# Memory Transition Model Design

Based on the Memory Transition Model, we can train the model itself to learn how to control and update memory. Given the design of SSM, we can easily see that SSM tends to cause a complete change in states, but we do not want this state transition to occur in TMNN, because the memory itself needs to be retained. That is why we designed this model to preserve the tensor's state, with everything stored in the tensor. In practice, however, this is a temporary memory in an LLM and can easily be saved as a tensor-based file using PyTorch.

The Event Vector is designed for memory because every memory is based on events. The memory block is a 2D tensor composed of event vectors, which are derived from the LLM's latent tokens. During the research process, we found something interesting: if an MLP is used as a vectorizer to transform the word vector matrix into an event vector, the activation function evidently affects the generation quality.

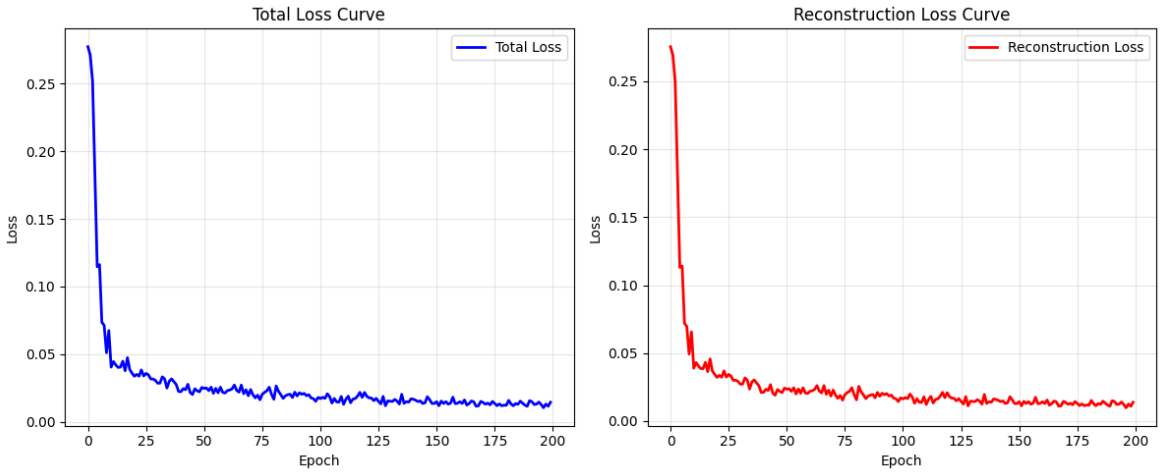
We tested some classic activation functions in the MLP-based vectorizer and found that Swish-Gated Linear Unit (SwiGLU) performs better than several others. We did not test all activation functions in this setting, but similar to other MLP experiments, it converges more rapidly than others.

Additionally, based on the principle of the attention mechanism, we use cosine similarity to search for the most similar event vectors. The cosine similarity matrix is passed through a softmax function and used for selection. If the memory block is very large, we can also draw inspiration from DeepSeek-V3.2 with its Lightning Indexer and Sparse Attention to select the top-k vectors for comparison. The top-k vectors are selected from the most frequently accessed vectors using the Access Frequency Logarithm Matrix. This matrix is computed as the negative logarithm of access frequency and normalized via softmax. It is also used in write control.

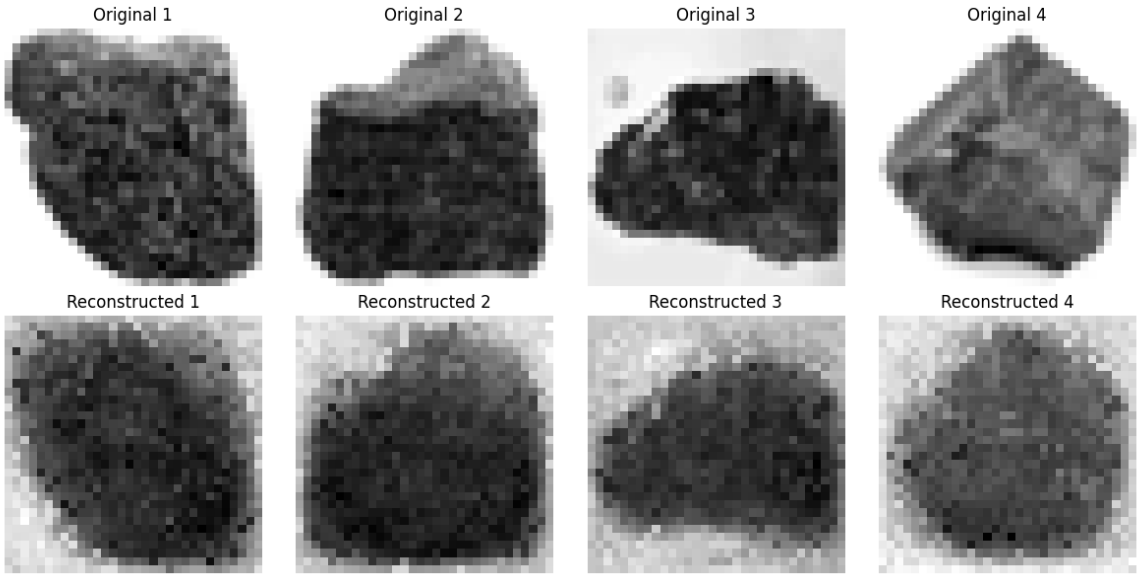
Write control is based on the Access Frequency Logarithm Matrix, where writes replace the vector with the lowest access frequency. We designed a write control equation to transform the memory states. This process can be used in stream-style computation.

Given the above, we have a memory read/write model. However, it is clear that we must use a specialized vectorizer to address the problem of data distortion during vector mapping. Therefore, we consider using an Invertible Neural Network (INN), which is an ideal choice for transforming raw data into vectors.

The INN produces a compressed output and an auxiliary output. Both can be saved to construct lossless data. However, preserving the data entirely intact compromises performance and storage efficiency. Thus, we designed a network to predict the other part of the INN's output. This process is somewhat similar to noise prediction in diffusion models, though diffusion models now directly predict and generate the content itself. In our memory compression process, this is unnecessary. Instead, during memory retrieval, we reconstruct the original data by predicting the auxiliary output from the compressed content. The loss can be controlled by training this prediction network, and with a large dataset, the prediction should achieve very high accuracy.




Scenario at 4:1 Compression Ratio: Comparison Based on 200 Rounds of Simple Training Results.




Memory Writing Equation:

$$AccessFrequencyFogarithmMatrix: \beta_i = \log\left(-\frac{counts}{Total}\right)$$

$$\beta_{1\times l} = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_l \end{bmatrix}$$

$$UpdateIndex: i = \arg\max(\beta)$$

$$MemoryOverWriteControlMatrix: C_{l\times l} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}, C_{(i,i)} = 0$$

$$WriteBroadcastControlMatrix: L_{l\times 1} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, L_i = 1$$

$$MemoryStateUpdateEquation: L_{l\times 1} \cdot V_{1\times d} + M_{l\times d}(x) \cdot C_{l\times l} = M_{l\times d}(x+1)$$

Memory Reading Equation:

$$MagnitudeReciprocalMatrix : A_{l\times 1} = \begin{bmatrix} \frac{1}{a_0 b_0} \\ \frac{1}{a_0 b_1} \\ \vdots \\ \frac{1}{a_0 b_l} \end{bmatrix}$$

$$CosineSimilarityMatrix : \alpha_{1\times l} = \text{softmax}\left(V_{1\times d} \cdot M_{d\times l} \circ A_{1\times l}\right)$$

$$ReadIndex : i = \arg\max(\alpha_{1\times l})$$