Слайд 1

Тема доклада — развитие подходов к прогнозированию временных рядов. А точнее эволюция методов прогнозирования последовательностей: от классических ARIMA и экспоненциального сглаживания к LSTM и трансформерам. Поговорим, как и зачем потребовался переход от традиционных статистических моделей к дип лернингу, а так же рассмотрим основные авторегрессионные и нейросетевые подходы и их математические основы, как они устроены и как работают.

**Слайд «Что такое временной ряд» (~45 секунд)**

«Временной ряд — это последовательность наблюдений некоторого показателя во времени. Например, ежедневная температура воздуха, недельные продажи товаров, почасовое потребление электроэнергии или колебания курсов валют. Такие данные позволяют анализировать динамику: наличие тренда, сезонных колебаний и случайных влияний среды. Главная задача — использовать прошлые наблюдения, чтобы сделать прогноз будущих значений.»

**Слайд «Постановка задачи прогнозирования» (~45–55 секунд)**

«Формально у нас есть временной ряд — это последовательность значений yt известная до момента T. Требуется построить функцию прогноза f, такую, что величина yT+h(с крышкой) – как можно лучше приближает значение истинного yT+h. То есть функция, по которой сможем предсказать значение ряда по этой истории через h шагов вперёд. H будем называть горизонт прогноза и он может быть разным: от одного шага до сотен.

В ряде задач помимо точечного значения важна оценка интервала, который отражает неопределённость прогноза

**Слайд «Неопределённость прогноза» (~30–40 сек)**  
Даже самый точный прогноз — это не гарантия конкретного значения. Всегда остаются случайные факторы, шум в данных, ошибки модели. Поэтому, вместо одной цифры, мы часто показываем **предсказательный интервал** — это такой диапазон, в который с вероятностью не ниже **альфа** попадёт настоящее будущее значение. По сути, он показывает, **насколько уверена модель** в своём прогнозе:  
если интервал узкий — неопределённость маленькая,  
если широкий — прогноз более рискованный.  
Это особенно важно там, где ошибка может дорого стоить — например, в **энергетике**, **финансах** или **планировании ресурсов**.

**Слайд «Иллюстративный пример и нотация» (~40–50 сек)**  
Пусть у нас есть значения за 30 дней, и нужно предсказать ещё на неделю вперёд.  
Тогда прогноз на первый день вперёд — y^31​, а на пятый — y^35  
Если спустя несколько дней появились новые данные, прогноз можно **уточнить** — например, используя точки до 33-го дня.  
Чтобы было понятно, когда именно построен прогноз, используют запись y^35∣30​ — прогноз на 35-й день, сделанный в момент 30-го дня. Эта нотация нам пригодится при объяснении методов дальше.»

**Классические статистические подходы**Классические методы прогнозирования строятся вокруг простой, но очень важной идеи: текущее значение временного ряда связано со своим прошлым.  
Если мы посмотрим на данные — например, на погодные измерения или потребление электроэнергии, — то увидим там несколько характерных структур:  
уровень тренд сезонность   
Наша задача — уметь извлекать эти элементы и использовать их для прогноза будущего.  
Методы, которые мы рассмотрим сейчас, просты в использовании, быстро считаются и хорошо интерпретируются.  
Часто именно они служат базовой линией, с которой сравнивают более сложные модели.

**Скользящее среднее**Первый подход — усреднить последние kkk значений.  
Это логично: недавние наблюдения лучше отражают текущее состояние системы.  
Формула очень простая: мы берём последние kkk измерений и считаем среднее.  
Параметр k задаёт, насколько далеко в прошлое мы «оглядываемся».  
Если окно маленькое — модель быстро реагирует на изменения, но сильнее чувствует шум.  
Если окно большое — получаем более стабильную оценку, но она будет запаздывать, если данные имеют тренд.  
Именно стремление уменьшить запаздывание привело к экспоненциальному сглаживанию

**Простое экспоненциальное сглаживание (SES) или Simple Exponential Smoothing**

Экспоненциальное сглаживание улучшает скользящее среднее тем, что использует всю историю данных,  
но веса уменьшаются экспоненциально при удалении в прошлое.  
Поэтому свежие значения влияют сильнее.  
За это отвечает параметр памяти α  
Если α близко к единице — модель моментально реагирует на изменения, но сглаживает слабее.  
Если α маленькое — модель игнорирует краткосрочные шумы, но начинает запаздывать.  
Такой способ хорош для рядов, где нет чётко выраженного тренда.

**Модель Холта — тренд**

Когда в данных появляется устойчивое направление изменения, простой уровень уже недостаточен.  
Модель Холта разделяет временной ряд на две компоненты:  
ℓt​ — текущий уровень,  
bt​ — оценка тренда, то есть скорости изменения уровня.  
Обе величины обновляются адаптивно на каждом шаге:  
чтобы учесть новые данные, но не забывать историю.  
За скорость адаптации отвечают параметры α\alphaα и β\betaβ, значения которых лежат в промежутке от 0 до 1. Их подбирают по данным так, чтобы прогнозы как можно лучше совпадали с реальными наблюдениями.  
Прогноз на h шагов вперёд строится как продолжение текущей линии тренда.  
Это уже заметный шаг к реальным задачам: от продаж и логистики до анализа трафика.  
Но если в данных ещё и сезонность — например, спрос растёт каждую субботу —  
модель Холта всё ещё недостаточна, и нам потребуется следующий метод.

**Модель Хольта–Уинтерса**

Модель Хольта–Уинтерса расширяет модель Холта, добавляя сезонный компонент st  
Она по-прежнему отслеживает уровень ℓt и тренд bt​,  
но теперь дополнительно учитывает повторяющийся паттерн с периодом m  
— например, дни недели или месяцы года.

Параметры α,β и γ управляют тем, как быстро модель адаптируется  
к изменениям уровня, тренда и сезонности.  
При каждом новом наблюдении все три компоненты обновляются,  
поэтому модель хорошо приспособлена к изменяющимся условиям.

Формула прогноза складывает уровень, тренд и сезонный эффект будущего момента.

Аддитивная форма применяется, когда сезонные колебания выражены **в одинаковых единицах** —  
например, разница в спросе в будни и в выходные остаётся примерно постоянной по величине.

**мультипликативная сезонность**

Если амплитуда сезонных колебаний зависит от уровня ряда,  
например: зимой продажи выше и колебания тоже больше —  
аддитивная модель уже некорректна.  
Тогда применяется **мультипликативный вариант**.  
Здесь сезонность действует как **масштабирующий множитель**.  
При росте уровня — пропорционально растёт и сезонная волатильность – то есть степень изменчивости.  
Поэтому мультипликативную форму используют при моделировании:  
— трафика,  
— экономических индикаторов,  
— спроса в периоды роста рынка.

**Как выбрать модель сезонности?**

мы просто **смотрим на данные**.  
Если сезонные всплески одинаковой высоты — аддитивный вариант.  
Если сезонные колебания усиливаются при росте уровня —  
лучше использовать мультипликативный.  
В реальной практике сначала анализируют график, затем пробуют оба варианта  
и выбирают тот, который даёт меньшую ошибку прогноза.

**Авторегрессионные модели: идея и маршрут**

Дальше нам нужны модели, которые формально описывают, как текущее значение формируется из прошлых значений и случайных возмущений. Для этого введём базовые понятия: лаги, автокорреляции и стационарность — они определяют, когда и какие модели уместны. После короткого ввода пройдём путь от AR и MA к ARMA, затем к ARIMA для нестационарных рядов и, наконец, к сезонным и расширенным версиям SARIMA и ARIMAX

### 3 .1 — Лаги и лаговый оператор

«Лаг — это сдвиг ряда назад на фиксированное число шагов. Лаговый оператор удобен тем, что компактно записывает сдвиги и разности. Обычная разность убирает тренд, сезонная — повторяющиеся циклы с известным периодом.»

**3.2 ACF и PACF (80–100 сек)**  
«Автокорреляционная функция rt​ показывает, насколько текущее значение ряда связано с прошлыми - то есть, на каких лагах сохраняется зависимость.  
Если пики повторяются через равные промежутки, значит, есть сезонность с таким периодом.  
PACF измеряет «чистое» влияние каждого лага, исключая промежуточные связи.  
По интерпретации затухания PACF можно определить порядок AR-модели, а по затуханию ACF — порядок MA-модели.

Пример:

* ACF может показать, что yty\_tyt​ связан с yt−2y\_{t-2}yt−2​,  
  но это может быть просто потому, что yt−2y\_{t-2}yt−2​ связан с yt−1y\_{t-1}yt−1​,  
  а yt−1y\_{t-1}yt−1​ связан с yty\_tyt​.
* PACF “вычитает” эти промежуточные связи и оставляет только **настоящую прямую зависимость**.

**3.3 Стационарность: интуиция и два уровня (80–100 сек)**  
«Большинство формул и оценок в AR/ARMA предполагают стационарность. Интуитивно это неизменность свойств во времени. В строгом смысле — совместные распределения не меняются при сдвиге. В широком смысле — достаточно, чтобы математическое ожидание было постоянным, дисперсия конечной, а ковариация зависела только от лага. В ARIMA-семействе чаще ориентируются на стационарность в широком смысле

**3.4 Пример и типичные нарушения (80–100 сек)**  
Есть процессы, которые стационарны в широком смысле, но не в узком. Среднее ноль, ковариация зависит только от разности, но распределения в разные моменты времени не совпадают. Нестационарность часто проявляется как случайное блуждание — дисперсия растёт со временем; как линейный тренд — среднее зависит от t; или как чистая сезонность с меняющимся средним.   
В таких случаях нужна подготовка данных.

**3.5 Приведение к стационарности (80–100 сек)**  
Сначала стабилизируем масштаб: преобразование Бокса–Кокса делает дисперсию более стабильной; для λ=0 это логарифм. Затем дифференцируем: сезонная разность убирает повторяющиеся паттерны, обычная — тренд. На практике часто сначала делают сезонную разность, и только потом при необходимости — обычную, пока ACF/PACF и тесты не подскажут, что ряд стал стационарным

**3.6 AR(p) (80–100 сек)**  
Авторегрессия описывает текущее значение через несколько предыдущих. Интуитивно — это модель инерции: если недавно было высоко, велик шанс остаться высоко, и наоборот. Белый шум отражает случайные колебания, которые модель не объясняет. У самой простой модели, где текущее значение зависит только от предыдущего, можно показать, что при коэффициенте зависимости меньше единицы по модулю процесс остаётся устойчивым, а влияние прошлого постепенно затухает.

Пример:  
 если ∣ϕ∣<1|\phi| < 1∣ϕ∣<1 → влияние прошлого **затухает**, и процесс **стабилен** (стационарен);

 если ∣ϕ∣≥1|\phi| \ge 1∣ϕ∣≥1 → процесс «разбегается» и становится нестабильным.

**3.7 MA(q) (60–80 сек)**  
В модели скользящего среднего текущее значение складывается из текущего случайного возмущения и «эхо» нескольких прошлых шумов. Такая модель хорошо описывает ситуации, когда наблюдаются кратковременные всплески — например, случайные колебания, чьё влияние быстро затухает.  
В отличие от авторегрессии, здесь память не по прошлым значениям самого ряда, а по прежним ошибкам — по тем самым шумам εt−i\varepsilon\_{t-i}εt−i​.

**3.8 ARMA(p,q) (60–80 сек)**  
Модель ARMA сочетает две идеи: инерцию по прошлым значениям и затухающие эффекты от прошлых случайных возмущений.  
Такое объединение позволяет описывать более сложные временные зависимости, когда данных недостаточно для чисто AR или MA модели.

**3.9 ARIMA(p,d,q) (80–100 сек)**  
Если в данных есть тренд, сначала берут разности — столько раз, сколько нужно, чтобы сделать ряд стационарным.  
После этого к преобразованным данным применяют ARMA.  
Так модель ARIMA позволяет работать с временными рядами, где значения со временем меняются, но сохраняются внутренние зависимости.

**3.10 SARIMA (80–100 сек)**  
Чтобы учесть повторяющуюся структуру данных, добавляют **сезонные лаги** — через m,2mm, 2mm,2m и т.д. шагов.  
Модель **SARIMA** (Seasonal ARIMA) объединяет обычные порядки (p,d,q) с сезонными (P,D,Q)m где m — длина сезона. Благодаря этому она одновременно учитывает **краткосрочную память** по обычным лагам и **долгосрочную — по сезонным**.  
Например, для месячных данных с годовой сезонностью m=12m = 12m=12: модель использует лаги через 12 месяцев, чтобы предсказывать повторяющиеся годовые пики и спады.

**3.11 ARIMAX / SARIMAX (80–100 сек)**  
Если, помимо самого ряда, нам известны внешние факторы — например, календарь, температура или цены, — их можно добавить в модель как объясняющие переменные.  
Тогда получаем ARIMAX или, при наличии сезонности, SARIMAX.  
Эти модели учитывают влияние внешних факторов, снимают часть систематической ошибки и делают прогноз точнее.

**3.12 Итоги блока (60–80 сек)**  
«Подведём итог. AR/MA/ARMA — для стационарных зависимостей; ARIMA — для рядов с трендом; SARIMA — явная сезонность; ARIMAX — учёт известных факторов. Эти инструменты покрывают широкий класс задач и формируют сильную базовую линию перед переходом к LSTM и трансформерам.

**Слайд 1 — От классики к RNN: идея рекуррентности (≈ 80–100 сек)**  
После классических моделей перейдем к нейросетевым подходам, которые способны реализовъыать нелинейные вычисления в ходе обработке рядов и которые умеют работать именно с последовательностями. Рекуррентная сеть отличается от обычного прецептрона тем, что у неё есть внутреннее скрытое состояние — по сути, ячейка с памятью о прошлом. На каждом шаге рннка смотрит на текущий вход и на память предыдущего шага, обновляет состояние и даёт выход. Модель как бы «читает» поток токен за токеном, постепенно накапливая контекст. Проблема в том, что при очень длинных последовательностях обучение становится нестабильным. В ходе вычисления градиента - он может затухать или взрываться. Новая информация почти не влияет на текущее состояние. В попытках решить эту проблему были избретены такие архитектура, как LSTM и GRU.

**Слайд 2 — LSTM: как стабилизировать память на длинных шагах (≈ 60–80 сек)**

В отличие от обычных RNN, LSTM использует специальные "ячейки памяти" и **гейты** (входные, забывающие и выходные), которые помогают сети решать, какие данные хранить, а какие сбрасывать.То есть как бы регулируют поток информации. Благодаря этому градиент не исчезает так быстро, и сеть может удерживать долгие зависимости.  
  
В подробное строение архитектуру и принцип работы в рамках данного доклада мы погружаться не будем, так как ранее на парах ребята уже выступали с разбором этой архитектуры. Вместо этого мы посмотрим на GRU.

**Слайд 3 — GRU: упрощённая альтернатива LSTM (≈ 90–110 сек)**  
«GRU сохраняет идею управления памятью, но делает архитектуру проще. Здесь нет отдельного вектора памяти — всё хранится в одном скрытом состоянии.

Последовательность вычислений такая: Сначала определяются два коэффициента — **обновления** и **сброса**.  
Коэффициент обновления zt показывает, какую часть информации из прошлого состояния нужно сохранить в новом.  
Коэффициент сброса rt​ регулирует, насколько учитывать предыдущее состояние при формировании кандидата на скрытое состояние

Затем на основе входа xt и скорректированного предыдущего состояния вычисляется **кандидат на новое состояние**

После этого формируется новое скрытое состояние ht​ как взвешенная сумма старого состояния и кандидата.

Такой механизм позволяет GRU гибко управлять накопленной информацией и адаптироваться к динамике временного ряда, избегая проблем затухающих градиентов.

**от RNN к Attention**

После рекуррентных моделей следующим шагом стали архитектуры, которые умеют не просто анализировать, а **генерировать последовательности произвольной длины**.

Рекуррентные сети действительно хорошо работают с зависимостями во времени, но они предполагают, что длина выхода совпадает с длиной входа. В реальных задачах это не так — например, в машинном переводе: одна фраза на русском может быть длиннее или короче английской, а соответствие между словами далеко не одно к одному.

Поэтому появилась архитектура **Sequence-to-Sequence**, состоящая из двух частей.  
Энкодер последовательно считывает вход и превращает его в компактный вектор — контекст.  
Декодер затем использует этот контекст, чтобы шаг за шагом формировать выходную последовательность.

Проблема в том, что при длинных фразах один контекстный вектор не способен сохранить всю информацию или же информация будет общей и не будет отобрадать тонкостей начала и конца последовательности.   
И именно здесь появляется идея **механизма внимания** — позволить декодеру на каждом шаге решать, на какие части исходной последовательности стоит обратить внимание.  
Это и стало основой дальнейшей эволюции — моделей с вниманием и трансформеров.

**Attention**

Теперь На каждом шаге декодер сам решает, **на какие части входной последовательности стоит смотреть**, чтобы сгенерировать текущий элемент выхода.

энкодер выдал набор скрытых состояний — по одному для каждого шага входа.  
На i-м шаге декодер имеет своё внутреннее состояние si и сравнивает его с каждым состоянием энкодера hj  
Результатом являются оценки внимания ei,j​, которые показывают, насколько текущее состояние декодера «соотносится» с конкретной позицией входа.

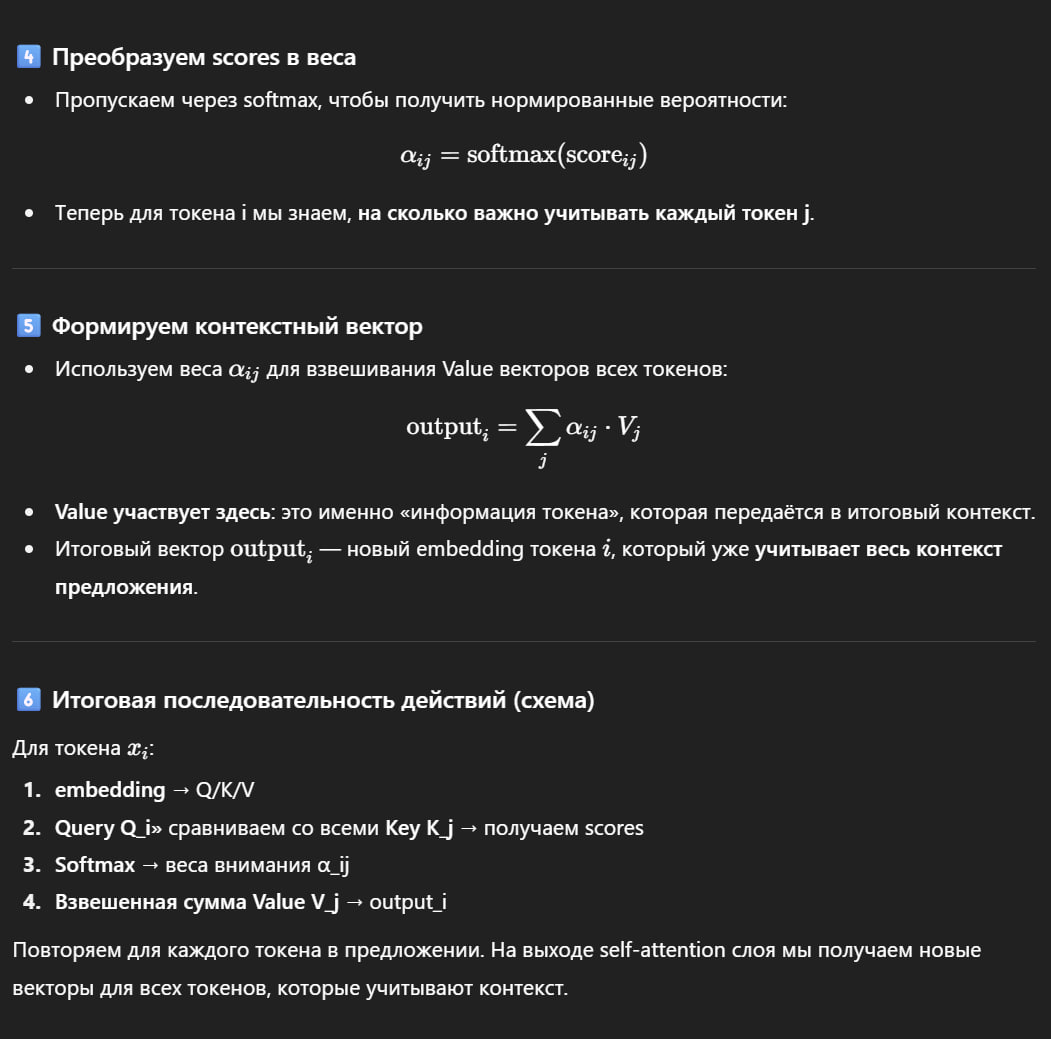
Затем эти оценки нормируются через softmax и превращаются в **веса внимания** αi,j​.  
Каждый вес отражает относительную важность соответствующего фрагмента входа при формировании следующего выхода.

После этого из всех скрытых состояний энкодера формируется **контекстный вектор** — это взвешенная сумма входных состояний, где вклад каждого определяется своим весом внимания.  
Этот контекст передаётся декодеру и используется для генерации очередного токена выходной последовательности.

Таким образом, attention позволяет модели **динамически фокусироваться** на нужных именно сейчас частях последовательности.

**Self**

После появления механизма внимания для связи энкодера и декодера стало понятно, что аналогичную идею можно использовать и внутри одной последовательности. Следовательно и в энкодере.

В реальных текстах или временных рядах значение текущего элемента зависит не только от ближайших соседей, но и от более дальних. Чтобы уловить эти связи, была предложена концепция **Self-Attention**

После нормировки через softmax они превращаются в **веса**, показывающие, насколько важен каждый другой токен.

Затем эти веса используются для вычисления **контекстного вектора** — это взвешенная сумма Value-векторов всех токенов. В результате каждый токен получает новое представление, которое уже учитывает контекст всего предложения.

Таким образом, self-attention позволяет модели одновременно видеть всю последовательность и динамически определять, какие части наиболее релевантны для каждого элемента. Это стало ключевым шагом к архитектуре трансформеров.

**MASK**

В задачах авторегрессии (например, языковое моделирование, прогнозирование временных рядов) модель не должна использовать будущие токены.

Поэтому перед softmax вводят \textit{причинную маску} $M$

Значения – бесконечность обнуляются после softmax, запрещая связь с будущими позициями.

Маскирование применяется до объединения голов, обеспечивая корректное поведение каждой self-attention головы в авторегрессионном режиме.

**Слайд «Transformer: обзор архитектуры»**  
«Transformer обрабатывает всю последовательность параллельно и строит модель вокруг механизма внимания. Базовые блоки: multi-head self-attention, позиционные кодировки, position-wise FFN и блоки Add&Norm. Энкодер и декодер состоят из повторяющихся слоёв этих компонентов.»

**Слайд «Scaled dot-product attention»**

«Для внимания используются матрицы запросов Q, ключей K и значений V. Считаем QKᵀ, делим на √d\_k для численной стабильности, применяем softmax и умножаем на V. Результат — для каждого запроса взвешенная сумма значений, где веса отражают релевантность ключей.»

**Слайд «Multi-Head self-attention»**  
Чтобы модель могла смотреть на данные под разными углами, attention выполняют **несколько раз параллельно** — это и есть multi-head.  
Каждый “голова” получает своё проецирование Q, K, V и изучает собственный тип зависимостей: кто-то фокусируется на близких элементах, кто-то на дальних.  
Потом результаты всех голов объединяются и проходят через линейную проекцию.  
В итоге модель одновременно видит и локальные, и глобальные связи, не теряя возможности вычислять всё параллельно — что и даёт трансформерам высокую эффективность.

**Слайд «Позиционные кодировки»**  
«Attention не учитывает порядок, поэтому в эмбеддинги добавляется позиционная информация. Оригинальный вариант — синусно-косинусные функции с разными частотами; альтернативы — обучаемые позиционные эмбеддинги или относительные кодировки. Позиционные кодировки дают модели информацию о порядке и относительных сдвигах.»

**Слайд «Position-wise Feed-Forward Network (FFN)»**

После слоя внимания каждый токен обрабатывается небольшой нейросетью — двухслойным полносвязным блоком.  
Он работает **независимо для каждого положения**: увеличивает размерность, применяет нелинейность, затем сжимает обратно.  
Такая операция добавляет модели способность выражать нелинейные зависимости, не смешивая токены между собой.  
То есть attention отвечает за взаимодействие между токенами, а FFN — за преобразование признаков внутри каждого токена.

**Слайд «Add & Norm (Residual + LayerNorm) — зачем»**  
Остаточное соединение просто добавляет вход к выходу блока. Нужно это для того, чтобы стабилизировать обучение модели. Грубо говоря, при обработке данных, мы можем перестараться и обновить вектора токенов так, что забудем их изначальное значение.

Теперь о нормализации.  
здесь обрабатывается **вся последовательность целиком**, а не по шагам, как в RNN. Это создаёт массу перекрёстных зависимостей, и без нормализации обучение было бы нестабильным.

Важно, что нормализуем мы **каждый токен отдельно — по его внутренним признакам**, а не всю последовательность и не батч.

Почему так?  
Во-первых, длина последовательностей разная, и если бы мы нормализовали все токены вместе, пришлось бы учитывать паддинги, которые искажают статистику.  
Во-вторых, у каждого токена своя семантика — нормализовать их между собой просто бессмысленно.  
В-третьих, при генерации, как в GPT-моделях, токены обрабатываются по одному, и LayerNorm ведёт себя одинаково как при обучении, так и при inference.  
И наконец, он **не зависит от размера батча**, поэтому работает стабильно даже при batch size = 1.

**Слайд «Краткое резюме»**  
«Self-attention дает параллельное моделирование зависимостей разной дальности; multi-head усиливает выразительность; позиционные кодировки вводят порядок; FFN добавляет нелинейность на уровне токена; Add&Norm обеспечивает стабильность обучения и корректное поведение при генерации. В совокупности эти компоненты делают Transformer эффективным и масштабируемым для задач с последовательностями.»

Подводя итог, можно сказать, что прогнозирование временных рядов прошло большую эволюцию.  
Мы начали с простых, интерпретируемых статистических методов — таких как экспоненциальное сглаживание и ARIMA.  
Эти модели просты, надёжны и до сих пор остаются отличной базовой линией, когда данных немного и поведение ряда относительно стабильно.

Но с ростом объёмов данных и усложнением зависимостей понадобились более гибкие инструменты.  
Рекуррентные сети — LSTM и GRU — научились учитывать долгосрочные связи, а появление механизма внимания и архитектуры Transformer полностью изменило подход к работе с последовательностями.

Сегодня трансформеры применяются не только в языковых задачах, но и для временных рядов, финансов, энергетики, медицины.  
Следующий шаг — это гибридные модели, которые объединяют интерпретируемость статистических подходов и мощь нейросетей.

И хотя методы становятся всё сложнее, цель остаётся прежней — понять закономерности во времени и научиться предсказывать будущее как можно точнее.»