Obsidian Memory Transformer: A Titan-Inspired Architecture for Large Language Models

Sahibzada Allahyar
Singularity Research and University of Cambridge

February 23, 2025

Abstract

This paper presents the *Obsidian Memory Transformer*, a novel Large Language Model (LLM) architecture that introduces a dedicated long-term memory (LTM) mechanism inspired by the Google Titan design for handling extensive context lengths. We merge a *latent memory bank* approach with a standard Transformer pipeline, and we apply *FlashAttention* to improve efficiency in both training and inference. Our implementation is carried out in low-level C++/CUDA, employing libraries such as cuBLAS, CUTLASS, and NVRTC to harness modern NVIDIA GPUs. We discuss the mathematical formulation behind our LTM module, the architectural layout of our Transformer blocks, and the core low-level optimizations that enable large-scale training and inference.

1 Introduction

Recent advancements in Large Language Models (LLMs) have revealed remarkable performance in natural language processing (NLP) tasks. However, most popular Transformer-based architectures suffer from prohibitive memory and computational costs when confronted with very large context windows. This limitation restricts the model's ability to maintain a coherent understanding of extended text passages.

Inspired by the *Google Titan* architecture [3], we propose the *Obsidian Memory Transformer*, which incorporates a novel *Long-Term Memory (LTM)* component for handling extensive context lengths. This auxiliary memory bank stores compressed representations from past segments, enabling longer-horizon context capture without exploding self-attention complexity.

Along with this architectural contribution, we incorporate *FlashAttention* [2] in our multi-head attention (MHA) kernels to reduce memory overhead and boost computation speed. We also provide a low-level implementation in C++/CUDA, leveraging libraries such as cuBLAS, NCCL, and CUTLASS, alongside warp-level optimizations and fused kernels for maximum throughput.

1.1 Contributions

- **Obsidian Memory Transformer.** We introduce an architecture that features a memory bank holding compressed representations of past tokens, inspired by Titan-based gating mechanisms. This *Obsidian Memory* mechanism mitigates the need for extremely large explicit context windows.
- **FlashAttention Integration.** We implement the Transformer's self-attention layer with a memory-friendly approach, enabling more efficient training and inference.

- **Low-Level Optimization.** We detail our C++/CUDA kernels that perform fused matrix multiply-accumulate (MMA), layer normalization, and memory attention, exploiting modern GPU Tensor Cores.

2 Related Work

Transformers. The Transformer [1] has become the de facto standard for NLP tasks. A conventional Transformer block applies scaled dot-product self-attention followed by a position-wise feed-forward network (FFN), each wrapped with residual connections and layer normalization.

Memory Mechanisms. Several approaches attempt to extend Transformers to long sequences. Methods like Longformer or BigBird use sparse attention to handle longer contexts. The *Google Titan* architecture [3] introduces additional gating and memory bridging modules that effectively store a compressed state to address large context problems.

FlashAttention. Tri Dao et al. [2] introduced FlashAttention, a memory- and speed-optimized attention kernel that computes partial softmax in small tiles. This reduces the memory footprint of conventional self-attention, making it suitable for large models or scenarios where GPU memory is a bottleneck.

3 Proposed Architecture: Obsidian Memory Transformer

3.1 Transformer Backbone

Our baseline Transformer layers are consistent with [1]:

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

where Q, K, V are queries, keys, and values. Each Transformer block also includes a position-wise feed-forward network (FFN):

$$\text{FFN}(\mathbf{x}) = \max(0,\, \mathbf{x}W_1 + b_1)W_2 + b_2.$$

We add layer normalization and residual connections around each sub-layer.

3.2 Long-Term Memory (LTM) Module

A key innovation in the Obsidian Memory Transformer is the *Long-Term Memory (LTM)* layer, inspired by gating mechanisms in the Titan design [3]. Instead of applying attention to the raw hidden states of past tokens (which scales poorly), we maintain a set of m *memory slots* $\mathbf{M} \in \mathbb{R}^{m \times d_{\text{model}}}$. Each slot holds a compressed representation of previously processed segments.

Compression Gate. At each segment (or context block), we compute a compressed vector $\mathbf{c}_{\text{segment}} \in \mathbb{R}^{d_{\text{model}}}$:

$$\mathbf{c}_{\text{segment}} = \text{Pool}\Big(\sigma(W_{\text{c}} \cdot \mathbf{H} + b_{\text{c}}) \odot \mathbf{H}\Big),$$

where $\mathbf{H}$ is the collection of hidden states for the segment, and $\text{Pool}(\cdot)$ is typically an average or sum pooling operation.

Memory Update. We then select or learn which slot $\mathbf{M}_i$ to update:

$$\mathbf{M}_i \leftarrow \alpha \mathbf{M}_i + (1 - \alpha)\, \mathbf{c}_{\text{segment}},$$

where $\alpha \in [0, 1]$ is a decay factor.

Memory Attention. Given a current batch of queries $\mathbf{Q}$, we attend over the memory bank's keys/values $(\mathbf{K}_M, \mathbf{V}_M)$:

$$\mathbf{O}_M = \text{softmax}\left(\frac{\mathbf{Q}\,\mathbf{K}_M^\top}{\sqrt{d_k}}\right)\mathbf{V}_M.$$

The LTM output $\mathbf{O}_M$ can be fused with standard self-attention output or used as a separate attention head in a multi-head scheme:

$$\mathbf{X}_{\text{out}} = \text{LayerNorm}\big(\mathbf{H} + \mathbf{O}_M + \text{FeedForward}(\mathbf{H})\big).$$

By preserving a compressed representation of older segments, the model effectively has "long-term memory" without directly expanding the self-attention window to thousands of tokens.

3.3 FlashAttention Integration

We apply the FlashAttention approach to our multi-head self-attention and memory attention computations. In standard attention, one might form and store $\mathbf{QK}^\top$ for all tokens, which is $O(n^2)$ in memory. FlashAttention avoids storing the entire matrix by computing partial sums on the fly. This approach is critical for scaling up context lengths without incurring massive memory overheads.

4 Low-Level Implementation

4.1 Core Technologies

Our implementation is primarily in C++/CUDA with:

- **CUDA C/C++:** Kernel writing for GPU-based parallel computation.
- **cuBLAS / cuBLAS Lt:** For high-performance matrix multiplication.
- **CUTLASS:** Provides warp-level primitives and Tensor Core MMA support.
- **NCCL & MPI:** Distributed parallelism for multi-GPU / multi-node training.
- **FlashAttention:** Implemented through custom or third-party fused attention kernels.

4.2 Kernel Highlights

Fused MHA. We fuse the attention softmax, scaling, and GEMM steps into a single kernel to minimize memory reads/writes:

$$\mathbf{O} = \text{FlashAttn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}),$$

using tile-based chunking to compute partial softmax in registers.

Long-Term Memory Attention. The memory attention kernel is also fused:

$$\mathbf{O}_M = \text{Attn}(\mathbf{Q}, \mathbf{K}_M, \mathbf{V}_M).$$

Since $m \ll n$ (the full sequence length), this kernel can be more efficient and requires fewer resources.

4.3 Quantization and Mixed Precision

We enable half-precision (`FP16`) and bfloat16 (`BF16`) for major matrix operations. For inference, optional post-training INT8 or INT4 quantization is provided, leveraging calibration steps or built-in CUDA functionality where possible.

4.4 Distributed Training

For large-scale training:

- **Data Parallelism:** Each GPU processes a subset of the batch, synchronized with all-reduce for gradients via `NCCL`.
- **Tensor Parallelism:** Split attention heads or FFN dimensions across GPUs to handle large models that do not fit on a single device.
- **Pipeline Parallelism:** Assign successive Transformer layers to different GPUs for balanced pipeline stages.

5 Experimental Observations

In preliminary experiments on language modeling tasks, we observe:

- **Improved Handling of Long Contexts:** The LTM module enables more coherent outputs for contexts exceeding thousands of tokens.
- **Competitive Throughput:** FlashAttention and fused kernels reduce memory consumption and speed up training relative to baseline attention.
- **Scalability:** Our design scales across multiple GPUs using MPI/NCCL, enabling the training of multi-billion parameter models.

6 Conclusion

We presented the *Obsidian Memory Transformer*, a Titan-inspired architecture for LLMs that incorporates a novel *long-term memory* (LTM) mechanism into the Transformer. Our approach selectively compresses and stores past segment representations in a memory bank, effectively extending context range without a quadratic cost in standard self-attention. Leveraging *FlashAttention* and a *C++/CUDA* core that harnesses GPU Tensor Cores, we achieve both scalability and efficiency in training and inference.

Future work includes investigating more sophisticated gating or retrieval schemes for the memory bank (e.g., dynamic segment hashing) and exploring advanced quantization methods for further speedups and memory savings.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. *Attention is all you need*. In *Advances in Neural Information Processing Systems*, 2017.

[2] T. Dao, D. Fu, S. K. Sabah, A. Rudra, C. Ré. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. arXiv preprint arXiv:2205.14135, 2022.

[3] *Behrouz, A., Zhong, P., & Mirrokni, V. (2024). Titans: Learning to Memorize at Test Time. arXiv. https://doi.org/10.48550/arXiv.2501.00663*