

# Текстовые эмбединги. Информационный поиск. RAG

[Contribute on GitHub](#)

Октябрь 2025.

## Содержание

<b>1</b>	<b>Математические предпосылки</b>	<b>2</b>
1.1	Разложение по базису в векторных пространствах . . . . .	2
1.2	Функциональные пространства и ортонормированные базисы . . . . .	2
1.3	Полиномиальные проекции . . . . .	3
1.4	Обыкновенные дифференциальные уравнения (ОДУ) . . . . .	3
1.5	Временные ряды . . . . .	4
1.6	Рекуррентные нейронные сети . . . . .	4
<b>2</b>	<b>HiPPO: Рекуррентная память с оптимальными полиномиальными проек-</b>	<b>5</b>
	<b>циями</b>	
2.1	Фреймворк . . . . .	5
2.2	Реализации HiPPO . . . . .	6
<b>3</b>	<b>От теории к практике</b>	<b>6</b>
3.1	Linear State-Space Layer (LSSL) . . . . .	6
3.2	Стратегии параметризации . . . . .	8
<b>4</b>	<b>Mamba: Селективные модели пространства состояний</b>	<b>9</b>
4.1	Мотивация . . . . .	9
4.2	Linear Attention . . . . .	9
4.3	Hungry Hungry Hippos (H3) . . . . .	10
4.4	Селективные модели пространства состояний . . . . .	10
4.5	Архитектура Mamba . . . . .	11
4.6	Экспериментальная оценка . . . . .	11
4.7	Характеристики производительности . . . . .	11

## Аннотация

Данная лекция представляет всесторонний обзор State-Space Models (SSMs), прослеживая их эволюцию от основополагающего фреймворка HiPPO до современной архитектуры Mamba. Мы начинаем с математических основ: полиномиальных проекций и дифференциальных уравнений; затем исследуем, как эти концепции были адаптированы в эффективные архитектуры для моделирования последовательностей. Лекция охватывает три основных развития: рекуррентную память HiPPO с оптимальными полиномиальными проекциями, переход к структурированным слоям (LSSL, S4, DSS, S4D), и наконец, селективные модели пространства состояний, воплощенные в Mamba. На протяжении всего изложения мы рассматриваем как теоретические основы, так и практические реализации, подчеркивая, как эти модели достигают линейной временной сложности, сохраняя при этом сопоставимую производительность с трансформерами на различных задачах моделирования последовательностей.

## 1. Математические предпосылки

### 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 признаковым описанием исходного вектора.

### 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	$L^2[0, +\infty)$	$w(x) = e^{-x}$	$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (e^{-x} x^n)$
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

### 1.3. 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.

### 1.4. 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:

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

## 1.5. 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.

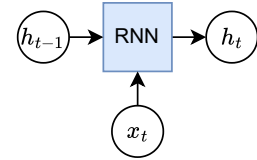
## 1.6. Рекуррентные нейронные сети

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.

## 2. HiPPO: Рекуррентная память с оптимальными полиномиальными проекциями

### 2.1. Фреймворк

Сделаем предположение: пусть за всяким одномерным временным рядом  $\{x_i\}_{i=1}^T$  стоит некоторая непрерывная функция  $f(t)$ , определенная при  $t \in [0, +\infty)$ . Временной ряд при этом будем называть дискретизацией функции  $f(t)$  по некоторой сетке значений  $t$ . Основная идея HiPPO в том, чтобы для временного ряда  $\{x_i\}_{i=1}^T$  получить коэффициенты  $\vec{c} \in \mathbb{R}^N$  полиномиальной 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:

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

где  $A \in \mathbb{R}^{N \times N}$  и  $B \in \mathbb{R}^N$  — некоторые константы, определяемые выбранным полиномиальным 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 ряда. Таким образом, для вычисления признакового описания временного ряда, достаточно произвести пересчет по рекуррентным формулам для  $c_{k+1}$ . В этом и заключается HiPPO — High Order Polynomial Projections. Этот метод стремится решать задачи на временных рядах с помощью рекуррентной памяти, используя коэффициенты разложения как скрытые состояния. Они служат математическим обоснованием оптимальности сжатия исторической информации.

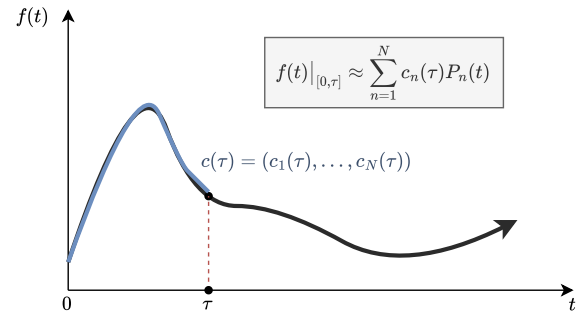


Рис. 1: Идея HiPPO: аппроксимировать функцию на отрезке с помощью полиномиальной проекции.

## 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 (таблица 2). Заметим, что выбор базиса влияет на то, как модель использует предыдущие наблюдения  $f(t)$  (рисунок 2).

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

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

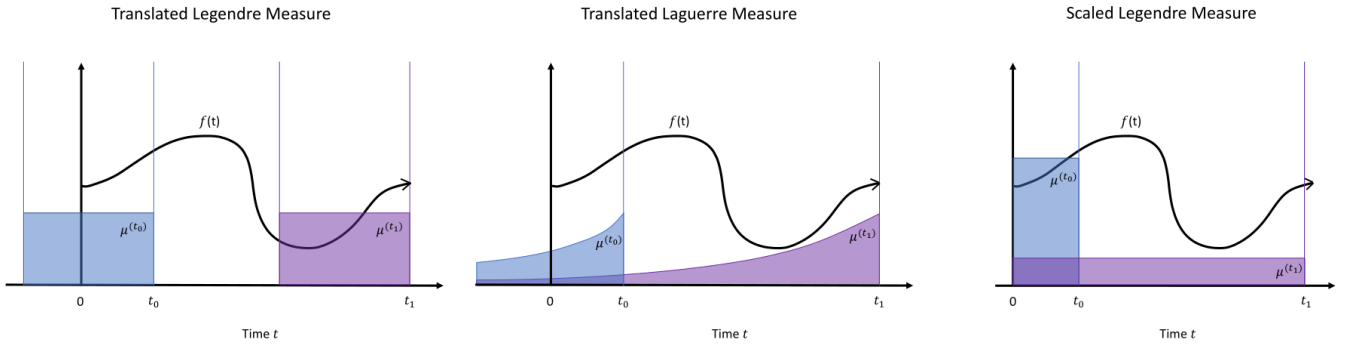


Рис. 2: Весовые функции и то, как они учитывают историю  $f(t)$ .

## 3. От теории к практике

### 3.1. Linear State-Space Layer (LSSL)

Основываясь на HiPPO, LSSL 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  $\{y_t \mid y_t \in \mathbb{R}^M\}$ :

$$\begin{aligned} x_t &= \bar{A}x_{t-1} + \bar{B}u_t \\ y_t &= Cx_t + Du_t \end{aligned}$$

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 параметрв.

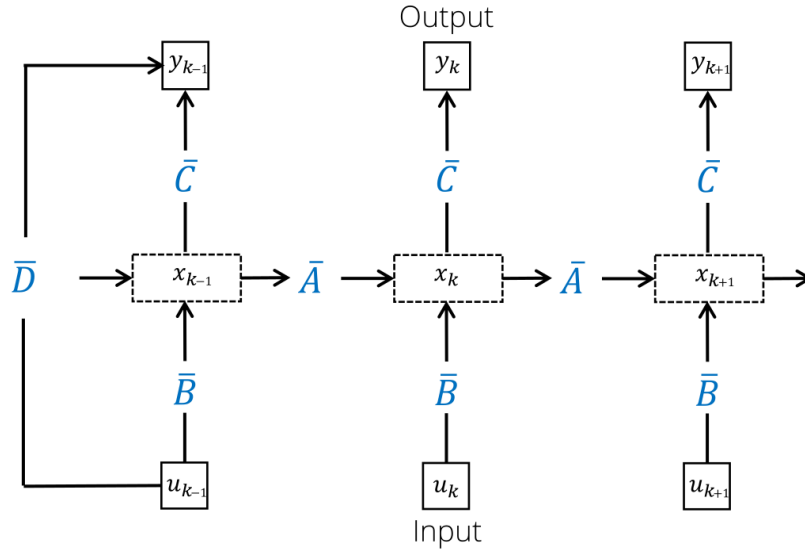


Рис. 3: Визуализация linear state-space layer (LSSL) как рекуррентной сети.

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

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

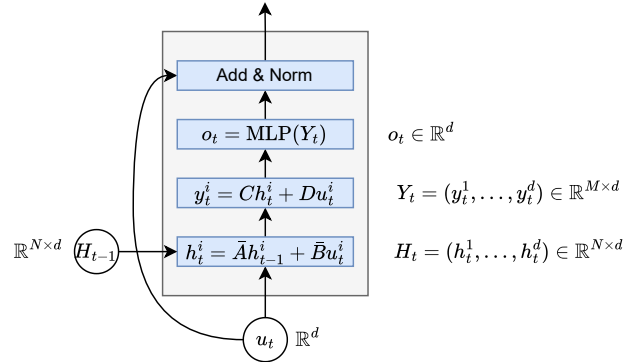


Рис. 4: Архитектура linear state-space layer.

Авторы 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 наперед, как на обучении. Однако у LSSL есть теоретический способ для ускорения.

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

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

Это может быть записано как:

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

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  $3 \times 3$  (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.

### 3.2. Стратегии параметризации

Выше упоминалось, что матрицу  $A$  можно обучать. Согласно экспериментам оригинальной статьи, 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 предсчитать  $\mathcal{K}$  один раз.

Чтобы решить проблему накладных расходов, связанных с высчитыванием  $\mathcal{K}$ , авторы применили трюк с репараметризацией. 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) использует параметризацию:

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

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

---

#### Algorithm 1 S4 CONVOLUTION KERNEL (SKETCH)

---

**Input:** S4 parameters  $\Lambda, P, Q, B, C \in \mathbb{C}^N$  and step size  $\Delta$

**Output:** SSM convolution kernel  $\bar{K} = \mathcal{K}_L(\bar{A}, \bar{B}, \bar{C})$  for  $A = \Lambda - PQ^*$  (equation (5))

---

- 1:  $\tilde{C} \leftarrow (I - \bar{A}^L)^* \bar{C}$  ▷ Truncate SSM generating function (SSMGF) to length  $L$
  - 2:  $\begin{bmatrix} k_{00}(\omega) & k_{01}(\omega) \\ k_{10}(\omega) & k_{11}(\omega) \end{bmatrix} \leftarrow [\tilde{C} Q]^* \left( \frac{2}{\Delta} \frac{1-\omega}{1+\omega} - \Lambda \right)^{-1} [B P]$  ▷ Black-box Cauchy kernel
  - 3:  $\hat{K}(\omega) \leftarrow \frac{2}{1+\omega} [k_{00}(\omega) - k_{01}(\omega)(1 + k_{11}(\omega))^{-1}k_{10}(\omega)]$  ▷ Woodbury Identity
  - 4:  $\hat{K} = \{\hat{K}(\omega) : \omega = \exp(2\pi i \frac{k}{L})\}$  ▷ Evaluate SSMGF at all roots of unity  $\omega \in \Omega_L$
  - 5:  $\bar{K} \leftarrow \text{iFFT}(\hat{K})$  ▷ Inverse Fourier Transform
- 

#### Диагональная параметризация (DSS, S4D)

Самый простой подход 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  $P_{ij} = \lambda_i j \cdot dt$ .

## 4. Mamba: Селективные модели пространства состояний

### 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.

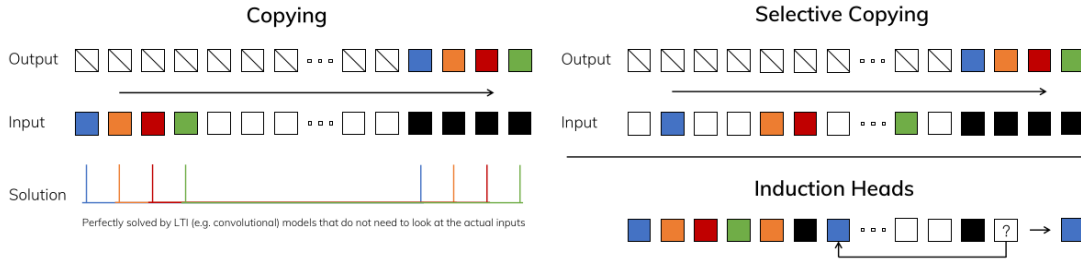


Рис. 5: Синтетические задачи языкового моделирования: копирование.

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.

### 4.2. Linear Attention

До Mamba, Linear Attention показал, как эффективно аппроксимировать внимание Transformer:

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

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

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

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

Определим величины  $S_i, Z_i$ :

$$O_i^T = \frac{\overbrace{\phi(Q_i)^T \sum_{j=1}^i \phi(K_j) V_j^T}^{S_i}}{\underbrace{\phi(Q_i)^T \sum_{j=1}^i \phi(K_j)}_{Z_i}} = \frac{\phi(Q_i)^T S_i}{\phi(Q_i)^T Z_i}.$$

Получили рекуррентный инференс:

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

### 4.3. Hungry Hungry Hippos (H3)

H3 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 Transformer средствами SSM (рисунок 7). Для эффективности авторы опираются на Flash Convolution для правильной утилизации GPU.

### 4.4. Селективные модели пространства состояний

Mamba 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  $x_t$ .

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

Поскольку свёртка в явном виде недоступна, для распараллеливания рекуррентности используется параллельное сканирование: оно обеспечивает  $O(N)$ -параллелизм на обучении, сохраняет линейно-временной инференс и адаптирует алгоритмы кумулятивной суммы к динамике SSM.

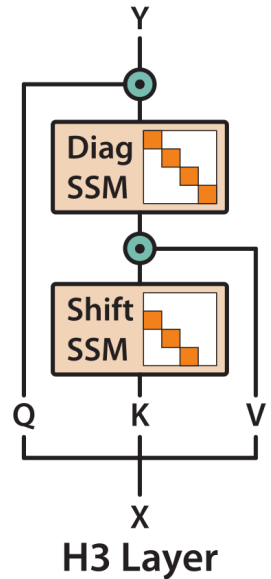


Рис. 7: Слой H3.

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

Рис. 8: Алгоритм SSM для Mamba.

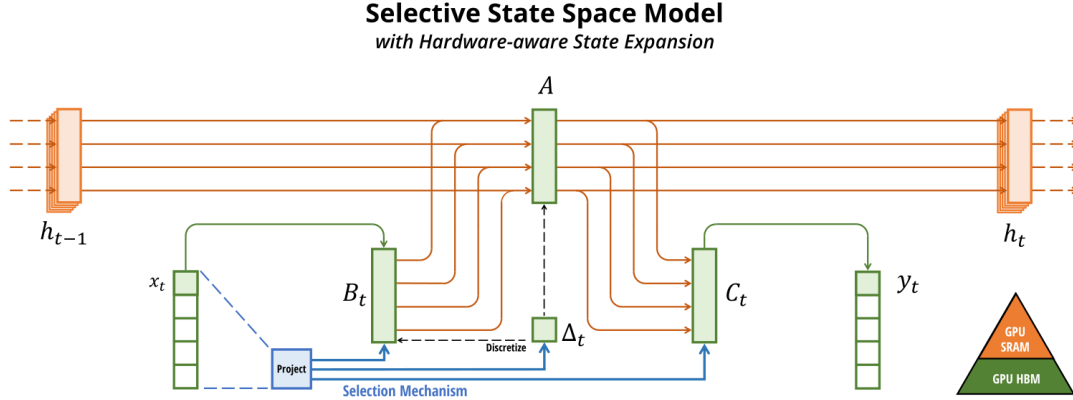


Figure 1: (**Overview.**) Structured SSMs independently map each channel (e.g.  $D = 5$ ) of an input  $x$  to output  $y$  through a higher dimensional latent state  $h$  (e.g.  $N = 4$ ). Prior SSMs avoid materializing this large effective state ( $DN$ , times batch size  $B$  and sequence length  $L$ ) through clever alternate computation paths requiring time-invariance: the  $(\Delta, A, B, C)$  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.

## 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) и заключительную проекцию обратно к исходной размерности (рисунк 10).

## 4.6. Экспериментальная оценка

### Задачи длинного контекста

На QA с длинными контекстами и в синтетических тестах (Phonebook, RULER) модель дает сопоставимое качество и лучше экстраполирует длину, чем Transformers.

### Гибридные архитектуры

Комбинирование с компонентами Transformer даёт «лучшее из двух миров»: эффективность Mamba сочетается с возможностями внимания, что обеспечивает более высокое качество и гибкий дизайн.

## 4.7. Характеристики производительности

Mamba обеспечивает линейно-временной инференс и постоянное использование памяти, хорошо экстраполирует длину и демонстрирует сопоставимую перплексию. Огра-

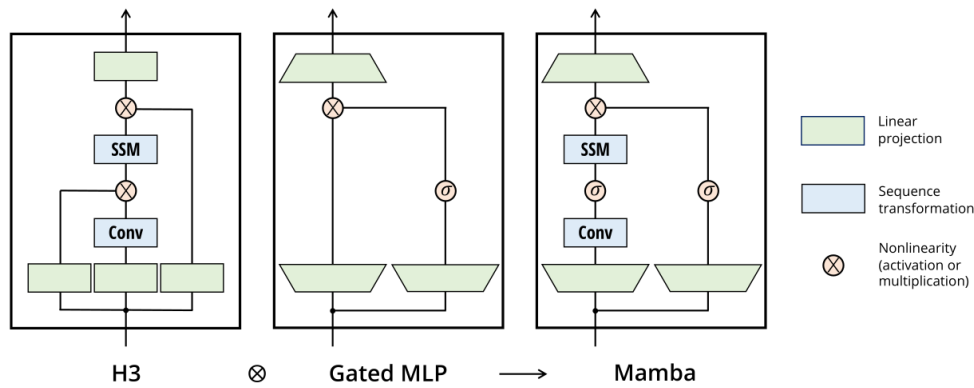


Figure 3: **(Architecture.)** Our simplified block design combines the H3 block, which is the basis of most SSM architectures, with the ubiquitous MLP block of modern neural networks. Instead of interleaving these two blocks, we simply repeat the Mamba block homogenously. Compared to the H3 block, Mamba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

Рис. 10: Архитектура Мамба.

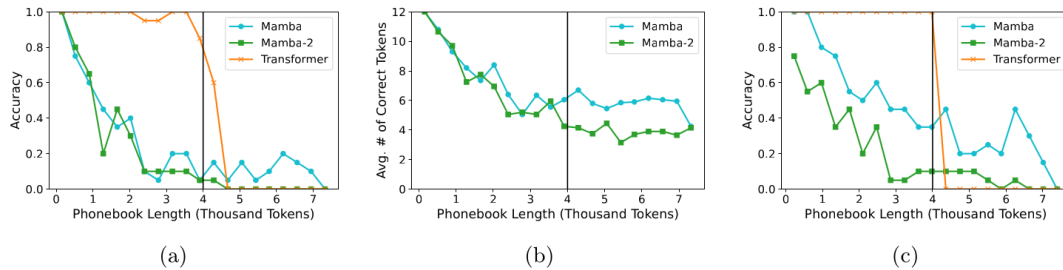


Figure 3: Evaluation results for pure SSM and Transformer models (trained for 1.1T tokens) on the Phonebook task illustrated in Figure 2b. **(a)** On the standard Phonebook task, Transformers are capable of in-context learning and answering questions that require copying from the input, but SSM models 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.

ничения касаются in-context learning (уступает Transformers), меньшей точности «памяти» по сравнению с вниманием и повышенной чувствительности к формату входа.

Оригинальную статью Мамба реджектнули с ICML. Сможете ли вы понять по таблице 12, почему?

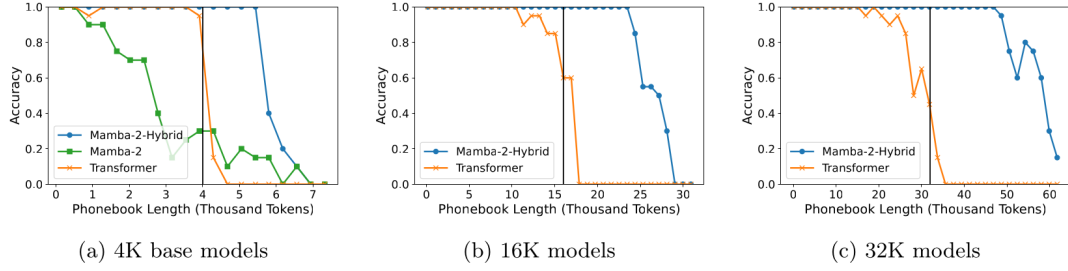


Figure 7: Phonebook evaluation results for 8B-parameter models trained on 3.5T tokens and their long-context extensions. We use the standard Phonebook formulation (Figure 2b). Mamba-2-Hybrid models can generalize beyond their pretraining sequence length and perform the Phonebook task on contexts longer 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. 11

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	<b>51.9</b>	40.6
<b>Mamba-130M</b>	NeoX	<b>10.56</b>	<b>16.07</b>	<b>44.3</b>	<b>35.3</b>	<b>64.5</b>	<b>48.0</b>	<b>24.3</b>	<b>51.9</b>	<b>44.7</b>
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
<b>Mamba-370M</b>	NeoX	<b>8.28</b>	<b>8.14</b>	<b>55.6</b>	<b>46.5</b>	<b>69.5</b>	<b>55.1</b>	<b>28.0</b>	<b>55.3</b>	<b>50.0</b>
Pythia-1B	NeoX	7.82	7.92	56.1	47.2	70.7	57.0	27.1	53.5	51.9
<b>Mamba-790M</b>	NeoX	<b>7.33</b>	<b>6.02</b>	<b>62.7</b>	<b>55.1</b>	<b>72.1</b>	<b>61.2</b>	<b>29.5</b>	<b>56.1</b>	<b>57.1</b>
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
<b>Mamba-1.4B</b>	NeoX	<b>6.80</b>	<b>5.04</b>	<b>64.9</b>	<b>59.1</b>	<b>74.2</b>	<b>65.5</b>	<b>32.8</b>	<b>61.5</b>	<b>59.7</b>
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
<b>Mamba-2.8B</b>	NeoX	<b>6.22</b>	<b>4.23</b>	<b>69.2</b>	<b>66.1</b>	<b>75.2</b>	<b>69.7</b>	<b>36.3</b>	<b>63.5</b>	<b>63.3</b>
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. 12: Результаты основного эксперимента из оригинальной статьи Мамба.