# Associative Recurrent Memory Transformer

Rodkin et al.,

ICML 2024 Next Generation of Sequence Modeling Architectures Workshop

November 2, 2025

**Presenter:** Haeyoung Lee

Seoul National University

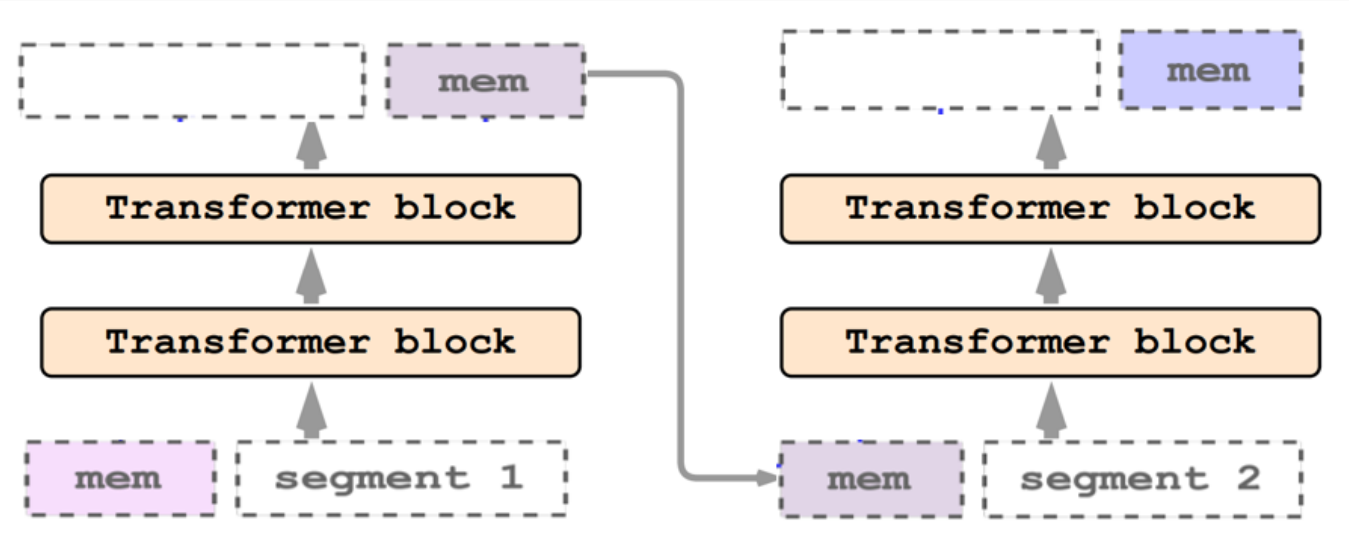
# Motivation

- Objective: process **extremely long** sequences.
- Standard Transformers:
  - Self-attention has $O(n^2)$ time and memory in sequence length.
  - Fixed context window (max sequence length) $\Rightarrow$ information beyond the window is truncated.
- Recurrent Memory Transformer (RMT) was proposed to **lift the context-length limitation** by introducing segment-level recurrence.
- However, RMT still suffers from (i) **limited capacity** because the memory state is restricted to few dense tokens, and (ii) **training difficulties** due to backpropagation through long recurrent chains.
- **Associative Recurrent Memory Transformer (ARMT)** mitigates these issues by introducing a **layer-wise associative (key-value) memory** with *separate projections*, so that segments are connected through an association matrix rather than only through dense memory tokens.

# Notation

- Input sequence is divided into $S$ segments $X_1, X_2, \ldots, X_S$.
- $l \in \{1, \ldots, L\}$ : Transformer layer index.
- $X_s^l$: token embeddings of segment $s$ entering layer $l$.
- $M_s^{l+1}$: recurrent **memory tokens** after processing segment $s$ at layer $l$ (RMT memory).
- $A_s^l$: **associative memory matrix** for layer $l$ after segment $s$.
- $z_s^l$ : corresponding **normalization** vector; updated as $z_s^l = z_{s-1}^l + \sum_i \gamma_i \, \phi(k_i)$.
- $\phi(\cdot)$: non-linear feature map used in the associative mechanism to obtain kernelized representations (the paper uses DPFP-3).
- $k_i = W_K m_i$ : **key** vector obtained from a memory token $m_i$.
- $v_i = W_V m_i$ : **value** vector paired with $k_i$, obtained from the same memory token.
- $q_j = W_Q x_j$ : **query** vector computed from a current token $x_j$.
- $y_j = \dfrac{A_s^l \, \phi(q_j)}{(z_s^l)^\top \phi(q_j)}$ : **retrieved vector** from the associative memory for query $q_j$.
- $\gamma_i = 1 - \dfrac{(z_{s-1})^\top \phi(k_i)}{\|\phi(k_i)\|^2}$ : **normalization correction term** used when updating $z_s^l$.
- $W_Q, W_K, W_V, W_\beta$ : learnable memory-specific linear projections in associative block.

# Recurrent Memory Transformer (RMT)

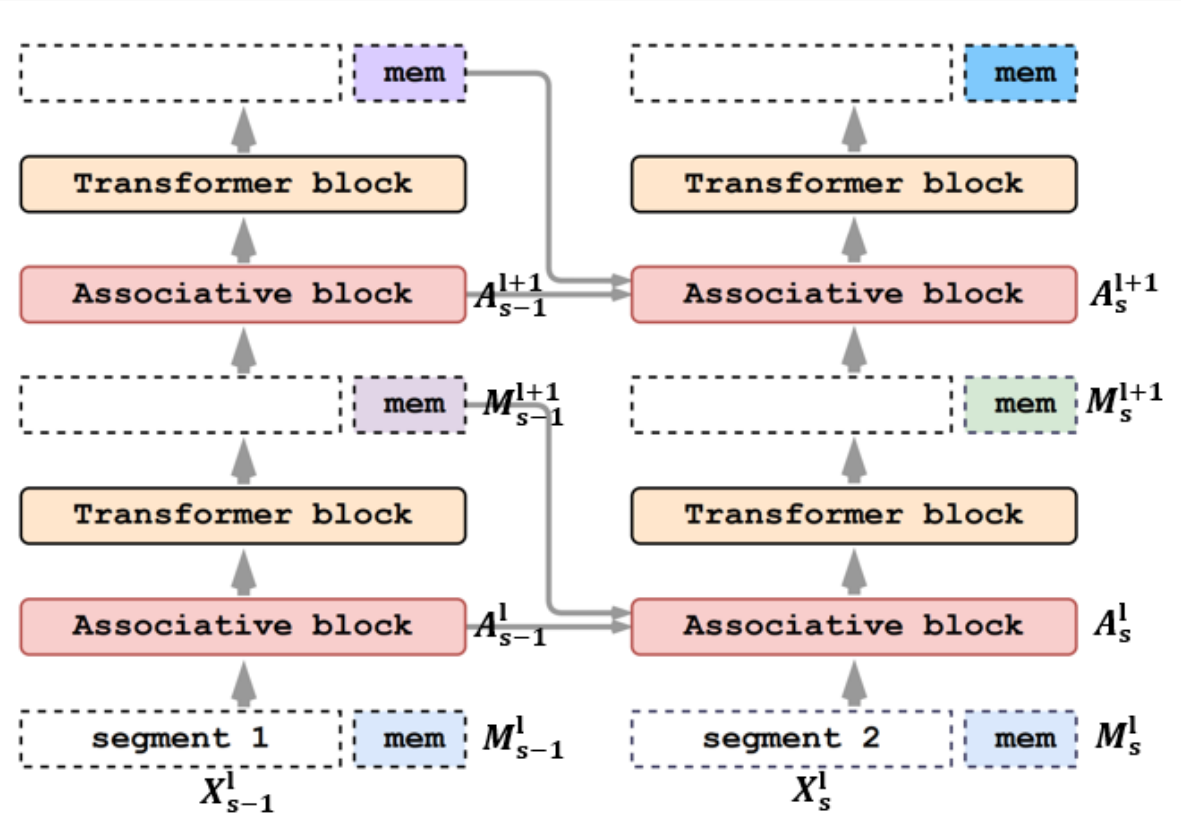



- RMT processes a long sequence **one segment at a time**. After segment $s-1$ is processed, the model produces a small set of **memory tokens** that summarize it.
- When the next segment $s$ is processed, these memory tokens from segment $s-1$ are **provided together with** the tokens of segment $s$ as the input to the layer.
- This procedure is repeated for every segment, so information can flow forward across the whole sequence.

# Recurrent Memory Transformer (RMT)

- Nonetheless, the memory that is passed across segments is always a **small, dense token set**, which means:
  - the actual **capacity** is bounded by the number of memory tokens.
  - access to past information is **implicit** (only through attention to those tokens).
  - long recurrent chains still require **backpropagation through many steps**.
  - and tasks requiring **precise associative retrieval** or **explicit rewriting** are not naturally supported.
- Associative Recurrent Memory Transformer (ARMT) keeps this segment-wise recurrence with a **layer-wise associative** memory.

# Associative Recurrent Memory Transformer (ARMT)

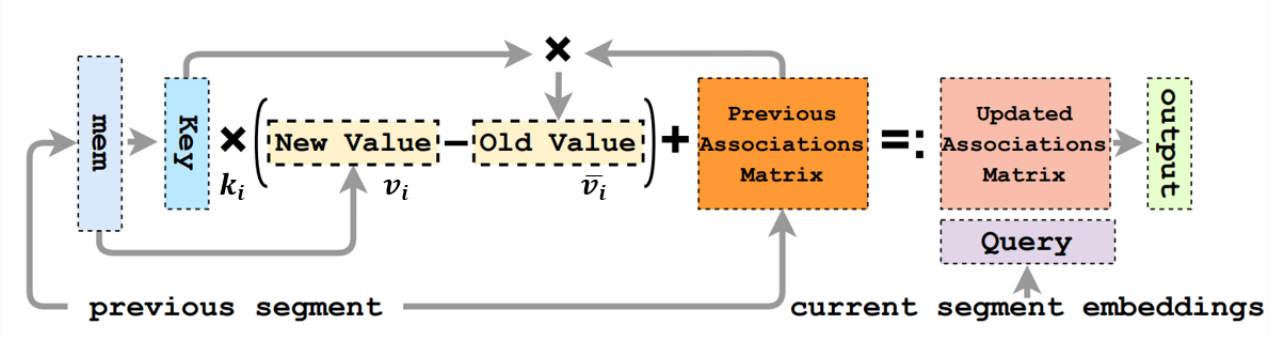



- ARMT inserts an **layer-wise associative block** *before* Transformer block.

$$[X_s^{l+1}; M_s^{l+1}] = \text{TrBlock}\Big(\text{AssocBlock}([X_s^l; M_s^l], A_s^l)\Big)$$

$$A_s^l = \text{MemUpdate}(A_{s-1}^l; M_{s-1}^{l+1})$$

# Associative Block




- **Associative Memory Update**

$$A_s^l = \text{MemUpdate}\big(A_{s-1}^l;\ M_{s-1}^{l+1}\big)$$

$$k_i = W_K m_i, \quad v_i = W_V m_i, \quad \beta_i = \sigma(W_\beta m_i), \quad \bar{v}_i = \frac{A_{s-1}^l \phi(k_i)}{(z_{s-1}^l)^\top \phi(k_i)}$$

$$A_s^l = A_{s-1}^l + \sum_i \beta_i \big(v_i - \bar{v}_i\big) \otimes \phi(k_i)$$

  - Project memory token to key/value, and recall the value already stored for the same key.
  - Overwrite: remove the old value and write the new one, scaled by $\beta_i$.

# Summary

- **RMT** removes the bottleneck for long sequences by processing segments recurrently and forwarding a small memory state.
- However, this token-based memory is **capacity-limited** and provides only **implicit** (attention-based) access, so precise associative operations and repeated rewrites are difficult.
- **ARMT** keeps the RMT-style segment recurrence but adds a **layer-wise associative memory** with separate linear projections for memory.
- This associative memory can
  - store information as key–value pairs,
  - read it by content in constant time per segment,
  - and overwrite an existing key explicitly.
- Thus ARMT preserves the efficiency of RMT while enabling **explicit, key-based access** to long context information.

# Thank you!