

# HPC Methodology: The FTT Bitwise Optimization

An Accelerator Framework for Mechanistic Interpretability

Timothy James Couch, Gemini

December 5, 2025

## I. Problem Statement: The Memory Wall Calculation

The Pattern Persistence Project requires analyzing activation traces from large language models (LLMs) in the 70B parameter class. Standard processing requires storing activations as float32, exceeding the available RAM on commodity hardware (e.g., Mac Studio 96GB), thereby halting the mechanistic search for the "Flinch".

### A. The OOM Bottleneck

To analyze a 70B model with 8192 hidden dimension, a standard dataset requires:

$$10,000,000 \text{ vectors} \times 8,192 \text{ dim/vector} \times 4 \text{ bytes/float32} \approx 327.68 \text{ GB}$$

This exceeds the 96GB system RAM constraint.

### B. Solution Target

The **\*\*FTT (Fast Tensor Transform)\*\*** system resolves this by compressing the data by 4x (float32  $\rightarrow$  int8), reducing the trace size to approximately 81.9 GB. This makes the data manageable via mmap streaming.

## II. FTT Architecture: Streaming and Interoperability

The FTT system fuses a Python host core with a C++/Metal device backend.

### A. The Virtual Tensor and Streaming

The Python layer transforms the compressed disk file into a **Virtual Tensor** using the torch.mmap wrapper. The SAE training loop then loads int8 data chunks directly from the disk into the GPU without staging the entire 327 GB as float32 in system RAM.

## B. C-API and Inter-Substrate Bridge

The C++ core exposes a C-API ('extern "C"') to enable seamless GPU acceleration:

- **Function:** The C functions accept raw host memory pointers (`const float*src, char*dst`).
- **Optimization:** This structure is designed to transition to `newBufferWithBytesNoCopy` to achieve **Zero-Copy** memory transfer between the host and the GPU compute unit.

## III. Metal Kernel Implementation: HPC Optimization

The two-pass kernel design, implemented in Metal, accelerates the reduction and quantization math, which is too slow for CPU execution.

### A. Pass 1: Row Max Reduction (`find_row_max`)

- **Function:** Calculates the  $\max(|x|)$  for the high-dimensional vector, determining the scaling factor.
- **Optimization:** A single CPU thread would take 8192 steps. On the GPU, this task is assigned to a single threadgroup, where  $\sim 256$  threads cooperatively execute the reduction in parallel (logarithmic time).

### B. Pass 2: Symmetric Quantization (`quantize_f32_int8`)

- **Function:** Applies the scaling factor and clamps the data to the int8 range ( $\pm 127$ ).
- **Optimization:** This is an **embarrassingly parallel** problem. The GPU is used to dispatch a thread per element, performing the conversion and rounding simultaneously, which is where the 4x compression is physically executed.