

Альтернативные методы NLP

[Contribute on GitHub](#)

Ноябрь 2025

Содержание

1 State-space models	2
1.1 Математические предпосылки	2
1.1.1 Разложение по базису в векторных пространствах	2
1.1.2 Функциональные пространства и ортонормированные базисы	2
1.1.3 Полиномиальные проекции	3
1.1.4 Обыкновенные дифференциальные уравнения (ОДУ)	3
1.1.5 Временные ряды	4
1.1.6 Рекуррентные нейронные сети	4
1.2 HiPPO: Рекуррентная память с оптимальными полиномиальными проекциями	5
1.2.1 Фреймворк	5
1.2.2 Реализации HiPPO	6
1.3 От теории к практике	6
1.3.1 Linear State-Space Layer (LSSL)	6
1.3.2 Стратегии параметризации	8
1.4 Mamba: Селективные модели пространства состояний	9
1.4.1 Мотивация	9
1.4.2 Linear Attention	9
1.4.3 Hungry Hungry Hippos (H3)	10
1.4.4 Селективные модели пространства состояний	10
1.4.5 Архитектура Mamba	11
1.4.6 Сравнение с трансформерами	11
2 Линейные трансформеры	12
2.1 Общие предпосылки	12
2.2 Примеры	13
3 RWKV	15

Аннотация

Данная лекция представляет всесторонний обзор State-Space Models (SSMs), прослеживая их эволюцию от фреймворка HiPPO [1] до современной архитектуры Mamba [2], а также расширяет обсуждение на родственные линейные архитектуры. Мы начинаем с математических основ (полиномиальные проекции, дифференциальные уравнения) и показываем, как они воплощаются в трёх направлениях развития: рекуррентная память HiPPO, структурированные и селективные SSM-слои (LSSL [3], S4 [4], DSS [5], S4D [6], Mamba). Далее рассматриваются линейные трансформеры [7, 8, 9], демонстрирующие, как идеи внимания переносятся на рекуррентные архитектуры, и сопоставляются с SSM через призму дуальности трансформеров и моделей состояний [10]. Финальный блок посвящён RWKV [11, 12]. На протяжении всего текста подчёркиваются теоретические связи, инженерные компромиссы и практические реализации, позволяющие достигать линейной временной сложности без потери качества на задачах языкового моделирования.

1. State-space models

1.1. Математические предпосылки

1.1.1. Разложение по базису в векторных пространствах

Рассмотрим конечномерное векторное пространство $V(\mathbb{R})$ с базисом $B = \{\vec{b}_1, \dots, \vec{b}_N\}$. Любой вектор $\vec{a} \in V$ может быть выражен как:

$$\vec{a} = c_1 \vec{b}_1 + \dots + c_N \vec{b}_N$$

где $\vec{c} = (c_1, \dots, c_N)$ — это вектор координат. Если базис ортонормированный:

$$\langle \vec{b}_i, \vec{b}_j \rangle = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

то коэффициенты могут быть вычислены как:

$$c_n = \langle \vec{a}, \vec{b}_n \rangle.$$

При любом базисе векторы координат \vec{c} образуют собственное векторное пространство \mathbb{R}^N . Но если базис ортонормированный, то между $V(\mathbb{R})$ и \mathbb{R}^N есть изометрический изоморфизм. Если допустить некоторую нестрогость интерпретации, то с точки зрения машинного обучения это означает, что вектор координат является идеальным признаком описанием исходного вектора.

1.1.2. Функциональные пространства и ортонормированные базисы

Те же принципы распространяются на функциональные пространства. Для функций $f, g \in L^2[-1, 1]$ мы определяем скалярное произведение как

$$\langle f, g \rangle = \int_{-1}^1 f(x)g(x)w(x)dx,$$

где $w(x)$ — весовая функция. Классическими примерами ортонормированных полиномиальных базисов служат полиномы Лежандра, полиномы Лагера, полиномы Чебышева и полиномы Эрмита.

Название	Простр.	Весовая ф-я	Явная формула
Тригонометрический ряд	$L^2[0, 2\pi]$	$w(x) = 1$	$\{\exp(inx)\}_{i=-\infty}^{+\infty}$
Полиномы Лежандра	$L^2[-1, 1]$	$w(x) = 1$	$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} (x^2 - 1)^n$
Полиномы Лагера	$L^2[0, +\infty)$	$w(x) = e^{-x}$	$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (e^n x^n)$
Полиномы Чебышева	$L^2[-1, 1]$	$w(x) = \frac{1}{\sqrt{1-x^2}}$	$T_n(x) = \sum_{j=0}^{\lfloor n/2 \rfloor} \binom{n}{2j} (x^2 - 1)^j x^{n-2j}$
Полиномы Эрмита	$L^2(-\infty, +\infty)$	$w(x) = e^{-x^2}$	$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$

Таблица 1: Ортонормированные полиномиальные базисы

1.1.3. Полиномиальные проекции

Особенность пространства Лебега в том, что базис L^2 бесконечный. Это означает, что идеальное признаковое описание функции будет бесконечномерным вектором. На первый взгляд, это ставит непреодолимое препятствие для практических применений... Однако мы можем воспользоваться аппроксимациями!

Для любой функции $f(x) \in L^2$ и ортонормированного базиса $\{P_n(x)\}_{n=1}^{\infty}$ справедлива формула аппроксимации:

$$f(x) = \sum_{n=0}^{\infty} c_n P_n(x) \approx \sum_{n=0}^N c_n P_n(x)$$

где коэффициенты:

$$c_n = \langle f(x), P_n(x) \rangle = \int_{-1}^1 f(x) P_n(x) w(x) dx$$

В терминах алгебры такая аппроксимация является проекцией пространства L^2 на N -мерное линейное подпространство, образованное конечным базисом $\{P_n\}_{n=1}^N$. Чем больше N , тем точнее аппроксимация.

1.1.4. Обыкновенные дифференциальные уравнения (ОДУ)

ОДУ описывают динамику системы:

$$\frac{d}{dt} x(t) = f(x(t), t)$$

Простейшим примером ОДУ являются уравнения движения двумерного коптера, которые можно составить пользуясь вторым законом Ньютона классической механики:

$$\begin{cases} \frac{d}{dt} x = v_x \\ \frac{d}{dt} v_x = -\frac{(u_1+u_2) \sin(\theta)}{m} \\ \frac{d}{dt} y = v_y \\ \frac{d}{dt} v_y = \frac{(u_1+u_2) \cos(\theta)}{m} - g \\ \frac{d}{dt} \theta = \omega \\ \frac{d}{dt} \omega = \frac{(u_1-u_2)r}{I} \end{cases}$$

Решение ОДУ (численно или аналитически) называется **интегрированием**. Простейшую схему численного интегрирования дает метод Эйлера:

$$\frac{x(t+dt) - x(t)}{dt} = f(x(t), t), x_{k+1} = x_k + f(x_k, k)dt$$

где dt — т.н. шаг дискретизации. Для интегрирования методом Эйлера достаточно задать начальное условие x_0 и запустить рекуррентный пересчет по формуле для x_{k+1} . Физически этот процесс можно интерпретировать как покадровую отрисовку движения некоторой динамической системы. Пример с динамикой коптера:

$$\begin{cases} x_{n+1} = x_n + v_{n,x} dt \\ v_{n+1,x} = v_{n,x} - \frac{(u_1+u_2) \sin(\theta)}{m} dt \\ y_{n+1} = y_n + v_{n,y} dt \\ v_{n+1,y} = v_{n,y} + \left(\frac{(u_1+u_2) \cos(\theta)}{m} - g \right) dt \\ \theta_{n+1} = \theta_n + \omega_n dt \\ \omega_{n+1} = \omega_n + \frac{(u_1-u_2)r}{I} dt \end{cases}$$

1.1.5. Временные ряды

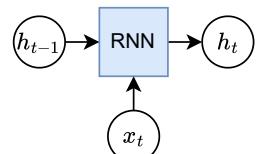
Одномерным временным рядом (univariate time series) мы будем называть числовую последовательность x_1, \dots, x_T . Число $x_t \in \mathbb{R}$ мы называем наблюдением, или замером (observation) в момент времени t (timestamp). Термин «наблюдение» отражает, в каких областях обычно возникают временные ряды: замер температуры воздуха, концентрации вещества и проч.

Многомерным временным рядом (multivariate time series) мы будем называть последовательность векторов x_1, \dots, x_T . В данном случае наблюдением будет целый вектор $x_t \in \mathbb{R}^C$. Он представляет собой замеры сразу нескольких величин. Например, вместе с температурой можно мерить давление и влажность воздуха. Отличие C -мерного временного ряда от набора из C одномерных временных рядов в том, что в многомерном ряде обычно предполагают, что компоненты вектора x_t относятся к одному моменту времени.

1.1.6. Рекуррентные нейронные сети

Простейшая рекуррентная сеть задается следующими формулами:

$$\begin{aligned} \text{RNN}(h, x) &= (1 - g) \circ h + g \circ \tanh(W_1 h + U_1 x + b_1), \\ g &= \sigma(W_2 h + U_2 x + b_2) \end{aligned}$$



где:

- $h \in \mathbb{R}^H$ — скрытое состояние (hidden state),
- $x \in \mathbb{R}^C$ — входной вектор (текущее наблюдение),
- $g \in \mathbb{R}^H$ — gate vector, управляющий обновлением скрытого состояния,
- $W_1, W_2 \in \mathbb{R}^{H \times H}$ — матрицы весов для преобразования скрытого состояния,
- $U_1, U_2 \in \mathbb{R}^{H \times C}$ — матрицы весов для преобразования входа,

- $b_1, b_2 \in \mathbb{R}^H$ — векторы смещения (bias),
- \circ — поэлементное умножение (произведение Адамара),
- $\sigma(\cdot)$ — сигмоидная функция активации $\sigma(z) = 1/(1 + e^{-z})$,
- $\tanh(\cdot)$ — гиперболический тангенс.

1.2. HiPPO: Рекуррентная память с оптимальными полиномиальными проекциями

1.2.1. Фреймворк

Сделаем предположение: пусть за всяkim одномерным времененным рядом $\{x_i\}_{i=1}^T$ стоит некоторая непрерывная функция $f(t)$, определенная при $t \in [0, +\infty)$. Времен-
ной ряд при этом будем называть дискре-
тизацией функции $f(t)$ по некоторой сет-
ке значений t . Основная идея HiPPO [1]
в том, чтобы для временного ряда $\{x_i\}_{i=1}^T$
получить коэффициенты $\vec{c} \in \mathbb{R}^N$ полиноми-
альной проекции функции $f(t)$ — тогда эти
коэффициенты можно использовать в ML
приложениях как признаковое описание
временного ряда. Проблема лишь в том,
что у нас нет доступа к $f(t)$, мы видим лишь
дискретизацию $\{x_i\}_{i=1}^N$.

Пусть $\vec{c}(\tau) \in \mathbb{R}^N$ — вектор коэффициентов полиномиальной проекции функции $f(t)|_{[0, \tau]}$ (рисунок 1). Оказывается, что векторная функция $\vec{c}(\tau)$ описывается ОДУ:

$$\frac{d}{dt} \vec{c}(t) = Ac(t) + Bf(t)$$

где $A \in \mathbb{R}^{N \times N}$ и $B \in \mathbb{R}^N$ — некоторые константы, определяемые выбранным полиноми-
альным базисом.

Применяя численное интегрирование методом Эйлера:

$$c_{k+1} = c_k + (Ac_k + Bf_k)dt = (I + Adt)c_k + (Bdt)f_k$$

Обозначая $\bar{A} = I + Adt$ и $\bar{B} = Bdt$, получаем:

$$c_{k+1} = \bar{A}c_k + \bar{B}f_k$$

В итоговом выражении участвует величина $f_k = x_k$ — один отсчет временного ря-
да. Таким образом, для вычисления признакового описания временного ряда, доста-
точно произвести пересчет по рекуррентным формулам для c_{k+1} . В этом и заключается
HiPPO [1] — High Order Polynomial Projections. Этот метод стремится решать задачи на
временных рядах с помощью рекуррентной памяти, используя коэффициенты разложе-
ния как скрытые состояния. Они служат математическим обоснованием оптимальности
сжатия исторической информации.

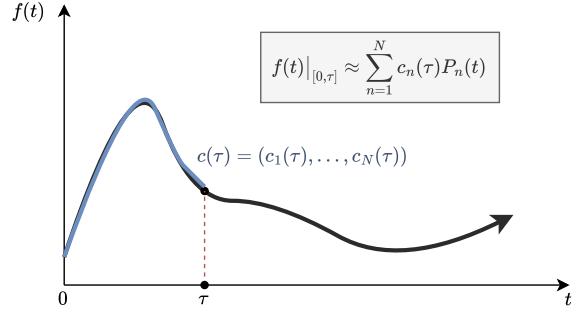


Рис. 1: Идея HiPPO: аппроксимировать функцию на отрезке с помощью полиноми-
альной проекции.

1.2.2. Реализации HiPPO

В зависимости от выбранного базиса, мы получаем разные \bar{A} и \bar{B} . Конкретные значения выводятся аналитически. Ниже приведены значения, которые получили авторы HiPPO [1] (таблица 2). Заметим, что выбор базиса влияет на то, как модель использует предыдущие наблюдения $f(t)$ (рисунок 2).

Метод	Пространство	Весовая функция	Элементы матрицы
LegT	$L^2[\tau - \theta, \tau]$	$w(t) = \frac{1}{\theta}[\tau - \theta \leq t \leq \tau]$	$A_{nk} = \frac{1}{\theta} \begin{cases} (-1)^{n-k}(2n+1), & n \geq k \\ 2n+1, & n \leq k \end{cases}$
LagT	$L^2[-\infty, \tau]$	$w(t) = \exp(t - \tau)[t \leq \tau]$	$A_{nk} = \begin{cases} 1, & n \geq k \\ 0, & n < k \end{cases}$
LegS	$L^2[0, \tau]$	$w(t) = \frac{1}{\tau}[0 \leq t \leq \tau]$	$A_{nk} = -\frac{1}{\tau} \begin{cases} \sqrt{(2n+1)(2k+1)}, & n > k \\ n+1, & n = k \\ 0, & n < k \end{cases}$

Таблица 2: Реализации HiPPO: Translated Legendre (LegT), Translated Laguerre (LagT) и Scaled Legendre (LegS)

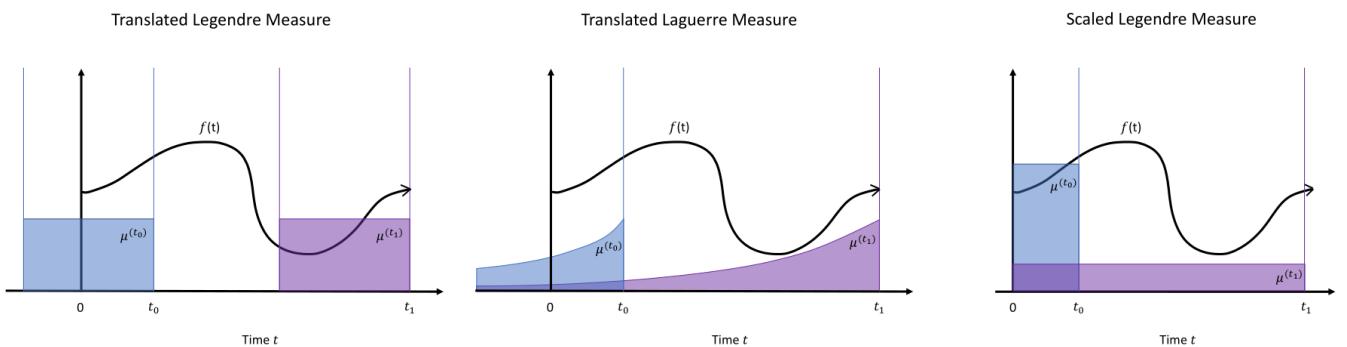


Рис. 2: Весовые функции и то, как они учитывают историю $f(t)$.

1.3. От теории к практике

1.3.1. Linear State-Space Layer (LSSL)

Основываясь на HiPPO, LSSL [3] трансформирует теоретический фреймворк HiPPO в слой нейронной сети, пригодный для практических приложений.

Авторы определяют отображение входа $\{u_t \mid u_t \in \mathbb{R}\}$ в выход $\{y_t \mid y_t \in \mathbb{R}^M\}$:

$$x_t = \bar{A}x_{t-1} + \bar{B}u_t$$

$$y_t = Cx_t + Du_t$$

где:

- $\bar{A} \in \mathbb{R}^{N \times N}$, $\bar{B} \in \mathbb{R}^N$ инициализируются из HiPPO матриц,
- $C \in \mathbb{R}^{M \times N}$, $D \in \mathbb{R}^M$ — обучаемые параметры.

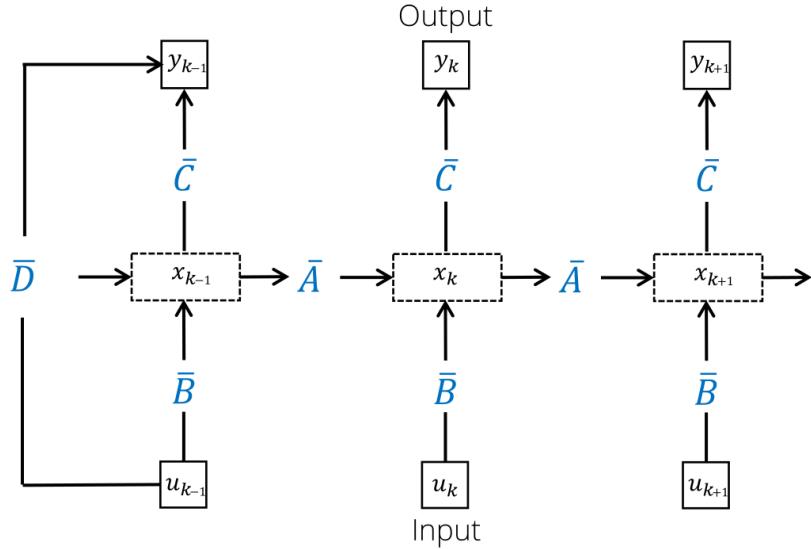


Рис. 3: Визуализация linear state-space layer (LSSL) как рекуррентной сети.

В теории оптимального управления подобные системы уравнений называют моделями пространства состояний — state-space model (SSM).

Используя SSM, авторы построили LSSL [3] — Linear State-Space Layer (рисунок 4).

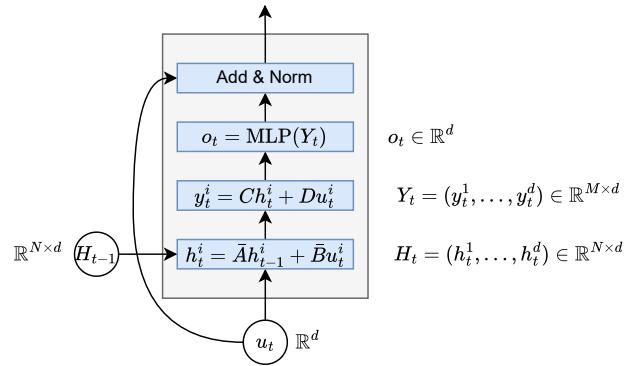


Рис. 4: Архитектура linear state-space layer.

Авторы [3] стремились реализовать компоненты, напоминающие компоненты трансформерного слоя: add & norm, коммуникация между токенами, position-wise обработка.

Рекуррентность как свертка

Один из минусов рекуррентной архитектуры в сравнении с трансформерами — это невозможность параллелизовать обучение. Наличие цикла по t обеспечивает быстрый инференс, например, для задачи next token prediction, но ужасно неэффективно, когда последовательность уже известна на перед, как на обучении. Однако у LSSL есть теоретический способ для ускорения.

Если развернуть рекуррентную формулу, то мы увидим что SSM реализуют свертку:

$$y_t = C(\bar{A})^t \bar{B} u_0 + C(\bar{A})^{t-1} \bar{B} u_1 + \dots + C \bar{B} u_t + Du_t$$

Это может быть записано как:

$$y = \mathcal{K}_L(\bar{A}, \bar{B}, C) * u + Du$$

где ядро:

$$\mathcal{K}_L(A, B, C) = (CB, CAB, \dots, CA^{L-1}B) \in \mathbb{R}^{M \times L}$$

Это ядро свертки отличается от того, что мы привыкли видеть в DL. Обычно мы имеем дело с ядрами фиксированного размера, такими как 3×3 (в компьютерном зрении) или одномерные ядра (в текстовых CNN-классификаторах). Тут же ядро \mathcal{K} имеет длину, равную длине входной последовательности.

Привычные ядра фиксированного размера хорошо параллелятся за счет сведения к матричным операциям. Ядра наподобие \mathcal{K} параллелятся за счет использования FFT convolve (Fast Fourier Transform, алгоритм "бабочка"). Тогда обучение выполняется за $O(L \log L)$. А на инференсе у нас есть выбор: либо последовательно за $O(L)$ для задач в духе next token prediction, либо $O(L \log L)$ для задач в духе классификации.

1.3.2. Стратегии параметризации

Выше упоминалось, что матрицу A можно обучать. Согласно экспериментам оригинальной статьи [3], это привносит ожидаемую специализацию под датасеты и дает небольшой прирост качества. Однако если делать матрицу A обучаемой, то мы обязаны пересчитывать ядро \mathcal{K} после каждого шага оптимизации, в то время как без обучения A достаточно предпосчитать \mathcal{K} один раз.

Чтобы решить проблему накладных расходов, связанных с вычислением \mathcal{K} , авторы применили трюк с репараметризацией [3]. Идея в том, чтобы ограничить A некоторым классом матриц, в котором находятся оригинальные матрицы HiPPO.

Трехдиагональная параметризация (LSSL)

Матрицы HiPPO могут быть представлены как:

$$A = P(D + T^{-1})Q$$

где D, P, Q — диагональные, а T — трехдиагональная. Это сокращает параметры с N^2 до $6N$, сохраняя теоретические свойства. Более того, подсчет ядра с таким A становится быстрее, если правильно определить порядок матрично-векторных операций.

Normal Plus Low-Rank (S4)

Одна из последующих работ (Structured State-Space Sequence Models, S4) [4] использует параметризацию:

$$A = V\Lambda V^* - PQ^*$$

где Λ — диагональная, V — унитарная, а $P, Q \in \mathbb{R}^{N \times r}$ — матрицы низкого ранга. Это позволяет эффективное вычисление ядра через специализированные алгоритмы.

Диагональная параметризация (DSS, S4D)

Самый простой подход [5, 6] использует диагональные матрицы:

$$A = \text{diag}(\lambda_1, \dots, \lambda_N)$$

Это дает чрезвычайно эффективное вычисление ядра: If $A = V\Lambda V^{-1}$, then $\exists w \in \mathbb{C}^N$:

$$\mathcal{K}_L(\bar{A}, \bar{B}, C) \Leftrightarrow \mathcal{K}_L(\Lambda, (e^{L\lambda_i dt} - 1)_{i=1}^N, w) = w\Lambda^{-1}\text{softmax}(P),$$

where $P \in \mathbb{C}^{N \times L}$ such that $P_{ij} = \lambda_i j \cdot dt$.

1.4. Mamba: Селективные модели пространства состояний

1.4.1. Мотивация

Хотя SSM показали перспективность, они страдали от фундаментального ограничения: **инвариантности по времени**. Параметры A, B, C оставались постоянными независимо от входа, что затрудняло селективное запоминание или забывание информации на основе контекста.

Рассмотрим задачу, где модель должна селективно копировать токены на основе контекста (рисунок 5). Традиционные SSM испытывают трудности, поскольку не могут адаптировать свой механизм памяти к содержимому входа.

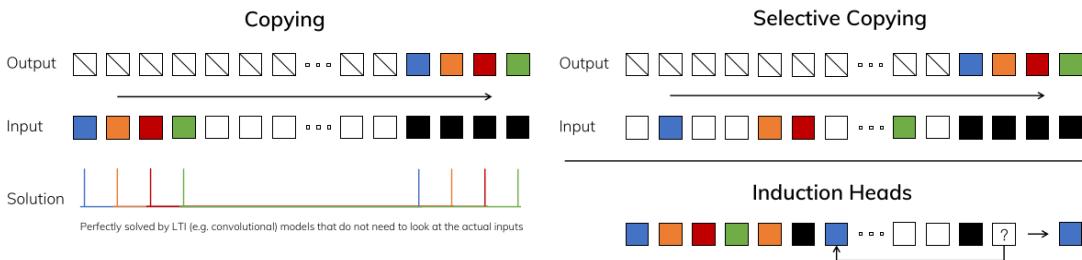


Рис. 5: Синтетические задачи языкового моделирования: копирование.

Подобные синтетические задачи созданы для того, чтобы изолированно проверять способности моделей, которые исследователям кажутся ключевыми. Аналогичная ситуация с другими двумя синтетическими задачами: *inductive heads* и *associative recall* (рисунок 6).

Table 1: Synthetic language modeling tasks.

Task	Input	Output	Sequence Length	Vocab Size
Induction Head	$a \ b \ c \ d \ e \vdash f \ g \ h \ i \dots \ x \ y \ z \vdash$	f	30	20
Associative Recall	$a \ 2 \ c \ 4 \ b \ 3 \ d \ 1 \ a$	2	20	10

Рис. 6: Синтетические задачи языкового моделирования: копирование.

1.4.2. Linear Attention

До Mamba, Linear Attention [7] показал, как эффективно аппроксимировать внимание трансформера:

Механизм внимания $\text{softmax}(QK^T)V$ можно расписать следующим образом:

$$O_i = \sum_{j=1}^i \frac{\text{sim}(Q_i, K_j)}{\sum_{t=1}^i \text{sim}(Q_i, K_t)} V_j = \frac{\sum_{j=1}^i \text{sim}(Q_i, K_j) V_j}{\sum_{t=1}^i \text{sim}(Q_i, K_t)}$$

где $\text{sim}(q, k) = \exp(q^T k)$. Если выбрать $\text{sim}(q, k) = \phi(q)^T \phi(k)$ с некоторой нелинейной функцией $\phi: \mathbb{R}^d \rightarrow \mathbb{R}^d$, то

$$O_i = \frac{\sum_{j=1}^i \phi(Q_i)^T \phi(K_j) V_j}{\sum_{t=1}^i \phi(Q_i)^T \phi(K_t)} = \left[\frac{\phi(Q_i)^T \sum_{j=1}^i \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^i \phi(K_j)} \right]^T$$

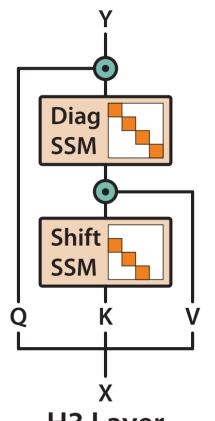


Рис. 7: H3.

Определим величины S_i, Z_i :

$$O_i^T = \frac{\phi(Q_i)^T \underbrace{\sum_{j=1}^i \phi(K_j) V_j^T}_{Z_i}}{\phi(Q_i)^T \sum_{j=1}^i \phi(K_j)} = \frac{\phi(Q_i)^T S_i}{\phi(Q_i)^T Z_i} \Rightarrow \begin{cases} S_i &= S_{i-1} + \phi(K_i) V_i^T, \\ Z_i &= Z_{i-1} + \phi(K_i). \end{cases}$$

Получили рекуррентный инференс.

1.4.3. Hungry Hungry Hippos (H3)

H3 [13] связал SSM и языковое моделирование, аппроксимируя внимание компонентами SSM:

$$Q \circ \text{SSM}_{\text{diag}}(\text{SSM}_{\text{shift}}(K) \circ V)$$

Эта архитектура аппроксимирует внимание трансформера средствами SSM (рисунок 7). Для эффективности авторы опираются на Flash Convolution для правильной утилизации GPU.

1.4.4. Селективные модели пространства состояний

Mamba [2] вводит входозависимые параметры (рисунок 8):

$$\begin{aligned} x_t &= \bar{A}(x_t)x_{t-1} + \bar{B}(x_t)u_t \\ y_t &= C(x_t)x_t + D(x_t)u_t \end{aligned}$$

где A, B, C, D теперь являются функциями от входа x_t .

Algorithm 1 SSM (S4)	Algorithm 2 SSM + Selection (S6)
<p>Input: $x : (\mathbf{B}, \mathbf{L}, \mathbf{D})$ Output: $y : (\mathbf{B}, \mathbf{L}, \mathbf{D})$</p> <p>1: $A : (\mathbf{D}, \mathbf{N}) \leftarrow \text{Parameter}$ ► Represents structured $N \times N$ matrix 2: $B : (\mathbf{D}, \mathbf{N}) \leftarrow \text{Parameter}$ 3: $C : (\mathbf{D}, \mathbf{N}) \leftarrow \text{Parameter}$ 4: $\Delta : (\mathbf{D}) \leftarrow \tau_\Delta(\text{Parameter})$ 5: $\bar{A}, \bar{B} : (\mathbf{D}, \mathbf{N}) \leftarrow \text{discretize}(\Delta, A, B)$ 6: $y \leftarrow \text{SSM}(\bar{A}, \bar{B}, C)(x)$ ► Time-invariant: recurrence or convolution 7: return y</p>	<p>Input: $x : (\mathbf{B}, \mathbf{L}, \mathbf{D})$ Output: $y : (\mathbf{B}, \mathbf{L}, \mathbf{D})$</p> <p>1: $A : (\mathbf{D}, \mathbf{N}) \leftarrow \text{Parameter}$ ► Represents structured $N \times N$ matrix 2: $B : (\mathbf{B}, \mathbf{L}, \mathbf{N}) \leftarrow s_B(x)$ 3: $C : (\mathbf{B}, \mathbf{L}, \mathbf{N}) \leftarrow s_C(x)$ 4: $\Delta : (\mathbf{B}, \mathbf{L}, \mathbf{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$ 5: $\bar{A}, \bar{B} : (\mathbf{B}, \mathbf{L}, \mathbf{D}, \mathbf{N}) \leftarrow \text{discretize}(\Delta, A, B)$ 6: $y \leftarrow \text{SSM}(\bar{A}, \bar{B}, C)(x)$ ► Time-varying: recurrence (<i>scan</i>) only 7: return y</p>

Рис. 8: Алгоритм SSM для Mamba.

Параметры A, B, C задаются простыми линейными проекциями от входа; вводится селективный механизм, управляющий потоком информации; реализация опирается на кастомные CUDA-ядра, учитывающие иерархию памяти (рисунок 9).

Поскольку свёртка в явном виде недоступна, для распараллеливания рекуррентности используется алгоритм parallel scan [14], который обычно в алгоритмах используют для вычисления кумулятивной суммы.

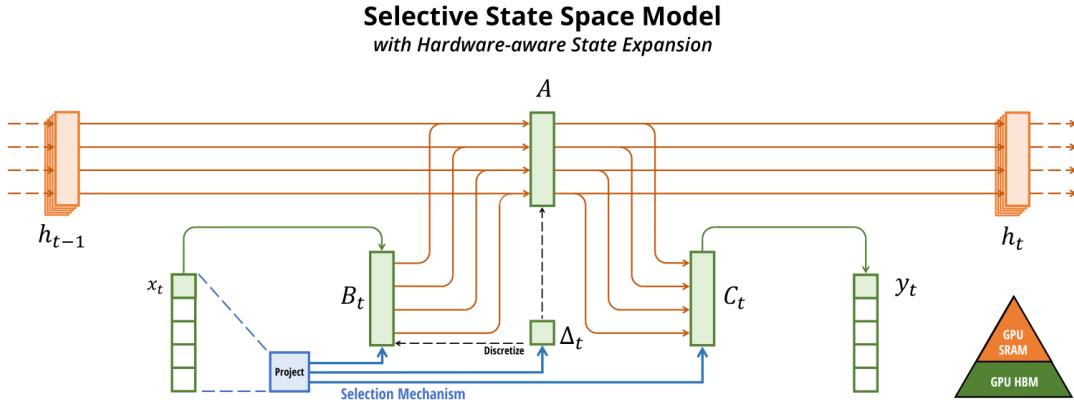


Figure 1: (Overview.) Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g. $N = 4$). Prior SSMs avoid materializing this large effective state (DN , times batch size B and sequence length L) through clever alternate computation paths requiring time-invariance: the (Δ, A, B, C) parameters are constant across time. Our selection mechanism adds back input-dependent dynamics, which also requires a careful hardware-aware algorithm to only materialize the expanded states in more efficient levels of the GPU memory hierarchy.

Рис. 9: Утилизация иерархии памяти в CUDA-реализации Mamba.

1.4.5. Архитектура Mamba

Блок Mamba включает линейную проекцию входа к скрытой размерности, селективный SSM (входозависимое обновление состояний), Gated MLP с современными активациями (SiLU/Swish [15, 16]) и заключительную проекцию обратно к исходной размерности (рисунок 10).

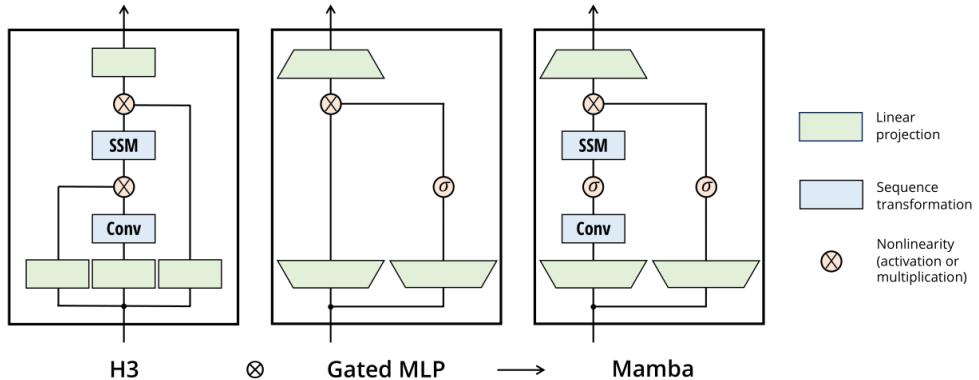


Figure 3: (Architecture.) Our simplified block design combines the H3 block, which is the basis of most SSM architectures, with the ubiquitous MLP block of modern neural networks. Instead of interleaving these two blocks, we simply repeat the Mamba block homogeneously. Compared to the H3 block, Mamba replaces the first multiplicative gate with an activation function. Compared to the MLP block, Mamba adds an SSM to the main branch. For σ we use the SiLU / Swish activation (Hendrycks and Gimpel 2016; Ramachandran, Zoph, and Quoc V Le 2017).

Рис. 10: Архитектура Mamba.

1.4.6. Сравнение с трансформерами

К сожалению, из-за своей рекуррентной природы Mamba [2] все равно плохо справляется с задачами, требующими точного запоминания контекста. Mamba обеспечивает линейно-временной инференс, хорошо экстраполирует длину входной последовательности и демонстрирует сопоставимую перплексию (рисунок 18). Однако Mamba слабо поддается in-context learning, и страдает от проблемы нечеткой памяти (рисунок 11).

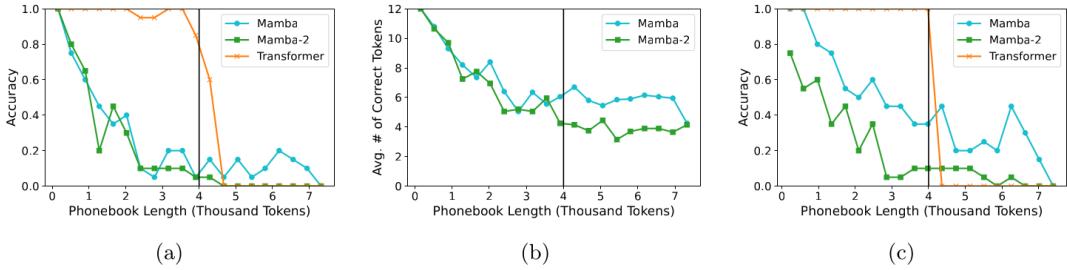


Figure 3: Evaluation results for pure SSM and Transformer models (trained for 1.1T tokens) on the Phonebook task illustrated in Figure 2b. (a) On the standard Phonebook task, Transformers are capable of in-context learning and answering questions that require copying from the input, but SSM models struggle with this task. (b) In the standard Phonebook setting (i.e., (a)), SSM models exhibit *fuzzy memory*—while they are unable to correctly predict the phone number, they predict phone numbers that share multiple digits (in the right locations) with the correct answer (see Section 3.3.3). (c) On the Reversed Phonebook formulation, even when notified at the beginning of the context which phone number they will be asked to recall, SSM models still lag behind Transformer models.

Рис. 11: Проблема нечеткой памяти в Mamba (fuzzy memory).

Комбинирование Mamba с трансформером даёт «лучшее из двух миров»: эффективность Mamba сочетается с возможностями внимания, что обеспечивает лучшее качество (рисунок 12).

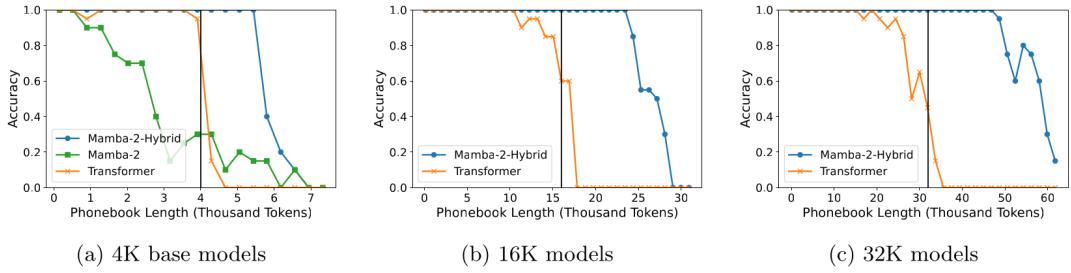


Figure 7: Phonebook evaluation results for 8B-parameter models trained on 3.5T tokens and their long-context extensions. We use the standard Phonebook formulation (Figure 2b). Mamba-2-Hybrid models can generalize beyond their pretraining sequence length and perform the Phonebook task on contexts longer than pure SSM or Transformer models. (a) Base 4K Mamba-2, Mamba-2-Hybrid, and Transformer model Phonebook evaluations. (b) 16K long-context extensions of the Mamba-2-Hybrid and Transformer model evaluated on Phonebook. (c) Phonebook evaluations for the Mamba-2-Hybrid and Transformer models extended to support 32K sequence lengths.

Рис. 12: Проблема нечеткой памяти (fuzzy memory) в Mamba решается в гибридной архитектуре, которая чередует Mamba слои со слоями трансформера.

В настоящий материал не включен рассказ про Mamba-2 [10] и Mamba-3 [17].

2. Линейные трансформеры

2.1. Общие предпосылки

Одним из многообещающих направлений развития альтернативных подходов являются так называемые линейные трансформеры. Вспомним идею линейного внимания, описанную в разделе 1.4.2. Если в формуле механизма внимания заменить $\text{sim}(q, k) = \exp(q^T k)$ на $\text{sim}(q, k) = \phi(q)^T \phi(k)$, то получаем рекуррентный инференс (линейный по време-

мени относительно длины последовательности):

$$\begin{aligned} S_i &= S_{i-1} + \phi(K_i)V_i^T, \\ Z_i &= Z_{i-1} + \phi(K_i). \end{aligned}$$

Эта идея развивалась, и сегодня мы имеем целый ряд LLM архитектур, основанных на линейном аттеншнене [8, 9, 18]. В целях краткости рассказа перечислим особенности этих архитектур тремя ключевыми апгрейдами.

Апгрейд 0: Без нелинейности. Во-первых, в формулах линейного трансформера можно убрать функцию $\phi(\cdot)$ (TODO cide):

$$\begin{aligned} S_i &= S_{i-1} + K_i V_i^T, \\ Z_i &= Z_{i-1} + K_i. \end{aligned}$$

Более того, некоторые работы показывают, что от знаменателя Z_i тоже можно избавиться, поскольку он вносит численную нестабильность. Тогда инференс станет линейным в смысле преобразований, а не только всмысле времени. Обоснование у такого подхода в первую эмпирическое. Однако полученную формулу $S_i = S_{i-1} + K_i V_i^T$ можно обосновать еще теоретически. Смысл в том, что она эквивалентна градиентной оптимизации следующего функционала:

$$\mathcal{L}_i = -\langle SK_i, V_i \rangle.$$

То есть можно сказать, что в ходе рекуррентного инференса мы используем память S как некоторый линейный оператор для предсказания V_i с помощью K_i . Обучение такого оператора происходит засчет минимизации скалярного произведения между предсказанием и таргетом.

Апгрейд 1: Delta-правило. Вместо $\mathcal{L}_i = -\langle SK_i, V_i \rangle$ можно использовать MSE-лосс:

$$\mathcal{L}_i = -\frac{1}{2} \|SK_i - V_i\|^2.$$

Тогда оптимизация будет выглядеть следующим образом:

$$\begin{aligned} S_i &= S_{i-1} - \beta_i \nabla \mathcal{L}_{i-1} \\ &= S_{i-1} - \beta_i (S_{i-1} K_i - V_i) K_i \end{aligned}$$

Поскольку здесь возникает разница между предсказанием и таргетом, эту формулу называют delta-rule. В качестве β_i берут $\beta_i = \sigma(W_\beta X_i)$ — обучаемую нелинейную функцию (с сигмоидой). Это правило интерпретируют как более точное обновление рекуррентной памяти S . Более того, существуют эффективные hardware-aware алгоритмы, которые позволяют параллелизировать обучение по этим формулам [8].

Апгрейд 2: Gated attention. В архитектуру современных трансформеров все чаще добавляют так называемый gated attention вместо обычного механизма внимания [19]. Прием заключается в добавлении специальной нелинейности после обычного аттеншнена перед конкатенацией выходов с разных голов (рис. 13). Необходимость в этой дополнительной нелинейности преимущественно объясняется эффектом под названием attention sink — когда внимание на начальные токены последовательности доминируют для всех токенов последовательности, как далеко от начала они бы не находились (рис. 14).

2.2. Примеры

Gated DeltaNet [20]. Это одна из первых работ, соединивших все перечисленные апгрейды. Дополнительно авторы ввели скалярный forget gate для рекуррентной памяти:

$$S_i = \alpha_i (I - \beta_i K_i K_i^T) S_{i-1} + \beta_i K_i V_i^T.$$

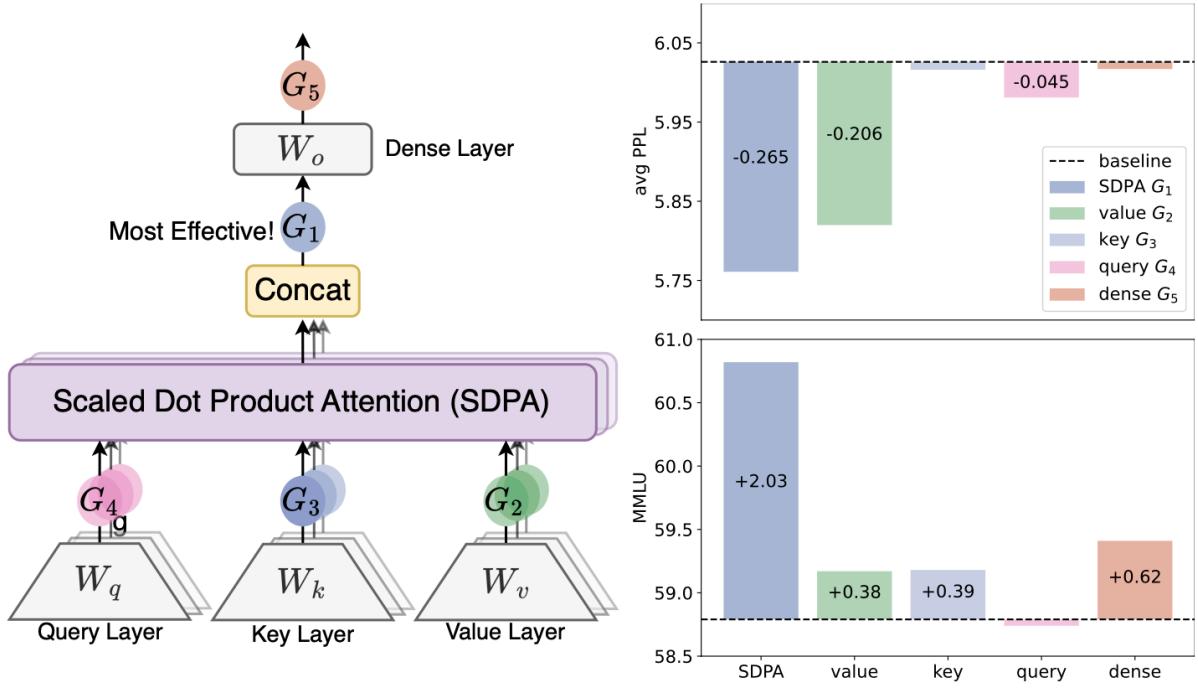


Рис. 13: Результаты основного эксперимента из статьи Gated Attention [19].

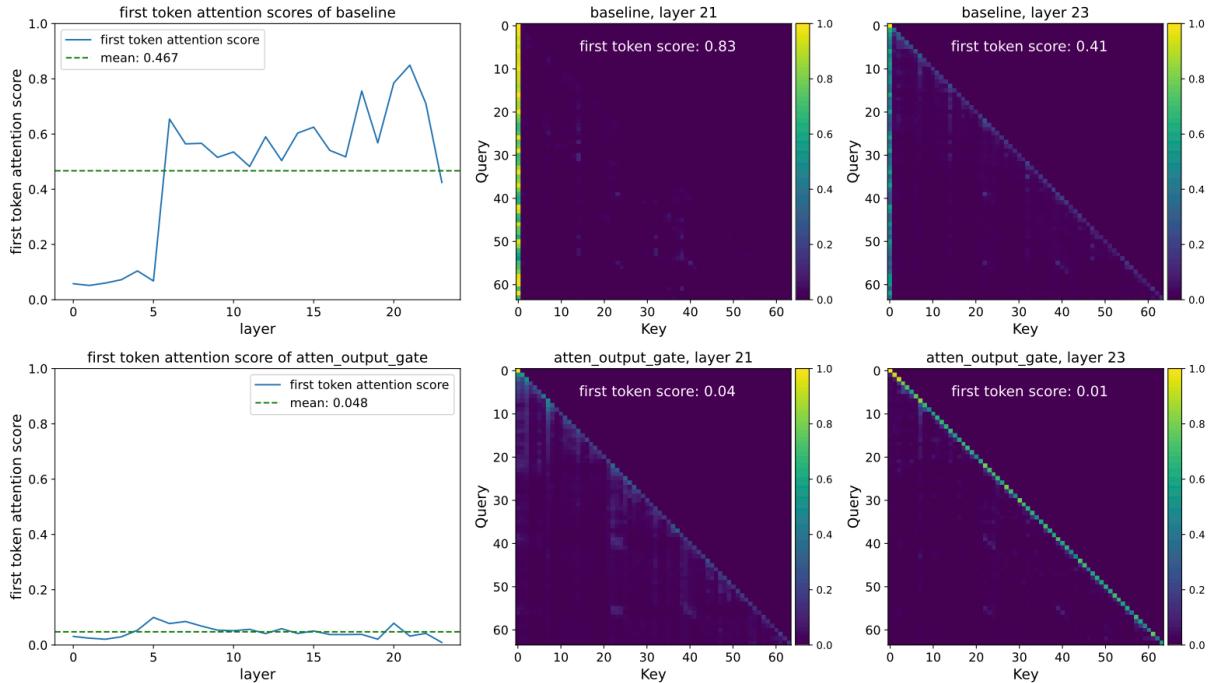


Figure 2: Left: Proportion of attention allocated to the initial token per layer (test perplexity dataset). The baseline model suffers from a significant attention sink, with an average of 46.7% of attention scores across layers directed towards the first token. Introducing a gate effectively alleviates this, reducing the proportion to 4.8%. Right: Average attention map weights for each head. Layer 21 in the baseline model demonstrates a strong attention sink (83% on the first token), which is substantially reduced by the gate (4%). In the final output layer, the gate amplifies the existing tendency for the model to attend to individual tokens within the sequence.

Рис. 14: Эффект attention sink из статьи Gated Attention [19].

Kimi Linear [9]. Во-первых, в этой работе скалярный α_i заменили на диагональную

матрицу $\text{diag}(\vec{\alpha}_i)$:

$$S_i = I - \beta_i K_i K_i^T \text{diag}(\vec{\alpha}_i) S_{i-1} + \beta_i K_i V_i^T.$$

Во-вторых, авторы детальнее проработали архитектуру всей модели, добавив свёртки и RMSProp-нормирование голов (рис. 15).

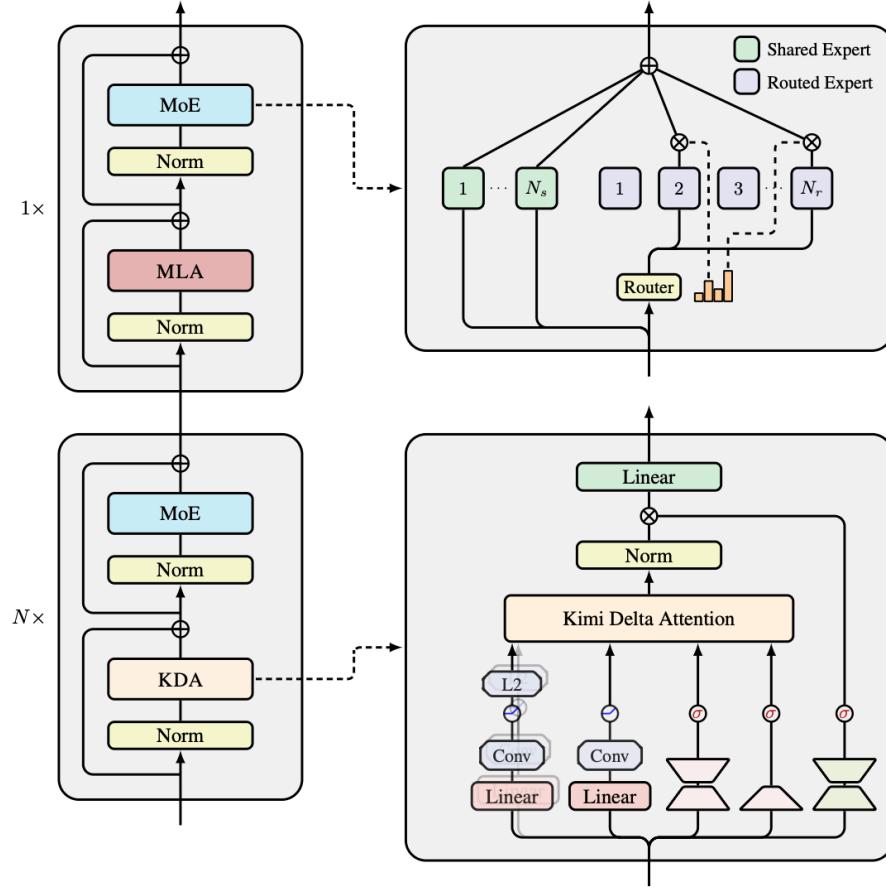


Figure 3: Illustration of our Kimi Linear model architecture, which consists of a stack of blocks containing a token mixing layer followed by a MoE channel-mixing layer. Specifically, we interleave N KDA layers with one MLA layer for token mixing, where N is set to 3 in our implementation.

Рис. 15: Архитектура Kimi Linear [9].

Qwen-3.2-next [18]. Эта модель очень похожа на Kimi Linear и по большей части отличается лишь тем, что не целиком отказывается от стандартного внимания взамен на линейное, а комбинирует оба подхода, образуя гибридную архитектуру (рис. 16).

3. RWKV

Название RWKV принадлежит целой серии моделей, которая сегодня насчитывает семь последовательно выпускавшихся версий [12]. Ниже поговорим о четвертой версии — первой, о которой вышла статья [11].

Во-первых, авторы используют измененную функцию близости между ключом и запросом: вместо $\text{sim}(q_i, k_j) = \exp(q_i^T k_j)$ считают $\text{sim}(q_i, k_j) = \exp(k_i - (i - j)w)$, где $w \in \mathbb{R}^d$ — обучаемый вектор. Интуиция такова, что чем дальше друг от друга ключ и запрос, тем менее они релевантны друг другу. Вместо вычисления сложных семантических отношений между ними с помощью скалярного произведения, достаточно просто знать,

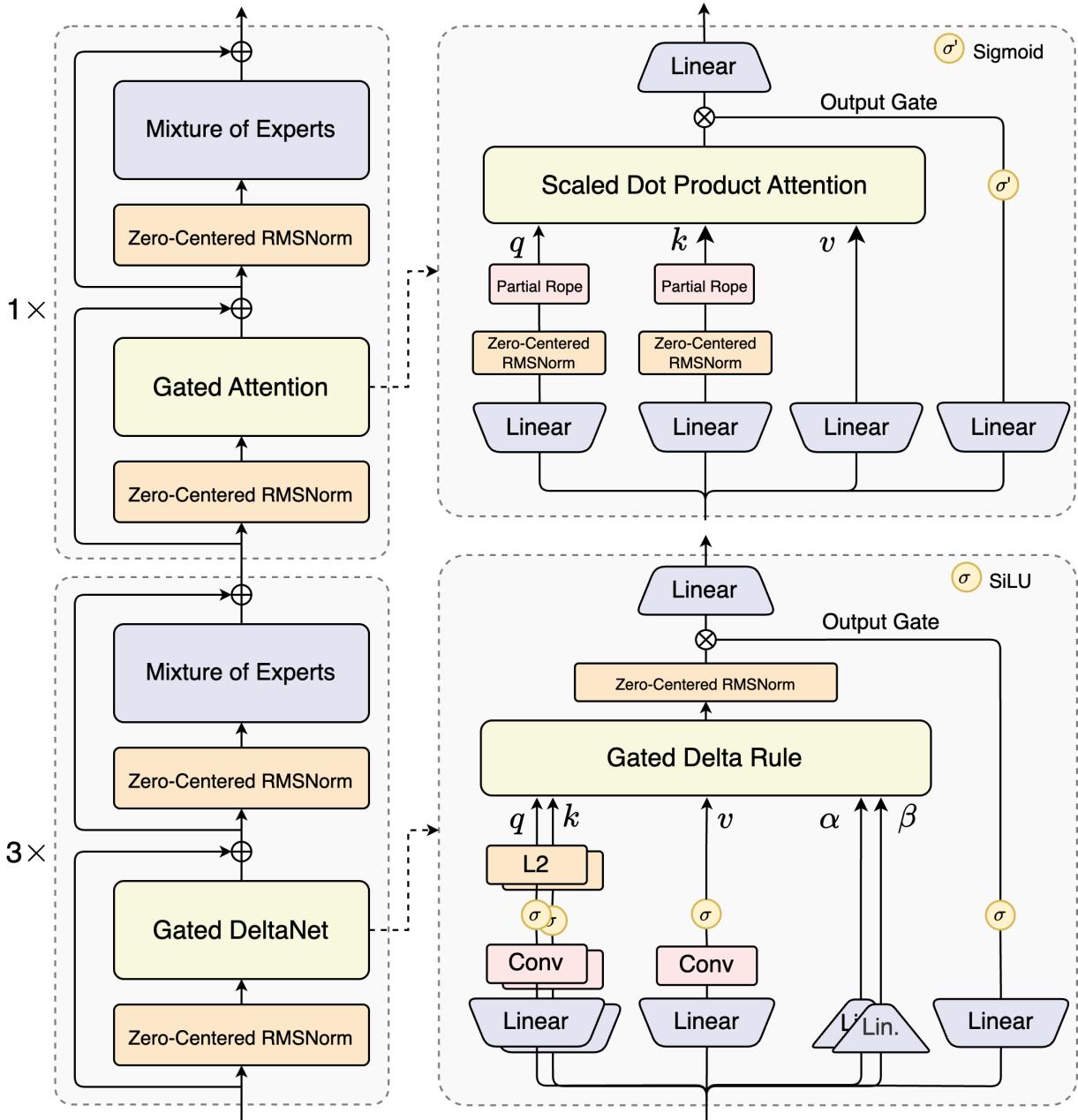


Рис. 16: Архитектура Qwen-3.2-next [18].

как далеко они друг от друга. Это сильное упрощение, но вместе с другими аспектами RWKV оно не сильно портит качество. При этом за счет избавления от q матрицу внимания легко считать на каждом новом шаге рекуррентного инференса.

Во-вторых, авторы используют архитектуру, имитирующую механизм внимания (рис. 17). Вектор R на этой схеме и в названии RWKV означает receptance и он заменяет векторы Q_i . Заметим, что в этой архитектуре предыдущие векторы не выкидываются, а хранятся, поскольку они используются в WKV операторе. Главный плюс в том, что благодаря упрощенной функции близости $\text{sim}(q_i, k_j) = \exp(k_i - (i - j)w)$ получается сохранить параллельность обучения и достичь линейный инференс.

В-третьих, помимо рекуррентной связи состояний в RWKV используется слаженное

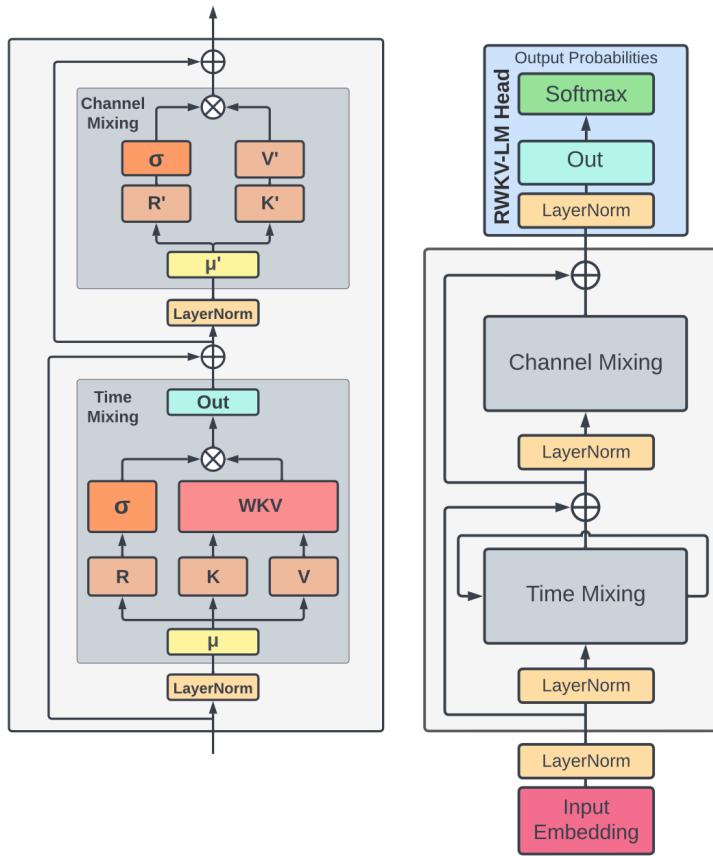


Figure 2: Elements within an RWKV block (left) and the complete RWKV residual block, equipped with a final head for language modeling (right).

Рис. 17: Архитектура RWKV-v4 [11]. На этом рисунке WKV означает формулу внимания, в которой $\text{sim}(q_i, k_j) = \exp(k_i - (i - j)w)$.

среднее входов (μ на схеме 17), которое авторы называют token shift:

$$\begin{aligned} r_i &= W_r(\mu_r \cdot x_i + (1 - \nu_r) \cdot x_{i-1}), \\ k_i &= W_k(\mu_k \cdot x_i + (1 - \nu_k) \cdot x_{i-1}), \\ v_i &= W_v(\mu_v \cdot x_i + (1 - \nu_v) \cdot x_{i-1}) \end{aligned}$$

В-четвертых, для стабилизации и ускорения обучения RWKV пришлось применить несколько маленьких оптимизаций:

1. кастомное CUDA-ядро для оператора WKV
2. специальная инициализация матрицы эмбедингов токенов
3. специальная инициализация весов сети

Список литературы

- [1] Albert Gu и др. *HiPPO: Recurrent Memory with Optimal Polynomial Projections*. 2020. arXiv: [2008.07669 \[cs.LG\]](https://arxiv.org/abs/2008.07669). URL: <https://arxiv.org/abs/2008.07669>.
- [2] Albert Gu и Tri Dao. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. 2024. arXiv: [2312.00752 \[cs.LG\]](https://arxiv.org/abs/2312.00752). URL: <https://arxiv.org/abs/2312.00752>.
- [3] Albert Gu и др. *Combining Recurrent, Convolutional, and Continuous-time Models with Linear State-Space Layers*. 2021. arXiv: [2110.13985 \[cs.LG\]](https://arxiv.org/abs/2110.13985). URL: <https://arxiv.org/abs/2110.13985>.
- [4] Albert Gu, Karan Goel и Christopher Ré. *Efficiently Modeling Long Sequences with Structured State Spaces*. 2022. arXiv: [2111.00396 \[cs.LG\]](https://arxiv.org/abs/2111.00396). URL: <https://arxiv.org/abs/2111.00396>.
- [5] Ankit Gupta, Albert Gu и Jonathan Berant. *Diagonal State Spaces are as Effective as Structured State Spaces*. 2022. arXiv: [2203.14343 \[cs.LG\]](https://arxiv.org/abs/2203.14343). URL: <https://arxiv.org/abs/2203.14343>.
- [6] Albert Gu и др. *On the Parameterization and Initialization of Diagonal State Space Models*. 2022. arXiv: [2206.11893 \[cs.LG\]](https://arxiv.org/abs/2206.11893). URL: <https://arxiv.org/abs/2206.11893>.
- [7] Angelos Katharopoulos и др. *Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention*. 2020. arXiv: [2006.16236 \[cs.LG\]](https://arxiv.org/abs/2006.16236). URL: <https://arxiv.org/abs/2006.16236>.
- [8] Songlin Yang и др. *Parallelizing Linear Transformers with the Delta Rule over Sequence Length*. 2025. arXiv: [2406.06484 \[cs.LG\]](https://arxiv.org/abs/2406.06484). URL: <https://arxiv.org/abs/2406.06484>.
- [9] Kimi Team и др. *Kimi Linear: An Expressive, Efficient Attention Architecture*. 2025. arXiv: [2510.26692 \[cs.CL\]](https://arxiv.org/abs/2510.26692). URL: <https://arxiv.org/abs/2510.26692>.
- [10] Tri Dao и Albert Gu. *Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality*. 2024. arXiv: [2405.21060 \[cs.LG\]](https://arxiv.org/abs/2405.21060). URL: <https://arxiv.org/abs/2405.21060>.
- [11] Bo Peng и др. *RWKV: Reinventing RNNs for the Transformer Era*. 2023. arXiv: [2305.13048 \[cs.CL\]](https://arxiv.org/abs/2305.13048). URL: <https://arxiv.org/abs/2305.13048>.
- [12] RWKV Wiki. *RWKV Architecture History*. URL: <https://wiki.rwkv.com/basic/architecture.html>.
- [13] Daniel Y. Fu и др. *Hungry Hungry Hippos: Towards Language Modeling with State Space Models*. 2023. arXiv: [2212.14052 \[cs.LG\]](https://arxiv.org/abs/2212.14052). URL: <https://arxiv.org/abs/2212.14052>.
- [14] Jimmy T. H. Smith, Andrew Warrington и Scott W. Linderman. *Simplified State Space Layers for Sequence Modeling*. 2023. arXiv: [2208.04933 \[cs.LG\]](https://arxiv.org/abs/2208.04933). URL: <https://arxiv.org/abs/2208.04933>.
- [15] Stefan Elfwing, Eiji Uchibe и Kenji Doya. *Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning*. 2017. arXiv: [1702.03118 \[cs.LG\]](https://arxiv.org/abs/1702.03118). URL: <https://arxiv.org/abs/1702.03118>.
- [16] Noam Shazeer. *GLU Variants Improve Transformer*. 2020. arXiv: [2002.05202 \[cs.LG\]](https://arxiv.org/abs/2002.05202). URL: <https://arxiv.org/abs/2002.05202>.

- [17] Anonymous. «Mamba-3: Improved Sequence Modeling using State Space Principles». B: *Submitted to The Fourteenth International Conference on Learning Representations*. under review. 2025. URL: <https://openreview.net/forum?id=HwCvaJ0iCj>.
- [18] qwen team. *Qwen-3.2-next*. 2025. URL: <https://qwen.ai/blog?id=4074cca80393150c248e508a&from=research.latest-advancements-list>.
- [19] Zihan Qiu и др. *Gated Attention for Large Language Models: Non-linearity, Sparsity, and Attention-Sink-Free*. 2025. arXiv: [2505.06708 \[cs.CL\]](https://arxiv.org/abs/2505.06708). URL: <https://arxiv.org/abs/2505.06708>.
- [20] Songlin Yang, Jan Kautz и Ali Hatamizadeh. *Gated Delta Networks: Improving Mamba2 with Delta Rule*. 2025. arXiv: [2412.06464 \[cs.CL\]](https://arxiv.org/abs/2412.06464). URL: <https://arxiv.org/abs/2412.06464>.

Table 3: (**Zero-shot Evaluations.**) Best results for each size in bold. We compare against open source LMs with various tokenizers, trained for up to 300B tokens. Pile refers to the validation split, comparing only against models trained on the same dataset and tokenizer (GPT-NeoX-20B). For each model size, Mamba is best-in-class on every single evaluation result, and generally matches baselines at twice the model size.

MODEL	TOKEN.	PILE PPL ↓	LAMBADA PPL ↓	LAMBADA ACC ↑	HELLASWAG ACC ↑	PIQA ACC ↑	ARC-E ACC ↑	ARC-C ACC ↑	WINOGRANDE ACC ↑	AVERAGE ACC ↑
Hybrid H3-130M	GPT2	—	89.48	25.77	31.7	64.2	44.4	24.2	50.6	40.1
Pythia-160M	NeoX	29.64	38.10	33.0	30.2	61.4	43.2	24.1	51.9	40.6
Mamba-130M	NeoX	10.56	16.07	44.3	35.3	64.5	48.0	24.3	51.9	44.7
Hybrid H3-360M	GPT2	—	12.58	48.0	41.5	68.1	51.4	24.7	54.1	48.0
Pythia-410M	NeoX	9.95	10.84	51.4	40.6	66.9	52.1	24.6	53.8	48.2
Mamba-370M	NeoX	8.28	8.14	55.6	46.5	69.5	55.1	28.0	55.3	50.0
Pythia-1B	NeoX	7.82	7.92	56.1	47.2	70.7	57.0	27.1	53.5	51.9
Mamba-790M	NeoX	7.33	6.02	62.7	55.1	72.1	61.2	29.5	56.1	57.1
GPT-Neo 1.3B	GPT2	—	7.50	57.2	48.9	71.1	56.2	25.9	54.9	52.4
Hybrid H3-1.3B	GPT2	—	11.25	49.6	52.6	71.3	59.2	28.1	56.9	53.0
OPT-1.3B	OPT	—	6.64	58.0	53.7	72.4	56.7	29.6	59.5	55.0
Pythia-1.4B	NeoX	7.51	6.08	61.7	52.1	71.0	60.5	28.5	57.2	55.2
RWKV-1.5B	NeoX	7.70	7.04	56.4	52.5	72.4	60.5	29.4	54.6	54.3
Mamba-1.4B	NeoX	6.80	5.04	64.9	59.1	74.2	65.5	32.8	61.5	59.7
GPT-Neo 2.7B	GPT2	—	5.63	62.2	55.8	72.1	61.1	30.2	57.6	56.5
Hybrid H3-2.7B	GPT2	—	7.92	55.7	59.7	73.3	65.6	32.3	61.4	58.0
OPT-2.7B	OPT	—	5.12	63.6	60.6	74.8	60.8	31.3	61.0	58.7
Pythia-2.8B	NeoX	6.73	5.04	64.7	59.3	74.0	64.1	32.9	59.7	59.1
RWKV-3B	NeoX	7.00	5.24	63.9	59.6	73.7	67.8	33.1	59.6	59.6
Mamba-2.8B	NeoX	6.22	4.23	69.2	66.1	75.2	69.7	36.3	63.5	63.3
GPT-J-6B	GPT2	—	4.10	68.3	66.3	75.4	67.0	36.6	64.1	63.0
OPT-6.7B	OPT	—	4.25	67.7	67.2	76.3	65.6	34.9	65.5	62.9
Pythia-6.9B	NeoX	6.51	4.45	67.1	64.0	75.2	67.3	35.5	61.3	61.7
RWKV-7.4B	NeoX	6.31	4.38	67.2	65.5	76.1	67.8	37.5	61.0	62.5

Рис. 18: Результаты основного эксперимента из оригинальной статьи Mamba [2]. Оригинальную статью Mamba реджектнули с ICML 2024. Сможете ли вы предположить, почему? Комментарии ревьюеров тут: <https://openreview.net/forum?id=AL1fq05o7H>