value) – три фундаментальных векторных проекции входа в механизме внимания.

- **Запрос (Q):** представляет текущий элемент, который «ищет» информацию.
- **Ключ (K):** представляет элементы, которые могут быть «найдены».
- **Значение (V):** содержит фактическую информацию, которая агрегируется.

Контекст использования: скалярное произведение QK^T определяет веса внимания, которые применяются к V. Матрицы проекций W_Q, W_K, W_V являются обучаемыми параметрами.

Маскирование (Masking) – техника ограничения видимости токенов в механизме внимания путем добавления $-\infty$ к логитам перед функцией softmax.

- **Каузальная маска (Causal Mask):** запрещает токенам обращать внимание на будущие позиции (используется в декодерах).
- **Маска заполнения (Padding Mask):** игнорирует специальные токены `pad` при обработке последовательностей разной длины.

Контекст использования: критически важно для авторегрессионной генерации (лабораторная работа №7, раздел 9) и работы с батчами переменного размера.

2.3. Архитектурные компоненты

Трансформер (Transformer) – архитектура нейронной сети, основанная исключительно на механизмах внимания без использования рекуррентных или сверточных слоев. Состоит из стека блоков энкодера или декодера.

Энкодер (Encoder) – часть архитектуры трансформера, которая преобразует входную последовательность в контекстуализированное представление высокой размерности. Использует двунаправленное внимание, позволяя каждому токену видеть весь контекст. *Контекст использования:* модели типа BERT (лабораторная работа №6, раздел 8), визуальные энкодеры в мультимодальных системах.

Декодер (Decoder) – часть архитектуры, генерирующая выходную последовательность на основе представлений энкодера (или предыдущих токенов). Использует каузальное внимание и кросс-внимание. *Контекст использования:* модели типа GPT (лабораторная работа №7, раздел 9), генерация текста в мультимодальных системах.

Бутылочное горлышко внимания (Attention Bottleneck) – ограничение масштабируемости трансформеров, вызванное квадратичной сложностью $O(L^2)$ вычисления матрицы внимания относительно длины последовательности L. *Контекст использования:* подробно анализируется в лабораторной работе №3 (раздел 5), где рассматриваются проблемы памяти и вычислений, а также методы оптимизации.

2.4. Оптимизация и инференс

Контекстное окно (Context Window) – максимальное количество токенов, которое модель может обработать за один проход. Информация за пределами окна теряется, что приводит к проблемам долгосрочной памяти. *Контекст использования:* ограничивает длину документов для анализа (лабораторная работа №3, раздел 5). Современные методы (RoPE, ALiBi) пытаются расширить эффективное окно.

KV-кэш (KV-Cache) – техника оптимизации инференса, при которой спроецированные матрицы ключей и значений для предыдущих токенов сохраняются в памяти GPU, чтобы избежать их повторного вычисления на каждом шаге генерации. *Контекст использования:* позволяет ускорить авторегрессивную генерацию (лабораторные работы №3, №7, раздел 5 и 9. Снижает сложность генерации нового токена с $O(L^2)$ до $O(L)$.

Линейное внимание (Linear Attention) – аппроксимация механизма внимания, позволяющая снизить вычислительную сложность с $O(L^2)$ до $O(L)$ за счет использования трюков с ядером (kernel trick) или разреженных структур. *Контекст использования:* реализуется в лабораторной

работе №4, раздел 6 (EfficientViT, QTViT). Позволяет обрабатывать изображения высокого разрешения.

Температура (Temperature) – гиперпараметр τ, контролирующий случайность выборки токенов при инференсе. Низкие значения делают вывод детерминированным, высокие более разнообразным. *Контекст использования:* настройка стратегий декодирования (лабораторная работа №7, раздел 9). Влияет на энтропию выходного распределения.

2.5. Таблица математических обозначений

В данном пособии используются следующие обозначения для матриц, векторов и скаляров. Рекомендуется придерживаться этой нотации при выполнении лабораторных работ. Все векторы по умолчанию считаются столбцами, однако в матричных обозначениях PyTorch часто используется представление батчей в формате $(Batch, Seq\_Len, Features)$. В формулах пособия размерность батча часто опускается для краткости.

Таблица 1: Основные математические обозначения

Обозначение	**Описание**	**Размерность**
L	Длина входной последовательности (токенов)	Скаляр
d_{model}	Размерность пространства моделей (скрытый размер)	Скаляр
d_k, d_v	Размерность векторов ключей и значений	Скаляр
X	Входная матрица эмбеддингов	$L \times d_{model}$
Q, K, V	Матрицы запросов, ключей и значений	$L \times d_k$ / $L \times d_v$
W_Q, W_K, W_V	Обучаемые весовые матрицы проекций	$d_{model} \times d_k$
A	Матрица весов внимания (Attention Weights)	$L \times L$
O	Выходная матрица механизма внимания	$L \times d_v$
$\text{softmax}(\cdot)$	Функция нормализации вероятностей	Операция
$\mathcal{L}$	Функция потерь (Loss Function)	Скаляр
τ	Параметр температуры при выборке	Скаляр
$P(x_t \mid x_{<t})$	Условная вероятность токена	Вероятность
$\phi(\cdot)$	Функция отображения признаков (Kernel)	Вектор
S_{KV}, S_K	Агрегированные статистики в линейном внимании	Матрица / Вектор
M_{causal}	Каузальная маска внимания	$L \times L$
E	Матрица встраивания патчей (ViT)	$D \times (P^2C)$
Z	Последовательность скрытых состояний	$L \times D$

Данная система обозначений позволяет компактно записывать уравнения прямого и обратного прохода нейронной сети. При реализации кода на Python/PyTorch рекомендуется использовать имена переменных, максимально близкие к данным математическим обозначениям, для улучшения читаемости и соответствия теоретическому материалу.

3. Лабораторная работа №1. Самовнимание

3.1. Математическое определение: скалярное произведение, масштабирование, softmax

Прежде чем вводить механизм внимания в его современной нейросетевой форме, полезно проследить его концептуальные корни до классической обработки сигналов. Особенно показательным предшественником является **двусторонняя фильтрация** [22] – нелинейный метод сглаживания, сохраняющий краевые структуры за счёт присвоения весов соседним выборкам на основе как *пространственной близости*, так и *сходства признаков*. Для сигнала $\{x_i\}_{i=1}^n$ отфильтрованное значение в позиции i вычисляется как

$$y_i = \frac{\sum_{j=1}^n w_{ij}\, x_j}{\sum_{j=1}^n w_{ij}}, \quad \text{где} \quad w_{ij} = \exp\left(-\frac{\|p_i - p_j\|^2}{2\sigma_p^2} - \frac{\|x_i - x_j\|^2}{2\sigma_x^2}\right), \tag{1}$$

где p_i обозначает позицию (например, координату пикселя), а x_i – значение признака (например, интенсивность). Таким образом, вес w_{ij} кодирует *контекстно-зависимую* агрегацию: удалённые или несходные выборки вносят меньший вклад.

Этот принцип, *адаптивное взвешивание, основанное на сходстве*, составляет суть механизма внимания в глубоком обучении. Однако, вместо ручного задания фиксированных правил взаимодействия, необходимо учиться вычислять эти веса непосредственно из данных с помощью параметрических функций.

Определение 3.1 (Внимание как взвешенная агрегация)**.** Пусть $\mathbf{Q} \in \mathbb{R}^{n\times d_k}$, $\mathbf{K} \in \mathbb{R}^{m\times d_k}$ и $\mathbf{V} \in \mathbb{R}^{m\times d_v}$ обозначают наборы *запросов*, *ключей* и *значений* соответственно. Механизм внимания выдаёт взвешенную сумму значений, где веса определяются степенью совместимости между запросами и ключами:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{A}\mathbf{V}, \tag{2}$$

где $\mathbf{A} \in \mathbb{R}^{n\times m}$ – стохастическая матрица (то есть $\mathbf{A}\mathbf{1} = \mathbf{1}$, $A_{ij} \geq 0$), элементы которой измеряют релевантность. Произведение $\mathbf{A}\mathbf{V}$ реализует операцию *агрегации контекста*: i-я строка результата представляет собой выпуклую комбинацию строк матрицы $\mathbf{V}$, взвешенную в соответствии с тем, насколько сильно i-й запрос «внимает» к каждому из m ключей. Таким образом, механизм динамически собирает наиболее релевантную информацию из всего множества значений для формирования нового представления.

Стандартным выбором в архитектуре Transformer является **масштабированное скалярное внимание**, которое определяет матрицу внимания

$\mathbf{A}$ следующим образом:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\mathsf{T}}{\sqrt{d_k}}\right). \tag{3}$$

Здесь d_k обозначает размерность векторов ключей (и запросов), а деление на $\sqrt{d_k}$ служит для масштабирования скалярных произведений, предотвращая насыщение функции softmax при больших значениях размерности. Рассмотрим это выражение пошагово.

Во-первых, **скалярное произведение** $\mathbf{Q}\mathbf{K}^\mathsf{T}$ вычисляет попарные сходства между каждым запросом $\mathbf{q}_i$ и ключом $\mathbf{k}_j$:

$$[\mathbf{Q}\mathbf{K}^\mathsf{T}]_{ij} = \langle \mathbf{q}_i \, , \, \mathbf{k}_j \rangle = \mathbf{q}_i^\mathsf{T}\mathbf{k}_j. \tag{4}$$

Это скалярное произведение служит немасштабированной мерой согласованности: большие значения указывают на большую совместимость. Однако при больших d_k дисперсия $\langle \mathbf{q}_i \, , \, \mathbf{k}_j \rangle$ растёт пропорционально d_k (при предположении о нулевом среднем и единичной дисперсии входов). Это приводит к тому, что функция softmax оказывается в режиме насыщения, где градиенты исчезают, а обучение замедляется.

Ремарка 3.2 (Проблема масштабирования). Суть ремарки заключается в том, что если $\mathbf{q}_i, \mathbf{k}_j \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, то $\mathbb{E}[\langle \mathbf{q}_i \, , \, \mathbf{k}_j \rangle] = 0$ и $\text{Var}[\langle \mathbf{q}_i \, , \, \mathbf{k}_j \rangle] = d_k$. Без масштабирование логиты (сырые оценки сходства до применения функции нормализации) становятся всё более пиковыми по мере роста d_k, что приводит к почти одноэлементным распределениям внимания с пренебрежимо малым потоком градиентов.

Для устранения этого эффекта Васвани и др. [24] ввели **масштабирующий множитель** $1/\sqrt{d_k}$. Он нормализует дисперсию скалярных произведений, не давая им расти пропорционально размерности d_k, что удерживает аргументы функции активации в области, чувствительной к изменениям. Затем оператор **softmax** применяется по строкам к полученной матрице масштабированных оценок: он экспоненциально усиливает наибольшие значения сходства и подавляет малые, гарантированно преобразуя их в корректное вероятностное распределение (где все веса неотрицательны и суммируются в единицу). Это позволяет интерпретировать итоговые коэффициенты как вероятность выбора соответствующего значения при агрегации информации для данного запроса.

$$A_{ij} = \frac{\exp\left(\langle \mathbf{q}_i \, , \, \mathbf{k}_j \rangle / \sqrt{d_k}\right)}{\sum_{l=1}^{m} \exp\left(\langle \mathbf{q}_i \, , \, \mathbf{k}_l \rangle / \sqrt{d_k}\right)}. \tag{5}$$

Это гарантирует, что $\sum_j A_{ij} = 1$ и $A_{ij} \in (0, 1)$, делая выход внимания выпуклой комбинацией векторов значений.

Не смотря на то, что механизм внимания обеспечивает мощный инструмент для контекстно-зависимой агрегации и взвешивания, аналогичного используемому в билатеральной фильтрации, он по своей природе **инвариантен к перестановке**: перестановка входных элементов не изменяет выходного результата. Это принципиально отличается от билатеральной фильтрации, где пространственные координаты p_i явно кодируют положение и напрямую влияют на веса w_{ij}. В последовательностях, таких как текст или временные ряды, информация о позиции не менее важна: без неё модель не может различить, например, фразы «кот гнался за собакой» и «собака гналась за котом». Для восстановления чувствительности к порядку в архитектурах трансформеров используются **позиционные кодировки** вспомогательные сигналы, внедряемые в представления токенов с целью передачи их позиции в последовательности.

Ранние подходы, такие как **фиксированная синусоидальная кодировка**, предложенная Васвани и соавт. [24], применяют детерминированные синусоидальные функции различных частот для встраивания абсолютных позиций, что позволяет модели извлекать относительные смещения посредством линейных комбинаций. Альтернативно, **обучаемые позиционные эмбеддинги** рассматривают позиции как обучаемые параметры, обеспечивая гибкость, но ограничивая обобщающую способность за пределами максимальной длины последовательности, наблюдаемой в ходе обучения. Более современные методы, в частности **вращательные позиционные кодировки (Rotary Position Embeddings, RoPE)**, встраивают позиционную информацию непосредственно в вычисление внимания с помощью матриц поворота, сохраняя относительные позиционные зависимости таким образом, который естественно согласуется со структурой скалярного произведения в механизме внимания.

Подобно тому, как билатеральная фильтрация объединяет геометрические (пространственные) и фотометрические (признаковые) сигналы в единый вес, эффективная позиционная кодировка интегрирует последовательную структуру с семантическим содержанием, обеспечивая, чтобы механизм внимания оставался одновременно **контекстно-зависимым** и **чувствительным к порядку**.

Таким образом, масштабированное скалярное внимание реализует *обучаемый*, *дифференцируемый* и *контекстно-чувствительный* механизм агрегации, обобщающий идеи двусторонней фильтрации на многомерные, сквозные представления. Его простота, параллелизуемость и выразительность сделали его основой современных архитектур.

В последующих разделах мы реализуем этот механизм с нуля, проанализируем его вычислительную сложность и исследуем его поведение с помощью визуализаций и аблационных исследований.

3.2. Свойства: перестановочная инвариантность, контекстная зависимость, параллелизм

Механизм внимания обладает рядом фундаментальных математических и вычислительных свойств, лежащих в основе его успеха в глубоком обучении. Ниже мы формализуем три ключевые характеристики поведения при перестановках, контекстную адаптивность и вычислительный параллелизм и обсудим два дополнительных структурных свойства: стохастичность строк матрицы внимания и неявную регуляризацию, вносимую оператором softmax.

Определение 3.3 (Стохастичность строк)**.** Матрица весов внимания $\mathbf{A} = \mathrm{softmax}\left(\mathbf{Q}\mathbf{K}^\mathsf{T}/\sqrt{d_k}\right) \in \mathbb{R}^{n\times m}$ является *стохастической по строкам*, то есть

$$\mathbf{A}\mathbf{1}_m = \mathbf{1}_n, \quad \text{и} \quad A_{ij} \geq 0 \ \ \forall\, i, j. \tag{6}$$

Это следует непосредственно из определения функции softmax, которая нормализует каждую строку логитов в вероятностное распределение по m ключам. Следовательно, выход $\mathbf{AV}$ представляет собой выпуклую комбинацию векторов значений для каждого запроса, обеспечивая гладкую интерполяцию вместо произвольного линейного смешивания.

Поведение при перестановках. Механизм внимания является *эквивариантным* относительно перестановки входной последовательности, но лишь при одновременной перестановке запросов, ключей и значений. А именно, пусть $\mathbf{P} \in \{0,1\}^{n\times n}$ – матрица перестановки. Тогда:

$$\mathrm{Attention}(\mathbf{PQ}, \mathbf{PK}, \mathbf{PV}) = \mathbf{P}\cdot \mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}). \tag{7}$$

Это означает, что при перестановке входных токенов выходные токены переставляются тем же образом, сохраняя реляционную структуру и игнорируя абсолютные позиции (если только не добавлены позиционные кодировки). Важно отметить, что внимание *не* является перестановочно *инвариантным* (что подразумевало бы одинаковый выход независимо от порядка); скорее, оно *эквивариантно*, что делает его подходящим для моделирования последовательностей, где порядок важен, но без рекуррентного смещения.

Контекстная зависимость. В отличие от фиксированных свёрток или статического пулинга, внимание вычисляет *динамические* веса, зависящие от всего контекста. Представление токена i является функцией *всех* остальных токенов (через ключи и значения), взвешенных по их релевантности запросу $\mathbf{q}_i$. Это позволяет осуществлять взаимодействия на больших расстояниях, основанные на содержании: два семантически связанных слова

(например, «нейронные» и «сети») могут обращаться друг к другу даже при наличии сотен промежуточных токенов. Именно эта контекстнозависимая агрегация является ключевой причиной выразительности механизма трансформеров при моделировании композиционной семантики.

Параллелизм. Поскольку внимание реализуется через матричные умножения $\mathbf{QK}^\mathsf{T}$, softmax и $\mathbf{AV}$, все попарные взаимодействия вычисляются *одновременно*. Это резко контрастирует с рекуррентными моделями (например, LSTM), которые обрабатывают токены последовательно и не могут эффективно использовать параллелизм современных GPU. Квадратичная по n стоимость памяти и вычислений ($O(n^2)$) является платой за такой массовый параллелизм, обеспечивающий ускоренное обучение на крупномасштабном оборудовании.

Ремарка 3.4 (Softmax вызывает неявную ℓ_2-нормализацию). Использование скалярного произведения $\langle \mathbf{q}_i \, , \, \mathbf{k}_j \rangle$ внутри softmax оказывает тонкий регуляризующий эффект. Рассмотрим градиент выхода внимания по запросу $\mathbf{q}_i$:

$$\frac{\partial (\mathbf{AV})_{i,:}}{\partial \mathbf{q}_i} \approx \frac{1}{\sqrt{d_k}} \sum_{j=1}^{m} A_{ij}(1 - A_{ij})(\mathbf{v}_j - \bar{\mathbf{v}}_i)\mathbf{k}_j^\mathsf{T}, \tag{8}$$

где $\bar{\mathbf{v}}_i = \sum_l A_{il}\mathbf{v}_l$. Член $A_{ij}(1 - A_{ij})$ максимален при $A_{ij} = 0.5$ и стремится к нулю при $A_{ij} \to 0$ или 1, подавляя градиентный поток при чрезмерной уверенности. Примечание: данная аппроксимация возникает при диагональном приближении якобиана softmax и игнорировании межтокенных корреляций.

Логиты имеют вид $\langle \mathbf{q}_i \, , \, \mathbf{k}_j \rangle / \sqrt{d_k} = \|\mathbf{q}_i\|_2 \, \|\mathbf{k}_j\|_2 \cos\theta_{ij} / \sqrt{d_k}$. При фиксированном угловом сходстве $\cos\theta_{ij}$ увеличение ℓ_2-норм запросов или ключей усиливает логиты, смещая softmax к одноэлементному распределению. Однако в процессе обучения это приводит к исчезающим градиентам (как отмечалось выше). Следовательно, оптимизационный процесс неявно препятствует большим ℓ_2-нормам в $\mathbf{Q}$ и $\mathbf{K}$, действуя как форма *мягкой ℓ_2-регуляризации*. Этот эффект дополняет явные методы, такие как нормализация по слоям и затухание весов.

В заключение можно отметить, что внимание сочетает в себе *структурированную интерполяцию* (благодаря стохастичности строк), *гибкость с учётом порядка* (через эквивариантность), *моделирование глобального контекста* и *параллелизм, дружественный к оборудованию*, при этом неявно регуляризуя представления через геометрию softmax. Эти свойства делают его уникально подходящим универсальным модулем для моделирования последовательностей в различных модальностях.

3.3. Маскирование causal mask, padding mask, их влияние на обучение

При практическом моделировании последовательностей не все попарные взаимодействия между токенами являются допустимыми или желательными. Для обеспечения структурных ограничений, таких как временной порядок при генерации или входы переменной длины, механизмы внимания используют *маскирование*. Маска выборочно отключает определённые взаимодействия «запрос–ключ», присваивая им логиты $-\infty$, которые функция softmax преобразует в нулевой вес. Ниже рассматриваются две канонические маски: *причинно-следственная (авторегрессионная) маска* и *маска заполнения*, а также анализируются их алгоритмические и теоретико-обучающие последствия.

Формально, пусть дана матрица логитов $\mathbf{L} = \mathbf{Q}\mathbf{K}^\mathsf{T}/\sqrt{d_k} \in \mathbb{R}^{n\times m}$. Тогда маска $\mathbf{M} \in \{0, -\infty\}^{n\times m}$ применяется до softmax:

$$\mathbf{A} = \mathrm{softmax}\left(\mathbf{L} + \mathbf{M}\right). \tag{9}$$

Поскольку $\exp(-\infty) = 0$, любой элемент с $M_{ij} = -\infty$ не вносит вклада в выход внимания.

Причинно-следственная маска (авторегрессионная маска). Используется в архитектурах только с декодером или с кодером-декодером для генерации последовательностей. Причинно-следственная маска обеспечивает *временную причинность*: токен i может обращаться только к токенам $j \le i$. Это предотвращает утечку информации из будущих токенов на этапе обучения, гарантируя, что модель учится предсказывать следующий токен исключительно на основе прошлого контекста. Маска имеет верхнетреугольную структуру:

$$M_{ij}^{\text{causal}} = \begin{cases} 0 & \text{если } j \le i, \\ -\infty & \text{если } j > i. \end{cases} \tag{10}$$

Для входной последовательности длины n это приводит к нижнетреугольной матрице внимания $\mathbf{A}$, что приводит к следующим особенностям обучения:

- Модель не может «списывать», заглядывая в будущие эталонные токены, что согласует обучение с инференсом (где будущие токены недоступны).
- Градиенты распространяются только по допустимым зависимостям, обеспечивая строгое распределение «кредита» слева направо.
- Вычисление функции потерь использует все позиции одновременно (принуждение учителя), однако каждое предсказание остаётся условно независимым при заданном прошлом.

Без причинно-следственного маскирования языковые модели переобучались бы на тривиальных корреляциях и не смогли бы выполнять авторегрессионную генерацию.

Маска заполнения. В реальных батчах встречаются последовательности разной длины, дополненные до общей максимальной длины L специальным токеном (например, `<pad>`). Чтобы предотвратить обращение модели к этим искусственным токенам, маска заполнения устанавливает $M_{ij} = -\infty$, если ключ j соответствует заполнению:

$$M_{ij}^{\text{pad}} = \begin{cases} 0 & \text{если токен } j \text{ не является заполнением,} \\ -\infty & \text{если токен } j \text{ заполнение.} \end{cases} \tag{11}$$

Эта маска применяется одинаково ко всем запросам в последовательности. Её эффекты включают:

- Сохранение осмысленного контекста: веса внимания перенормируются только по валидным токенам.
- Стабильное обучение: токены заполнения, не несущие семантической информации, не искажают агрегацию значений.
- Корректное вычисление потерь: позиции заполнения обычно исключаются из функции потерь с помощью дополнительной целевой маски.

Ремарка 3.5 (Комбинированные маски). В внимании кодера-декодера (например, при машинном переводе) запрос поступает из декодера (с причинной маской), а ключи из кодера (с маской заполнения). Две маски логически комбинируются следующим образом:

$$\mathbf{M} = M^{\text{causal}} \oplus M^{\text{pad}}, \tag{12}$$

где $\oplus$ обозначает транслированное сложение (на практике обе маски прибавляются к матрице логитов). Это гарантирует, что модель обращается только к *валидным токенам кодера* и только *до текущего шага декодера.*

Ремарка 3.6 (Влияние на динамику градиентов). Маскирование изменяет носитель распределения внимания, что, в свою очередь, влияет на величины градиентов. В областях заполнения градиенты по ключам и значениям токенов заполнения точно равны нулю, предотвращая обновления параметров для нерелевантных эмбеддингов. Аналогично, причинное маскирование ограничивает обратное распространение только прошлыми моментами времени, имитируя индуктивное смещение рекуррентных сетей, но с полным параллелизмом по допустимому контексту.

Таким образом, маскирование – это не просто техническая деталь реализации, а *фундаментальное архитектурное ограничение*, определяющее, чему может научиться модель. Причинное маскирование обеспечивает связную генерацию; маскирование заполнения гарантирует устойчивость к входам переменной длины. Вместе они позволяют трансформерам эффективно работать в реальных условиях, сохраняя согласованность между целями обучения и условиями развёртывания.

3.4. Реализация с нуля: forward pass, градиенты, эффективные трюки (flash attention)

Реализация механизма масштабированного скалярного внимания с нуля – важное упражнение для понимания его вычислительной структуры, объёма занимаемой памяти и дифференцируемости. Начнём с базового прямого прохода, выведем градиенты, а затем обсудим современные оптимизации, в первую очередь *FlashAttention*.

Прямой проход

Пусть заданы матрицы запросов $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$, ключей $\mathbf{K} \in \mathbb{R}^{m \times d_k}$ и значений $\mathbf{V} \in \mathbb{R}^{m \times d_v}$. Стандартное внимание вычисляется как

$$\mathbf{A} = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right), \quad \mathbf{O} = \mathbf{A}\mathbf{V}. \tag{13}$$

Здесь $\mathbf{O} \in \mathbb{R}^{n \times d_v}$ представляет собой итоговую выходную матрицу. Это можно реализовать всего в несколько строк кода, напоминающего PyTorch.

Прямой проход: базовое внимание

```
def scaled_dot_product_attention(Q, K, V, mask=None):
    """
    Q: (n, dk), K: (m, dk), V: (m, dv)
    """
    dk = Q.size(-1)
    logits = Q @ K.t() / (dk ** 0.5)     # (n, m)
    if mask is not None:
        logits = logits.masked_fill(mask == 0, float('-inf'))
    A = logits.softmax(dim=-1)           # (n, m)
    O = A @ V                            # (n, dv)
    return O, A
```

Эта реализация проста, читаема и полностью дифференцируема, однако она материализует полную матрицу внимания $\mathbf{A}$ размера $n \times m$, что становится неприемлемым для длинных последовательностей.

Вычисление градиентов

Для проверки корректности и возможности ручной оптимизации выведем градиенты по $\mathbf{Q}$, $\mathbf{K}$ и $\mathbf{V}$. Пусть $\mathbf{O} = \mathbf{AV}$, а $\mathbf{G} = \partial\mathcal{L}/\partial\mathbf{O} \in \mathbb{R}^{n\times d_v}$ восходящий градиент.

Во-первых, $\frac{\partial\mathcal{L}}{\partial\mathbf{V}} = \mathbf{A}^\top\mathbf{G}$.

Для $\mathbf{A}$, поскольку $\mathbf{A} = \mathrm{softmax}(\mathbf{L})$, где $\mathbf{L} = \mathbf{QK}^\top/\sqrt{d_k}$, градиент softmax даёт

$$\frac{\partial\mathcal{L}}{\partial\mathbf{L}} = \mathbf{A}\odot\left(\mathbf{GV}^\top - \mathrm{diag}\left((\mathbf{A}\odot(\mathbf{GV}^\top))\mathbf{1}\right)\right), \tag{14}$$

где $\odot$ обозначает поэлементное умножение. На практике это упрощается до

$$\mathbf{D} = \mathbf{GV}^\top, \quad \mathbf{D}_{centered} = \mathbf{D} - \mathrm{mean}_{\mathrm{row}}(\mathbf{D}) \quad \text{(неявно в autograd)}, \tag{15}$$

и далее

$$\frac{\partial\mathcal{L}}{\partial\mathbf{Q}} = \frac{1}{\sqrt{d_k}}\left(\frac{\partial\mathcal{L}}{\partial\mathbf{L}}\right)\mathbf{K}, \quad \frac{\partial\mathcal{L}}{\partial\mathbf{K}} = \frac{1}{\sqrt{d_k}}\left(\frac{\partial\mathcal{L}}{\partial\mathbf{L}}\right)^\top\mathbf{Q}. \tag{16}$$

Эти градиенты подтверждают, что внимание является гладким и хорошо обусловленным при условии, что $\mathbf{A}$ не насыщено (отсюда важность масштабирования).

Проблемы эффективности и FlashAttention

Основная проблема наивного внимания – это потребление $O(nm)$ памяти и вычислительных ресурсов для хранения и обработки $\mathbf{A}$. Например, при $n = m = 10^4$ требуется 400 МБ (в формате float32), что быстро исчерпывает видеопамять GPU.

FlashAttention решает эту проблему следующим образом:

- Избегает материализации $\mathbf{A}$,
- Использует тайлинг и повторное вычисление, чтобы удерживать все операции в быстрой SRAM (внутричиповой памяти),
- Эффективно использует иерархию памяти GPU для оптимальной пропускной способности.

Хотя полная реализация на CUDA выходит за рамки данного текста, основная идея заключается в блочном вычислении $\mathbf{O}$:

$$\mathbf{O}_i = \sum_j \mathrm{softmax}\left(\frac{\mathbf{Q}_i\mathbf{K}_j^\top}{\sqrt{d_k}}\right)\mathbf{V}_j, \tag{17}$$

с одновременной фьюжн-нормализацией softmax по тайлам с использованием трюка *онлайн-softmax*.

На практике следует использовать оптимизированные ядра:

Эффективное внимание на практике

```
# PyTorch 2.0
from torch.nn.functional import scaled_dot_product_attention

O = scaled_dot_product_attention(Q, K, V, attn_mask=mask)

# Или используйте flash-attn (сторонняя библиотека)
from flash_attn import flash_attn_func
O = flash_attn_func(Q, K, V, causal=True)
```

Эти реализации не только ускоряют вычисления в 2–5 раз, но и снижают потребление памяти до 10 раз, позволяя обрабатывать контексты длиной 8K–32K токенов на одном GPU.

Когда стоит реализовывать с нуля?

Написание собственного внимания полезно для:

- исследований (например, пользовательские ядра внимания, разреженные паттерны),
- отладки (визуализация $\mathbf{A}$, тестирование масок),
- обучения (понимание потока градиентов).

Однако в продакшене всегда предпочтительнее использовать фьюжн-ядра с эффективным расходованием памяти, такие как FlashAttention или встроенная в PyTorch SDPA.

Далее в лабораторной работе вы реализуете базовую версию, профилируете её задержку и потребление памяти и сравните с оптимизированными альтернативами, измерив реальную стоимость узкого места внимания.

3.5. Эксперимент: измерение времени и VRAM в зависимости от длины последовательности

Теоретические вопросы

1. Покажите, что выход внимания $\mathbf{O} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ является выпуклой комбинацией векторов значений. Почему стохастичность строк матрицы внимания существенна для стабильного обучения?

2. Объясните, почему скалярное произведение $\mathbf{q}_i^\top \mathbf{k}_j$ имеет дисперсию, пропорциональную d_k, при изотропных входах. Как масштабирование на $1/\sqrt{d_k}$ устраняет проблему исчезающих градиентов?

3. Докажите, что внимание является перестановочно *эквивариантным*, но не перестановочно *инвариантным*. Что изменится, если убрать позиционные кодировки?

4. Опишите, как причинно-следственное маскирование обеспечивает согласованность между обучением и инференсом в авторегрессионных моделях. Что произойдёт, если случайно использовать немаскированное (не причинное) внимание при обучении языковой модели?

5. Почему применение softmax к большим логитам приводит к почти одноэлементным весам внимания? Как это связано с неявной ℓ_2-регуляризацией запросов и ключей?

Задачи по реализации и профилированию

6. **Реализация базового внимания**. Реализуйте функцию `scaled_dot_product_attention(Q, K, V, mask=None)` с нуля, используя только тензорные операции PyTorch (без `nn.MultiheadAttention`). Обеспечьте поддержку причинной и маски заполнения.

7. **Профилирование памяти и задержки**. Для длин последовательностей $n \in \{128, 256, 512, 1024, 2048\}$ измерьте:

   - Задержку прямого прохода (усреднённую по 50 запускам),
   - Пиковое потребление видеопамяти GPU (с помощью `torch.cuda.max_memory_allocated()`),
   - Объём памяти, занимаемый матрицей внимания $\mathbf{A}$ (теоретически и эмпирически).

   Постройте графики зависимости задержки и объёма VRAM от n. Убедитесь, что потребление памяти растёт как $O(n^2)$.

8. **Сравнение эффективности**. Сравните вашу реализацию со встроенной реализацией PyTorch (или FlashAttention). Приведите ускорение и снижение потребления памяти при $n = 2048$.

9. **MNIST как задача моделирования последовательностей**. Рассматривайте каждое изображение MNIST размером 28×28 как последовательность из 784 пикселей (развёрнутых построчно). Спроектируйте компактный энкодер на основе Transformer:

- Преобразуйте каждый пиксель с помощью обучаемого эмбеддинга размерности $d = 64$,
- Добавьте позиционные кодировки (обучаемые или синусоидальные),
- Примените 1–2 слоя самовнимания (без полносвязного блока для упрощения),
- Выполните пулинг выхода (например, усреднение или через специальный токен `[CLS]`) и классифицируйте на 10 классов.

Обучите модель в течение 10 эпох на MNIST (с разбиением на обучающую и валидационную выборки).

Требования к оформлению

- Один Colab/Kaggle Notebook с ячейками, чётко помеченными номерами заданий.
- Все графики должны быть включены непосредственно в ноутбук с подписями осей и легендами.
- Код должен быть модульным, снабжён комментариями и не содержать избыточных вычислений.
- Модель MNIST должна обучаться с нуля (без предобученных весов).
- Включите краткое описание (не более 150 слов) практических ограничений квадратичного внимания в реальных приложениях.

Критерии оценки. Для успешного выполнения лабораторной работы необходимо: (1) корректно ответить на все теоретические вопросы, (2) достичь точности не ниже 80% на MNIST и (3) экспериментально подтвердить квадратичную ($O(n^2)$) зависимость потребления памяти от длины последовательности при профилировании.

4. Лабораторная работа №2. Кросс-внимание (Cross-Attention)

4.1. Математическая формулировка: запросы из одного источника, ключи/значения

В отличие от *self-attention*, где запросы (queries), ключи (keys) и значения (values) формируются из одной и той же входной последовательности, в **cross-attention** их источники разделяются: *запросы* поступают из одной модальности или вычислительного потока (например, декодера), тогда как *ключи* и *значения* предоставляются другим (например, энкодером). Эта асимметрия позволяет модели динамически выравнивать и извлекать релевантную информацию из различных представлений. Эта способность лежит в основе задач условного моделирования последовательностей, таких как машинный перевод, генерация описаний к изображениям или преобразование речи в текст.

Формально пусть

$$\mathbf{Q} \in \mathbb{R}^{n \times d_k}, \quad \mathbf{K} \in \mathbb{R}^{m \times d_k}, \quad \mathbf{V} \in \mathbb{R}^{m \times d_v} \tag{18}$$

обозначают матрицы запросов, ключей и значений, где:

- $\mathbf{Q} = f_{\text{query}}(\mathbf{X})$ вычисляется по *целевой* последовательности размера n $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ (например, частично сгенерированному выходу),
- $\mathbf{K} = f_{\text{key}}(\mathbf{Y})$ и $\mathbf{V} = f_{\text{value}}(\mathbf{Y})$ вычисляются по *исходной* последовательности $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_m]$ (например, закодированному входу).

Тогда выход cross-attention определяется следующим образом:

$$\text{CrossAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}. \tag{19}$$

Каждая строка результирующей матрицы представляет собой контекстно-зависимое свёрнутое представление исходной последовательности $\mathbf{Y}$, адаптированное под конкретный вектор запроса из $\mathbf{X}$. По сути, i-й выходной токен интегрирует информацию от всех исходных токенов, взвешенную по их релевантности относительно $\mathbf{q}_i$.

Эта формулировка сохраняет основные компоненты масштабированного скалярного произведения сходство через скалярное произведение, нормировку на $\sqrt{d_k}$ и softmax-нормализацию, но переосмысливает их как операцию *поиска* или *выравнивания* между двумя различными последовательностями. Следует отметить следующие особенности:

- Матрица внимания $\mathbf{A} \in \mathbb{R}^{n \times m}$, как правило, *не является квадратной* ($n \neq m$ типичная ситуация).

- Механизм принципиально *асимметричен*:
  $\text{CrossAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \neq \text{CrossAttention}(\mathbf{K}, \mathbf{Q}, \mathbf{V})$.
- Он обеспечивает *глобальное условие*: каждая позиция целевой последовательности имеет прямой доступ ко всему контексту исходной последовательности.

В архитектурах типа «энкодер–декодер» cross-attention служит основным мостом между этапами кодирования и декодирования, позволяя декодеру «обращать внимание» на наиболее релевантные части закодированного входа при генерации каждого выходного токена. Такой подход заменяет рекуррентные или свёрточные механизмы выравнивания полностью дифференцируемой, параллелизуемой и основанной на содержании альтернативой, заложившей основу современного моделирования последовательностей «многие-ко-многим».

4.2. Реализация и отличия от self-attention (разные проекционные головы)

Хотя self-attention и cross-attention используют одинаковое математическое ядро, масштабированное скалярное произведение с последующей агрегацией с весами softmax, их реализации различаются способом формирования векторов запросов, ключей и значений. Это различие обусловлено тем, что в cross-attention запросы и пары (ключ, значение) происходят из *разных входных последовательностей* и, следовательно, требуют *независимых обучаемых проекций*.

В стандартном блоке Transformer self-attention вычисляет все три компонента из единого входного тензора $\mathbf{X} \in \mathbb{R}^{n \times d_{\text{model}}}$ с использованием общих или отдельных линейных проекций:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V, \tag{20}$$

где $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ (или d_v) обучаемые весовые матрицы.

Напротив, cross-attention работает с *двумя разными входами*: *целевым* представлением $\mathbf{X} \in \mathbb{R}^{n \times d_{\text{model}}}$ (например, состояниями декодера) и *исходным* представлением $\mathbf{Y} \in \mathbb{R}^{m \times d_{\text{model}}}$ (например, выходами энкодера). Соответственно, он использует *отдельные проекционные подпространства*:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q_{\text{dec}}, \quad \mathbf{K} = \mathbf{Y}\mathbf{W}^K_{\text{enc}}, \quad \mathbf{V} = \mathbf{Y}\mathbf{W}^V_{\text{enc}}, \tag{21}$$

где индексы подчёркивают, что:

- $\mathbf{W}^Q_{\text{dec}}$ принадлежит стеку декодера,
- $\mathbf{W}^K_{\text{enc}}$ и $\mathbf{W}^V_{\text{enc}}$ обычно совпадают с матрицами, используемыми в self-attention энкодера (либо представляют собой отдельную копию, если веса не связаны).

Такое архитектурное разделение гарантирует, что:

1. Пространство запросов оптимизировано для определения *того, что искать* (на основе контекста декодера),

2. Пространство ключей/значений оптимизировано для представления *того, что доступно для извлечения* (на основе семантики энкодера).

С практической точки зрения, прямой проход (forward pass) cross-attention почти идентичен self-attention отличается лишь источник матриц $\mathbf{K}$ и $\mathbf{V}$:

Прямой проход Cross-Attention

```
def cross_attention(Q_from_decoder, K_from_encoder,
                    V_from_encoder  , mask=None):
    """
    Q_from_dec: (n, dk)  ← из состояний декодера
    K_from_enc: (m, dk)  ← из выхода энкодера
    V_from_enc: (m, dv)
    """
    dk = Q_from_decoder.size(-1)
    logits = Q_from_dec @ K_from_enc.t() / (dk ** 0.5) # (n, m)
    if mask is not None:
        logits = logits.masked_fill(mask == 0, float('-inf'))
    A = logits.softmax(dim=-1)  # (n, m)
    O = A @ V_from_encoder     # (n, dv)
    return O, A
```

Основные практические различия включают:

- **Отсутствие саморекуррентности**. В отличие от self-attention, cross-attention не моделирует внутренние зависимости внутри последовательности запросов. Он моделирует только зависимости от запросов к значениям источника.

- **Правила маскирования**. В cross-attention декодера маска, как правило, представляет собой *маску заполнения* (padding mask) над входом энкодера (чтобы игнорировать заполненные токены источника). Каузальное маскирование здесь, как правило, не применяется, оно используется только в подслое self-attention декодера.

- **Независимость параметров**. Матрицы проекций для запросов (со стороны декодера) и ключей/значений (со стороны энкодера) обучаются независимо, что позволяет модели формировать специализированные семантические пространства для поиска и представления.

Наконец, отметим, что если self-attention используется как в энкодерах, так и в декодерах (для моделирования внутренней структуры последовательности), то cross-attention появляется только в слоях декодера (или в асимметричных мультимодальных архитектурах, таких как CLIP или Flamingo), где он обеспечивает взаимодействие между различными модальностями или этапами обработки. Такая модульность self-attention для внутренней согласованности, cross-attention для внешней привязки является краеугольным камнем универсальности архитектуры трансформеров.

4.3. Примеры использования: машинный перевод, генерация описаний к изображениям, генерация с внешней памятью

Cross-attention является архитектурной основой, позволяющей трансформерам моделировать взаимодействия между *разнородными* или *асимметричными* последовательностями. Его способность позволять одному потоку («декодеру») динамически запрашивать информацию у другого («энкодера») делает его незаменимым в широком спектре задач условной генерации и мультимодального выравнивания. Ниже рассматриваются три канонических применения, демонстрирующих его универсальность.

Машинный перевод. В нейросетевом машинном переводе (NMT) цель состоит в отображении исходного предложения $\mathbf{Y} = [y_1, \ldots, y_m]$ на одном языке в целевое предложение $\mathbf{X} = [x_1, \ldots, x_n]$ на другом. Трансформер с архитектурой «энкодер–декодер» использует cross-attention, чтобы каждый целевой токен x_i мог обращаться ко всем исходным токенам y_j. Конкретно, на этапе декодирования:

$$\mathbf{Q} = \mathrm{DecoderLayer}_\ell(\mathbf{X}_{<i})\mathbf{W}^Q, \quad \mathbf{K}, \mathbf{V} = \mathrm{Encoder}(\mathbf{Y})\mathbf{W}^{K/V}, \tag{22}$$

поэтому веса внимания A_{ij} отражают, насколько предсказание x_i должно зависеть от y_j. Этот механизм эффективно заменяет мягкие модели выравнивания из ранних RNN с вниманием, но обеспечивает полный параллелизм и глобальный контекст. Ключевым преимуществом является то, что cross-attention позволяет разрешить неоднозначность: например, перевод слова «bank» может быть корректно интерпретирован как *финансовое учреждение* или *берег реки* на основе всего исходного контекста, а не только локальных подсказок.

Генерация описаний к изображениям. В задачах компьютерного зрения и обработки естественного языка, таких как генерация подписей к изображениям, cross-attention связывает модальности: «энкодером» выступает визуальная модель (например, ViT или CNN), формирующая пространственные признаковые карты $\mathbf{Y} \in \mathbb{R}^{m \times d}$, а «декодером» – языковая модель, генерирующая словесные токены $\mathbf{X}$. Здесь cross-attention позволяет

каждому словесному запросу $\mathbf{q}_i$ фокусироваться на релевантных областях изображения:

$$\text{Caption}_i \leftarrow \text{CrossAttention}\big(\mathbf{q}_i,\ \{\mathbf{k}_{\text{patch}_j}, \mathbf{v}_{\text{patch}_j}\}_{j=1}^{m}\big). \tag{23}$$

Например, при генерации слова «собака» модель может присвоить высокие веса внимания патчам, содержащим собаку, игнорируя фоновые области. Современные архитектуры, такие как BLIP или GIT, используют несколько слоёв cross-attention для итеративного уточнения выравнивания между визуальной и языковой информацией.

Генерация с внешней памятью (Retrieval-Augmented Generation, RAG). RAG расширяет языковые модели внешними знаниями, извлекая релевантные документы $\{\mathbf{d}^{(1)}, \ldots, \mathbf{d}^{(K)}\}$ из крупного корпуса и условно генерируя ответ на их основе. Cross-attention позволяет генератору обращаться к извлечённым фрагментам:

$$\mathbf{Y} = \big[\text{Encode}(\mathbf{d}^{(1)}); \ldots; \text{Encode}(\mathbf{d}^{(K)})\big] \in \mathbb{R}^{m \times d}, \tag{24}$$

а декодер использует cross-attention для извлечения свидетельств из $\mathbf{Y}$ при генерации токенов. Это особенно эффективно для обеспечения фактологической точности и привязки к реальным данным: при ответе на вопрос «Когда родилась Мария Кюри?» модель может сфокусироваться на извлечённом предложении вида «Мария Кюри (1867–1934) была физиком. . . » и извлечь правильную дату. В отличие от дообучения, RAG обновляет знания без повторного обучения cross-attention выступает в роли дифференцируемого интерфейса к внешней памяти.

Во всех этих случаях cross-attention функционирует как *адресуемая по содержимому головка чтения*: запрос кодирует потребность («что мне сказать дальше?»), а ключ-значение память предоставляет релевантные свидетельства («вот что важно в источнике»). Такое разделение *рассуждения* (декодер) и *памяти/представления* (энкодер) лежит в основе успеха современных мультимодальных и знаниеёмких ИИ-систем.

4.4. Эксперимент: сравнение cross- и self-внимание для seq2seq

Опираясь на концепции, рассмотренные в лабораторной работе №1 (self-attention), и формулировку cross-attention, представленную в данной главе, в этом эксперименте предлагается эмпирически исследовать, как тип механизма внимания влияет на производительность и поведение в простой задаче преобразования последовательности в последовательность (seq2seq). Предлагается реализовать две минимальные архитектуры, одну с использованием только self-attention, другую с cross-attention и сравнить их динамику обучения, паттерны внимания и итоговую точность на синтетической мультимодальной задаче.

Теоретические и аналитические вопросы

1. Объясните, почему чистый декодер на основе self-attention (без cross-attention) не может решить стандартную задачу seq2seq, такую как машинный перевод. Какой информации ему не хватает?

2. Предположим, вы заменяете cross-attention простой конкатенацией эмбеддингов энкодера и декодера с последующим применением self-attention. Какое фундаментальное ограничение это вносит в доступ к контексту?

3. Опишите, чем отличается форма матрицы внимания в self-attention (в декодере) и в cross-attention (декодер-к-энкодеру). Что это означает для потребления памяти, когда длины исходной и целевой последовательностей различаются?

4. Почему каузальное маскирование применяется в self-attention декодера, но *не применяется* в cross-attention декодера? Что пошло бы не так, если бы вы добавили каузальное маскирование в cross-attention?

Практические задания

1. **Синтетическая задача Seq2Seq MNIST как парные последовательности**: Рассматривайте каждое изображение MNIST как *исходную* последовательность из 784 пикселей (вытянутых по строкам). Определите *целевую* последовательность как 10-мерный one-hot вектор метки цифры, преобразованный в последовательность длины 10 (каждая позиция соответствует классу цифры). Ваша цель – обучить компактную модель «энкодер–декодер», отображающую 784-элементную последовательность пикселей в 10-элементную последовательность меток.

2. **Модель A Декодер с Cross-Attention**: Реализуйте минимальную трансформероподобную модель:

   - *Энкодер*: один слой self-attention над 784-пиксельным входом (с позиционными кодировками).
   - *Декодер*: один слой cross-attention, в котором 10 обучаемых запросов (по одному на класс) обращаются к выходу энкодера. Self-attention в декодере отсутствует.
   - Выход: линейная проекция выхода декодера в логиты по 10 классам.

3. **Модель B Только Self-Attention (абляция)**: Реализуйте вторую модель, в которой cross-attention заменён на *только self-attention в декодере*. Чтобы это стало возможным, конкатенируйте представление энкоде-

ра (усреднённое до одного вектора) с 10 обучаемыми «псевдотокенами» – это будет вход декодера. Используйте только self-attention (с каузальным маскированием) в декодере. Эта модель не может обращаться к отдельным пикселям, только к усреднённому сводному представлению.

4. **Обучение и оценка**: Обучите обе модели на наборе данных MNIST. Используйте идентичные гиперпараметры (скорость обучения, оптимизатор, размерность эмбеддингов и т.д.). Предоставьте:
   - Итоговую точность на валидации,
   - Графики обучения (потери и точность),
   - Задержку при инференсе.
5. **Визуализация внимания**: Для 5 тестовых изображений визуализируйте веса cross-attention от 10 запросов к 784 пикселям входа. Определите, какие пиксели получают высокие веса для правильного класса цифры. Сравните это с внутренними паттернами self-attention модели (если они интерпретируемы).

Требования к сдаче работы

- Один Jupyter/Colab Notebook с чётко обозначенными разделами для каждого задания.
- Весь код должен быть самодостаточным (без загрузки внешних моделей).
- Включите графики кривых обучения и тепловые карты внимания (используйте `matplotlib` или `seaborn`).
- Напишите краткое описание абляции (до 200 слов): почему cross-attention превосходит (или уступает) базовой модели с self-attention? Что это говорит об индуктивных смещениях?
- Убедитесь, что позиционные кодировки используются как в энкодере, так и в декодере (обучаемые или синусоидальные).

Критерии оценки: Для успешной сдачи лабораторной работы необходимо: (1) корректно ответить на все теоретические вопросы, (2) достичь точности на валидации 85% для модели с cross-attention, (3) продемонстрировать разрыв в производительности между Моделью A и Моделью B, и (4) представить интерпретируемые визуализации внимания.

5. Лабораторная работа №3. Бутылочное горлышко механизма внимания

5.1. Квадратичная временная и память-сложность: теоретический и практический анализ

Механизм масштабированного скалярного произведения (scaled dot-product attention), несмотря на свой эмпирический успех, страдает от фундаментального вычислительного узкого места: его требования к времени и памяти растут *квадратично* с длиной последовательности . Это напрямую связано с вычислением матрицы внимания $\mathbf{A} = \text{softmax}(\mathbf{QK}^\top/\sqrt{d_k}) \in \mathbb{R}^{n\times m}$, которое требует оценки попарных взаимодействий между всеми запросами и ключами.

Теоретическая сложность. Рассмотрим стандартный случай самовнимания, где $n = m = L$ (длина последовательности). Вычислительные этапы следующие:

1. Вычисление логитов: $\mathbf{QK}^\top$ – матричное умножение размерностей $(L\times d_k)\cdot(d_k\times L)$, требующее $O(L^2 d_k)$ операций с плавающей точкой (FLOPs).

2. Применение softmax: $O(L^2)$ операций (поэлементное возведение в экспоненту и нормализация по строкам).

3. Вычисление выхода: $\mathbf{AV}$ – матричное умножение размерностей $(L \times L) \cdot (L \times d_v)$, требующее $O(L^2 d_v)$ FLOPs.

Таким образом, общая временная сложность составляет $O(L^2(d_k+d_v))$. Поскольку d_k и d_v не зависят от длины входа, сложность механизма внимания по длине входа равна $O(L^2)$.

Потребление памяти определяется следующими компонентами:

- Матрица внимания $\mathbf{A}$: L^2 чисел с плавающей точкой,

- Промежуточные градиенты при обратном распространении ошибки (например, $\partial\mathcal{L}/\partial\mathbf{A}$, $\partial\mathcal{L}/\partial(\mathbf{QK}^\top)$), которые также требуют $O(L^2)$ памяти для автоматического дифференцирования.

Следовательно, пиковое потребление памяти масштабируется как $O(L^2)$, независимо от глубины модели, но критически зависит от длины контекста.

В случае кросс-внимания с длиной источника m и длиной цели n сложность становится $O(nm)$, что остаётся квадратичным в худшем случае (например, при $n \approx m$).

Практические последствия. На практике узкое место L^2 проявляется уже при умеренных значениях L:

- При $L = 1024$ матрица $\mathbf{A}$ занимает 4МБ (float32).
- При $L = 8192$ объём $\mathbf{A}$ достигает 256МБ.
- При $L = 32\,768$ объём $\mathbf{A}$ превышает 4ГБ даже для одного слоя в прямом проходе, что выходит за пределы видеопамяти многих устройств.

Более того, пропускная способность памяти часто становится ограничивающим фактором, поскольку матрица внимания многократно считывается и записывается в процессе прямого и обратного проходов.

Это ограничение вынуждает реальные системы:

- Обрезать длинные документы (например, в больших языковых моделях с окном контекста 4K),
- Использовать разреженное или приближённое внимание (например, Longformer, BigBird),
- Переносить вычисления на CPU или диск (замедляя обучение),
- Применять экономичные по памяти ядра, такие как FlashAttention, которые снижают, но не устраняют асимптотическую сложность.

Эмпирическая проверка. Как показано в лабораторной работе №1, профилирование внимания при увеличении L подтверждает масштабирование $O(L^2)$ как по времени выполнения, так и по видеопамяти. Примечательно, что скорость роста потребления памяти оказывается выше, чем рост числа FLOPs, поскольку память расходуется даже в периоды простоя вычислений (например, при хранении $\mathbf{A}$ для обратного распространения).

Это квадратичное узкое место не является недостатком самой *идеи* внимания, которая обеспечивает ценную глобальную контекстуализацию, а следствием её *наивной реализации*. Поэтому значительная часть современных исследований направлена на разработку вариантов внимания, сохраняющих выразительность при достижении почти линейной сложности, что будет рассмотрено в последующих подразделах. На данном этапе важно осознавать, что изящество механизма внимания имеет высокую вычислительную стоимость, определяющую архитектурные решения во всех масштабных моделях обработки последовательностей.

5.2. Центральная предельная теорема и концентрация внимания: почему softmax «сжимается»

Тонкое, но критически важное узкое место в механизмах внимания возникает не только из-за вычислительной сложности, но и из-за *статистической вырожденности* оператора softmax при применении к длинным контекстам. По мере роста длины последовательности распределение весов внимания становится всё более *пикообразным* или *сконцентрированным*, часто вырождаясь в эффективный one-hot вектор даже при отсутствии проблем с масштабированием. Это явление обусловлено взаимодействием геометрии высоких размерностей, случайных проекций и центральной предельной теоремы (ЦПТ), и означает, что на практике лишь исчезающе малая доля токенов действительно вносит вклад в выход внимания.

Скалярные произведения как суммы случайных величин. Предположим, что запросы и ключи являются случайными векторами нулевого среднего и изотропными в $\mathbb{R}^{d_k}$ (например, после нормализации по слоям и линейного преобразования). Тогда скалярное произведение между запросом $\mathbf{q}_i$ и ключом $\mathbf{k}_j$ имеет вид:

$$s_{ij} = \mathbf{q}_i^\top \mathbf{k}_j = \sum_{\ell=1}^{d_k} q_i^{(\ell)} k_j^{(\ell)}. \tag{25}$$

Если компоненты $q_i^{(\ell)}, k_j^{(\ell)}$ независимы и одинаково распределены с конечной дисперсией, то центральная предельная теорема (ЦПТ) утверждает, что s_{ij} приближённо гауссовская величина:

$$s_{ij} \xrightarrow{d} \mathcal{N}(0, \sigma^2 d_k), \quad \text{при } d_k \to \infty, \tag{26}$$

с дисперсией, пропорциональной d_k. Важно отметить, что даже при умеренных значениях d_k (например, 64) это гауссово приближение уже достаточно точно благодаря быстрой сходимости ЦПТ.

Теперь рассмотрим фиксированный запрос $\mathbf{q}_i$ и m ключей $\{\mathbf{k}_j\}_{j=1}^m$. Логиты $\{s_{ij}\}_{j=1}^m$ ведут себя как m выборок из $\mathcal{N}(0, \sigma^2 d_k)$. Максимум среди m независимых гауссовых величин растёт как $\Theta(\sqrt{\log m})$ – известный результат теории экстремальных значений. Следовательно, по мере роста m (длины контекста) разрыв между наибольшим логитом и основной массой распределения увеличивается.

Softmax усиливает экстремумы. Функция softmax экспоненциально чувствительна к различиям между логитами:

$$A_{ij} = \frac{e^{s_{ij}/\sqrt{d_k}}}{\sum_{l=1}^m e^{s_{il}/\sqrt{d_k}}}. \tag{27}$$

Поскольку числитель экспоненциально растёт с s_{ij}, даже небольшое преимущество нескольких верхних логитов доминирует в знаменателе. В результате при больших m распределение внимания становится *сильно асимметричным*: горсть токенов получает почти всю вероятностную массу, тогда как остальным присваиваются пренебрежимо малые веса.

Эффективный контекст значительно меньше m. Эмпирические исследования (например, [24, 5]) показывают, что в реальных моделях эффективный ранг или энтропия распределений внимания резко падают с ростом длины последовательности. При $m = 512$ значимый вклад могут вносить лишь 20–50 токенов; при $m = 8192$ это число может увеличиться лишь до 50–100 не линейно, а логарифмически. Это означает, что большая часть $O(m)$ контекста представляет собой *потерянные вычисления.*

Критически важно, что такая концентрация возникает *задолго до* предела $m \to \infty$. Фактически, при типичном $d_k = 64$ моделирование показывает, что уже при $m \approx 200$ случайных ключах топ-5 токенов часто аккумулируют более 80% массы внимания даже при корректном масштабировании на $1/\sqrt{d_k}$. Таким образом, узкое место заключается не только в памяти или FLOPs, но и в *информационной избыточности*: модель не может эффективно использовать длинные контексты, поскольку softmax по своей природе подавляет разнообразие в условиях высокой размерности и большого m.

Почему это важно. Этот статистический коллапс объясняет, почему:

- Модели с длинным контекстом часто демонстрируют более плохие результаты, нежели те, которые ожидаемы без специальной регуляризации (например, с ℓ_2-нормализованными запросами и ключами),
- Разреженное внимание (например, внимание только к локальным окнам или топ-k токенам) работает удивительно хорошо,
- Альтернативные формулировки внимания (например, ϕ-внимание, линейное внимание через ядерные трюки) стремятся заменить softmax на более равномерные или структурированные агрегаторы.

По сути, настоящее узкое место внимания заключается не только в квадратичной стоимости, но и в *неспособности softmax поддерживать диффузное, информативное распределение на длинных последовательностях.* Механизм становится саморазрушительным: чем длиннее контекст, тем меньше токенов действительно имеют значение. Это понимание лежит в основе многих современных архитектур, ориентированных на эффективность, которые рассматриваются далее.

5.3. Проблема конечного контекстного окна: потеря информации, катастрофическое забывание

Несмотря на теоретическую способность механизма внимания моделировать глобальные зависимости, практическое применение моделей на основе трансформеров ограничено жёстким пределом длины входной последовательности обычно от 512 до 32 768 токенов в зависимости от архитектуры и доступного оборудования. Это *конечное контекстное окно* создаёт фундаментальное узкое место: любая информация за пределами окна просто *отбрасывается*, что приводит к двум взаимосвязанным режимам отказа **потере информации** и **катастрофическому забыванию**, серьёзно ограничивающим способность модели рассуждать над длинными документами, сохранять историю диалога или выполнять многошаговые выводы.

Потеря информации за пределами окна. Когда вход превышает максимально допустимую длину контекста $L_{\max}$, стандартной практикой является усечение последовательности с сохранением только самых свежих (или первых) $L_{\max}$ токенов. В таких задачах, как:

- *Ответы на вопросы по длинным документам*: релевантный факт может находиться на странице 1, а вопрос на странице 20, за пределами контекстного окна.
- *Чат-боты*: изначальное предпочтение пользователя (например, «Я вегетарианец») вытесняется из контекста после нескольких реплик.
- *Генерация кода*: определение функции, используемой позже в файле, оказывается невидимым при генерации её вызова.

модель не имеет механизма для восстановления утраченной информации. В отличие от рекуррентных архитектур с состоятельными скрытыми векторами (которые теоретически могут сжимать длинные истории), внимание в трансформере является чисто *реактивным* – оно может обращаться только к тому, что явно присутствует во входном буфере. Таким образом, критически важное семантическое или фактическое содержание теряется безвозвратно.

Катастрофическое забывание в потоковых сценариях. При авторегрессивной генерации длинных последовательностей (например, написании романа или продолжительного диалога) модель обрабатывает токены в скользящем окне. По мере поступления новых токенов старые отбрасываются для поддержания длины $L_{\max}$. Поскольку трансформер не обладает внешней памятью или рекуррентностью, он не может сохранять сжатые резюме прошлого содержимого. Это приводит к *катастрофическому забыванию* – внезапным несогласованностям, таким как:

- изменение имени персонажа в середине повествования,
- противоречие ранее заявленным фактам («Раньше вы сказали X, теперь говорите не X»),
- повторение фраз или потеря структуры дискурса.

В отличие от человека или даже РНС с тщательно настроенными механизмами затворов, обычные трансформеры рассматривают каждое окно как независимый эпизод, не имея постоянного состояния, связывающего временные разрывы.

Почему одно лишь внимание не может решить эту проблему? Можно было бы надеяться, что кросс-внимание к внешней памяти (например, в генерации с извлечением, Retrieval-Augmented Generation, RAG) смягчит эту проблему. Хотя RAG помогает в фактической привязке, он не решает проблему *внутренней согласованности* в длинных генерируемых последовательностях, поскольку собственные предыдущие выходы модели не сохраняются и не поддаются извлечению, если это специально не реализовано (например, через буферизацию логов или банки памяти).

К недавним попыткам расширения контекста относятся:

- **Экстраполяция позиций** (например, масштабирование RoPE): позволяет моделям, обученным на коротких контекстах, *экстраполировать* на более длинные, но часто ухудшает качество из-за сдвига распределения.
- **Сжатие памяти**: использование обучаемых резюме для передачи ключевой информации.
- **Аппроксимации бесконечного контекста**: нейросетевые архитектуры, например, Mamba, или линейного внимания снижают затраты памяти, но жертвуют полной парной выразительностью softmax-внимания.

Однако ни один из подходов полностью не решает основную проблему: *механизм внимания по своей сути близорук, когда ограничен конечной памятью.*

Эмпирические свидетельства. Исследования показывают резкое падение точности ответов на вопросы по длинным документам, как только релевантный фрагмент выходит за пределы контекстного окна. Аналогично, в диалоговых тестах модели демонстрируют «смещение к новизне»: они хорошо справляются с вопросами о недавних репликах, но терпят неудачу при запросах, требующих долгосрочной памяти. Это подтверждает, что

конечное контекстное окно является не бесполезной инженерной деталью, а *фундаментальным архитектурным ограничением.*

В заключение, хотя механизм внимания обеспечивает мощные возможности контекстно-зависимых вычислений, его полезность ограничена физическими и алгоритмическими рамками контекстного окна. Без механизмов сохранения, сжатия или извлечения памяти трансформеры остаются уязвимыми к потере информации и несогласованности в задачах с длинным горизонтом, что подчёркивает необходимость гибридных архитектур, сочетающих внимание с системами постоянной памяти.

5.4. Кэширование ключей и значений при автогрессивном декодировании: вычислительная избыточность и сложность

При автогрессивной генерации последовательностей с использованием трансформерных моделей распределение выхода на временном шаге $t \in \{1, \ldots, T\}$ обусловлено префиксом $\mathbf{x}_{1:t} = (x_1, \ldots, x_t)$. Механизм самовнимания вычисляет для каждого слоя проекции запросов, ключей и значений:

$$\mathbf{Q}_t = \mathbf{X}_t \mathbf{W}^Q, \quad \mathbf{K}_t = \mathbf{X}_t \mathbf{W}^K, \quad \mathbf{V}_t = \mathbf{X}_t \mathbf{W}^V,$$

где $\mathbf{X}_t \in \mathbb{R}^{t \times d_{\text{model}}}$ обозначает контекстуализированные представления токенов до позиции t, а $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ (или d_v) обучаемые матрицы проекций.

Наивная реализация, при которой $\mathbf{K}_t$ и $\mathbf{V}_t$ пересчитываются с нуля на каждом шаге декодирования, приводит к вычислительной избыточности: представления токенов $1, \ldots, t-1$ повторно пропускаются через все предшествующие слои, несмотря на то что при причинно-маскирующем условии они инвариантны относительно будущих входов. Хотя операция вычисления внимания для текущего запроса $\mathbf{q}_t$ относительно всего множества ключей $\mathbf{K}_{1:t}$ имеет стоимость $O(td_k)$ на шаг, на практике доминирующим фактором времени выполнения становится многократное прямое распространение через слои эмбеддингов и промежуточные представления.

Кэширование ключей и значений (KV-кэширование) устраняет эту избыточность за счёт поддержания постоянных буферов $\mathcal{K}^{(\ell)}$ и $\mathcal{V}^{(\ell)}$ для каждого слоя ℓ, в которые инкрементально сохраняются спроецированные ключи и значения:

$$\mathcal{K}^{(\ell)} \leftarrow [\mathcal{K}^{(\ell)}, \mathbf{k}_t^{(\ell)}], \quad \mathcal{V}^{(\ell)} \leftarrow [\mathcal{V}^{(\ell)}, \mathbf{v}_t^{(\ell)}].$$

На шаге t через стек сети пропускается лишь представление нового токена x_t; ранее вычисленные $\mathbf{k}_j^{(\ell)}$ и $\mathbf{v}_j^{(\ell)}$ для $j < t$ повторно используются из кэша. В результате суммарная вычислительная сложность за T шагов масштабируется как

$$\sum_{t=1}^{T} O(t) = O(T^2),$$

что совпадает со сложностью одного неавтогрессивного прямого прохода по последовательности длины T.

Важно отметить, что KV-кэширование не изменяет асимптотический объём потребляемой памяти: совокупный объём хранимых данных для $\mathcal{K}$ и $\mathcal{V}$ во всех слоях остаётся $O(Ld_k + Ld_v)$ на слой при длине контекста L, а матрица весов внимания (в случае её материализации) по-прежнему занимает $O(L^2)$ памяти при обратном распространении ошибки или использовании неспециализированных (non-fused) ядер. Более того, кэширование никак не смягчает проблему статистической концентрации весов внимания, рассмотренную ранее; оно решает исключительно задачу устранения вычислительной избыточности при инференсе.

Таким образом, несмотря на то что KV-кэширование является необходимым компонентом эффективного автогрессивного декодирования, оно функционирует в рамках тех же фундаментальных квадратичных ограничений (как вычислительных, так и статистических), которые характеризуют канонический механизм скалированного скалярного произведения внимания.

5.5. Визуализация: тепловые карты внимания при увеличении длины последовательности

В данной лабораторной работе исследуется, как поведение механизма внимания изменяется с ростом длины последовательности, с акцентом на статистические и вычислительные узкие места, обсуждаемые в этой главе. В рамках лабораторной работы предлагается не обучать модели, а использовать фиксированный, случайно инициализированный (или заранее заданный) модуль внимания для вычисления и анализа паттернов внимания в контролируемых условиях.

Теоретические вопросы

1. Объясните, почему логиты скалярного произведения во внимании становятся всё более пикообразными по мере роста длины контекста m, даже при масштабировании на $1/\sqrt{d_k}$. Как центральная предельная теорема способствует этому эффекту?

2. Дайте определение энтропии распределения внимания. Почему она уменьшается при удлинении последовательностей при случайных входах? Что означает низкая энтропия с точки зрения эффективного использования контекста?

3. Почему в конечном контекстном окне информация за его пределами невосстановима в стандартном трансформере? Чем это отличается от РНС или моделей с внешней памятью?

4. Почему квадратичная стоимость памяти внимания становится аппаратным узким местом раньше, чем FLOPs? Проиллюстрируйте конкретными числами для $L = 4096$ в формате float16.

5. Изменяет ли FlashAttention статистические свойства распределения внимания (например, энтропию, разреженность)? Обоснуйте свой ответ.

Задачи анализа и визуализации

1. **Вычисление синтетического внимания**. Для длин последовательностей $L \in \{128, 256, 512, 1024, 2048, 4096\}$ сгенерируйте случайные матрицы запросов и ключей:

$$\mathbf{Q}, \mathbf{K} \sim \mathcal{N}(0, 1) \quad \text{с размерностью } (L, d_k),\ d_k = 64.$$

   Вычислите матрицу внимания как

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right).$$

   Используйте только базовые операции PyTorch без обучения и без векторов значений.

2. **Тепловые карты внимания**. Для каждого L визуализируйте веса внимания от *последнего* токена-запроса (строка $L-1$ матрицы $\mathbf{A}$) ко всем позициям ключей. Создайте подграфики, показывающие, как распределение становится острее с ростом L. Выделите позиции с топ-5 значениями внимания.

3. **Количественная оценка концентрации**. Для каждого L вычислите:

   - Энтропию: $H = -\sum_{j=1}^{L} A_{L-1,j} \log A_{L-1,j}$,
   - Вес топ-1: $\max_j A_{L-1,j}$,
   - Покрытие топ-10: $\sum_{j \in \text{топ-10}} A_{L-1,j}$.

   Постройте графики этих метрик в зависимости от L (рекомендуется логарифмический масштаб по оси L).

4. **Профилирование памяти и времени выполнения**. Для тех же значений L измерьте:

   - Пиковое потребление видеопамяти GPU при вычислении внимания (используйте `torch.cuda.reset_peak_memory_stats()` и `max_memory_allocated()`),

- Время (в мс) прямого прохода для $\mathbf{QK}^\top$, softmax и создания $\mathbf{A}$.

Наложите теоретическую кривую $O(L^2)$ на графики памяти и времени.

5. **Моделирование контекстного окна**. Установите $L = 2048$, но смоделируйте контекстное окно $L_{\max} = 512$. Вычислите внимание дважды:

   - (a) По полной матрице 2048×2048,
   - (b) Только по последним 512 токенам, т. е. $\mathbf{A}_{\text{усеч}} = \text{softmax}(\mathbf{Q}_{512:}\mathbf{K}_{512:}^\top/\sqrt{d_k})$.

   Сравните энтропию и вес топ-1 в случаях (a) и (b). Обсудите, «улучшает» ли усечение концентрацию искусственно.

Требования к оформлению

- Один Colab/Kaggle Notebook со всеми визуализациями внутри.
- Тепловые карты должны содержать цветовые шкалы, подписи осей и маркеры для наиболее значимых позиций.
- Все графики должны иметь легенды, заголовки осей и соответствующие масштабы.
- Код должен быть эффективным и избегать одновременного хранения всех матриц $\mathbf{A}$ в памяти (освобождать память, когда матрица больше не нужна).
- Включите краткое описание (до 150 слов): *При какой длине L внимание становится эффективно разреженным при случайных входах? Каковы последствия этого для моделирования длинного контекста?*

Критерии оценки: Необходимо: (1) корректно ответить на все теоретические вопросы, (2) построить точные тепловые карты, демонстрирующие возрастающую концентрацию, (3) экспериментально подтвердить масштабирование памяти как $O(L^2)$, и (4) представить тренды энтропии и веса топ-1, подтверждающие коллапс внимания с ростом L.

6. Лабораторная работа №4. Линейное внимание и приближения softmax

6.1. Теория: kernal trick

Квадратичная сложность стандартного масштабированного скалярного произведения в механизме внимания возникает из-за явного вычисления полной матрицы внимания

$$\mathbf{A} = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\mathsf{T}}{\sqrt{d_k}}\right) \in \mathbb{R}^{n\times m},$$

что требует $O(nm)$ памяти и времени для хранения полной матрицы весов. Однако, если взвешенную с помощью softmax сумму можно алгебраически переписать без изменения её математического смысла в такую форму, которая не требует материализации $\mathbf{A}$ (то есть явного создания и хранения в памяти всей промежуточной матрицы размером $n \times m$), то становится возможной линейная (или почти линейная) сложность.

Мощный подход к достижению этого основан на kernal trick: экспоненциальная функция внутри softmax заменяется отображением признаков $\phi(\cdot)$, таким что

$$\exp\left(\langle \mathbf{q}_i\,,\,\mathbf{k}_j\rangle/\sqrt{d_k}\right) \approx \langle \phi(\mathbf{q}_i)\,,\,\phi(\mathbf{k}_j)\rangle.$$

Если такое отображение существует, выход внимания для i-го запроса можно переписать как

$$\mathbf{o}_i = \frac{\sum_{j=1}^m \exp\left(\langle \mathbf{q}_i\,,\,\mathbf{k}_j\rangle/\sqrt{d_k}\right)\,\mathbf{v}_j}{\sum_{j=1}^m \exp\left(\langle \mathbf{q}_i\,,\,\mathbf{k}_j\rangle/\sqrt{d_k}\right)} \tag{28}$$

$$= \frac{\langle \phi(\mathbf{q}_i)\,,\,\sum_{j=1}^m \phi(\mathbf{k}_j)\mathbf{v}_j^\mathsf{T}\rangle}{\langle \phi(\mathbf{q}_i)\,,\,\sum_{j=1}^m \phi(\mathbf{k}_j)\rangle}. \tag{29}$$

Определим агрегированные статистики «ключ–значение» и «только ключ» как

$$\mathbf{S}_{KV} = \sum_{j=1}^m \phi(\mathbf{k}_j)\mathbf{v}_j^\mathsf{T} \in \mathbb{R}^{D\times d_v}, \quad \mathbf{S}_K = \sum_{j=1}^m \phi(\mathbf{k}_j) \in \mathbb{R}^D,$$

(где D размерность пространства признаков, индуцированного ϕ), тогда весь выход внимания принимает вид

$$\mathbf{O} = \mathrm{LinearAttention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = (\Phi(\mathbf{Q})\mathbf{S}_{KV}) \oslash (\Phi(\mathbf{Q})\mathbf{S}_K)\,, \tag{30}$$

где $\Phi(\mathbf{Q}) = [\phi(\mathbf{q}_1),\ldots,\phi(\mathbf{q}_n)]^\mathsf{T} \in \mathbb{R}^{n\times D}$, а $\oslash$ обозначает построчное деление (с трансляцией по столбцам).

Ключевым преимуществом такой формулировки является развязывание зависимости между запросами и ключами: все взаимодействия с памятью «ключ–значение» предварительно агрегируются в $\mathbf{S}_{KV}$ и $\mathbf{S}_K$, что можно выполнить за один проход по последовательности. Окончательный вывод затем требует лишь двух матричных умножений и одного деления, обеспечивая $O(n+m)$ памяти и $O((n+m)D)$ времени при условии, что D постоянно или растёт медленно с ростом d_k.

Успех такого преобразования зависит от выбора ϕ. Наиболее распространённые аппроксимации включают:

- Случайные признаки Фурье (Random Fourier Features, RFF): для гауссовского ядра $k(\mathbf{q}, \mathbf{k}) = \exp(-\|\mathbf{q}-\mathbf{k}\|^2/2\sigma^2)$, хотя оно напрямую не совпадает с softmax,
- Положительные отображения признаков: например, $\phi(\mathbf{x}) = \exp(\mathbf{x}/\sqrt{d_k})$, применяемое поэлементно (используется в Performer [6]), которое удовлетворяет

$$\langle \phi(\mathbf{q}) \, , \, \phi(\mathbf{k}) \rangle = \exp\left(\langle \mathbf{q} \, , \, \mathbf{k} \rangle / \sqrt{d_k} \right) \quad \text{только если } \mathbf{q}, \mathbf{k} \geq 0,$$

  но служит практичной заменой при соответствующей нормализации,
- Детерминированные ядра: такие как $\phi(\mathbf{x}) = \mathrm{elu}(\mathbf{x}) + 1$, обеспечивающие неотрицательность и гладкость.

Хотя эти аппроксимации вносят смещение по сравнению с точным вниманием через softmax, они сохраняют основную индуктивную предвзятость взвешивания на основе содержания, позволяя достичь линейного масштабирования по длине последовательности. Этот компромисс между точностью и эффективностью лежит в основе современных архитектур для длинных контекстов.

6.2. Современные аппроксимации

Чтобы обойти квадратичную вычислительную сложность и концентрацию внимания, вызванную softmax, в недавних работах точное ядро внимания заменяется структурированными аппроксимациями, допускающими линейные или почти линейные вычисления.

Два представительных подхода **EfficientViT** и **QTViT** иллюстрируют различные философии: первый использует простое положительное отображение признаков для реализации ядерного внимания, тогда как второй опирается на разложение softmax в ряд Тейлора.

EfficientViT: ReLU как положительное отображение признаков. Архитектура EfficientViT [3] полностью отказывается от softmax, переосмысливая внимание через призму неотрицательной ядерной аппроксимации.

Отмечается, что если отображение признаков $\phi(\cdot)$ выбрано поэлементно неотрицательным, то ядерная формулировка внимания

$$\mathbf{O} = \frac{\phi(\mathbf{Q}) \left(\sum_j \phi(\mathbf{k}_j) \mathbf{v}_j^\top \right)}{\phi(\mathbf{Q}) \left(\sum_j \phi(\mathbf{k}_j) \right)}$$

остаётся корректно определённой и численно устойчивой без явной нормализации.
EfficientViT предлагает исключительно простой выбор (функцию линейного выпрямления (ReLU), определяемую как $\mathrm{ReLU}(x) = \max(0, x)$):

$$\phi(\mathbf{x}) = \mathrm{ReLU}(\mathbf{x}). \tag{31}$$

Хотя $\langle \mathrm{ReLU}(\mathbf{q}) \, , \, \mathrm{ReLU}(\mathbf{k}) \rangle$ не равно $\exp(\langle \mathbf{q} \, , \, \mathbf{k} \rangle / \sqrt{d_k})$, это отображение сохраняет два ключевых свойства:

1. *Неотрицательность*: $\phi(\mathbf{x}) \geq 0$, что гарантирует интерпретируемость агрегированных статистик $\mathbf{S}_K = \sum_j \phi(\mathbf{k}_j)$ и $\mathbf{S}_{KV} = \sum_j \phi(\mathbf{k}_j) \mathbf{v}_j^\top$ как ненормированных «счётчиков» и «взвешенных сумм»,

2. *Разреженность*: ReLU обнуляет активации для отрицательных предактиваций, порождая разреженные $\phi(\mathbf{x})$ и снижая фактические вычислительные затраты.

На практике EfficientViT применяет нормализацию по слоям перед ReLU, чтобы центрировать входы и смягчить чрезмерную разреженность. Полученный механизм внимания называемый *линейным вниманием на основе признаков ReLU* достигает сложности $O(L)$ и позволяет трансформерам для компьютерного зрения масштабироваться до изображений высокого разрешения с минимальными потерями качества. Его успех показывает, что точное воспроизведение softmax не требуется; важно сохранить *относительное взвешивание* между релевантными токенами.

QTViT: Аппроксимация второго порядка с диагональным квадратичным приближением. Альтернативный подход стремится аппроксимировать ядро softmax *аналитически*. Модель QTViT [25] начинает с разложения экспоненты в функции сходства до второго порядка:

$$\exp\left(\frac{\langle \mathbf{q}_i \, , \, \mathbf{k}_j \rangle}{\sqrt{d_k}} \right) \approx 1 + \frac{\langle \mathbf{q}_i \, , \, \mathbf{k}_j \rangle}{\sqrt{d_k}} + \frac{1}{2} \left(\frac{\langle \mathbf{q}_i \, , \, \mathbf{k}_j \rangle}{\sqrt{d_k}} \right)^2 . \tag{32}$$

Квадратичный член можно переписать с использованием тождества $\langle \mathbf{q} \, , \, \mathbf{k} \rangle^2 = \langle \mathbf{q} \otimes \mathbf{q} \, , \, \mathbf{k} \otimes \mathbf{k} \rangle$, где $\otimes$ обозначает произведение Кронекера. Это наводит на

мысль об отображении признаков:

$$\phi^{(2)}(\mathbf{x}) = \begin{bmatrix} 1 \\ \mathbf{x}/\sqrt{d_k} \\ (\mathbf{x} \otimes \mathbf{x})/(\sqrt{2}\, d_k) \end{bmatrix}, \tag{33}$$

которое удовлетворяет $\langle \phi^{(2)}(\mathbf{q}) \,,\, \phi^{(2)}(\mathbf{k}) \rangle \approx \exp(\langle \mathbf{q} \,,\, \mathbf{k} \rangle / \sqrt{d_k})$. Однако $\mathbf{x} \otimes \mathbf{x}$ имеет размерность d_k^2, делая такое представление вычислительно неприемлемым.

Ключевое наблюдение QTViT состоит в том, что основной вклад в квадратичный член дают *диагональные элементы* внешнего произведения $\mathbf{x}\mathbf{x}^\mathsf{T}$, то есть поэлементные квадраты $\{x_i^2\}_{i=1}^{d_k}$. Внедиагональные смешанные члены $x_i x_j$ $(i \neq j)$ на практике оказывают незначительное влияние на итоговую производительность. Поэтому QTViT отбрасывает их и вводит компактное отображение признаков:

$$\tilde{\phi}(\mathbf{x}) = \left[\, \alpha \cdot \mathbf{x}^2,\ \beta \cdot \mathbf{x},\ \gamma \,\right] \in \mathbb{R}^{2d_k+1}, \tag{34}$$

где $\mathbf{x}^2$ обозначает поэлементное возведение в квадрат, а α, β, γ обучаемые скаляры. В окончательной реализации авторы дополнительно упрощают модель, полагая $\beta = 0$ (то есть отбрасывая линейный член), получая:

$$\tilde{\phi}(\mathbf{x}) = \left[\, \alpha \cdot \mathbf{x}^2,\ \gamma \,\right] \in \mathbb{R}^{d_k+1}. \tag{35}$$

Такая аппроксимация сохраняет основное квадратичное взаимодействие второго порядка $\langle \mathbf{q}^2 \,,\, \mathbf{k}^2 \rangle = \sum_i q_i^2 k_i^2$, соответствующее диагонали матриц $\mathbf{q}\mathbf{q}^\mathsf{T}$ и $\mathbf{k}\mathbf{k}^\mathsf{T}$, одновременно сокращая размерность признаков с $O(d_k^2)$ до $O(d_k)$.

С этим отображением внимание вычисляется как

$$\mathbf{O} = \frac{\tilde{\phi}(\mathbf{Q}) \left(\sum_j \tilde{\phi}(\mathbf{k}_j) \mathbf{v}_j^\mathsf{T} \right)}{\tilde{\phi}(\mathbf{Q}) \left(\sum_j \tilde{\phi}(\mathbf{k}_j) \right)}, \tag{36}$$

что требует лишь $O(Ld_k)$ времени и памяти. Эмпирически эта простая диагональная квадратичная аппроксимация последовательно превосходит методы первого порядка (например, линейное разложение Тейлора) и соответствует или превосходит EfficientViT на различных масштабах моделей без необходимости в дистилляции знаний или использовании высокоуровневых остаточных соединений.

В совокупности EfficientViT и QTViT демонстрируют, что масштабируемое внимание не требует сложных механизмов: *положительность на основе ReLU* и *диагональные взаимодействия второго порядка* достаточны для сохранения большей части выразительной способности softmax-внимания

при достижении линейной сложности. Эти идеи открывают путь к эффективным, не требующим дополнительного обучения и дружелюбным к аппаратному обеспечению трансформерам, пригодным для практического развёртывания.

6.3. Эксперимент: точность, задержка, объём занимаемой памяти

В данном эксперименте вы проведёте эмпирическую оценку трёх механизмов внимания в рамках единой архитектуры Vision Transformer (ViT): (1) стандартного самовнимания на основе softmax, (2) QT-ViT с квадратичным разложением Тейлора (диагональное ядро второго порядка) и (3) EfficientViT с линейным вниманием на основе ReLU.

Все модели имеют идентичную архитектурную основу (глубина, размерность эмбеддингов, размер патча и т. д.) и отличаются лишь ядром внимания. Ваша задача – количественно оценить компромиссы между точностью, вычислительной сложностью (FLOPs), задержкой при выводе и потреблением памяти, а также определить режим разрешения входных данных, при котором линейное внимание становится практически выгодным.

Теоретические и аналитические вопросы

1. Почему квадратичное разложение Тейлора в QT-ViT сохраняет большую выразительную способность по сравнению с аппроксимациями первого порядка? Какое конкретное взаимодействие оно захватывает, чего не может сделать EfficientViT на основе ReLU?

2. Объясните, почему методы линейного внимания (QT-ViT, EfficientViT) имеют сложность $O(Ld^2)$ вместо $O(L^2d)$. При каком соотношении между L и d это становится выгодным?

3. В пределе очень высокого разрешения входных данных, почему линейное внимание может не только экономить память, но и улучшать устойчивость обучения?

4. И QT-ViT, и EfficientViT избегают softmax. Как это влияет на распространение градиентов при обратном проходе по сравнению со стандартным вниманием? Рассмотрите роль насыщения и разреженности.

5. Предположим, вы фиксируете количество FLOPs у всех моделей, регулируя глубину или ширину. Ожидаете ли вы, что модели с линейным вниманием превзойдут внимание на основе softmax при высоких разрешениях? Обоснуйте свой ответ.

Практические задания

6. **Базовая архитектура ViT**: Реализуйте минимальный энкодер Vision Transformer для классификации изображений:

   - Входные данные: MNIST (28×28) или CIFAR-10 (32×32) на ваш выбор.
   - Разбейте изображение на неперекрывающиеся патчи (например, $4 \times 4 \to 49$ токенов для MNIST).
   - Преобразуйте патчи в эмбеддинги размерности $d = 64$.
   - Добавьте обучаемые позиционные эмбеддинги.
   - Используйте 4 трансформерных блока, каждый с одним головным вниманием и без полносвязной сети (для упрощения).
   - Выполните глобальное усреднение и линейную классификацию.

7. **Модель A – стандартное внимание с softmax**: Используйте масштабированное скалярное произведение.

8. **Модель B – QT-ViT**: Замените механизм внимания на диагональное квадратичное ядро из QT-ViT:

   $$\phi(\mathbf{x}) = [\alpha \cdot \mathbf{x}^2,\ \gamma] \in \mathbb{R}^{d+1},$$

   и вычисляйте внимание через ядерную агрегацию:

   $$\mathbf{O} = \frac{\phi(\mathbf{Q})(\sum_j \phi(\mathbf{k}_j)\mathbf{v}_j^\top)}{\phi(\mathbf{Q})(\sum_j \phi(\mathbf{k}_j))}.$$

   Инициализируйте $\alpha = 1/\sqrt{2d}$, $\gamma = 1/\sqrt{2}$ (в соответствии с масштабированием разложения Тейлора). Не используйте дистилляцию знаний или резидуальный softmax.

9. **Модель C EfficientViT**: Используйте $\phi(\mathbf{x}) = \mathrm{ReLU}(\mathbf{x})$ с нормализацией по слоям перед ReLU для стабилизации активаций. Примените ту же формулу ядерного внимания, что и выше.

10. **Протокол обучения**: Обучите все три модели с одинаковыми гиперпараметрами:

    - Оптимизатор: AdamW ($\mathrm{lr} = 1 \cdot 10^{-3}$, $\mathrm{weight\_decay} = 0{,}05$),
    - Размер батча: 128,
    - Эпохи: 30,
    - Аугментация данных отсутствует.

Сообщите итоговую точность на валидационной выборке.

11. **Оценка эффективности**: Для каждой модели измерьте на одном GPU:

    - Задержку при выводе (мс) на одно изображение (усреднённая по 1000 образцам),
    - Пиковое потребление видеопамяти (МБ) во время прямого прохода,
    - Общее число FLOPs.

12. **Исследование масштабирования: когда линейное внимание выигрывает?** Повторите весь эксперимент при более высоких разрешениях, увеличив MNIST до 56×56 и 112×112. Сохраняйте размер патча фиксированным (4×4), так что длина последовательности L возрастает до 196 и 784 соответственно. Постройте графики:

    - точность в зависимости от разрешения,
    - задержка в зависимости от разрешения,
    - потребление памяти в зависимости от разрешения.

    Определите пороговое разрешение, начиная с которого модели с линейным вниманием (B и C) становятся строго предпочтительнее по скорости и памяти при сохранении конкурентоспособной точности.

Требования к оформлению

- Один Colab Notebook с чётко обозначенными разделами для каждого задания.
- Все модели должны быть реализованы с нуля исключительно с использованием PyTorch.
- Включите таблицы, сравнивающие точность, FLOPs, задержку и объём памяти для всех трёх моделей при каждом разрешении.
- Предоставьте линейные графики, показывающие, как метрики зависят от разрешения.
- Добавьте краткое абляционное описание (не более 200 слов): *При каком разрешении линейное внимание становится практически предпочтительным? Какой метод QT-ViT или EfficientViT обеспечивает наилучший компромисс и почему?*
- Обеспечьте воспроизводимость: зафиксируйте начальные значения (seed) генераторов случайных чисел, укажите версию PyTorch и избегайте смешанной точности, если она не применяется единообразно ко всем моделям.

Критерии оценивания: Для успешного выполнения лабораторной работы необходимо: (1) корректно ответить на все теоретические вопросы, (2) достичь точности не ниже 80% у модели со стандартным вниманием на выбранном наборе данных, (3) продемонстрировать эмпирические тенденции масштабирования, согласующиеся со сложностями $O(L^2)$ и $O(L)$, и (4) идентифицировать режим разрешения, при котором линейное внимание даёт чёткое преимущество в эффективности.

7. Лабораторная работа №5. Трансформеры в компьютерном зрении

7.1. Архитектуры ViT Encoder

Трансформер для изображений (Vision Transformer, ViT) [8] переосмысливает задачу понимания изображений через призму моделирования последовательностей: вместо иерархических локальных свёрток он рассматривает изображение как последовательность визуальных токенов и обрабатывает их с помощью стека энкодерных слоёв. Такая конструкция наследует глобальное рецептивное поле и контекстно-зависимую агрегацию внимания, что обеспечивает высокую эффективность как в задачах классификации изображений, так и в других задач компьютерного зрения при достаточном объёме данных и вычислительной мощности модели.

Базовая архитектура. Для входного изображения $\mathbf{X} \in \mathbb{R}^{H\times W\times C}$ модель ViT сначала разбивает его на регулярную сетку из $N = \frac{HW}{P^2}$ неперекрывающихся патчей размером $P \times P$. Каждый патч выравнивается в вектор и проецируется в D-мерное пространство с помощью обучаемого линейного преобразования:

$$\mathbf{z}_i = \mathbf{E} \cdot \mathrm{vec}(\mathbf{x}_i) \in \mathbb{R}^D, \quad i = 1, \ldots, N, \tag{37}$$

где $\mathbf{E} \in \mathbb{R}^{D\times(P^2C)}$ – матрица встраивания патчей. В начало последовательности добавляется обучаемый токен `[CLS]` $\mathbf{z}_0 \in \mathbb{R}^D$, который служит глобальным представлением для задачи классификации (аналогично BERT). Позиционная информация вводится посредством аддитивных обучаемых позиционных вложений $\mathbf{p}_i \in \mathbb{R}^D$:

$$\mathbf{Z}^{(0)} = \begin{bmatrix} \mathbf{z}_0 + \mathbf{p}_0 \\ \mathbf{z}_1 + \mathbf{p}_1 \\ \vdots \\ \mathbf{z}_N + \mathbf{p}_N \end{bmatrix} \in \mathbb{R}^{(N+1)\times D}. \tag{38}$$

Полученная последовательность пропускается через L идентичных блоков энкодера трансформера. Каждый блок состоит из двух подслоёв:

1. **Многоголовое самовнимание (Multi-head Self-Attention, MSA)**:

$$\mathrm{MSA}(\mathbf{Z}) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_h)\mathbf{W}^O, \tag{39}$$

где каждая голова вычисляет масштабированное скалярное произведение с вниманием:

$$\mathrm{head}_i = \mathrm{Attention}\left(\mathbf{Z}\mathbf{W}_i^Q, \mathbf{Z}\mathbf{W}_i^K, \mathbf{Z}\mathbf{W}_i^V\right), \tag{40}$$

причём $\mathbf{W}_i^Q, \mathbf{W}_i^K \in \mathbb{R}^{D\times d_k}$, $\mathbf{W}_i^V \in \mathbb{R}^{D\times d_v}$ и $hd_k = hd_v = D$.

2. **Позиционно-независимая полносвязная сеть (Feed-Forward Network, FFN)**:

$$\mathrm{FFN}(\mathbf{Z}) = \sigma(\mathbf{Z}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2, \tag{41}$$

обычно с функцией активации $\sigma = \mathrm{GELU}$, $\mathbf{W}_1 \in \mathbb{R}^{D\times 4D}$, $\mathbf{W}_2 \in \mathbb{R}^{4D\times D}$.

Каждый подслой окружён остаточным соединением и нормализацией по слоям (LayerNorm):

$$\mathbf{Z}' = \mathbf{Z} + \mathrm{MSA}(\mathrm{LayerNorm}(\mathbf{Z})), \tag{42}$$
$$\mathbf{Z}'' = \mathbf{Z}' + \mathrm{FFN}(\mathrm{LayerNorm}(\mathbf{Z}')). \tag{43}$$

После L слоёв финальный токен `[CLS]` $\mathbf{z}_0^{(L)}$ используется для классификации с помощью линейного классификатора.

Ключевые проектные решения.

- **Отсутствие индуктивного смещения в пользу локальности или эквивариантности к сдвигу**: в отличие от свёрточных нейронных сетей (CNN), ViT не предполагает пространственной структуры, кроме фиксированного позиционного кодирования. Все взаимодействия между патчами изучаются через механизм внимания.

- **Глобальный контекст уже на первом слое**: каждый патч сразу же взаимодействует со всеми остальными, что позволяет учитывать длиннодистантные зависимости без иерархического построения признаков.

- **Высокая потребность в данных**: ViT требует масштабной предварительной подготовки на больших наборах данных для достижения результатов, сопоставимых с CNN; на малых наборах данных лучше работают гибридные архитектуры CNN–трансформер.

Основные архитектуры на основе ViT. С момента появления ViT было предложено множество модификаций, направленных на повышение эффективности, масштабируемости или универсальности базовой архитектуры:

- **DeiT** [23]: вводит дистилляционный токен и обучение с учителем, что позволяет достигать конкурентоспособных результатов без предобучения на миллиардных наборах данных.

- **Swin Transformer** [16]: заменяет глобальное самовнимание на *сдвинутое оконное внимание*, ограничивая взаимодействие патчей локальными окнами и обеспечивая иерархическое построение карт признаков (аналогично CNN). Сложность снижается до $O(HW)$.

- **ConvNeXt** [17]: демонстрирует, что модернизированные CNN с элементами, заимствованными из ViT (например, LayerNorm, GELU, крупные ядра свёрток), могут соответствовать или превосходить ViT по качеству.

- **FasterViT** [9]: сочетает иерархическую свёрточную подвыборку с гибридным локально-глобальным вниманием для ускорения сходимости и снижения потребления памяти.

- **MobileViT** [18]: объединяет лёгкие CNN с блоками ViT для развёртывания на мобильных устройствах, применяя локальные свёртки с последующим глобальным вниманием над пониженными по разрешению признаками.

Несмотря на различия в деталях, почти все эти модели сохраняют основную структуру ViT: разбиение на патчи → встраивание → добавление позиционных вложений → применение блоков трансформера. Основные инновации касаются способа вычисления внимания (глобального или локального), методов понижения разрешения признаков и способов повторного введения индуктивных смещений.

7.2. Архитектуры DETR Decoder

Detection Transformer (DETR) [4] предложил первую полностью сквозную архитектуру обнаружения объектов, исключающую ручные компоненты, такие как генерация якорей, подавление немаксимумов (NMS) или сети предложений регионов. Вместо этого DETR формулирует задачу обнаружения как прямую задачу предсказания множества, решаемую с помощью архитектуры трансформера «энкодер–декодер». Такой подход знаменует смену парадигмы: переход от эвристической постобработки к обучаемому, инвариантному к перестановке декодированию множеств.

Общая архитектура. DETR принимает изображение $\mathbf{X} \in \mathbb{R}^{H\times W\times 3}$ и пропускает его через свёрточный backbone (например, ResNet-50), получая карту признаков $\mathbf{F} \in \mathbb{R}^{h\times w\times C}$. Эта карта выравнивается в последовательность из $N = hw$ токенов и проецируется в пространство размерности d:

$$\mathbf{Z}_{\text{enc}} = \text{Linear}(\text{vec}(\mathbf{F})) + \mathbf{P},$$

где $\mathbf{P} \in \mathbb{R}^{N\times d}$ двумерные позиционные кодировки. Энкодер, состоящий из стека стандартных слоёв трансформера, уточняет эти контекстуализированные признаки.

Декодер получает два входа:

1. Фиксированный набор из M обучаемых объектных запросов $\mathbf{Q} \in \mathbb{R}^{M\times d}$, которые играют роль «слотов» для потенциальных объектов.

2. Закодированные признаки изображения $\mathbf{Z}_{\text{enc}}$ от энкодера.

Каждый слой декодера выполняет:

- **Многоголовое самовнимание** над объектными запросами (для моделирования взаимосвязей между объектами),

- **Многоголовое кросс-внимание** между запросами и признаками энкодера (для локализации и классификации объектов).

После L слоёв декодера каждый запрос независимо формирует предсказание:

$$\hat{\mathbf{y}}_i = (\hat{c}_i, \hat{\mathbf{b}}_i) \in \mathcal{Y}, \quad i = 1, \ldots, M,$$

где $\hat{c}_i$ – распределение вероятностей по $K+1$ классам (включая класс «нет объекта»), а $\hat{\mathbf{b}}_i = (x, y, w, h)$ – ограничивающий прямоугольник в нормализованных координатах.

Таким образом, DETR выдаёт множество фиксированного размера $\hat{\mathbf{Y}} = \{\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_M\}$.

Предсказание множества через двудольное согласование. Ключевой инновацией DETR является инвариантная к перестановке функция потерь, основанная на оптимальном двудольном согласовании. Поскольку эталонное множество $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_{|\mathbf{Y}|}\}$ имеет переменный размер ($|\mathbf{Y}| \ll M$), DETR должен сопоставить предсказания целям без использования якорей или NMS.

Это достигается поиском биекции $\sigma \in \Sigma_M$ (перестановки M элементов), минимизирующей суммарную стоимость согласования:

$$\hat{\sigma} = \arg\min_{\sigma\in\Sigma_M} \sum_{i=1}^{|\mathbf{Y}|} \mathcal{L}_{\text{match}}(\mathbf{y}_i, \hat{\mathbf{y}}_{\sigma(i)}) + \sum_{i=|\mathbf{Y}|+1}^{M} 1_{\text{noobj}}, \tag{44}$$

где $\mathcal{L}_{\text{match}}$ взвешенная сумма стоимостей классификации и локализации:

$$\mathcal{L}_{\text{match}}(\mathbf{y}, \hat{\mathbf{y}}) = \lambda_{\text{cls}} \cdot 1_{c\neq\varnothing} \cdot \text{CE}(c, \hat{c}) - \lambda_{\text{giou}} \cdot 1_{c\neq\varnothing} \cdot \text{GIoU}(\mathbf{b}, \hat{\mathbf{b}}). \tag{45}$$

Здесь CE – кросс-энтропия, GIoU – обобщённый IoU, а $\lambda_{\text{cls}}, \lambda_{\text{giou}}$ — весовые коэффициенты. Венгерский алгоритм эффективно решает эту задачу назначения за время $O(M^3)$.

После нахождения $\hat{\sigma}$ итоговая функция потерь DETR записывается как

$$\mathcal{L}_{\text{DETR}} = \sum_{i=1}^{M} \Big[\lambda_{\text{cls}} \cdot \text{CE}(c_{\hat{\sigma}(i)}, \hat{c}_i) \tag{46}$$

$$+ 1_{c_{\hat{\sigma}(i)} \neq \varnothing} \cdot \big(\lambda_{\text{l1}} \cdot \|\mathbf{b}_{\hat{\sigma}(i)} - \hat{\mathbf{b}}_i\|_1 + \lambda_{\text{giou}} \cdot (1 - \text{GIoU}(\mathbf{b}_{\hat{\sigma}(i)}, \hat{\mathbf{b}}_i)) \big) \Big]. \tag{47}$$

CIoU и уточнение локализации. Хотя изначально DETR использовал GIoU, в последующих работах (и на практике) его часто заменяют на CIoU (Complete IoU), который добавляет три геометрических компонента к функции потерь IoU:

$$\text{CIoU} = \text{IoU} - \frac{\rho^2(\mathbf{b}, \hat{\mathbf{b}})}{c^2} - \alpha v, \tag{48}$$

где:

- $\rho^2(\mathbf{b}, \hat{\mathbf{b}})$ – квадрат евклидова расстояния между центрами прямоугольников,
- c – длина диагонали наименьшего охватывающего прямоугольника,
- $v = \frac{4}{\pi^2} \left(\arctan \frac{w}{h} - \arctan \frac{\hat{w}}{\hat{h}} \right)^2$ измеряет согласованность соотношения сторон,
- $\alpha = \frac{v}{(1-\text{IoU})+v}$ регулирует компромисс между компонентами.

CIoU ускоряет сходимость и повышает точность локализации за счёт явного штрафа за несовпадение центров и соотношений сторон факторов, критически важных для высокоточного обнаружения.

Принципы дизайна декодера.

- **Объектные запросы – это обучаемые вложения**, не зависящие от изображения; они кодируют «что искать» и «где», используя кросс-внимание.
- **Отсутствие рекуррентности или итеративного уточнения**: в отличие от детекторов на основе RNN, DETR предсказывает все объекты параллельно за один проход.
- **Фиксированная мощность множества**: количество предсказаний M – гиперпараметр; «пустые» слоты обучаются предсказывать класс «нет объекта».

- **Медленная сходимость**: из-за глобального характера внимания и слабых сигналов обратной связи на ранних этапах обучения DETR обычно требует длительного обучения на COCO; эта проблема позже была решена в Deformable DETR [26].

Расширения DETR:

1. Deformable DETR: использует разреженное деформируемое внимание для снижения сложности и ускорения сходимости,

2. Conditional DETR: усиливает содержание запросов пространственными приоритетами,

3. DAB-DETR: разделяет запрос на явные позиционную и содержательную компоненты,

4. DN-DETR: добавляет вспомогательные задачи по подавлению шума для ускорения обучения.

В совокупности эти работы подтверждают ключевую идею механизма DETR: задачу обнаружения объектов можно рассматривать как прямую задачу предсказания множества, решаемую с помощью декодера трансформера и оптимального согласования, заменяющего устаревшие эвристические методы специфичной для детекции постобработки в единой унифицированной архитектуре.

7.3. Практика: реализация

Предложеное практическое задание ориентировано на получение опыта работы с двумя фундаментальными архитектурами трансформеров в компьютерном зрении: (1) DETR для сквозного обнаружения объектов и (2) Vision Transformer (ViT) для классификации изображений. Вы можете выбрать любой общедоступный набор данных, соответствующий каждой задаче.

Теоретические и проектные вопросы

1. Почему DETR требует значительно больше эпох обучения по сравнению со свёрточными детекторами (например, Faster R-CNN)? Как двудольное согласование влияет на поток градиентов?

2. Почему в ViT предобучение на крупных наборах данных (например, ImageNet-21k) часто необходимо для достижения высоких результатов на малых целевых задачах? Какую роль играют позиционные вложения при дообучении на не квадратных изображениях?

3. Объясните, почему фиксированное предсказание множества в DETR исключает необходимость в подавлении немаксимумов (NMS). Может ли NMS всё же быть полезен в конвейере DETR? Обоснуйте.

Практические задания

1. **Обнаружение объектов с помощью DETR**:
   - Выберите набор данных для детекции (например, COCO, Pascal VOC, BDD100K или любой пользовательский набор).
   - Используйте предобученную модель DETR (например, из библиотеки Hugging Face `transformers`).
   - Дообучите модель на выбранном наборе данных в течение нескольких эпох.
   - Замените голову классификации, если в вашем наборе данных другое количество классов.
   - Убедитесь в корректной нормализации ограничивающих прямоугольников (координаты в диапазоне $[0, 1]$).
2. **Классификация изображений с помощью ViT**:
   - Выберите набор данных для классификации (например, CIFAR-10/100, Food-101, Stanford Cars или любой пользовательский набор).
   - Используйте предобученную модель ViT.
   - Дообучите всю модель (или хотя бы последние несколько слоёв и классификатор) в течение нескольких эпох.
   - Измените размер изображений до ожидаемого разрешения модели (например, 224×224).
   - Примените стандартные аугментации (случайное обрезание, отражение, нормализация).
3. **Метрики оценки**: Для DETR:
   - Сообщите значения **mAP@.5** и **mAP@.5:.95** на валидационной выборке с использованием официальной оценки в стиле COCO (через `pycocotools` или аналог).
   - Укажите AP по классам, если в наборе данных не более 10 классов.

   Для ViT:

- Сообщите значения **точности top-1** и **точности top-5** (если применимо).
- Постройте графики обучения/валидации (потери и точность).

4. **Качественный анализ**:
   - Для DETR: визуализируйте 5 предсказаний с ограничивающими прямоугольниками и метками классов. Отметьте случаи ложных срабатываний и пропусков.
   - Для ViT: используйте attention rollout или Grad-CAM для визуализации областей изображения, повлиявших на предсказание, для 3 правильных и 2 неправильных примеров.

Требования к оформлению

- Один Jupyter/Colab NoteBook с чётко разделёнными секциями для DETR и ViT.
- Весь код должен использовать открытые библиотеки (PyTorch, Hugging Face `transformers`, `torchvision` и др.). Проприетарные фреймворки не должны использоваться.
- Укажите ссылки на наборы данных и обоснование выбора (не более 50 слов на набор).
- Все метрики должны быть вычислены на официальной валидационной выборке (без утечки из тестовой).
- Визуализации должны быть чёткими, снабжены подписями и встроены непосредственно в ноутбук.
- Добавьте краткое описание (не более 200 слов):

Критерии сдачи: Для успешного выполнения лабораторной работы необходимо: (1) корректно ответить на все теоретические вопросы, (2) успешно дообучить обе модели без ошибок, (3) представить корректные метрики mAP и точности, (4) включить качественные визуализации.

8. Лабораторная работа №6. Трансформеры в NLP

8.1. Модели только с энкодером и только с декодером

Архитектуры на основе трансформеров в области обработки естественного языка (NLP) условно делятся на три парадигмы: только с энкодером, только с декодером и энкодер–декодер. В данном подразделе мы сосредоточимся на первых двух, лежащих в основе современного понимания и генерации языка, и рассмотрим их архитектурные различия, целевые функции обучения и прикладные задачи.

Модели только с энкодером: BERT и двунаправленный контекст. Двунаправленные представления энкодера на основе трансформеров (BERT, Bidirectional Encoder Representations from Transformers) [7] популяризировали парадигму «только энкодер» для задач понимания языка. Модель BERT состоит исключительно из стека слоёв энкодера трансформера и за один проход обрабатывает входную последовательность, формируя контекстуализированные представления для каждого токена за счёт одновременного внимания ко всем другим токенам как слева, так и справа.

Пусть входная последовательность токенов задана как $\mathbf{X} = [x_1, \ldots, x_n]$. BERT сначала преобразует токены во вложения:

$$\mathbf{E} = \mathbf{T} + \mathbf{P} + \mathbf{S},$$

где: $\mathbf{T} \in \mathbb{R}^{n \times d}$ – токенные эмбеддинги, $\mathbf{P} \in \mathbb{R}^{n \times d}$ – позиционные эмбеддинги (обучаемые), $\mathbf{S} \in \mathbb{R}^{n \times d}$ – сегментные (пары предложений) эмбеддинги.

Полученная последовательность эмбеддингов затем пропускается через L идентичных блоков энкодера трансформера, каждый из которых содержит многоголовое самовнимание и позиционно-зависимую полносвязную сеть с нормализацией по слоям (LayerNorm) и остаточными связями.

Важно отметить, что BERT не обучается автогрессивно, вместо этого используется маскированное языковое моделирование (MLM): на этапе предобучения 15% входных токенов случайным образом маскируются (заменяются на `[MASK]`), и модель должна восстановить исходные значения этих токенов, используя полный двунаправленный контекст. Дополнительной задачей является предсказание следующего предложения (NSP, next sentence prediction), при котором модель учится классифицировать, следуют ли два сегмента друг за другом в исходном корпусе.

Благодаря своей двунаправленной природе BERT демонстрирует высокую эффективность в дискриминативных задачах, требующих глубокого понимания семантики предложения или текстового фрагмента, таких как: (1) классификация текста (например, анализ тональности), (2) распознавание именованных сущностей (NER), (3) ответы на вопросы (например, SQuAD), (4) логический вывод на естественном языке (NLI).

Однако BERT напрямую не может использоваться для генерации текста, поскольку не обладает механизмом последовательного порождения токенов: его выход представляет собой набор статических представлений, а не распределение вероятностей над будущими токенами.

Такие варианты, как RoBERTa [15], ALBERT [12] и DeBERTa [10], улучшают стратегию предобучения или архитектуру BERT, но сохраняют базовую структуру «только энкодер».

Модели только с декодером: GPT и автогрессивная генерация. Напротив, серия моделей Generative Pretrained Transformer (GPT) [19, 20, 2] использует архитектуру «только декодер». Она состоит исключительно из блоков декодера трансформера, но с существенным изменением: подслой самовнимания делается каузальным (применяется маскирование верхнетреугольной матрицы, чтобы токен i обращал внимание только к токенам $j \leq i$). Это обеспечивает автогрессивное моделирование: модель учится совместному распределению последовательности по правилу цепочки:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid x_1, \ldots, x_{i-1}). \tag{49}$$

На этапе предобучения GPT обучается с помощью стандартного языкового моделирования (LM): по заданной последовательности токенов предсказывается следующий токен. Маскирование не применяется; модель видит всю историю до текущей позиции.

Выход последнего слоя декодера проецируется через классификатор с привязкой весов к токенным эмбеддингам, формируя логиты над словарём:

$$\mathbf{Z} = \mathrm{DecoderOnly}(\mathbf{X}), \quad \mathbf{L} = \mathbf{Z}\mathbf{W}_E^{\mathsf{T}}, \quad P(x_i \mid x_{<i}) = \mathrm{softmax}(\mathbf{L}_{i-1}), \tag{50}$$

где $\mathbf{W}_E$ матрица токенных эмбеддингов (привязка весов повышает эффективность обучения по данным).

Благодаря автогрессивной природе GPT изначально является генеративной моделью: она может генерировать связный и беглый текст, последовательно предсказывая следующий токен. Это делает её идеальной для следующих задач: (1) завершение текста и генерация рассказов, (2) диалоговые системы, (3) синтез кода (например, Codex), (4) промптинг без примеров (zero-shot) и с несколькими примерами (few-shot) (особенно в GPT-3 и последующих версиях).

В отличие от BERT, GPT не имеет доступа к будущему контексту, что ограничивает её эффективность в задачах, требующих анализа всей последовательности (например, извлечение фрагментов). Однако простота (единая целевая функция) и хорошие свойства масштабируемости сделали её основой крупных языковых моделей (LLM).

Последующие версии (GPT-2, GPT-3, GPT-4) отличаются в основном масштабом (миллиарды параметров), объёмом обучающих данных и инженерными оптимизациями (например, разреженное внимание, улучшенная токенизация), но сохраняют ту же самую автогрессивную архитектуру «только декодер».

Таблица 2: Сравнение архитектур BERT (только энкодер) и GPT (только декодер).

Свойство	**BERT (только энкодер)**	**GPT (только декодер)**
Тип внимания	Полное самовнимание (двунаправленное)	Каузальное самовнимание (однонаправленное)
Цель предобучения	Маскированное языковое моделирование + NSP	Стандартное языковое моделирование (предсказание следующего токена)
Видимость токенов	Все токены (включая будущие)	Только прошлые и текущие токены
Основное применение	Задачи понимания (классификация, ответы на вопросы)	Задачи генерации (завершение, диалог)
Поддержка генерации текста «из коробки»	Нет (требуются модификации)	Да (изначально автогрессивна)
Кодирование позиций	Обучаемые эмбеддинги	Обучаемые (или синусоидальные в ранних версиях)

Сравнение ключевых архитектурных особенностей. Эти две парадигмы, двунаправленное кодирование для понимания и каузальная декодировка для генерации, составляют основу современной NLP. Хотя гибридные модели (например, T5, BART) объединяют оба подхода для задач типа «последовательность-в-последовательность», многие практические системы по-прежнему используют модели только с энкодером или только с декодером в зависимости от того, является ли главной целью понимание или создание.

8.2. Энкодер–декодер

Хотя трансформеры только с энкодером и только с декодером отлично справляются с пониманием и генерацией соответственно, многие реальные задачи требуют тесной связи между ними: сначала *закодировать* структурированный вход (например, предложение, изображение или таблицу), а затем *декодировать* зависимый от него выход (например, перевод, подпись или ответ). Архитектура энкодер–декодер, впервые предложенная в оригинальном трансформере [24] и усовершенствованная в таких моделях, как T5, предоставляет унифицированную основу для подобных задач типа «последовательность-в-последовательность» (seq2seq).

T5: Text-to-Text Transfer Transformer. Модель T5 [21] переформулирует все NLP-задачи как преобразования текста в текст: вход и выход всегда являются строками, дополненными префиксами, указывающими на тип задачи (например, «переведи с английского на немецкий: Hello» $\rightarrow$ «Hallo»). Такая унификация позволяет использовать одну архитектуру для классификации, суммаризации, перевода и других задач.

Архитектурно T5 состоит из: (1) стека энкодера (обычно 12–24 слоя), который преобразует входную последовательность $\mathbf{X} = [x_1, \ldots, x_n]$ в контекстуализированные представления $\mathbf{H}^{\text{enc}} \in \mathbb{R}^{n \times d}$, (2) стека декодера (той

же глубины), который автогрессивно генерирует выходную последовательность $\mathbf{Y} = [y_1, \ldots, y_m]$, используя:

- самовнимание декодера (с каузальной маской),
- кросс-внимание к $\mathbf{H}^{\text{enc}}$ на каждом слое.

Важно, что T5 использует общие токенные эмбеддинги и относительное позиционное смещение вместо абсолютного позиционного кодирования, что улучшает обобщение на более длинные последовательности. Успех T5 показывает, что хорошо обученная архитектура энкодер–декодер может соответствовать или превосходить специализированные модели в самых разных задачах при достаточном масштабе и объёме данных.

Мультимодальные модели энкодер–декодер: выравнивание модальностей в пространстве LLM. Недавние достижения в области ИИ расширяют парадигму энкодер–декодер за пределы текста, используя парадигму модально-специфичных энкодеров для проецирования немовербальных данных (изображений, аудио, видео) в эмбеддинговое пространство крупной языковой модели (LLM), которая выступает в роли универсального декодера.

Общий конвейер выглядит следующим образом:

$$\text{Модальность } \mathcal{M} \xrightarrow{\text{Энкодер } f_{\mathcal{M}}} \mathbf{Z}_{\mathcal{M}} \in \mathbb{R}^{L \times d} \xrightarrow{\text{Проектор } \mathbf{W}_{\mathcal{M}}} \mathbf{E}_{\mathcal{M}} \in \mathbb{R}^{L \times d_{\text{LLM}}} \tag{51}$$

$$\xrightarrow{\text{Декодер LLM}} \text{Текстовый вывод.} \tag{52}$$

Ключевые принципы проектирования:

- LLM остаётся замороженной (или слегка дообучается), сохраняя свои лингвистические априорные знания.
- Энкодер модальности (например, ViT для изображений, Whisper для речи) обучается или адаптируется так, чтобы его признаки были совместимы с ожидаемым распределением входа LLM.
- Обучаемый линейный или MLP-проектор связывает пространства признаков, часто обучаясь с использованием контрастивных потерь или предсказания следующего токена.

Известные примеры включают:

- **Flamingo** [1]: использует предобученный энкодер изображений, подобный CLIP; визуальные токены чередуются с текстовыми и подаются в замороженную LLM Chinchilla через затворённое кросс-внимание.

- **BLIP-2** [13]: применяет Q-Former (трансформер с запросами) для сжатия признаков ViT в небольшой набор «мягких промптов», которые добавляются перед входом LLM.
- **LLaVA** [14]: проецирует эмбеддинги патчей ViT напрямую в пространство токенных эмбеддингов LLaMA с помощью простого линейного слоя, обеспечивая сквозное обучение по инструкциям.
- **Kosmos-1/2** [11]: интегрирует текст, изображения и сигналы привязки в единую мультимодальную LLM с кросс-модальным выравниванием на этапе предобучения.

Такой подход, включающий замороженную LLM и обучаемый адаптер модальности, позволяет быстро разрабатывать мультимодальные системы без повторного обучения миллиардных декодеров, одновременно используя способности LLM к возникающему рассуждению и композиционности.

Применения. Архитектуры энкодер–декодер и мультимодальные энкодер–LLM лежат в основе широкого спектра приложений:

- **Машинный перевод и суммаризация**: классические seq2seq-задачи, где преуспевают T5 и mT5.
- **Ответы на вопросы по изображениям (VQA)**: кодирование изображения и вопроса и, затем, генерация ответа (например, LLaVA, PaLI).
- **Генерация подписей к изображениям**: кодирование изображения и генерация описательного текста (например, BLIP, GIT).
- **Речь-в-текст / речь-в-речь**: аудио-энкодер, LLM, текст или синтезированный голос (например, Whisper + LLM, VALL-E).
- **Мультимодальные рассуждения**: решение текстовых математических задач с диаграммами, интерпретация графиков, рассуждения по видеокадрам.
- **Робототехника и воплощённый ИИ**: объединение данных с камер, лидаров и проприоцепции в последовательности действий, обусловленные языком.

Критически важно, что такие системы наследуют способность LLM к обобщению без обучения (zero-shot): после выравнивания они могут выполнять новые инструкции (например, «опиши это изображение, как будто ты поэт») без дообучения под конкретную задачу.

Проблемы и компромиссы. Несмотря на гибкость, мультимодальные модели энкодер–декодер сталкиваются с рядом проблем:

- **Разрыв выравнивания**: несоответствие между семантикой изображений/аудио и лингвистическим пространством токенов может вызывать галлюцинации.
- **Узкое место контекста**: визуальные признаки часто требуют сотен токенов, конкурируя с длинными текстовыми промптами за ограниченный контекст LLM.
- **Эффективность обучения**: сквозное (дорогостояще по ресурсам) обучение; большинство систем используют поэтапное обучение (контрастивное предобучение → дообучение по инструкциям).

Тем не менее, архитектура энкодер–декодер с чётким разделением восприятия (энкодер) и познания (декодер) остаётся доминирующей парадигмой при создании универсальных мультимодальных интеллектуальных систем.

8.3. Практическое задание

В рамках данного задания необходимо реализовать и дообучить лёгкую мультимодальную модель (VLM) для генерации подписей к изображениям канонической кросс-модальной задачи, требующей согласования визуального восприятия с лингвистической генерацией. Для этого необходимо выбрать любой общедоступный набор данных «изображение–подпись».

Теоретические и проектные вопросы

1. Почему архитектура энкодер–декодер лучше подходит для генерации подписей к изображениям, чем LLM только с декодером? Какова роль визуального энкодера?
2. Объясните, как кросс-внимание позволяет языковому декодеру «привязывать» свои предсказания к визуальному содержимому. На каком уровне стека декодера обычно вставляется кросс-внимание?
3. Какова цель обучаемых «токенов-запросов» или «префиксных эмбеддингов», соединяющих визуальный энкодер и языковый декодер? Можно ли заменить их усреднёнными визуальными признаками? Почему да или почему нет?
4. Почему замороженный декодер LLM (например, LLaMA, OPT) всё ещё может генерировать осмысленные подписи при использовании обучаемого визуального адаптера? Какие лингвистические априорные знания он предоставляет?

Задачи по реализации

1. **Выбор набора данных**: Выберите один набор данных «изображение–подпись»:
   - COCO Captions (120 тыс. изображений, по 5 подписей на каждое),
   - Flickr30k (30 тыс. изображений, по 5 подписей),
   - Conceptual Captions (упрощённое подмножество),
   - Пользовательский набор (например, отфильтрованный LAION или предметно-ориентированный, например, медицинские изображения с подписями).
2. **Архитектура модели**: Постройте минимальную VLM, используя следующий подход:
   (на основе LLM): (1) визуальный энкодер: CLIP ViT-B/16 или ViT-Tiny, (2) языковой декодер: из Hugging Face (замороженный или частично дообучаемый), (3) адаптер: обучаемый линейный/MLP-проектор из визуальных токенов в пространство токенов LLM. Также допускается использование предобученной VLM, например, Qwen-VL.
3. **Метрики оценки**: Приведите результаты на валидационной выборке:
   - BLEU-1/4, METEOR, ROUGE-L, CIDEr (используйте библиотеку `pycocoevalcap`),
   - Качественные примеры: покажите 5 изображений с эталонными и предсказанными подписями,
   - Анализ ошибок: укажите 2 случая, когда модель галлюцинирует или пропускает ключевые объекты.
4. **Профилирование эффективности**: Для финальной модели измерьте:
   - Время вывода на одно изображение (мс),
   - Пиковое потребление видеопамяти при прямом проходе (МБ),
   - Общее число обучаемых параметров (в сравнении с общим числом параметров при использовании замороженной LLM).

Требования к оформлению

- Один Jupyter/Colab Notebook с разделами, помеченными номерами заданий.

- Весь код должен использовать открытые библиотеки (PyTorch, Hugging Face `transformers`, `timm` и др.).
- Укажите ссылку на набор данных и детали предобработки.
- Визуализации должны включать изображения с эталонными и предсказанными подписями.
- Добавьте краткое описание (не более 200 слов).

Критерии оценки работы: Для успешного прохождения лабораторной работы необходимо: (1) корректно ответить на все теоретические вопросы, (2) успешно обучить VLM, генерирующую связные и релевантные подписи, (3) привести стандартные метрики качества подписей, (4) включить качественные примеры с анализом ошибок.

9. Лабораторная работа №7. Авторегрессионные трансформеры: обучение и инференс

9.1. Вероятностные основы: от теоремы Байеса к цепному правилу

В основе больших языковых моделей (Large Language Models, LLM) лежит вероятностная формулировка языка. Несмотря на то, что современные архитектуры опираются на глубокие нейронные сети для аппроксимации сложных функций, их целевая задача остаётся укоренённой в классической теории вероятностей. Чтобы понять, как трансформеры генерируют текст, необходимо прежде всего формализовать понятие вероятности последовательности. Хотя математический вывод опирается преимущественно на **цепное правило вероятности**, интуицию можно эффективно оформлен через байесовскую линзу: модель поддерживает состояние убеждённости (распределение) над словарём и последовательно обновляет его по мере поступления нового контекста.

Определение 9.1 (Вероятность последовательности)**.** Пусть последовательность токенов обозначена как $X = (x_1, x_2, \ldots, x_T)$, где каждый x_t принадлежит конечному словарю $\mathcal{V}$. Совместная вероятность наблюдения этой конкретной последовательности определяется как $P(X)$. В контексте языкового моделирования цель состоит в оценке этого совместного распределения на основе данных.

Используя цепное правило вероятности, любое совместное распределение может быть разложено в произведение условных вероятностей:

$$P(X) = P(x_1) \cdot P(x_2|x_1) \cdot P(x_3|x_1, x_2) \cdots P(x_T|x_{<T}) = \prod_{t=1}^{T} P(x_t|x_{<t}), \quad (53)$$

где $x_{<t}$ обозначает последовательность всех токенов, предшествующих позиции t. Это разложение является точным и не требует предположений о независимости. Однако прямая оценка $P(x_t|x_{<t})$ является вычислительно неразрешимой для длинных последовательностей из-за экспоненциального роста количества возможных контекстов. Традиционные n-граммные модели решали эту проблему через марковское предположение, ограничивая контекст фиксированным окном, что приводило к потере долгосрочных зависимостей.

Ремарка 9.2 (Байесовская интерпретация). Хотя уравнение 53 выведено из цепного правила, оно тесно согласуется с байесовским обновлением. Рассмотрим предсказание на шаге t как вывод апостериорной вероятности следующего токена наблюдаемых свидетельств (контекста). В байесовских терминах:

$$P(x_t|x_{<t}) \propto P(x_{<t}|x_t) \cdot P(x_t), \quad (54)$$

где $P(x_t)$ представляет априорную уверенность в токене (например, его унарную частоту), а $P(x_{<t}|x_t)$ представляет правдоподобие контекста при данном токене. Архитектура трансформера выступает в роли аппроксиматора функции правдоподобия, трансформируя априорный контекст в уточнённое вероятностное распределение над словарём. По мере роста окна контекста модель обновляет свою «уверенность» в том, какой токен должен следовать далее, подавляя маловероятные кандидаты и усиливая те, которые согласуются с семантической траекторией.

Эта интерпретация носит **концептуальный характер**: авторегрессионные трансформеры **не вычисляют явно** члены правдоподобия $P(x_{<t} \mid x_t)$, а **непосредственно обучаются моделировать условное распределение** $P_\theta(x_t \mid x_{<t})$.

Данная вероятностная формулировка требует **авторегрессионный** подход. Авторегрессия подразумевает, что выход в момент времени t регрессирует на предыдущие выходы. В трансформерах это обеспечивается посредством **каузального маскирования** , гарантируя, что предсказание для позиции t зависит только от позиций $1 \dots t$.

Определение 9.3 (Авторегрессионный трансформер)**.** Авторегрессионный трансформер моделирует условную вероятность $P(x_t|x_{<t})$ используя архитектуру только с декодером (decoder-only). Параметры модели θ оптимизируются для максимизации логарифмического правдоподобия обучающих данных:

$$\mathcal{J}(\theta) = \sum_{t=1}^{T} \log P_\theta(x_t|x_{<t}). \tag{55}$$

Эта формулировка позволяет модели генерировать связный текст путём итеративной выборки из предсказанного распределения. В отличие от моделей только с энкодером (например, BERT), которые маскируют токены и полагаются на двунаправленный контекст, авторегрессионные модели строго придерживаются временного порядка, определённого в уравнении 53.

Ремарка 9.4 (Предположение Маркова против Трансформеров). Традиционные n-граммные модели аппроксимируют уравнение 53, предполагая свойство Маркова, где $P(x_t|x_{<t}) \approx P(x_t|x_{t-n+1}, \dots, x_{t-1})$. Это ограничивает контекст фиксированным окном n. Трансформеры, однако, используют механизмы самовнимания для обуславливания x_t на всей истории $x_{<t}$ (в пределах максимального окна контекста). Это устраняет строгое предположение Маркова, позволяя зависимостям охватывать сотни или тысячи токенов, хотя практические ограничения (см. Лабораторную работу №3) всё ещё ограничивают эффективную длину контекста.

Переход от теоретической вероятности к практическому обучению включает оптимизацию уравнения 55. В следующих разделах мы подробно опи-

шем, как эта цель реализуется с использованием функции потерь перекрёстной энтропии со сдвинутыми метками, и как современные LLM масштабируют этот фундаментальный принцип до миллиардов параметров.

9.2. Механика обучения: перекрёстная энтропия и сдвинутые метки

Установив вероятностный фундамент авторегрессионного моделирования в Разделе 7.1, мы переходим к практической механике обучения таких моделей. Теоретическая цель максимизация логарифмического правдоподобия данных должна быть переведена в дифференцируемую функцию потерь, пригодную для стохастического градиентного спуска. В этом разделе подробно описывается построение обучающих пар посредством сдвига меток, формулировка функции потерь перекрёстной энтропии и педагогическая стратегия, известная как teacher forcing.

Определение 9.5 (Сдвинутые метки)**.** Пусть обучающая последовательность токенов обозначена как $X = (x_1, x_2, \ldots, x_T)$. Для обучения авторегрессионной модели предсказанию следующего токена на основе прошлого, последовательность разделяется на входной и целевой тензоры путём сдвига индексов на одну позицию. Входная последовательность X_{in} и целевая последовательность X_{out} определяются как

$$X_{\text{in}} = (x_1, x_2, \ldots, x_{T-1}), \tag{56}$$

$$X_{\text{out}} = (x_2, x_3, \ldots, x_T). \tag{57}$$

На каждом временном шаге $t \in \{1, \ldots, T-1\}$ модель получает x_t в качестве входа и должна предсказать x_{t+1} в качестве цели. Это выравнивание гарантирует, что модель обучается условной вероятности $P(x_{t+1}|x_{\leq t})$ для каждой позиции в последовательности одновременно во время обучения.

Функция потерь. Стандартной целью для обучения авторегрессионных трансформеров является **функция потерь перекрёстной энтропии (Cross-Entropy Loss)**, что эквивалентно минимизации отрицательного логарифмического правдоподобия (NLL), выведенного в уравнении 55. Для батча из N последовательностей потеря $\mathcal{L}$ вычисляется как средняя отрицательная логарифмическая вероятность, присвоенная целевым токенам

$$\mathcal{L}(\theta) = -\frac{1}{N(T-1)} \sum_{i=1}^{N} \sum_{t=1}^{T-1} \log\left(P_\theta(x_{i,t+1}|x_{i,\leq t})\right). \tag{58}$$

На практике модель выдаёт логиты $Z \in \mathbb{R}^{(T-1)\times|\mathcal{V}|}$, где $|\mathcal{V}|$ – размер словаря. Распределение вероятностей получается посредством функции softmax

по размерности словаря. Функция потерь перекрёстной энтропии для одного шага предсказания имеет вид:

$$\mathrm{CE}(z, y) = -\log\left(\frac{\exp(z_y)}{\sum_{j=1}^{|\mathcal{V}|}\exp(z_j)}\right), \tag{59}$$

где z – вектор логитов, а y – индекс целевого токена. Эта формулировка штрафует модель пропорционально отрицательному логарифму вероятности правильного класса, поощряя уверенные и точные предсказания.

Ремарка 9.6 (Параллелизация во время обучения). Ключевым преимуществом формулировки со сдвинутыми метками является возможность **параллельных вычислений** во время обучения. Хотя модель является авторегрессионной (что означает, что предсказание на шаге t зависит от шага $t-1$), входные данные ground truth X_{in} известны заранее. Следовательно, модель может обработать всю последовательность X_{in} за один прямой проход, используя каузальное маскирование вместо генерации токенов по одному, как требуется во время инференса. Это значительно ускоряет обучение по сравнению с рекуррентными архитектурами.

Teacher Forcing. Стратегия использования токенов ground truth X_{in} в качестве входа вместо собственных предсказанных моделью токенов известна как **teacher forcing**. Формально, на шаге t входом является x_t (из данных), а не $\hat{x}_t$ (сгенерированный моделью):

$$\text{Input}_t = \begin{cases} x_t & \text{(Обучение)} \\ \hat{x}_t & \text{(Инференс)} \end{cases}. \tag{60}$$

Ремарка 9.7 (Смещение экспозиции). В то время как teacher forcing стабилизирует градиенты обучения, предоставляя правильный контекст на каждом шаге, это вносит расхождение между условиями обучения и инференса, известное как *смещение экспозиции (exposure bias)*. Во время обучения модель никогда не сталкивается с собственными ошибками; во время инференса, однако, единственное ошибочное предсказание может распространиться и усугубиться, приводя к ухудшению качества генерации. Для смягчения этого эффекта иногда применяются продвинутые техники, такие как плановая выборка (scheduled sampling) или обучение на уровне последовательности (например, Reinforcement Learning from Human Feedback), хотя стандартная перекрёстная энтропия с teacher forcing остаётся доминирующей парадигмой для предобучения LLM.

Детали реализации. В PyTorch операция сдвига эффективно реализуется с использованием slicing тензоров. Функция потерь обычно игнорирует токены заполнения (padding), чтобы предотвратить их влияние на обновления градиентов.

Подготовка сдвинутых меток в PyTorch

```
# Example: Preparing shifted labels in PyTorch
# Assume input_ids has shape (batch_size, seq_length)

# Input to the model: all tokens except the last
model_input = input_ids[:, :-1]

# Target for loss calculation: all tokens except the first
target_labels = input_ids[:, 1:]

# Forward pass
logits = model(model_input)
# Shape: (batch, seq_length-1, vocab_size)

# Compute loss (PyTorch's CrossEntropyLoss expects
shape (N, C) and (N,))
criterion = nn.CrossEntropyLoss()
loss = criterion(logits.view(-1, logits.size(-1)),
                 target_labels.view(-1))
```

Связь с современными LLM. В масштабных моделях (например, LLaMA, GPT) этот фундаментальный механизм масштабируется до миллиардов параметров и триллионов токенов. К этому процессу применяется ряд инженерных оптимизаций:

1. **Токенизация:** сабвордная токенизация (например, BPE, SentencePiece) уменьшает размер словаря и обрабатывает слова, отсутствующие в словаре, влияя на энтропию целевого распределения.

2. **Смешанная точность:** обучение часто проводится в FP16 или BF16 для уменьшения использования памяти во время прямого и обратного проходов вычисления потерь.

3. **Накопление градиентов:** для симуляции больших размеров батчей без превышения памяти GPU, градиенты накапливаются по нескольким микро-батчам перед обновлением весов, обеспечивая стабильную сходимость потерь.

4. **Маскирование потерь:** при дообучении на инструкциях или многоходовых диалогах потеря часто вычисляется только для токенов ответа, а не токенов промпта. Это достигается установкой целевых меток для позиций промпта в специальный индекс игнорирования.

Понимание этих механик критически важно для практического эксперимента в данной лабораторной работе, где вы реализуете цикл обучения

для мини-LLM и проанализируете, как сходится потеря по мере того, как модель изучает статистическую структуру текста.

9.3. Стратегии инференса и современные LLM

Установив цели обучения и вероятностные основы в Разделах 7.1 и 7.2, мы переходим к фазе инференса. Во время инференса модель генерирует последовательности токен за токеном без доступа к целевым меткам ground truth. Стратегия, используемая для выборки из предсказанного распределения $P(x_t|x_{<t})$, существенно влияет на качество, разнообразие и связность сгенерированного текста. В этом разделе формализуются распространённые стратегии декодирования и проводится связь с инженерными оптимизациями, встречающимися в современных больших языковых моделях (LLM).

Стратегии декодирования

На каждом шаге генерации t модель генерирует вектор логитов $z_t \in \mathbb{R}^{|\mathcal{V}|}$. Распределение вероятностей получается через $p_t = \text{softmax}(z_t)$. Метод выбора следующего токена x_t из p_t определяет стратегию декодирования.

Жадный поиск (Greedy Search). Простейшая стратегия выбирает токен с наибольшей вероятностью на каждом шаге:

$$x_t = \underset{v \in \mathcal{V}}{\text{argmax}}\, P(v|x_{<t}). \tag{61}$$

Несмотря на вычислительную эффективность и детерминированность, жадный поиск часто приводит к повторяющемуся и шаблонному тексту. Всегда выбирая наиболее вероятный путь, модель может попасть в циклы с высокой вероятностью и не исследует разнообразные семантические траектории.

Температурное масштабирование. Для контроля случайности процесса выборки вводится параметр температуры $\tau > 0$. Логиты масштабируются перед применением softmax:

$$P_\tau(v) = \frac{\exp(z_v/\tau)}{\sum_{j \in \mathcal{V}} \exp(z_j/\tau)}. \tag{62}$$

Ремарка 9.8 (Влияние температуры). При $\tau \to 0$ распределение приближается к one-hot вектору на argmax (поведение жадного поиска). При $\tau \to \infty$ распределение становится равномерным, максимизируя энтропию, но теряя семантическую связность. Типичные значения для креативной генерации находятся в диапазоне $\tau \in [0.7, 1.2]$.

Выборка Top-K. Чтобы предотвратить выборку из длинного хвоста маловероятных токенов (которые часто соответствуют шуму), выборка

Top-K ограничивает набор кандидатов K наиболее вероятными токенами. Вероятностная масса перераспределяется среди этих K токенов:

$$\mathcal{V}_K = \mathrm{TopK}(\mathcal{V}, K), \quad P'(v) = \begin{cases} \frac{P(v)}{\sum_{j \in \mathcal{V}_K} P(j)} & \text{if } v \in \mathcal{V}_K \\ 0 & \text{otherwise} \end{cases}. \tag{63}$$

Это гарантирует, что маловероятные токены никогда не будут выбраны, улучшая безопасность и связность, но может отсечь допустимые варианты, если K слишком мало.

Выборка Top-P (Nucleus Sampling). Динамическая альтернатива Top-K, выборка Top-P выбирает наименьшее множество токенов, чья кумулятивная вероятность превышает порог $p \in (0, 1]$:

$$\mathcal{V}_p = \left\{ v \in \mathcal{V} \,\middle|\, \sum_{j \in \mathrm{TopSort}(\mathcal{V})} P(j) \geq p \right\}. \tag{64}$$

Это адаптируется к уверенности модели: когда распределение острое (высокая уверенность), рассматривается меньше токенов; когда плоское (высокая неопределённость), допускается больше токенов. Это часто даёт более естественный текст, чем фиксированный Top-K.

Инженерные оптимизации в современных LLM

Современные LLM (например, LLaMA, Qwen, Mistral) масштабируют авторегрессионный фреймворк до миллиардов параметров. Чтобы сделать инференс осуществимым, применяется ряд архитектурных и системных оптимизаций.

Кэширование ключей и значений (KV-Caching). Как обсуждалось ранее, авторегрессионная генерация включает в себя пересчёт ключей и значений для прошлых токенов на каждом шаге. KV-Caching хранит спроецированные ключи K и значения V в памяти GPU во время генерации. На шаге t только новый токен x_t обрабатывается через проекции запросов, в то время как прошлые K, V извлекаются из кэша. Это уменьшает сложность вычислений на токен, хотя потребление памяти остаётся.

Токенизация. Современные модели используют алгоритмы сабвордной токенизации, такие как Byte-Pair Encoding (BPE) или Unigram. В отличие от моделей на уровне символов (используемых в этой лабе для простоты), сабвордные токенизаторы балансируют размер словаря и длину последовательности. Они обрабатывают слова, отсутствующие в словаре, путём разложения их на известные под-слова, уменьшая энтропию целевого распределения и улучшая эффективность выборки.

Вращательные позиционные кодировки (RoPE). В то время как ранее мы фокусировались на обучаемых или синусоидальных позиционных эмбеддингах, современные LLM преимущественно используют RoPE.

RoPE кодирует информацию о позиции путём вращения векторов запроса и ключа в комплексной плоскости перед скалярным произведением. Это сохраняет информацию об относительной позиции и позволяет лучше экстраполировать на длины последовательностей, не виденные во время обучения, что является критической функцией для длинноконтекстных LLM.

Простой цикл инференса с температурой и Top-K

```
@torch.no_grad()
def generate(model, prompt_ids, max_length, temperature=1.0,
top_k=50):
    model.eval()
    generated = prompt_ids.clone()

    for _ in range(max_length):
        # Forward pass (with KV-cache in production)
        logits = model(generated)[:, -1, :] / temperature

        # Top-K filtering
        if top_k > 0:
            indices_to_remove = (logits
                < torch.topk(logits, top_k)[0][..., -1, None])
            logits[indices_to_remove] = float('-inf')

        # Sampling
        probs = torch.softmax(logits, dim=-1)
        next_token = torch.multinomial(probs, num_samples=1)

        generated = torch.cat([generated, next_token], dim=1)

        if next_token.item() == EOS_TOKEN_ID:
            break

    return generated
```

Системные оптимизации. Помимо архитектуры, движки инференса используют:

- **Непрерывный батчинг (Continuous Batching):** вместо обработки запросов последовательно, новые запросы вставляются в батч, как только предыдущие завершают свою генерацию, максимизируя утилизацию GPU (асинхронная генерация).
- **Квантование:** веса хранятся в меньшей точности (например, INT8, INT4) для уменьшения footprint памяти и увеличения эффективности

пропускной способности, часто с минимальной потерей перплексии.

- **Спекулятивное декодирование:** маленькая модель-«черновик» генерирует несколько токенов быстро, которые затем верифицируются параллельно большей целевой моделью, ускоряя вывода.

Понимание этих стратегий необходимо для практического эксперимента в данной лабораторной работе. Вы реализуете цикл генерации, позволяющий настраивать параметры температуры и Top-K, наблюдая из первых рук, как эти гиперпараметры формируют семантическое качество выхода.

Теоретические вопросы

1. Выведите градиент функции потерь перекрёстной энтропии по логитам z. Почему эта формулировка поощряет уверенные предсказания?

2. Объясните разницу между *перплексией обучения* и *качеством генерации*. Почему модель с низкой перплексией всё ещё может генерировать повторяющийся текст?

3. Как температура τ математически влияет на энтропию выходного распределения? Покажите, что $\tau \to 0$ приближается к жадному поиску.

4. Почему каузальное маскирование недостаточно само по себе для предотвращения утечки информации во время обучения, если предобработка данных выполнена неверно (например, смешивание сэмплов в батче)?

5. Обсудите байесовскую перспективу: как размер окна контекста ограничивает «апостериорную» уверенность, которую модель может сформировать следующего токена?

Задачи по реализации

1. **Подготовка данных.** Загрузите небольшой текстовый корпус (например, TinyShakespeare или WikiText-2). Реализуйте токенизатор на уровне символов или токенов (размер словаря ≈ 256 или 1000). Создайте входные/целевые тензоры с логикой сдвига.

2. **Архитектура модели.** Реализуйте блок трансформера только с декодером (Causal Self-Attention + MLP + LayerNorm). Используйте обучаемые позиционные эмбеддинги. Общее количество параметров должно быть < 10M для быстрого обучения.

3. **Цикл обучения.** Реализуйте цикл обучения с функцией потерь перекрёстной энтропии. Отслеживайте потерю обучения и валидационную перплексию. Обучайте в течение 50 эпох или до сходимости.

4. **Инференс и выборка.** Реализуйте следующую функцию генерации `generate(model, prompt, max_length, temperature, top_k)`. Сгенерируйте следующие текстовые сэмплы с разными температурами ($\tau \in \{0.5, 1.0, 1.5\}$). Реализуйте выборку Top-K.

5. **Профилирование.** Измерьте задержку инференса на токен с KV-кэшированием и без него (опциональный бонус). Сравните использование памяти во время обучения и инференса.

Требования к сдаче

- Один Colab/Kaggle Notebook с чёткими разделами.
- Графики зависимости потери обучения от эпох.
- Сгенерированные текстовые сэмплы для разных температур (включая анализ качества).
- Код должен быть модульным (отдельные классы для Model, DataLoader, Trainer).
- Краткий отчёт (200 слов): Как температура повлияла на семантическую связность сгенерированного текста?

Критерии оценки

Для успешного выполнения лабораторной работы необходимо:

1. Корректно ответить на все теоретические вопросы.
2. Продемонстрировать функциональную генерацию с контролируемым разнообразием (temperature/top-k).
3. Предоставить чёткий анализ компромисса между перплексией и разнообразием генерации.

10. Заключение

Настоящее учебно-методическое пособие, *Введение в нейронные имитации механизмов внимания»*, представляет собой систематическое погружение в теоретические основы, вычислительные ограничения и современные оптимизации механизма внимания – ключевого компонента архитектур глубокого обучения последнего десятилетия. Пособие структурировано как последовательность лабораторных работ, каждая из которых сочетает формальный анализ, реализацию с нуля и эмпирическую верификацию, тем самым формируя у студентов целостное понимание механизма внимания не как чёрного ящика, а как модифицируемого и профилируемого вычислительного примитива.

Первая лабораторная работа закладывает математический фундамент самовнимания: масштабированное скалярное произведение, свойства перестановочной эквивариантности, стохастичности строк матрицы внимания и влияние маскирования (каузального и заполнения). Студенты реализуют прямой и обратный проходы, подтверждая квадратичную сложность $O(L^2)$ как по времени, так и по потреблению видеопамяти.

Вторая работа посвящена кросс-вниманию как асимметричному механизму выравнивания между разнородными последовательностями. Через сравнительный эксперимент на задаче seq2seq демонстрируется его незаменимость для условной генерации, где запросы из декодера динамически извлекают релевантную информацию из энкодера, что невозможно при использовании только самовнимания.

Третья лабораторная работа диагностирует фундаментальные узкие места механизма внимания: квадратичную вычислительную сложность, статистический коллапс распределений softmax вследствие центральной предельной теоремы и проблему конечного контекстного окна, ведущую к катастрофическому забыванию. Визуализации тепловых карт и количественные метрики (энтропия, вес топ-k) эмпирически подтверждают, что эффективный контекст растёт лишь логарифмически с длиной последовательности.

Четвёртая работа предлагает конструктивное решение этим ограничениям через линейное внимание. На основе трюков с ядром (kernal trick) и аппроксимаций второго порядка (QT-ViT) или положительных отображений (EfficientViT) студенты реализуют механизмы со сложностью $O(L)$, сравнивая их по точности, задержке и потреблению памяти на задачах компьютерного зрения. Эксперименты показывают, что при высоком разрешении входа линейные методы обеспечивают конкурентоспособное качество при значительном выигрыше в эффективности.

Пятая и шестая лабораторные работы демонстрируют применение механизма внимания в двух доминирующих модальностях: компьютерном зрении (ViT, DETR) и обработке естественного языка (BERT, GPT, T5,

мультимодальные LLM). Студенты осваивают архитектурные паттерны: глобальное кодирование изображений через патчи, сквозное детектирование объектов как задачу предсказания множества, двунаправленное понимание текста и автогрессивную генерацию, а также интеграцию модальностей через замороженные LLM и обучаемые адаптеры.

Ожидаемый результат освоения курса – способность студента не только применять готовые реализации внимания, но и:

- анализировать вычислительные и статистические свойства новых вариантов внимания;
- проектировать эффективные архитектуры для длинных последовательностей;
- реализовывать и профилировать attention-механизмы в мультимодальных системах;
- критически оценивать компромиссы между выразительностью, масштабируемостью и устойчивостью обучения.

Для дальнейшего развития рекомендуется:

1. Изучение передовых методов расширения контекста (RoPE, ALiBi, YaRN);
2. Исследование гибридных архитектур, сочетающих внимание с рекуррентными или state-space моделями (например, Mamba);
3. Участие в следующих международных соревнованиях по длинному контексту (LongBench, InfiniteBench) и мультимодальному рассуждению (MMMU, MathVista);
4. Вклад в open-source библиотеки (Hugging Face, xFormers) через реализацию новых attention-ядер.

Таким образом, данное пособие служит не просто введением, а отправной точкой для самостоятельных исследований и инженерных инноваций в области масштабируемых, интерпретируемых и мультимодальных систем искусственного интеллекта.

Список литературы

[1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob L. Menick, Sebastian Borgeaud, Andy Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karén Simonyan. Flamingo: a visual language model for few-shot learning. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/960a172bc7fbf0177ccccbb411a7d800-Abstract-Conference.html.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[3] Han Cai, Junyan Li, Muyan Hu, Chuang Gan, and Song Han. Efficientvit: Lightweight multi-scale attention for high-resolution dense prediction. In *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*, pages 17256–17267. IEEE, 2023. doi: 10.1109/ICCV51070.2023.01587. URL https://doi.org/10.1109/ICCV51070.2023.01587.

[4] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings,*

Part I, volume 12346 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2020. doi: 10.1007/978-3-030-58452-8_13. URL https://doi.org/10.1007/978-3-030-58452-8_13.

[5] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[6] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with performers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=Ua6zuk0WRH.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/V1/N19-1423. URL https://doi.org/10.18653/v1/n19-1423.

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

[9] Ali Hatamizadeh, Greg Heinrich, Hongxu Yin, Andrew Tao, José M. Álvarez, Jan Kautz, and Pavlo Molchanov. Fastervit: Fast vision transformers with hierarchical attention. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=kB4yBiNmXX.

[10] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: decoding-enhanced bert with disentangled attention. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event,*

Austria, May 3-7, 2021. OpenReview.net, 2021. URL https://openreview.net/forum?id=XPZIaotutsD.

[11] Shaohan Huang, Li Dong, Wenhui Wang, Yaru Hao, Saksham Singhal, Shuming Ma, Tengchao Lv, Lei Cui, Owais Khan Mohammed, Barun Patra, Qiang Liu, Kriti Aggarwal, Zewen Chi, Nils Johan Bertil Bjorck, Vishrav Chaudhary, Subhojit Som, Xia Song, and Furu Wei. Language is not all you need: Aligning perception with language models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/e425b75bac5742a008d643826428787c-Abstract-Conference.html.

[12] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=H1eA7AEtvS.

[13] Junnan Li, Dongxu Li, Silvio Savarese, and Steven C. H. Hoi. BLIP-2: bootstrapping language-image pre-training with frozen image encoders and large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 19730–19742. PMLR, 2023. URL https://proceedings.mlr.press/v202/li23q.html.

[14] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/6dcf277ea32ce3288914faf369fe6de0-Abstract-Conference.html.

[15] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

[16] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 9992–10002. IEEE, 2021. doi: 10.1109/ICCV48922.2021.00986. URL https://doi.org/10.1109/ICCV48922.2021.00986.

[17] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 11966–11976. IEEE, 2022. doi: 10.1109/CVPR52688.2022.01167. URL https://doi.org/10.1109/CVPR52688.2022.01167.

[18] Sachin Mehta and Mohammad Rastegari. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=vh-0sUt8HlG.

[19] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[20] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[21] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020. URL https://jmlr.org/papers/v21/20-074.html.

[22] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 839–846, 1998. doi: 10.1109/ICCV.1998.710815.

[23] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers amp; distillation through attention. In *International Conference on Machine Learning*, volume 139, pages 10347–10357, July 2021.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention

is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[25] Yixing Xu, Chao Li, Dong Li, Xiao Sheng, Fan Jiang, Lu Tian, and Emad Barsoum. Qt-vit: Improving linear attention in vit with quadratic taylor expansion. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 83048–83067. Curran Associates, Inc., 2024. doi: 10.52202/079017-2642. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/971f1e59cd956cc094da4e2f78c6ea7c-Paper-Conference.pdf.

[26] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: deformable transformers for end-to-end object detection. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=gZ9hCDWe6ke.

Али Аммар
Кашевник Алексей Михайлович
Othman Валаа
Лефкиммиатис Стаматис

Введение в модели трансформеров: нейросетевые имитации механизмов внимания

Учебно-методическое пособие

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49, литер А