System-Level Optimization for Sparse Autoencoder Training

1.Problem Statement

Sparse Autoencoders (SAEs) and their variants (e.g., Top-K SAE, JumpReLU SAE) are widely used in mechanistic interpretability and feature extraction tasks. However, training these models is computationally expensive due to the heavy use of sparse tensors.

In current implementations, most frameworks (e.g., SAELens) treat these sparse intermediate tensors as dense tensors in both forward and backward passes. As a result:

- Sparse decoding operations waste compute on zero activations.
- Backpropagation still executes dense GEMM operations.
- Large SAEs (e.g., 32k–512k dictionary size) cause memory pressure and slow training.

This leads to high latency, suboptimal hardware utilization, and increased training cost, making SAE training a system bottleneck in current mechanistic interpretability pipelines.

Our project aims to optimize the training pipeline by leveraging sparsity at both algorithm and system levels.

2. Proposed Method

We propose to accelerate SAE training via sparse optimization, targeting both the decoder and the backward pass.

A. Sparse-Aware Decoder Computation

During decoding, each activation vector in an SAE typically contains only k non-zero entries (Top-K or thresholded). We plan to:

- Replace dense matrix multiplication with Sparse-dense matrix multiplication.
- Reduce memory of activations using compressed formats (COO / CSR).

This reduces compute and memory proportional to the num of **activated features** instead of **total features**.

B. Sparse Backward Propagation

Backward computation also wastes time differentiating through zero entries. We will: - Derive a sparse backward formula that only propagates gradients along active features. - Use masked matrix multiplication and sparse-dense matrix multiplication to avoid GEMM.

C. Kernel Fusion

When applying activation functions to dense feature activations, some intermediate results can help construct a sparse feature activation tensor. These two parts can be computed together.

3. Evaluation Plan

We will systematically evaluate the following metrics:

- Latences of each step
- SM occupancy
- Memory bandwidth saturation
- Peak GPU memory usage

Although this is a system-focused project, we will confirm whether sparse acceleration introduces deviation from the original model by comparing MSE.

4. Expected Outcomes

We expect to deliver: - A prototype implementation of sparse-aware SAE training using custom kernels and sparse formats. - A detailed system-performance analysis (latency, memory, hardware utilization). - Demonstration of speedup for large SAEs under realistic training workloads. - Insights into how sparse interpretability models should be implemented efficiently in ML systems.