Текстовые эмбединги. Информационный поиск. RAG

Contribute on GitHub

Октябрь 2025.

Содержание

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 пространства состояний (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

Сперва мы вспомним математические основы, которые пригодятся для понимания State-Space Models.

1.1. Математические основы

1.1.1. Разложение по базису в векторных пространствах

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

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

где $\vec{c} = (c_1, \ldots, c_N)$ — это вектор координат, представляющий $\vec{a}$ в базисе B.
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 ортонормированном базисе векторы координат $\vec{c}$ образуют векторное пространство $\mathbb{R}^N$. Если допустить некоторую нестрогость интерпретации, то с точки зрения машинного обучения это означает, что вектор координат является идеальным признаковым 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

Отличие пространства Лебега от векторного пространства в предыдущем разделе в том, что базис L^2 бесконечный, в то время как базис $V(\mathbb{R})$ конечный. 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 $f(x) \in L^2[-1, 1]$ и ортонормированного базиса $\{P_n(x)\}_{n=1}^{\infty}$ справедлива формула оптимальной аппроксимации:

$$f(x) = \sum_{n=1}^{\infty} c_n P_n(x) \approx \sum_{n=1}^{N} c_n P_n(x)$$

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 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:

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

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.

1.1.5. Временные ряды

Одномерным временным рядом (univariate time series) мы будем называть числовую последовательность $x_1, \ldots, x_T$. Число $x_t \in \mathbb{R}$ мы называем наблюдением, или замером, (observation) в момент времени t (timestamp). Эти названия отражают, 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 $x_1, \ldots, x_T$. 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:

$$\text{RNN}(h, x) = (1 - g) \circ h + g \circ \tanh(W_1 h + U_1 x + b_1), \tag{1}$$

$$g = \sigma(W_2 h + U_2 x + b_2) \tag{2}$$

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 $f(t)\big|_{[0,\tau]}$. 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 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:

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

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.

2.2. Реализации HiPPO

В зависимости от выбранного базиса, мы получаем разные $\bar{A}$ и $\bar{B}$. Конкретные значения выводятся аналитически. Ниже мы приводим значения, которые получили авторы HiPPO.

Translated Legendre (LegT)

- Пространство: $L^2[\tau - \theta, \tau]$ (скользящее окно)
- Весовая функция: $w(t) = \frac{1}{\theta}[\tau - \theta \leq t \leq \tau]$
- Элементы матрицы:

$$A_{nk} = \frac{1}{\theta}\begin{cases}(-1)^{n-k}(2n+1), & n \geq k \\ 2n+1, & n \leq k\end{cases}$$

Translated Laguerre (LagT)

- Пространство: $L^2[-\infty, \tau]$ (экспоненциальное взвешивание)
- Весовая функция: $w(t) = \exp(t - \tau)[t \leq \tau]$
- Элементы матрицы:

$$A_{nk} = \begin{cases}1, & n \geq k \\ 0, & n < k\end{cases}$$

Scaled Legendre (LegS)

- Пространство: $L^2[0, \tau]$ (полная история)
- Весовая функция: $w(t) = \frac{1}{\tau}[0 \leq t \leq \tau]$
- Элементы матрицы:

$$A_{nk} = -\frac{1}{\tau}\begin{cases}\sqrt{(2n+1)(2k+1)}, & n > k \\ n+1, & n = k \\ 0, & n < k\end{cases}$$

2.3. Интеграция с нейронными сетями и эксперименты

Для экспериментальной проверки HiPPO был интегрирован с RNN путем использования коэффициентов разложения как дополнительного контекста:

$$\text{RNN}(h, [c, x])$$

где $c_t = \bar{A}c_{t-1} + \bar{B}f_t$ представляет коэффициенты HiPPO для $f(t) = w^T h_t$ (с обучаемым w).

В свое время HiPPO достиг SOTA на ряде задач: на Permuted MNIST измерялась точность классификации изображений как классификация временного ряда; в задаче прогнозировании для хаотической системы Mackey–Glass; на IMDB Sentiment классификация текстов. Ключевые преимущества HiPPO

быстрый рекуррентный инференс и теоретические гарантии точности аппроксимации. На задачах временных рядов HiPPO показал результаты, сопоставимые не только с RNN, но и с трансформером, превосходя их по скорости инференса в десятки и сотни раз.

3. От теории к практике

3.1. Linear State-Space Layer (LSSL)

Эксперименты HiPPO проводились на практически синтетических и игрушечных датасетах. Основываясь на HiPPO, LSSL трансформирует теоретический фреймворк HiPPO в слой нейронной сети, пригодный для более серьезных применений.

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

$$x_t = \bar{A}x_{t-1} + \bar{B}u_t \tag{3}$$

$$y_t = Cx_t + Du_t \tag{4}$$

где:

- $\bar{A} \in \mathbb{R}^{N \times N}$, $\bar{B} \in \mathbb{R}^N$ инициализируется из HiPPO матриц (опционально обучаемые, но об этом позже)
- $C \in \mathbb{R}^{M \times N}$, $D \in \mathbb{R}^M$ (обучаемые)

В теории оптимального управления подобные системы называют моделями пространства состояний

state-space model (SSM).

Используя SSM, авторы HiPPO построили Linear State-Space Layer (LSSL):

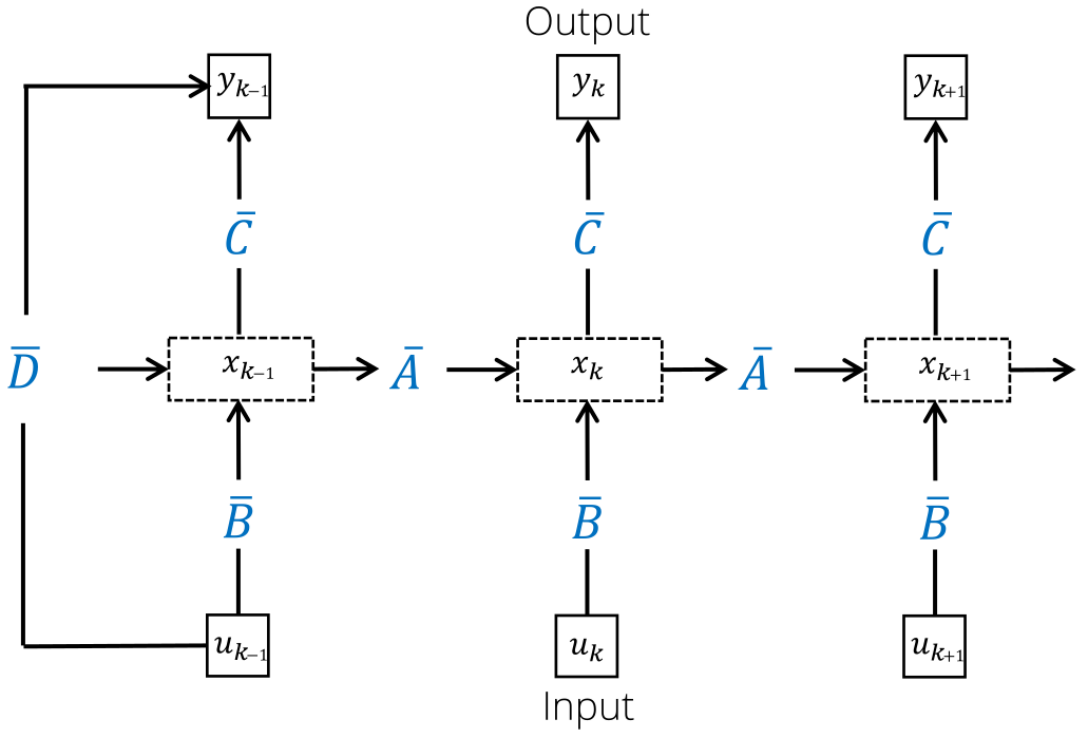
Авторы стремились реализовать компоненты, напоминающие компоненты трансформерного слоя:

- коммуникация между токенами
- position-wise обработка
- add & norm

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:

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

Это может быть записано как:

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

где ядро:

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

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 (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 предпосчитать $\mathcal{K}$ один раз.

Table 5: (**Modeling and Computational Benefits of LSSLs.**) In each benchmark category, we compare the number of epochs (ep.) it takes a LSSL-f to reach the previous SoTA (PSoTA) results as well as a near-SoTA target. We also report the wall clock time it took to reach PSoTA relative to the previous best model.

	Permuted MNIST			BDIMC Heart Rate			Speech Commands RAW		
	98% Acc.	PSoTA	Time	1.5 RMSE	PSoTA	Time	65% Acc.	PSoTA	Time
LSSL-fixed	16 ep.	104 ep.	0.19×	9 ep.	10 ep.	0.07×	9 ep.	10 ep.	0.14×
CKConv	118 ep.	200 ep.	1.0×	✗	✗	✗	188 ep.	280 ep.	1.0×
UnICORNN	75 ep.	✗	✗	116 ep.	467 ep.	1.0×	✗	✗	✗

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 Λ — 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 $\boldsymbol{\Lambda}, \boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{C}^N$ and step size Δ
Output: SSM convolution kernel $\overline{\boldsymbol{K}} = \mathcal{K}_L(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}})$ for $\boldsymbol{A} = \boldsymbol{\Lambda} - \boldsymbol{P}\boldsymbol{Q}^*$ (equation (5))

1: $\widetilde{\boldsymbol{C}} \leftarrow \left(\boldsymbol{I} - \overline{\boldsymbol{A}}^L\right)^* \overline{\boldsymbol{C}}$ ▷ Truncate SSM generating function (SSMGF) to length L

2: $\begin{bmatrix} k_{00}(\omega) & k_{01}(\omega) \\ k_{10}(\omega) & k_{11}(\omega) \end{bmatrix} \leftarrow \left[\widetilde{\boldsymbol{C}}\ \boldsymbol{Q}\right]^* \left(\frac{2}{\Delta}\frac{1-\omega}{1+\omega} - \boldsymbol{\Lambda}\right)^{-1} [\boldsymbol{B}\ \boldsymbol{P}]$ ▷ Black-box Cauchy kernel

3: $\hat{\boldsymbol{K}}(\omega) \leftarrow \frac{2}{1+\omega}\left[k_{00}(\omega) - k_{01}(\omega)(1 + k_{11}(\omega))^{-1}k_{10}(\omega)\right]$ ▷ Woodbury Identity

4: $\hat{\boldsymbol{K}} = \{\hat{\boldsymbol{K}}(\omega) : \omega = \exp(2\pi i \frac{k}{L})\}$ ▷ Evaluate SSMGF at all roots of unity $\omega \in \Omega_L$

5: $\overline{\boldsymbol{K}} \leftarrow \mathsf{iFFT}(\hat{\boldsymbol{K}})$ ▷ Inverse Fourier Transform

Диагональная параметризация (DSS, S4D)

Самый простой подход использует диагональные матрицы:

$$A = \mathsf{diag}(\lambda_1, \ldots, \lambda_N)$$

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

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

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

3.3. Экспериментальные результаты

На Long-Range Arena (классификация очень длинных последовательностей) S4 показывает прорывное качество, заметно опережая прежние подходы. В сравнении с трансформерами, в языковом моделировании демонстрируется сопоставимая перплексия при

генерации, которая выполняется на порядки быстрее (до 60×). Кроме того, модель успешно обучается напрямую на сырых аудиосигналах с дискретизацией 16 кГц, без спектральной предобработки.

Table 8: (**WikiText-103 language modeling**) S4 approaches the performance of Transformers with much faster generation. (*Top*) Transformer baseline which our implementation is based on, with attention replaced by S4. (*Bottom*) Attention-free models (RNNs and CNNs).

Model	Params	Test ppl.	Tokens / sec
Transformer	247M	**20.51**	0.8K (1×)
GLU CNN	229M	37.2	-
AWD-QRNN	151M	33.0	-
LSTM + Hebb.	-	29.2	-
TrellisNet	180M	29.19	-
Dynamic Conv.	255M	25.0	-
TaLK Conv.	240M	23.3	-
S4	249M	**20.95**	**48K (60×)**

---

4. Mamba: Селективные пространства состояний

4.1. Мотивация и постановка проблемы

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.

Задача селективного копирования

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

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.

4.2. Предыстория: Linear Attention

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

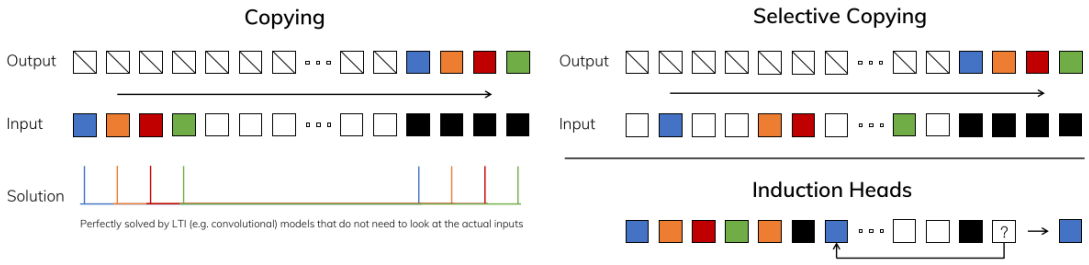


Table 1: Synthetic language modeling tasks.

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

Transformer attention $\text{softmax}(QK^T)V$ can be viewed as the following:

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

where $\text{sim}(q, k) = \exp(q^T k)$. If you choose $\text{sim}(q, k) = \phi(q)^T \phi(k)$ with some non-linear $\phi : \mathbb{R}^d \to \mathbb{R}^d$, then

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

Let us define S_i, Z_i:

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

Note, that

$$S_i = S_{i-1} + \phi(K_i) V_i^T, \tag{5}$$

$$Z_i = Z_{i-1} + \phi(K_i). \tag{6}$$

This gives us linear-time inference.

4.3. Hungry Hungry Hippos (H3)

H3 связал SSM и языковое моделирование, аппроксимируя внимание компонентами SSM:

Слой H3

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

Эта архитектура аппроксимирует внимание Transformer средствами SSM, достигает сопоставимых результатов на SuperGLUE и опирается на Flash Convolution для эффективной утилизации GPU.

Table 2: Evaluation of 2-layer models on synthetic language tasks.

Task	Random	S4D	Gated State Spaces	H3	Attention
Induction Head	5.0	35.6	6.8	**100.0**	**100.0**
Associative Recall	25.0	86.0	78.0	99.8	**100.0**

4.4. Архитектура Mamba

Селективные пространства состояний
Mamba вводит входозависимые параметры:

$$x_t = \bar{A}(x_t)x_{t-1} + \bar{B}(x_t)u_t \tag{7}$$
$$y_t = C(x_t)x_t + D(x_t)u_t \tag{8}$$

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

Детали реализации
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.

Алгоритм параллельного сканирования
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.

4.5. Компоненты архитектуры Mamba

Полная архитектура 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.

Оптимизация иерархии памяти
Для повышения утилизации оборудования Mamba различает быструю on-chip SRAM для активных вычислений и HBM для хранения параметров, а также использует объединённые (fused) операции, минимизирующие пересылки данных.

4.6. Экспериментальная оценка

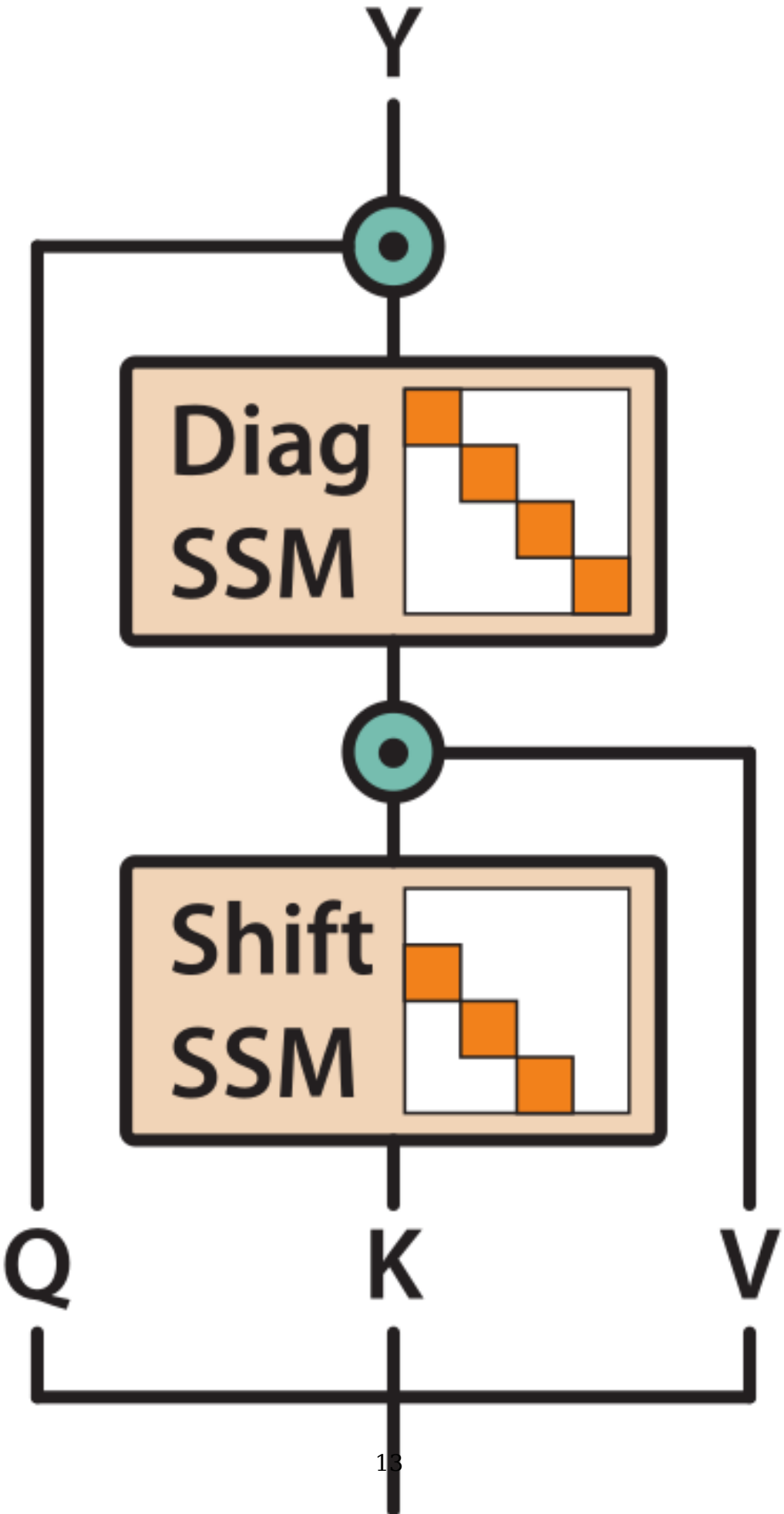
Задачи длинного контекста
На QA с длинными контекстами и в синтетических тестах (Phonebook, RULER) модель дает сопоставимое качество и лучше экстраполирует длину, чем Transformers.

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

Оригинальную статью Mamba реджектнули с ICML. Сможете ли вы понять по этой таблице, почему?


Y
Diag
SSM
Shift
SSM
Q
K
V

Algorithm 1 SSM (S4)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
1: $\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
3: $\boldsymbol{C} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
4: $\Delta : (\mathsf{D}) \leftarrow \tau_\Delta(\text{Parameter})$
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{D}, \mathsf{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
7: **return** y

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
Output: $y : (\mathsf{B}, \mathsf{L}, \mathsf{D})$
1: $\boldsymbol{A} : (\mathsf{D}, \mathsf{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_B(x)$
3: $\boldsymbol{C} : (\mathsf{B}, \mathsf{L}, \mathsf{N}) \leftarrow s_C(x)$
4: $\Delta : (\mathsf{B}, \mathsf{L}, \mathsf{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathsf{B}, \mathsf{L}, \mathsf{D}, \mathsf{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
7: **return** y

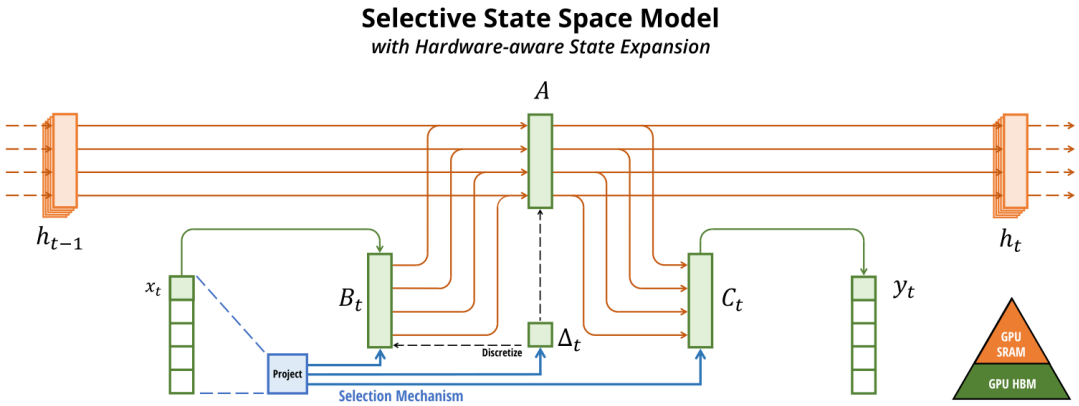


Figure 1: (**Overview**.) Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g. $N = 4$). Prior SSMs avoid materializing this large effective state (DN, times batch size B and sequence length L) through clever alternate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

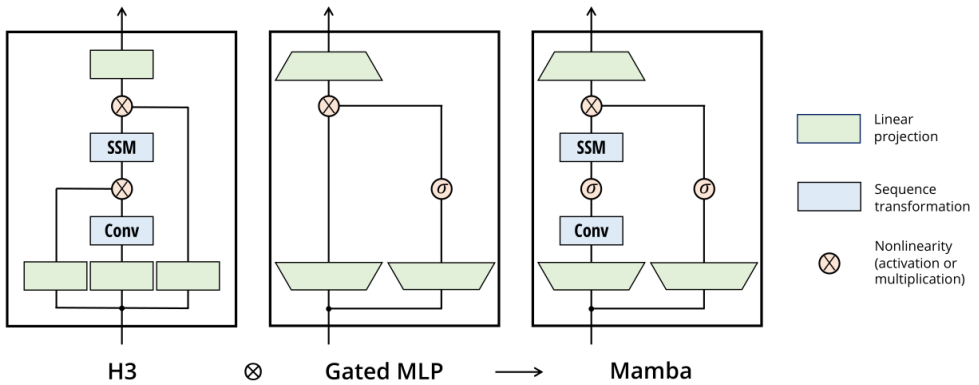
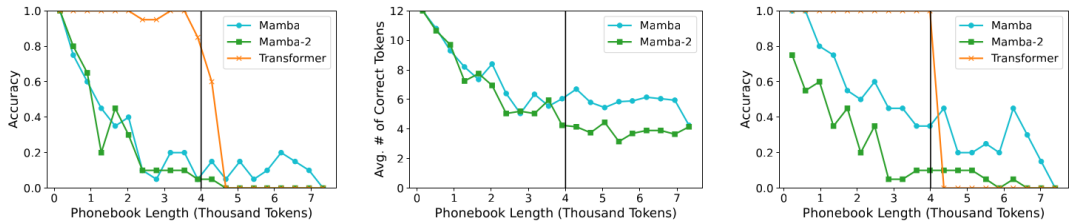


Figure 3: (**Architecture**.) Our simplified block design combines the H3 block, which is the basis of most SSM architectures, with the ubiquitous MLP block of modern neural networks. Instead of interleaving these two blocks, we simply repeat the Mamba block homogenously. Compared to the H3 block, Mamba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



(a) (b) (c)

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.

Table 10: Evaluation results of 8B-parameter Mamba-2-Hybrid and Transformer models trained for 3.5T tokens, plus their long-context extensions, on the synthetic RULER long-context benchmark suite (Hsieh et al. 2024). Mamba-2-Hybrid models achieve higher accuracy than Transformers on these tasks, highlighting their improved ability to recall, trace, and aggregate information across long inputs.

Model	Synthetic Tasks					Needle-In-A-Haystack Tasks (NIAH)								Avg
	HotpotQA	SquadQA	CWE	VT	KWE	NIAH-1	NIAH-2	NIAH-3	MK-NIAH-1	MK-NIAH-2	MK-NIAH-3	MV-NIAH	MQ-NIAH	
Mamba-2	31.75	35.5	32.87	76.45	**76.58**	100	98	83	40.25	13.75	5.5	35	49.12	52.14
Transformer-4K	42.25	56.5	**36.42**	79.2	76.33	100	100	75.25	**99.5**	94	58	96.56	95.06	77.62
Mamba-2-Hybrid-4k	**48.75**	56.5	32.2	**90.55**	65.58	100	100	**95.75**	89.5	**95.5**	**96**	**97.94**	**97.62**	**81.99**
Transformer-16K	37.25	45	3	7.25	**58.33**	100	99.5	75.75	**93.75**	56.5	57.5	88.5	87.94	62.33
Mamba-2-Hybrid-16k	**44**	**48.75**	**12.88**	**83.2**	46.83	100	100	**81.5**	92	**92.25**	**83**	**89.81**	**90.19**	**74.19**
Transformer-32K	35	41.5	4.42	0.35	**53.5**	100	99	76.5	70.75	57.75	41.25	69.69	**84.69**	56.49
Mamba-2-Hybrid-32K	**38.5**	41.75	**8.4**	**79.9**	36.5	100	100	**96.75**	**84**	**76.5**	**81.5**	**84.31**	80.94	**69.93**

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