落合式プログラミングコミュニティー

KimiK3から学ぶLLMのattention機構の進化

落合式講義

speaker: にころ@nikoro_is_coder

にころ

@nikoro_is_coder

- 東京大学大学院 情報理工学系研究科 **修士1年**
- 自然言語処理、特に **LLMの解釈性** を研究
- **Kaggle Expert**
- AtCoder **Algorithm 青 / Heuristic 青**

Kaggle **アルゴ** **ヒューリスティック**

の話はついていけます。

画像：@nikoro_is_coder 公開プロフィール画像（note.com/nikoro256）

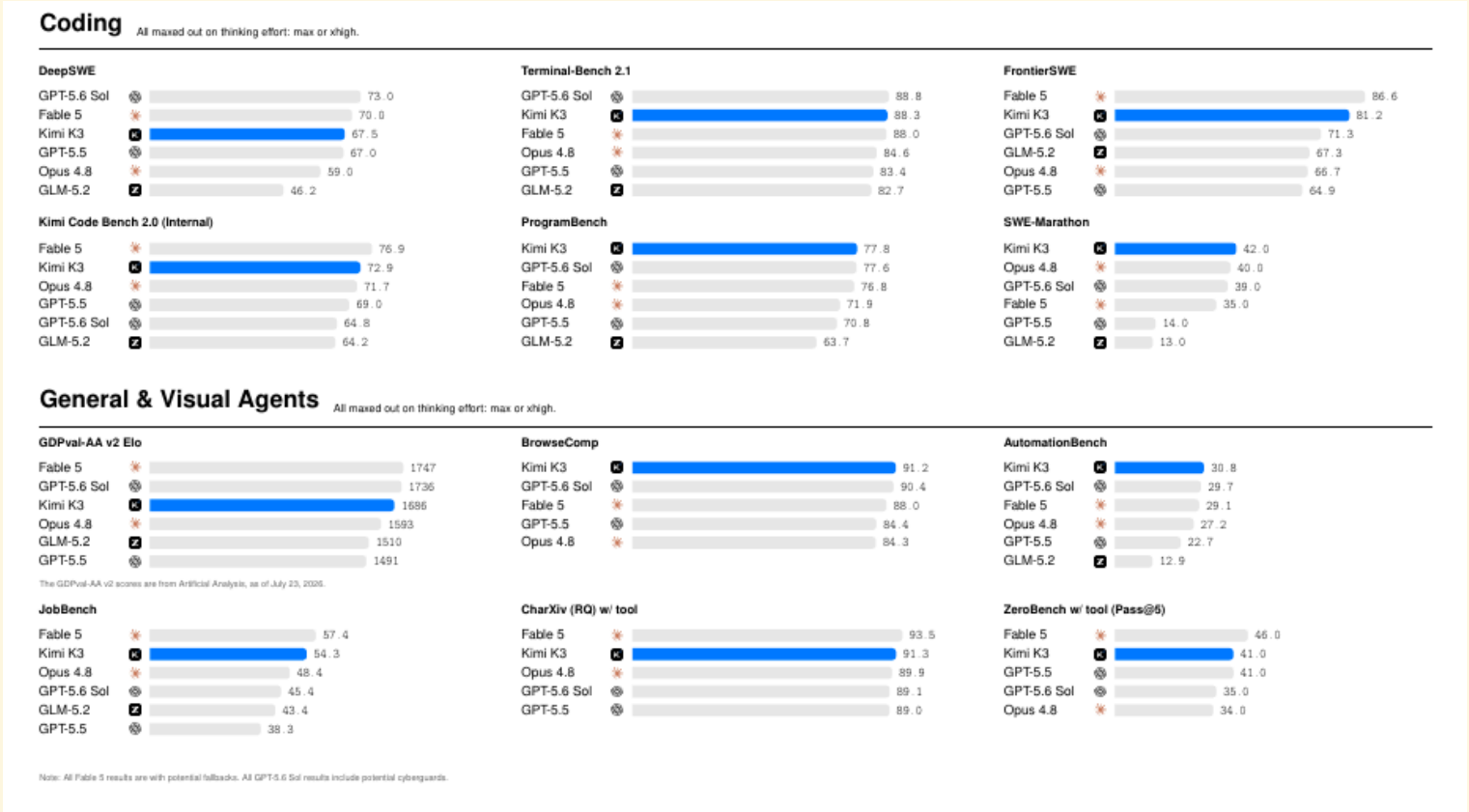
Kimi K3とは？

・中国・北京のムーンショットAI（月之暗面）が開発したLLM
・公式Coding Agent 「**Kimi Code**」

画像：Kimi Code公式ページ（2026-08-18取得）, https://www.kimi.com/code/　K3対応：[1] Kimi Code Docs, Model Configuration, https://www.kimi.com/code/docs/en/kimi-code/models.html

Kimi K3すごい

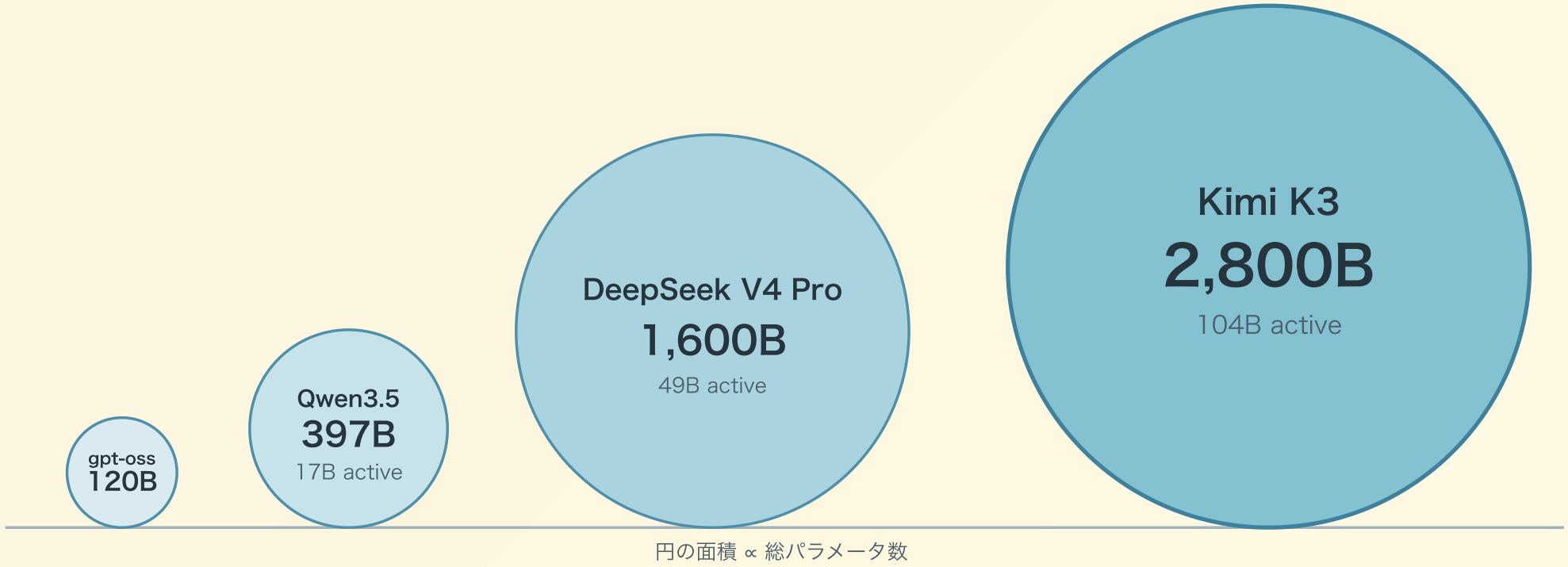
モデルの重み・アーキテクチャが公開されている、公開パラメータなのに、これまでの最高性能のモデルに匹敵する。
最高性能のモデルのアーキテクチャ・重みが公開されるのは異常



図：Kimi Team, “Kimi K3: Open Frontier Intelligence,” Figure 1 (2026), https://arxiv.org/abs/2607.24653　※青色がKimi K3

Kimi K3すごい

・総パラメータ数は **2800B**、1トークン当たりの活性化は **104B**
・これまでの公開パラメータモデルよりかなりでかい。



[1] OpenAI, gpt-oss (2025)　[2] Qwen Team, Qwen3.5 (2026)　[3] DeepSeek, V4 Preview (2026)　[4] Moonshot AI, Kimi K3 (2026)／ロゴ：netroom.ai

KimiK3から何がわかるか？

KimiK3が出た今こそ、フロンティアレベルのモデルの中身がどうなっているか知ることができる。
→ 意外とこれまでのLLMの発展上にあるアーキテクチャで特殊な工夫はない。

あじぇんだ

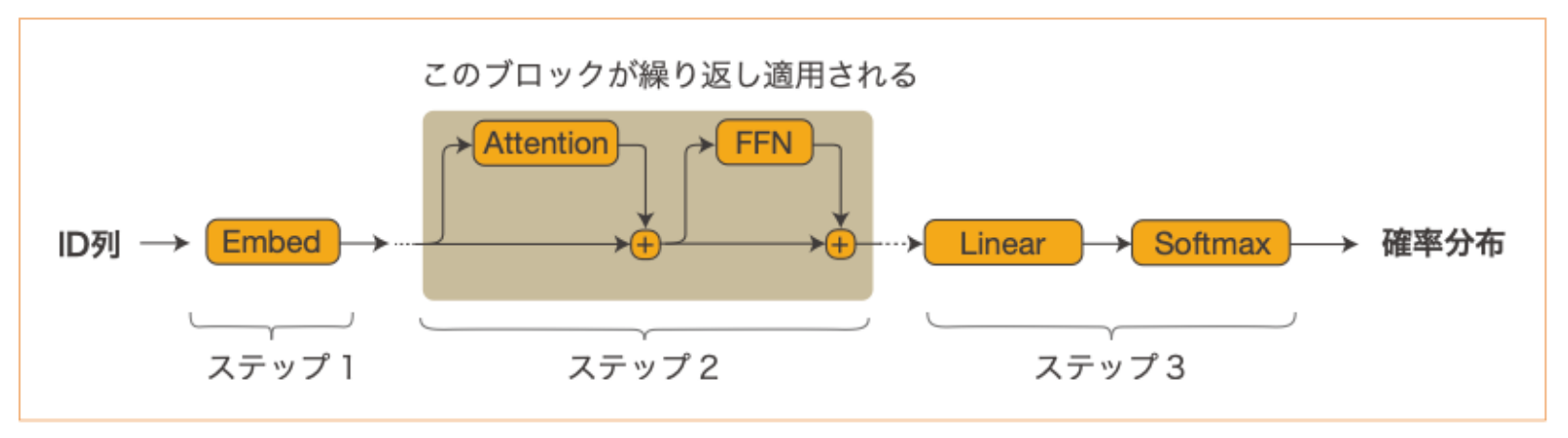
1. LLMの基礎になっているTransformer・attention機構
2. attention機構の高速化、Linear Attention
3. DeltaNetとMamba
4. Kimiに使われているKimiDeltaAttentionへ

Kimiの中核である**KimiDeltaAttention**を理解することを目標に主にLLMの主要な部分の一つの**Attention**に絞って解説します。

Transformerとは？

Attention Is All You Need (Googleなど)で提案された今のLLMのベースとなっているモデル
Transformer(デコーダー)は以下のような構成になっている。

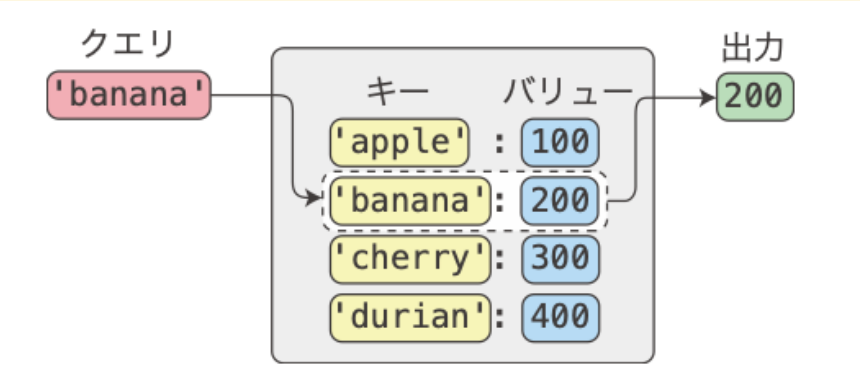
1. 埋め込みベクトルでトークン(単語より少し小さな分割の単位)をそれぞれ数値ベクトルに変換
2. **Attention**機構でトークン同士の関係性を学習・FFNで更なる知識を学習
3. Linear + softmaxで単語の確率に戻す



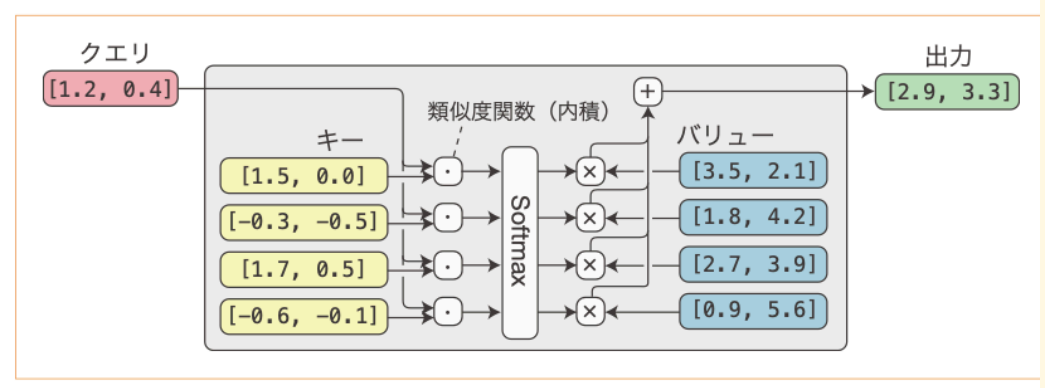
図：『ゼロから作るDeep Learning ❹』

Attentionとは？

あなたはディクショナリーを知っていますか？



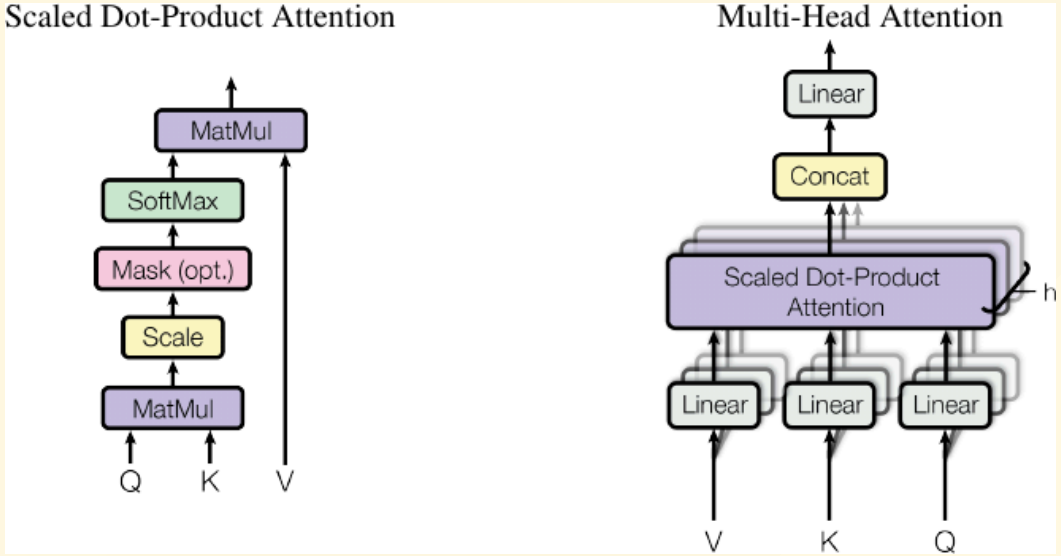
Attentionは答えを一意に決めるのではなく、QueryとKeyの**類似度**でValueを選ぶ



図：『ゼロから作るDeep Learning ❹』

Attentionとは？

・QueryとKeyの類似度から、Valueをどの割合で読むか決める
・Multi-Head Attentionでは、異なる関係を複数のHeadで捉える

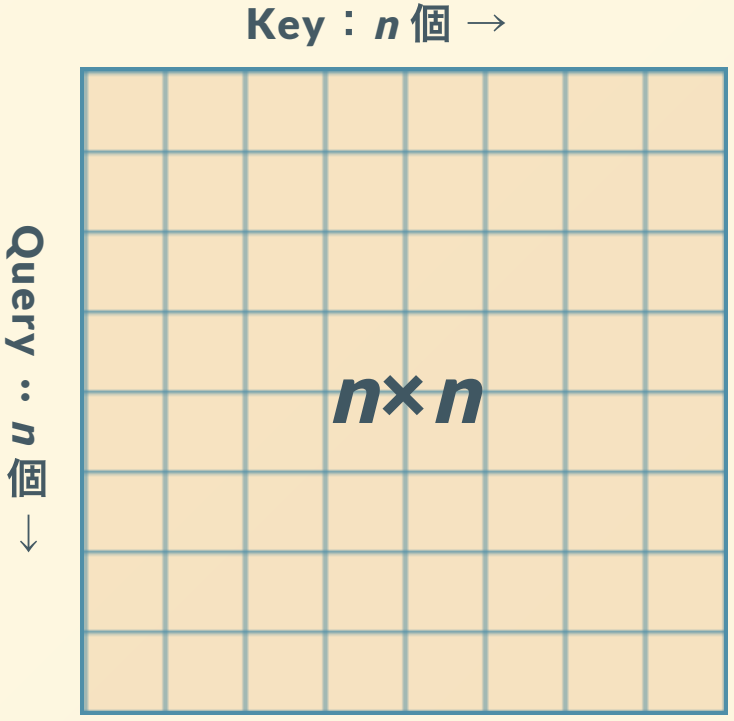


$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

・$QK^\top$：QueryとKeyの全組み合わせの類似度を計算
・softmax：類似度を「どのValueを読むか」の割合へ変換
・最後に V を掛け、Valueの加重和を出力

図・式：Vaswani et al., “Attention Is All You Need,” Figure 2 / Eq. (1) (2017), https://arxiv.org/abs/1706.03762

Attentionは長文になると急激に重くなる



全Queryと全Keyの組み合わせを計算

各Queryは、系列中のすべてのKeyと比較する

$$A = QK^\top \in \mathbb{R}^{n \times n}$$

- Queryが n 個 × Keyが n 個
- Attention scoreは n^2 **個**
- 計算量と学習時メモリは $O(n^2)$

例：4,096 tokens → 約1,678万個のscore／Head／層

→ 計算量を O(n) にできないか？

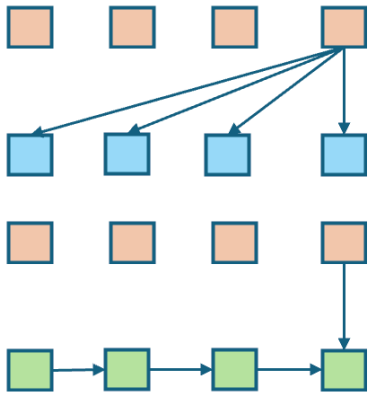
式：Vaswani et al., “Attention Is All You Need,” Eq. (1) (2017), https://arxiv.org/abs/1706.03762　計算量：Katharopoulos et al., “Transformers are RNNs” (ICML 2020), https://proceedings.mlr.press/v119/katharopoulos20a.html

Linear Attention

・softmaxをFeature Map ϕ に置き換える
・積の順序を変更し、過去のKey–Valueを状態 S に要約する(累積和のような発想)

$$S_t = S_{t-1} + \phi(k_t)v_t^\top, \qquad o_t = \phi(q_t)^\top S_t$$

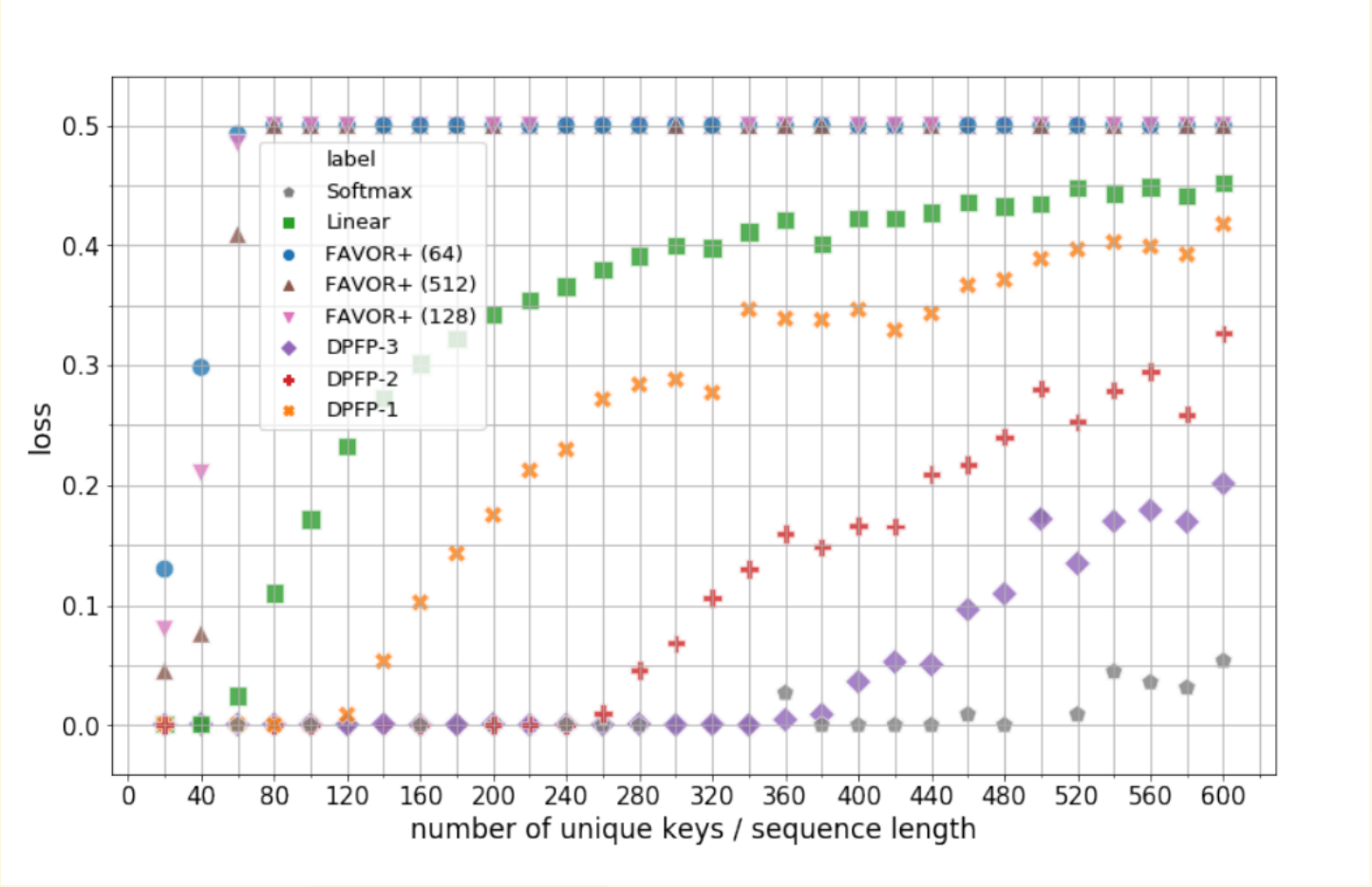
・計算量がattentionの $O(n^2)$ から $O(n)$ に減らせる



通常のattentionは過去の全ての参照が必要

Linear attentionでは累積されている行列だけ参照すればいい！

実験：記憶容量を超えると性能が崩れる



横軸：覚えるKey–Value数　縦軸：検索loss（低いほどよい）

Associative Retrieval

Keyを渡したとき、対応するValueを正しく返せるかを測る実験

- 状態・Keyの次元は **64**
- 緑の■がLinear Attention
- 灰色がSoftmax Attention

> Linear Attentionは、記憶数が**約60個**を超えるとlossが急増

固定サイズの状態へ足し続けるため、容量を超えるとコンフリクトする

図：Schlag, Irie & Schmidhuber, “Linear Transformers Are Secretly Fast Weight Programmers,” Figure 2 / Sec. 4.1, 6.1.1 (ICML 2021), https://proceedings.mlr.press/v139/schlag21a.html

Delta Net (予測誤差分だけ修正する機構をつける)

Linear Attention

新しいKey–Valueを、そのまま状態へ足す

$$S_t = S_{t-1} + k_t v_t^\top$$

$$o_t = q_t^\top S_t$$

→

Delta Net

まず、同じKeyに現在紐づく値を読む

$$S_t = S_{t-1} + \beta_t k_t (v_t - \bar{v}_t)^\top$$

$$\bar{v}_t = S_{t-1}^\top k_t$$

変わったのは書き込むValue：

$$\boxed{v_t \quad \Longrightarrow \quad \beta_t (v_t - \bar{v}_t)}$$

加算する時の干渉をできるだけ防ぐために
単純な加算ではなく、現在の記憶に対する**予測誤差だけを訂正**する

[1] Katharopoulos et al., “Transformers are RNNs,” Eq. (9)–(11) (ICML 2020), https://proceedings.mlr.press/v119/katharopoulos20a.html　[2] Schlag, Irie & Schmidhuber, “Linear Transformers Are Secretly Fast Weight Programmers,” Eq. (20)–(25) (ICML 2021), https://proceedings.mlr.press/v139/schlag21a.html　※前ページの状態行列 S の向きに合わせて転置

Mamba2 (過去に減衰をかける)

Linear Attention

過去の状態を残し、新しい情報を足し続ける

$$S_t = S_{t-1} + \phi(k_t)v_t^\top$$

$$o_t = \phi(q_t)^\top S_t$$

→

Mamba2（SSD）

過去を減衰させてから、新しい情報を書く

$$S_t = \alpha_t S_{t-1} + \beta_t k_t v_t^\top$$

$$o_t = q_t^\top S_t$$

変わったのは **保持** と **書き込み**：

$$\boxed{S_{t-1} \Longrightarrow \alpha_t S_{t-1}} \qquad \boxed{\phi(k_t)v_t^\top \Longrightarrow \beta_t k_t v_t^\top}$$

・α_t：入力依存の忘却率。$\alpha_t \approx 1$なら保持、$\alpha_t \approx 0$なら忘れる
・β_t：書き込み強度。k_tをアドレス、v_tを内容として状態へ追加

[1] Dao & Gu, “Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality,” Eq. (2), Sec. 2.4 (ICML 2024), https://proceedings.mlr.press/v235/dao24a.html　※原論文の A_t H_(t−1) + B_t X_t^T, Y_t = C_t^T H_t を前ページの記号へ対応。α_t=A_t, k_t=B_t, v_t=X_t, q_t=C_t とし、β_tは書き込み係数へ分離して表記。

Kimi Delta Attention = 過去の減衰 + 予測誤差の訂正

① Mamba2：過去を減衰

$$S_t = \alpha_t S_{t-1} + \beta_t k_t v_t^\top$$

前ページの式。α_tを掛け、過去の状態を弱めてから書き込む

+

② DeltaNet：誤差を訂正

$$S_t = S_{t-1} + \beta_t k_t (v_t - \bar{v}_t)^\top$$

$$\bar{v}_t = S_{t-1}^\top k_t$$

前々ページの式。現在の予測と正しいValueとの差だけを書き込む

③ KDA：減衰させた後に、予測誤差を訂正

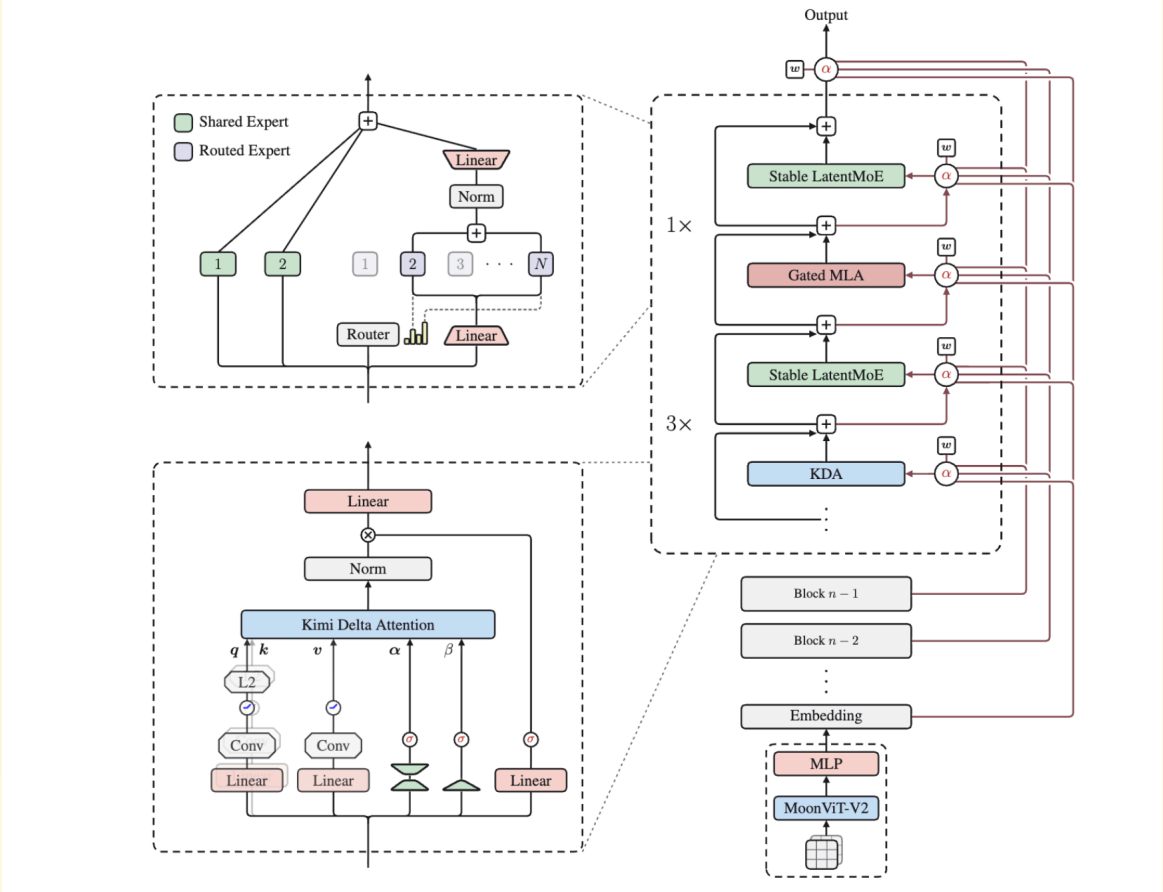
$$\begin{aligned}
\widetilde{S}_{t-1} &= \mathrm{Diag}(\boldsymbol{\alpha}_t) S_{t-1} && \text{過去を減衰} \\
\bar{v}_t &= \widetilde{S}_{t-1}^\top k_t && \text{現在のValueを予測} \\
S_t &= \widetilde{S}_{t-1} + \beta_t k_t (v_t - \bar{v}_t)^\top && \text{予測誤差を訂正}
\end{aligned}$$

$$= (I - \beta_t k_t k_t^\top)\mathrm{Diag}(\boldsymbol{\alpha}_t) S_{t-1} + \beta_t k_t v_t^\top, \qquad o_t = S_t^\top q_t$$

[1] Kimi Team, “Kimi Linear: An Expressive, Efficient Attention Architecture,” Eq. (1), Sec. 2.2–3 (2025), https://arxiv.org/abs/2510.26692

KimiK3

- Kimi Delta AttentionでやっとKimiのAttentionまで辿り着いた。
- 今日はattentionに絞って説明したがまだKimiの一部分
- FeedFowardNetworkに対応する部分など他の部分にも大量の改善が積み重ねられている



まとめ

・最近のLLMの性能向上はすざまじく、LLM開発にまるで革命が起こったかのように感じられる。
→ 実は革新的な技術が多く生まれたわけではなく、細かい性能改善の積み重ねで今の性能を達成している。

少なくともアーキテクチャ面ではまだ2017年に生まれたTransformer・attentionをベースとしている

・Kaggle・研究・AHCでも**天才的な発想**をすることより、ちょっとした違和感を見つけて少しずつ改善していくのが大事かもしれない？