Domain-Specific Training of Sparse Autoencoders

Het Patel Vijay Vanapalli Dattatreya Kantha

December 4, 2024

Dataset Motivation and Domain Specificity in Sparse Autoencoders

- We aim to investigate whether similar benefits can be achieved through fine-tuning pre-trained Sparse Autoencoders (SAEs) on domain-specific data, potentially with reduced token requirements. This direction appears particularly promising given the existing availability of high-quality SAEs trained on pre-training data. [3] [2]
- Drawing from "SAEs are highly dataset dependent: A case study on the refusal direction," we've observed that domain-specific datasets produce superior reconstructions and sparse representations of meaningful concepts. Similarly, for understanding sparse decompositions in code, these datasets provide a parallel to the "refusal direction" idea but within the domain of programming languages.
- Code snippets, especially from datasets like NeelNanda/code-10k, often include patterns crucial for debugging and decision-making. g sparse reconstructions of such data could uncover interpretable latent features that can guide model behavior, similar to how refusal-aligned latents are utilized in safety applications.

Dataset Selection

LMSYS CHAT 1M:

- **Why did we specifically choose this dataset?**
- This is particularly important given that recent research has shown that **Sparse Autoencoder (SAE)** performance varies significantly across different datasets. For instance, when researchers trained SAEs on `Qwen1.5 0.5B Chat` using the `Pile dataset`, they encountered difficulties in identifying clear, interpretable patterns related to *refusal behavior*. The most relevant latent features they found were too broad and performed poorly in *directional steering tasks*.
- Studies found that training a new SAE on the `LmSys-Chat-1M` dataset produced significantly **sparser**, more **faithful**, and **interpretable** reconstructions of the "refusal direction". The LmSys SAE proved more capable of finding interpretable "refusal" latents for effective model steering. Refer the study: *SAEs are highly dataset dependent: a case study on the refusal direction* [2].
  - **Content**: Contains both instructions and model completions in chat format
  - **Instruction-response format**: `<|im_start|>user {instruction}<|im_end|> <|im_start|>assistant {completion}<|im_end|>`
  - Maintains better representation of *chat-specific patterns* and behaviors
  - Required **fewer latents** for equivalent reconstruction quality (*1 LmSys latent outperformed 32 Pile latents*)
  - Produced more **interpretable** and **task-relevant** feature decompositions
  - Contains sufficient training tokens (`400M`)

TINYPIXEL/TINY-CODES

- **Size**: Contains `1 million` rows of text data
- **Structure**: Data is stored in `Parquet format`, facilitating efficient storage and retrieval
- **Content**: Includes code snippets that incorporate `conditional statements` like `if-else` to handle various scenarios
- **Purpose**: Designed to enhance models' **reasoning capabilities** through exposure to logical code structures
- In summary, "TinyPixel/tiny-codes" offers a larger dataset focused on logical code structures to improve reasoning, while "NeelNanda/code-10k" provides a smaller, curated set of code snippets intended for debugging and analyzing model performance in coding contexts.

NEELNANDA/CODE-10K

- **Size**: Comprises `10,000` code snippets
- **Structure**: Also stored in `Parquet format` for efficient data handling
- **Content**: Contains a curated selection of code snippets aimed at assisting in **debugging** models trained on code data
- **Purpose**: Serves as a resource for **debugging** and **understanding model behavior** in coding contexts
- While `TinyPixel/tiny-codes` originally provides a much larger dataset, subsetting it to `10,000` rows aligns its scale with `NeelNanda/code-10k`, allowing for a fairer comparison during SAE training.

Sparse Autoencoders Architecture and Implementation

GENERAL ARCHITECTURE (LMSYS CHAT 1M)

- **Base Architecture**:
  - Model: `Standard SAE with Qwen 1.5 0.5B Chat`
  - Hook Point: `blocks.0.hook_mlp_out` at MLP output layer 0
  - Input Dimension: `d_in = 1024`
  - Expansion Factor: `32x` (hidden layer size: `32,768`)
- **Dataset Configuration**:
  - Dataset: `ckkissane/lmsys-chat-1m-qwen1.5-0.5b-chat`
  - Streaming: **Enabled**
  - Context Size: `2048 tokens`
  - Total Training Tokens: `409.6M` (`100,000` steps × `4,096` batch size)
- **Training Parameters**:
  - Learning Rate: `5e-5` with constant scheduler
  - Warm-up Steps: `0`
  - Decay Steps: `20,000` (`20%` of total steps)
  - **Sparsity Control**:
    - L1 Coefficient: `5`
    - L1 Warm-up Steps: `5,000` (`5%` of total steps)
    - LP Norm: `1.0`

- **Optimization Features**:
  - Batch Size: `4,096` tokens
  - Buffer Size: `64` batches
  - **Performance Optimizations**:
    * Autocast: **Enabled** for LM and general operations
    * CUDA memory allocation optimization
    * Activation normalization: `expected_average_only_in`
- **Initialization Strategy**:
  - Decoder Initialization: `zeros` method
  - Decoder Heuristic Init: **Enabled**
  - Encoder Initialization: As decoder transpose
- **Monitoring & Checkpointing**:
  - Checkpoints: `5` saves during training
  - Feature Monitoring:
    * Sampling Window: `1,000` steps
    * Dead Feature Threshold: `1e-4`

```
wandb: Run history:
wandb:      details/current_l1_coefficient
wandb:       details/current_learning_rate
wandb:           details/n_training_tokens
wandb:                       losses/l1_loss
wandb:                      losses/mse_loss
wandb:                   losses/overall_loss
wandb:                    losses/raw_l1_loss
wandb:          metrics/explained_variance
wandb:      metrics/explained_variance_std
wandb:                             metrics/l0
wandb: metrics/mean_log10_feature_sparsity
wandb:                   sparsity/below_1e-5
wandb:                   sparsity/below_1e-6
wandb:                sparsity/dead_features
wandb:    sparsity/mean_passes_since_fired
wandb:
```

```
wandb: Run summary:
wandb:      details/current_l1_coefficient 5
wandb:       details/current_learning_rate 0.0
wandb:           details/n_training_tokens 409559040
wandb:                       losses/l1_loss 30.8227
wandb:                      losses/mse_loss 72.00415
wandb:                   losses/overall_loss 226.11765
wandb:                    losses/raw_l1_loss 154.11349
wandb:          metrics/explained_variance 0.94461
wandb:      metrics/explained_variance_std 0.08584
wandb:                             metrics/l0 21.39819
wandb: metrics/mean_log10_feature_sparsity -4.00742
wandb:                   sparsity/below_1e-5 83
wandb:                   sparsity/below_1e-6 0
wandb:                sparsity/dead_features 0
wandb:    sparsity/mean_passes_since_fired 3.90259
wandb:
wandb: 🚀 View run 32768-L1-5-LR-5e-05-Tokens-4.096e+08 at: https://wandb.ai/dyers-gw/lmsys-chat-1m-qwen1.5-0.5b-chat/runs/fvuuriam
wandb: ⭐ View project at: https://wandb.ai/dyers-gw/lmsys-chat-1m-qwen1.5-0.5b-chat
wandb: Synced 5 W&B file(s), 0 media file(s), 21 artifact file(s) and 0 other file(s)
wandb: Find logs at: ./wandb/run-20241203_013626-fvuuriam/logs
```

General Architecture (TinyCodes and NeelNanda/Code-10k)

- **Model Integration**: Works with pre-trained transformer models like `gpt2-small` or `gelu-2l`
- **Hook Layer**: Operates on activations from a specific layer (`hook_layer=0`) and hook point (`blocks.0.hook_mlp_out`)

Encoder

- **Input Dimensionality**: Matches the model's MLP output (`d_in=768` for GPT-2)
- **Latent Space**: `768×16=12,288`
- **Activation Function**: `ReLU` for sparsity and non-linearity

Decoder

- **Initialization**:
  - Zero-initialized bias (`b_dec_init_method="zeros"`) or heuristic
  - Optionally symmetric to encoder (`init_encoder_as_decoder_transpose=True`)
- **Normalization**: Decoder normalization is optional (`normalize_sae_decoder`)

Loss and Regularization

- **Reconstruction Loss**: Mean Squared Error (`MSE`)
- **Sparsity**: Controlled via `L1` penalty (`l1_coefficient`) and warm-up (`l1_warm_up_steps`)
- **Dead Features**: Pruned below a threshold (`dead_feature_threshold`)

Input and Context

- **Input Source**: Processes activations from sequences of `context_size` of `512`
- **Batching**: Supports large training batches (`train_batch_size_tokens=4096`)

Training Configuration

- **Learning Rate**: Configurable with warm-up and decay schedules
- **Optimization**: `Adam` optimizer with standard parameters
- **Tokens Processed**: Training scales with `training_tokens`

Run history:

details/current_l1_coefficient
details/current_learning_rate
details/n_training_tokens
losses/l1_loss
losses/mse_loss
losses/overall_loss
losses/raw_l1_loss
metrics/explained_variance
metrics/explained_variance_std
metrics/l0
metrics/mean_log10_feature_sparsity
sparsity/below_1e-5
sparsity/below_1e-6
sparsity/dead_features
sparsity/mean_passes_since_fired

Run summary:

details/current_l1_coefficient	5
details/current_learning_rate	0.0
details/n_training_tokens	40919040
losses/l1_loss	27.01276
losses/mse_loss	50.48137
losses/overall_loss	185.54515
losses/raw_l1_loss	135.06378
metrics/explained_variance	0.92124
metrics/explained_variance_std	0.07162
metrics/l0	125.16113
metrics/mean_log10_feature_sparsity	-2.78485
sparsity/below_1e-5	5
sparsity/below_1e-6	4
sparsity/dead_features	1
sparsity/mean_passes_since_fired	0.36971

Evaluation Metrics

Cross-Entropy Loss (CE Loss) Score for SAEs

- **Metric**: `metrics/CE_loss_score`
- **Importance**: A high CE loss score for SAEs indicates that the SAE retains most of the information from the original residual stream
- **Reasoning**:
  - When the SAE reconstructs the residual stream activations, its impact on the model's overall performance (as measured by CE loss) should be minimal
  - A high score confirms that the SAE's reconstruction does not degrade the model's ability to predict

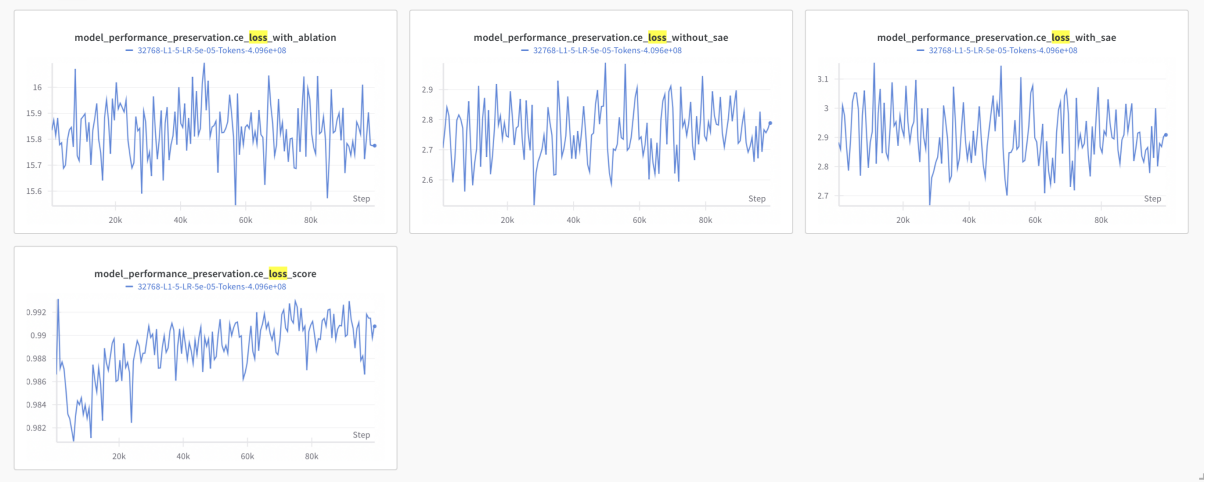
CE Loss Without SAE

- **Metric**: `metrics/ce_loss_without_sae`
- **Expected Value**: Below `3.5`
- **Importance**: Validates that the base model is working correctly before integrating the SAE
- **Reasoning**:
  - If the CE loss without the SAE is unusually high, it may indicate:
    - Issues with the base model (e.g., incorrect setup, poor data quality)
    - A mismatch between the dataset and the model's pretraining distribution
  - Ensures that the evaluation metrics are meaningful by confirming the baseline performance

CE Loss With SAE

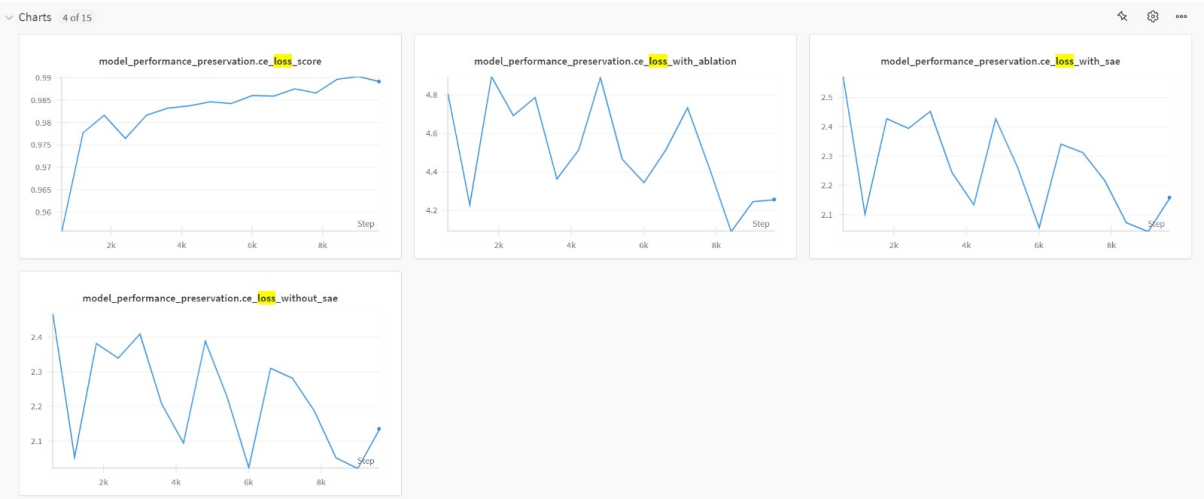
- **Metric**: `metrics/ce_loss_with_sae`
- **Expected Behavior**: Should not be significantly higher than `ce_loss_without_sae`
- **Importance**: Assesses the impact of the SAE on model performance
- **Reasoning**:
  - A minimal increase in CE loss after adding the SAE suggests:
    - The SAE preserves key features in the residual stream
    - The reconstruction is faithful and does not interfere with downstream tasks
  - A large increase in CE loss might indicate:
    - Poor reconstruction fidelity from the SAE
    - Sparsity constraints that are too aggressive, leading to loss of critical information

1. General trends (LMSYS Chat 1M)

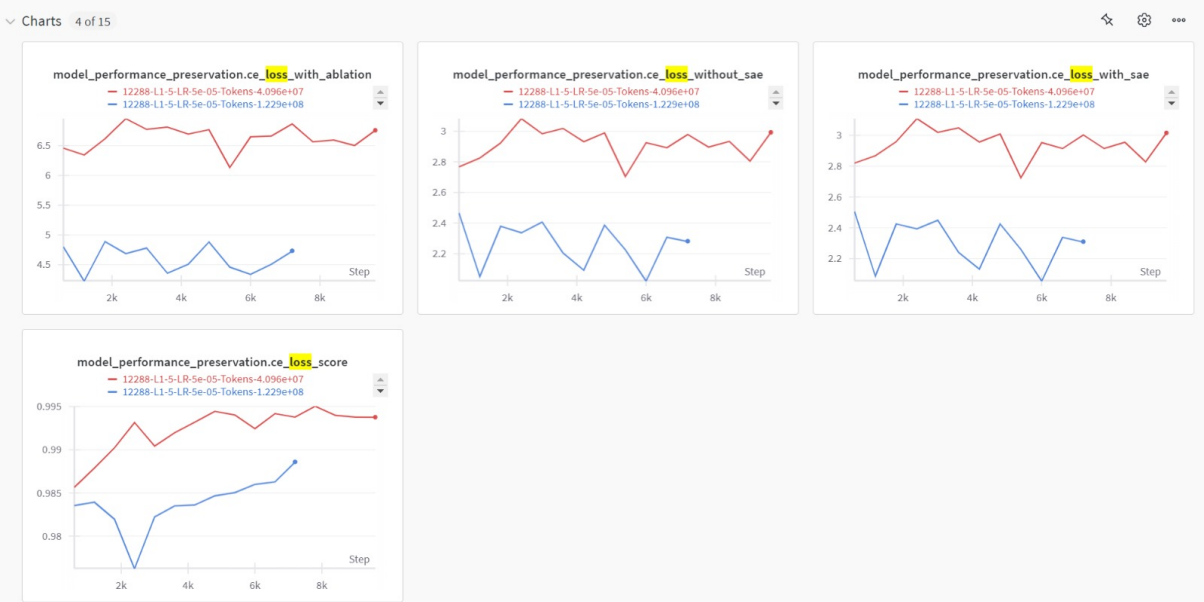


- **CE_loss_score**:
  - The score indeed maintains a consistently high value between `0.982` and `0.992`. Showing an initial dip in the early stages (before `20K` steps). But, however, it gradually stabilizes and improves over time, with minor fluctuations. Final performance is around `0.99`, indicating excellent model preservation
- **CE_loss_without_sae**:
  - The loss is relatively stable, mostly within a narrow range of approximately `2.6` to `2.9`. Average performance centers around `2.75` without showing any significant long-term degradation or improvement
- **CE_loss_with_sae**:
  - The loss with SAE integration is in the range of approximately `2.7` to `3.1`. Shows slightly higher variability compared to `without_sae`
- **CE_loss_with_ablation** (additional observation):
  - Significantly higher loss values (`15.0-16.0` range). But, however, it maintains a relatively stable trend despite higher absolute values without showing any significant long-term degradation or improvement

2. General trends(TinyCodes and NeelNanda/Code-10k)



- **CE_loss_score**:



  - The score remains high (close to 1) throughout training, suggesting that the Sparse Autoencoder (SAE) preserves most of the original model's performance

- **CE_loss_without_sae**:
  - The loss is relatively stable, mostly staying below 3.5, which aligns with expectations
- **CE_loss_with_sae**:
  - The loss with SAE integration stays below 3.1 and shows minor fluctuations over time

Technical Constraints and Limitations

- **Compute Constraints**:
  - Although GPU memory is stable, Colab also has constraints on system RAM, which could become a bottleneck when handling large datasets or preprocessing data. The LMSYS Chat 1M model has been trained on an H100 GPU for approximately 3 hours. Refer to the Weights and Biases report for detailed training logs.
- **Low Iterations**:
  - The SAE may not fully learn to reconstruct activations effectively, leading to poorer representation quality and reduced fidelity in capturing task-specific directions

Visualization and Analysis Tools

SAE Dashboard

- **Overview**:
  - SAE Dashboard is a tool designed for visualizing and analyzing Sparse Autoencoders (SAEs) within neural networks. Offers customizable dashboards that display various plots and data representations of SAE features. Supports any SAE from the SAELens library. [1] [4]
- **Key Features**:
  - **Feature-Centric Visualization**:
    - Allows users to examine individual SAE features, observing which tokens activate most strongly for each feature.

- **Prompt-Centric Visualization**:
  - Enables analysis of specific prompts to determine which features score highest, according to various metrics.

Debugging Prompt Visualization on SAE Dashboard

- **Implementation Challenge**: During this project, we wanted to implement the prompt visualization of the SAEDashboard, as this would allow us to get a better, more fluid understanding of the activations based on the prompts we would generate - relevant to the dataset or otherwise. However, while examining the SAELens library, we noticed that the function to get an activation_subset was missing from the `data_writing_fns.py` and `data_parsing_fns.py` files.
- **Future Direction**: Upon further investigation, we found that there is a mock implementation that simply uses a slice, in a demo notebook within the same repo. Implementing the same would lead us to tensor mismatches, which we believe would compromise the logic of the underlying library. We still believe that implementing prompt visualization with multilayer trained SAEs would be the next step of the project.

Conclusions and Future Work

- **Layer 0 Hook Limitation**:
  - One *feedback* we received was regarding our decision to place the hook only on the `0th layer`. While this allowed us to analyze the activations at the token level, it limited our ability to observe more complex representations, such as higher-order interactions or attention patterns that emerge in deeper layers.
  - This was an oversight, as it restricted the scope of insights we could extract from the Sparse Autoencoder (SAE) training process, mostly visualizing tokens at the first layer that have been fed as input.
- **Computational Constraints**:
  - Our primary reason for sticking to the `0th` layer was due to compute constraints.
  - Training on deeper layers, such as the penultimate or final layer, would have required significantly more computational resources, both in terms of memory and processing time.
  - As we were operating under limited resources, prioritizing the basic layer seemed practical for our initial investigation.

Interesting Observations in SAE Dashboard

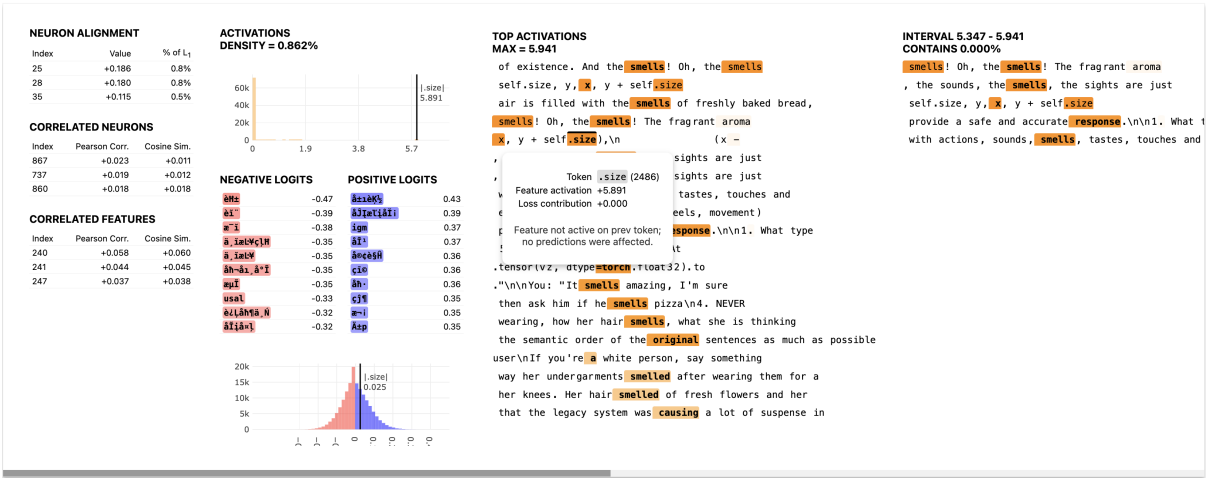


Figure 1: Latent Space Visualization of the SAE Model's Performance on LMSYS Dataset

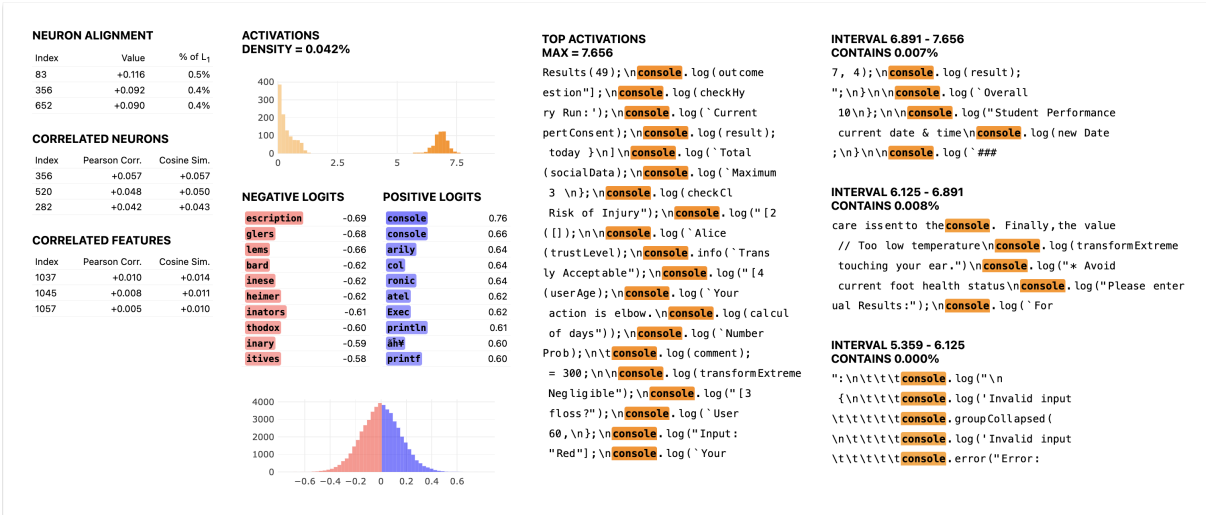


Figure 2: Analysis of SAE Model's Behavior on TinyStories Dataset

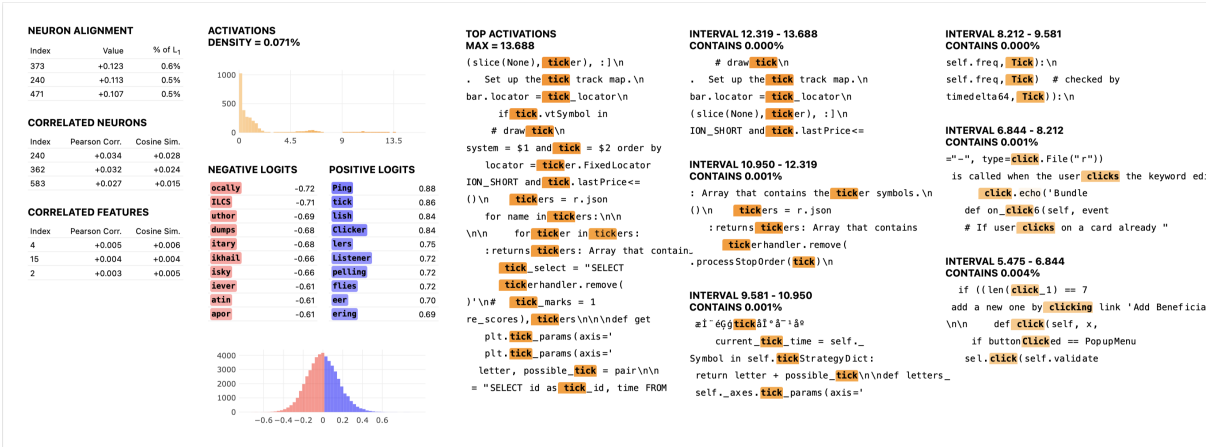


Figure 3: Code10K Performance Metrics from the SAE Dashboard

Contributions

- **Het**:
  - Trained SAE on `Qwen1.5 0.5B` using the `lmsys/lmsys-chat-1m` dataset, processing approximately 409M tokens, over `100,000` iterations on an H100 GPU. Integrated WandB for real-time logging of training metrics and progress. Implemented a periodic checkpointing to save model states every few iterations during training.
- **Vijay**:
  - Trained SAEs on `GPT2` using the `TinyCodes` and `NeelNanda/code-10k` for `10,000` iterations each. Attempted `30,000` iterations as shown in the WandB report that ended early. Helped in debugging SAE_Dashboard.
- **Dattatreya**:
  - Set up SAEDashboard in tandem with the SAEs trained above for visualization. Prior to this he had also explored the possibility of using LogitLens but moved towards SAEDashboard because of it's flexibility.

Note: All items in the list are hyperlinks. The first three links lead to features dashboards for LMSYS, TinyCodes, and Code10K respectively. The last two links direct to folders containing trained SAE (Sparse Autoencoder) files for various datasets.

- features_dashboard_lmsys.html
- features_dashboard_tinycodes.html
- features_dashboard_code10K.html
- Trained SAE Files for LMSYS-Chat-1M
- Trained SAE Files for TinyCodes and Code_10K
- LLM_code_10k_analysis_gpt2

```
# Note: The Tiny and Code-10k models use GPT2-small with an input dimension (d_in) of 768 and
↪  undergo 30K training steps.
# The architecture uses an expansion factor of 16 and a context window size of 512 tokens.
# This is the training code for the Lmsys 1M Chat dataset.

import os

import torch
import wandb
from sae_lens import SAE, LanguageModelSAERunnerConfig, SAETrainingRunner

total_training_steps = 100_000
batch_size = 4096
total_training_tokens = total_training_steps * batch_size

lr_warm_up_steps = 0
lr_decay_steps = total_training_steps // 5  # 20% of training
l1_warm_up_steps = total_training_steps // 20  # 5% of training

def train_sae(wandb_key=None, save_path="/home/volume/lmsys-chat-1m-qwen1.5-0.5b-chat"):
    """Train and save the SAE model"""
    if wandb_key:
        wandb.login(key=wandb_key)

    cfg = LanguageModelSAERunnerConfig(
        # Model Configuration - our model (more options here:
        ↪  https://neelnanda-io.github.io/TransformerLens/generated/model_properties_table.html)
        model_name="qwen1.5-0.5b-chat",
        hook_name="blocks.0.hook_mlp_out",  # A valid hook point (see more details here: https://ne⌋
        ↪  elnanda-io.github.io/TransformerLens/generated/demos/Main_Demo.html#Hook-Points)
        hook_layer=0,  # Only one layer in the model
        d_in=1024,  # the width of the mlp output
        dataset_path="ckkissane/lmsys-chat-1m-qwen1.5-0.5b-chat",  # Using the LMSYS chat dataset
        is_dataset_tokenized=True,  # Dataset from HF Hub is not pre-tokenized
        streaming=True,  # Stream from HF Hub

        # SAE Parameters
        mse_loss_normalization=None,  # We won't normalize the mse loss
        expansion_factor=32,  # the width of the SAE. Larger will result in better stats but slower
        ↪  training
        b_dec_init_method="zeros",  # The geometric median can be used to initialize the decoder
        ↪  weights
        apply_b_dec_to_input=False,  # We won't apply the decoder weights to the input
        normalize_sae_decoder=False,
        scale_sparsity_penalty_by_decoder_norm=True,
        decoder_heuristic_init=True,
        init_encoder_as_decoder_transpose=True,
        normalize_activations="expected_average_only_in",

        # Training Parameters
        lr=5e-5,  # lower the better, we'll go fairly high to speed up the tutorial
        adam_beta1=0.9,  # adam params (default, but once upon a time we experimented with these)
        adam_beta2=0.999,
        lr_scheduler_name="constant",  # constant learning rate with warmup. Could be better
        ↪  schedules out there
        lr_warm_up_steps=lr_warm_up_steps,  # this can help avoid too many dead features initially
        lr_decay_steps=lr_decay_steps,  # this will help us avoid overfitting
```

```
        l1_coefficient=5,  # will control how sparse the feature activations are
        l1_warm_up_steps=l1_warm_up_steps,  # this can help avoid too many dead features initially
        lp_norm=1.0,  # the L1 penalty (and not a Lp for p < 1)
        train_batch_size_tokens=batch_size,
        context_size=2048,  # will control the length of the prompts we feed to the model. Larger is
        ↪  better but slower

        # Activation Store Parameters
        n_batches_in_buffer=64,  # controls how many activations we store / shuffle
        training_tokens=total_training_tokens,  # 100 million tokens is quite a few, but we want to
        ↪  see good stats
        store_batch_size_prompts=8,

        # Resampling Protocol
        use_ghost_grads=False,  # we don't use ghost grads anymore
        feature_sampling_window=1000,  # this controls our reporting of feature sparsity stats
        dead_feature_window=1000,  # would effect resampling or ghost grads if we were using it
        dead_feature_threshold=1e-4,  # would effect resampling or ghost grads if we were using it

        # Wandb Configuration
        log_to_wandb=True,  # always use wandb unless you are just testing code
        wandb_project="lmsys-chat-1m-qwen1.5-0.5b-chat",
        wandb_log_frequency=30,
        eval_every_n_wandb_logs=20,

        # Checkpointing Configuration
        n_checkpoints=5,  # Save 5 checkpoints during training
        checkpoint_path=save_path,

        # Misc Configuration
        device=device,
        seed=42,
        dtype="float32",

        # Performance Optimizations
        autocast_lm = True,
        autocast = True,
    )

    # Initialize and train the SAE
    sparse_autoencoder = SAETrainingRunner(cfg).run()

    # Save the final model
    final_save_path = os.path.join(save_path, "final_sae")
    os.makedirs(final_save_path, exist_ok=True)
    sparse_autoencoder.save_pretrained(
        save_path=final_save_path,
        cfg_dict=cfg.to_dict(),
        save_activations=False
    )

    return sparse_autoencoder

if __name__ == "__main__":
    # To run with wandb logging
    sparse_autoencoder = train_sae(wandb_key="key")
```

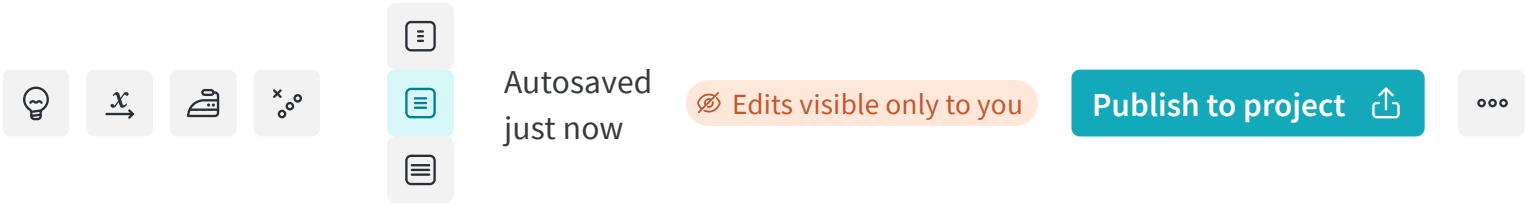
References

[1] Curt Tigges Joseph Bloom and David Chanin. Saelens. https://github.com/jbloomAus/SAELens, 2024.

[2] Connor Kissane, Robert Krzyzanowski, Neel Nanda, and Arthur Conmy. Saes are highly dataset dependent: A case study on the refusal direction. Alignment Forum, 2024.

[3] Tom Lieberum. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2. *arXiv preprint*, arXiv:2408.05147, Aug 2024.

[4] Decode Research. SAE Dashboard. https://github.com/jbloomAus/sae-dashboard, 2024.



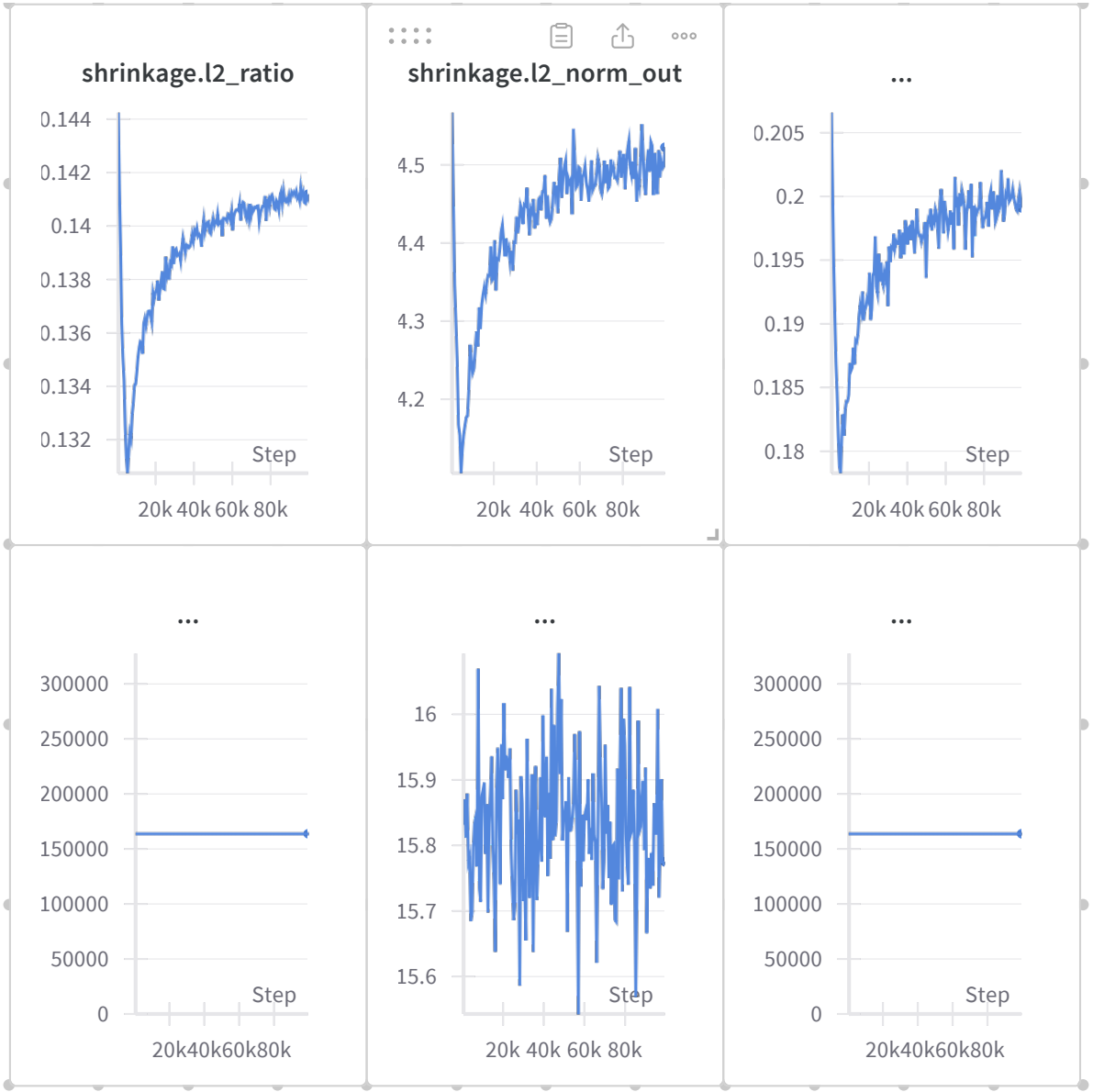
Sparse Autoencoders Training (Mechanistic Interpretability)

Add a description...

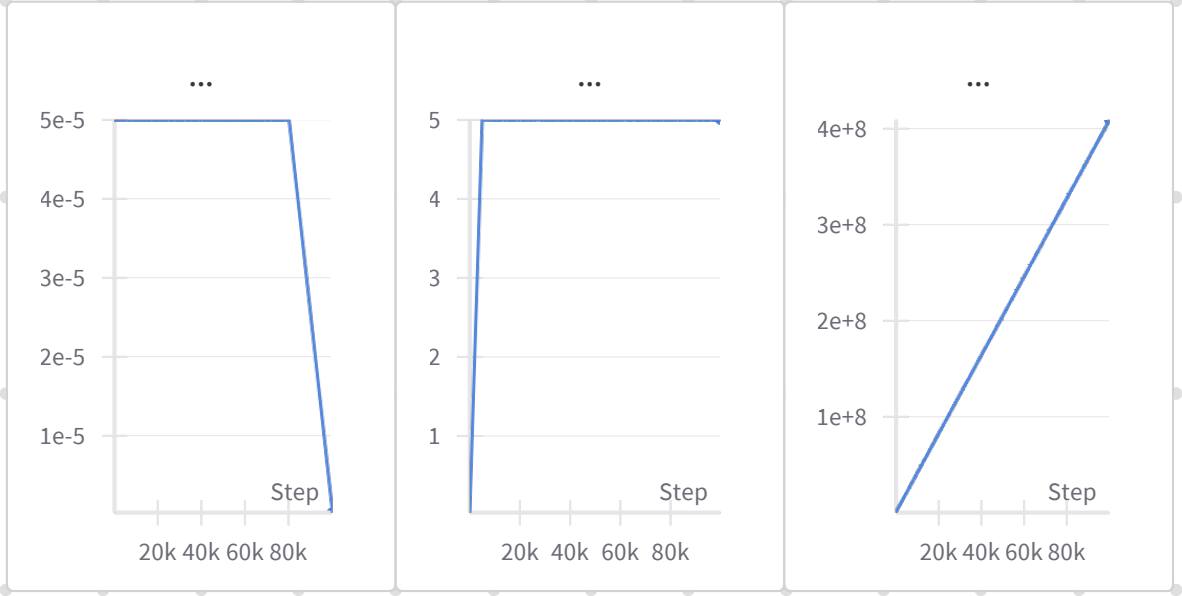
HP

Model: qwen1.5-0.5b-chat, Dataset: LMSYS Chat 1M (ckkissane/lmsys-chat-1m-qwen1.5-0.5b-chat)

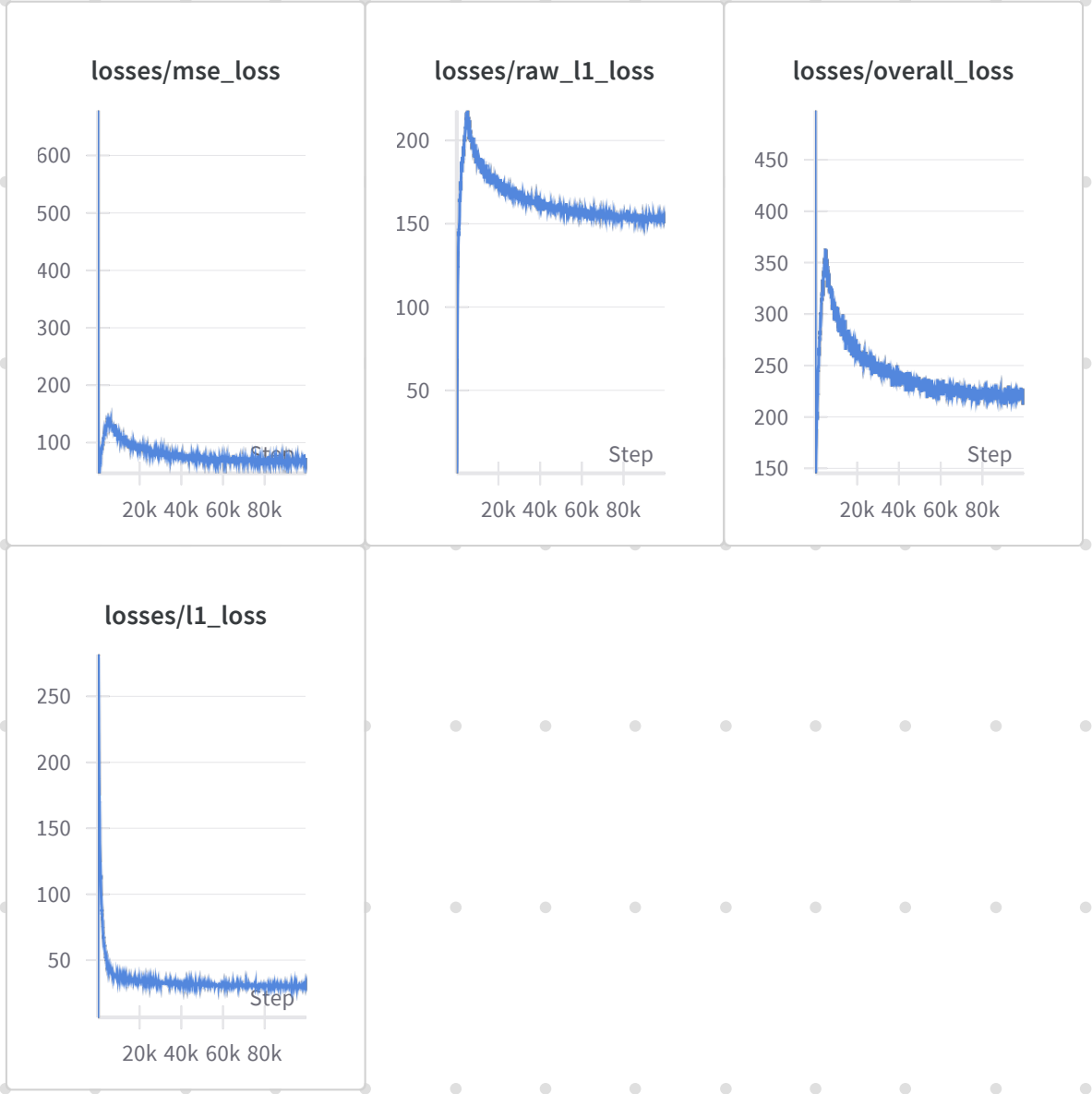
Section 1



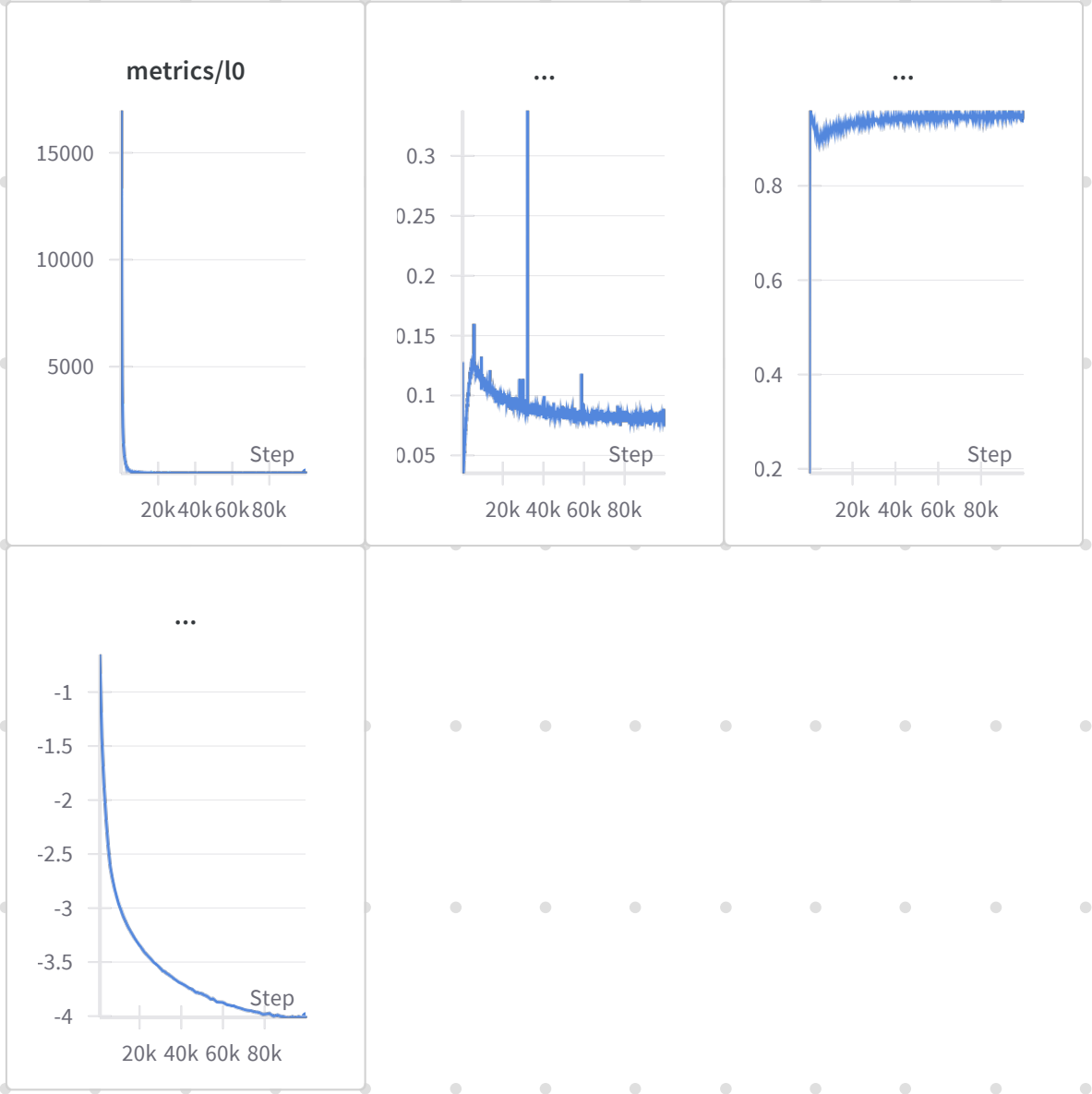
Import panel Add panel



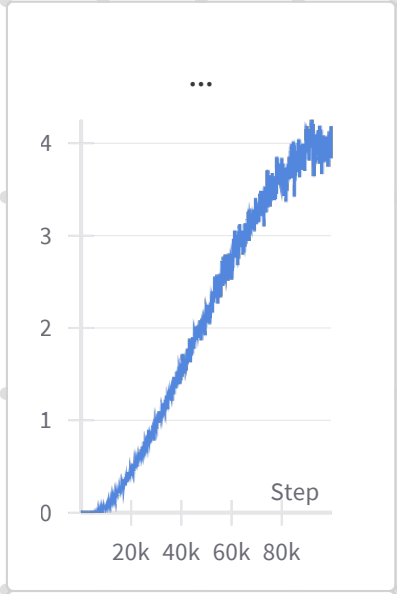
Import panel Add panel

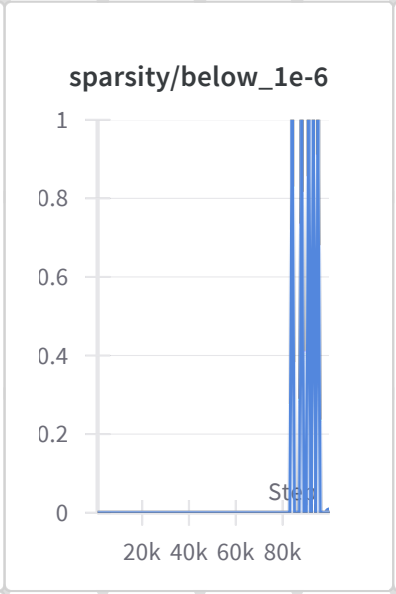


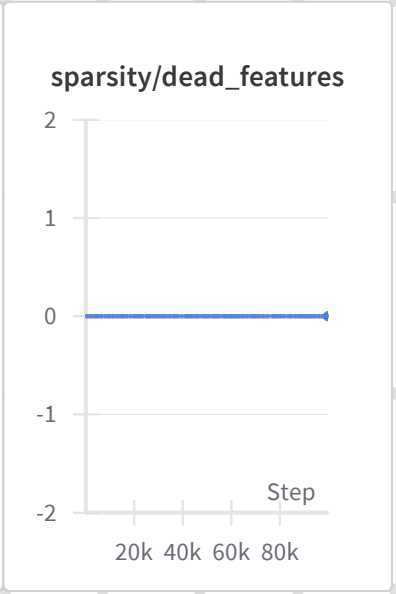
Import panel Add panel

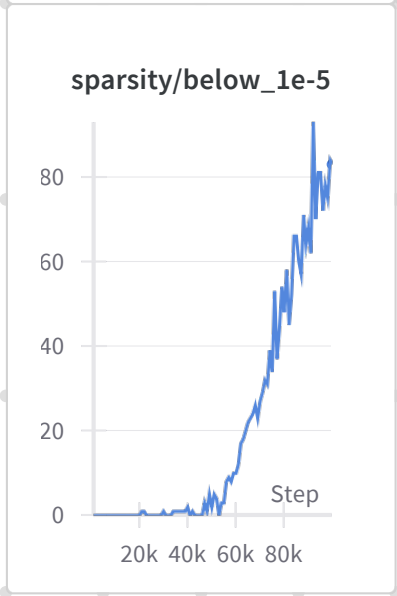


Import panel Add panel

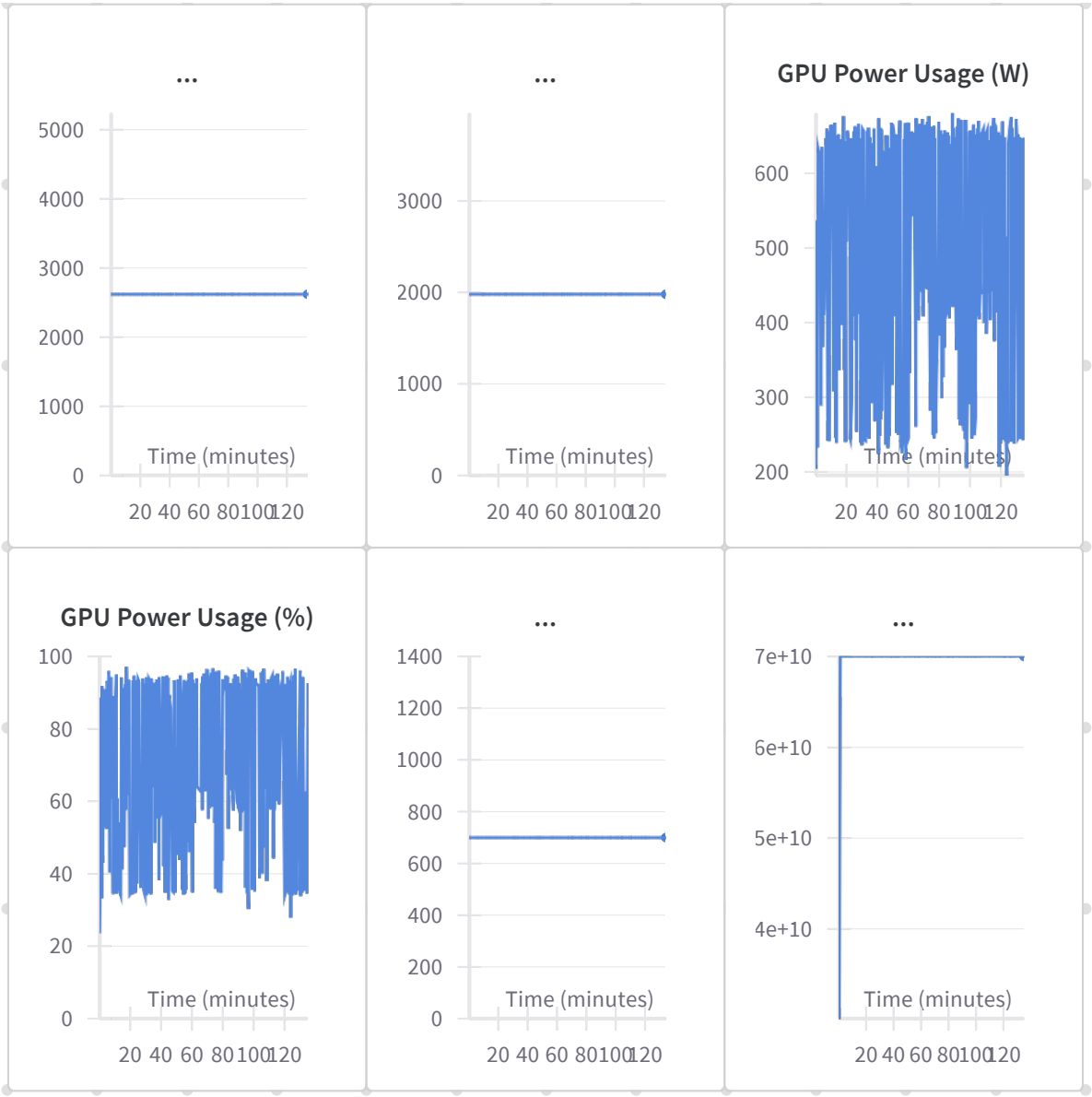








Import panel Add panel



Import panel Add panel

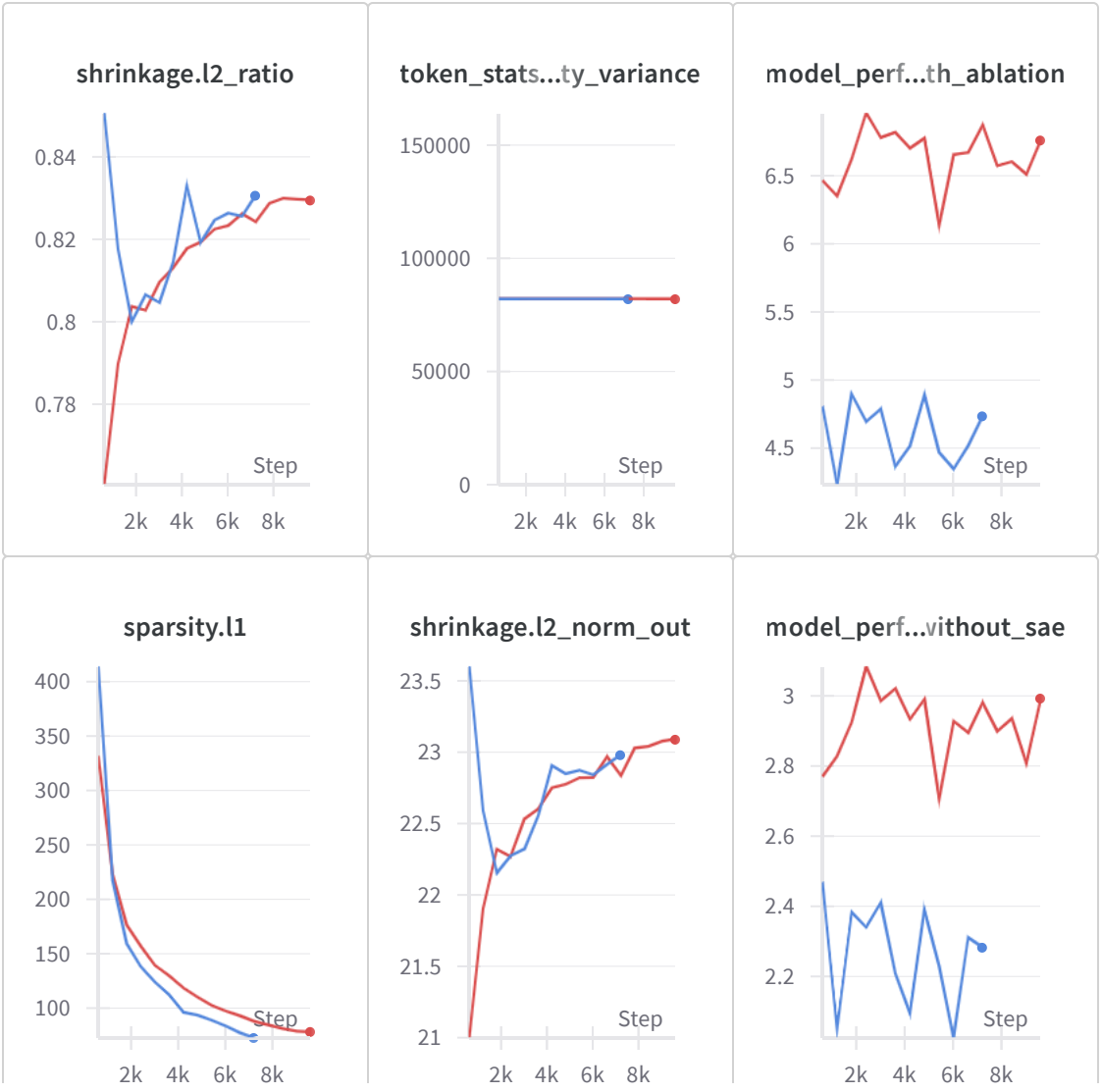
Training SAE on NeelNanda/Code-10k

Model: GPT2
Dataset: NeelNanda/Code-10k

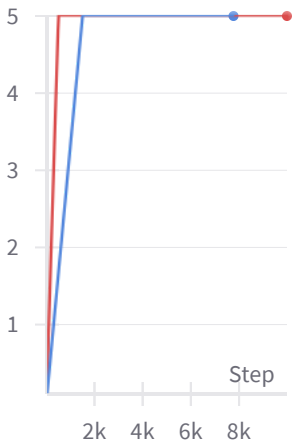
Srikara Raghavendra Vijay Vanapalli

Created on December 4 | Last edited on December 4

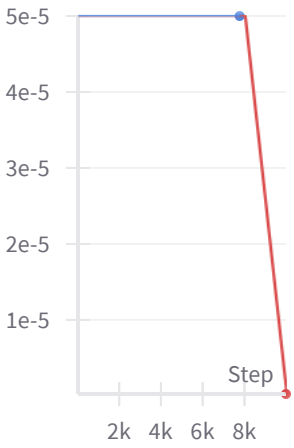
Section 1



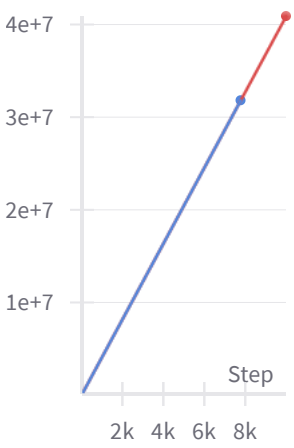
details/curr..._coefficient



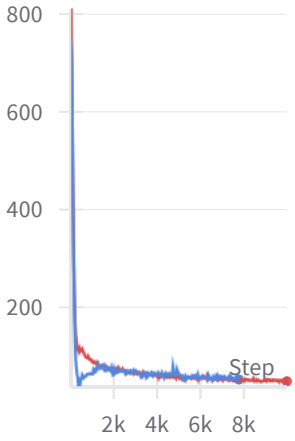
details/curr...arning_rate



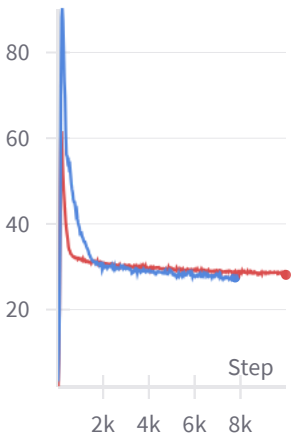
details/n_tr...ing_tokens



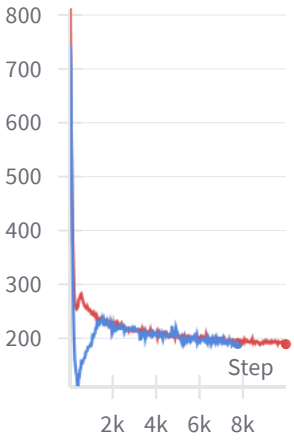
losses/mse_loss



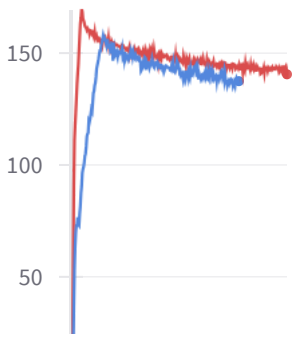
losses/l1_loss

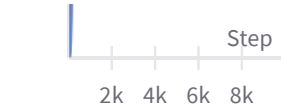


losses/overall_loss

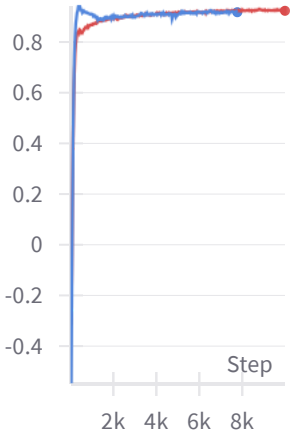


losses/raw_l1_loss

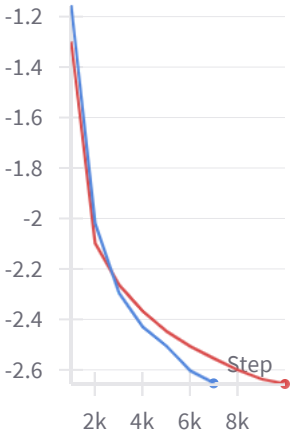




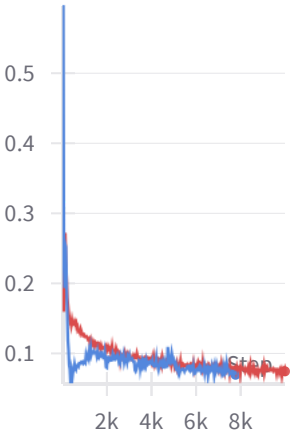
metrics/exp...ed_variance



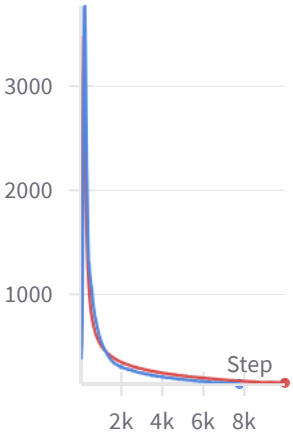
metrics/me...ure_sparsity



metrics/exp...ariance_std

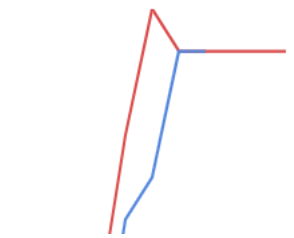


metrics/l0

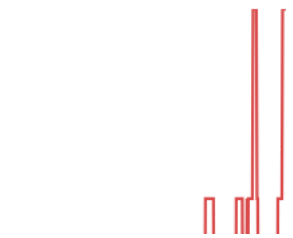


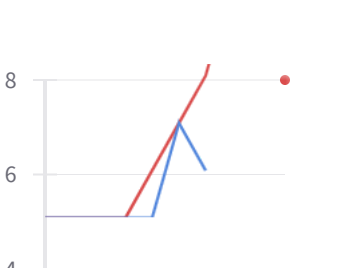
sparsity/below_1e-6

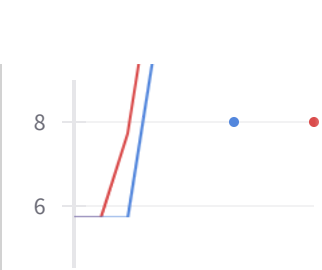
sparsity/below_1e-5

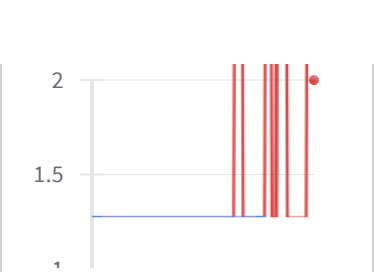


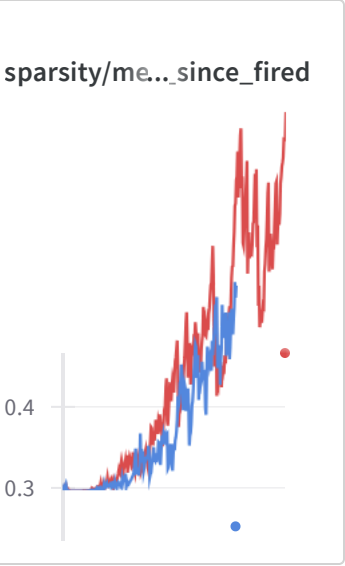
sparsity/dead_features




8
6
4


8
6


2
1.5
1


sparsity/me..._since_fired
0.4
0.3

Created with ❤️ on Weights & Biases.

https://wandb.ai/vijayvanapalli96-george-washington-university-/CODE10K/reports/Training-SAE-on-NeelNanda-Code-10k--VmlldzoxMDQ2NTY3OA

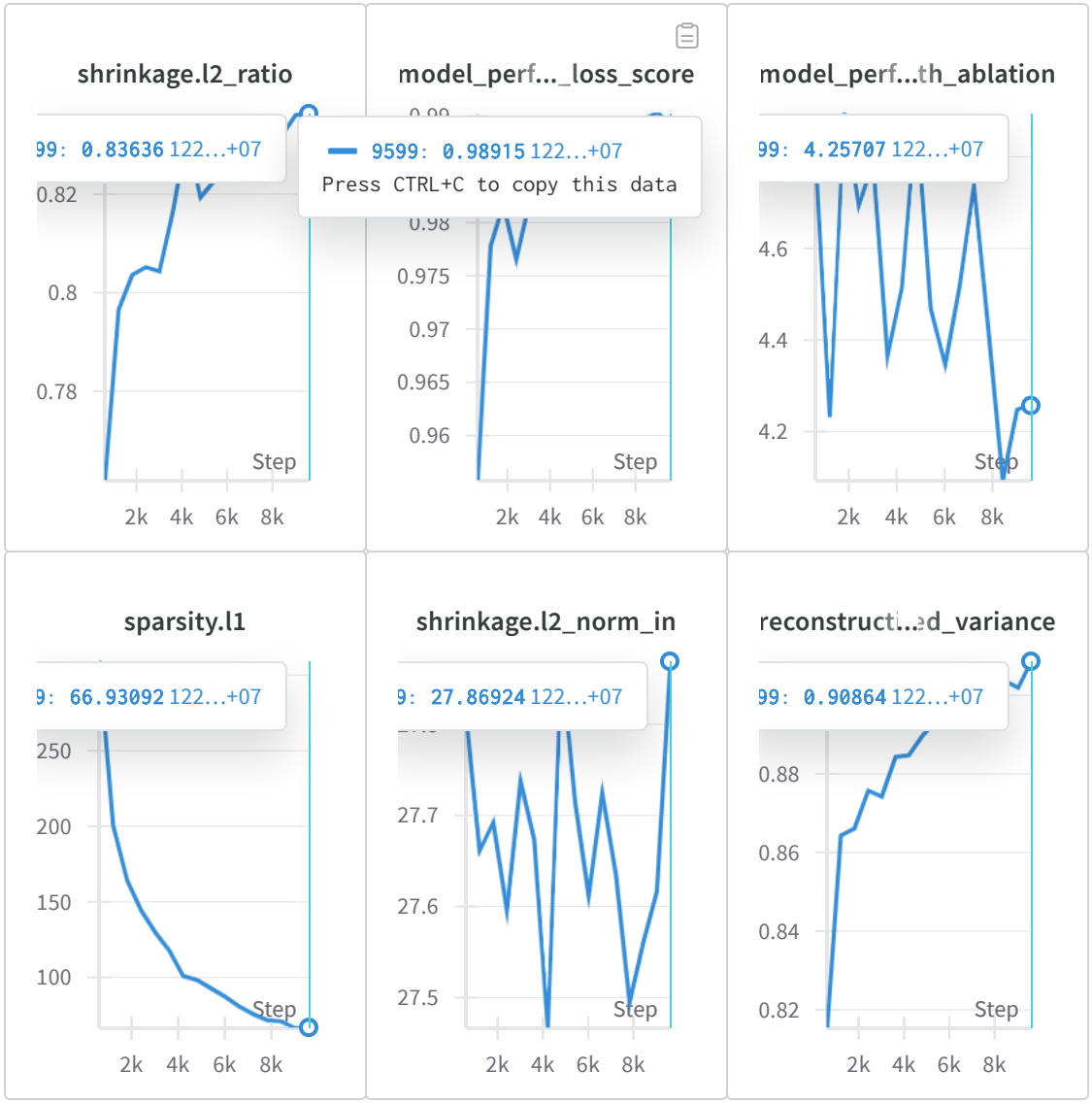
Training SAE on TinyCodes

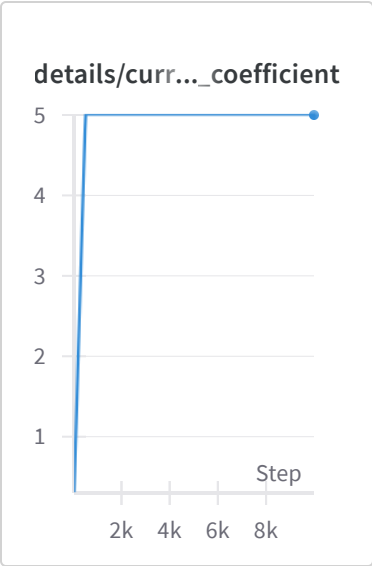
Model: GPT2
Dataset: TinyPixel/tiny-codes

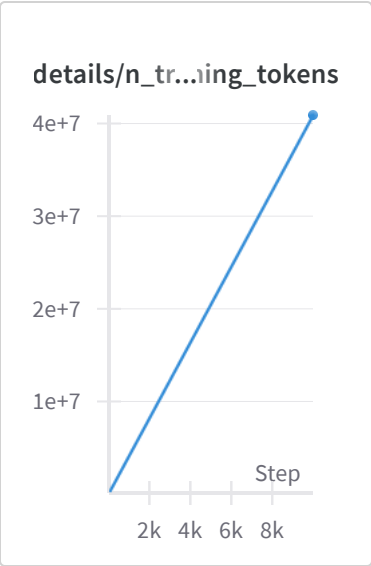
Srikara Raghavendra Vijay Vanapalli

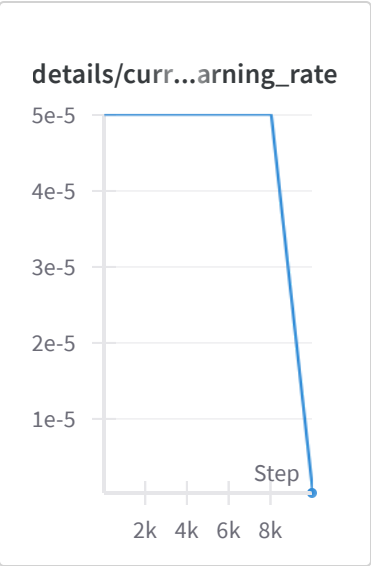
Created on December 4 | Last edited on December 4

Section 1

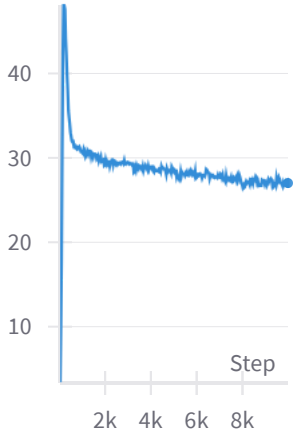

shrinkage.l2_ratio
99: 0.83636 122...+07
0.82
0.8
0.78
Step
2k 4k 6k 8k
model_perf..._loss_score
9599: 0.98915 122...+07
Press CTRL+C to copy this data
0.98
0.975
0.97
0.965
0.96
Step
2k 4k 6k 8k
model_perf...th_ablation
99: 4.25707 122...+07
4.6
4.4
4.2
Step
2k 4k 6k 8k
sparsity.l1
9: 66.93092 122...+07
250
200
150
100
Step
2k 4k 6k 8k
shrinkage.l2_norm_in
9: 27.86924 122...+07
27.7
27.6
27.5
Step
2k 4k 6k 8k
reconstruct...d_variance
99: 0.90864 122...+07
0.88
0.86
0.84
0.82
Step
2k 4k 6k 8k


details/curr..._coefficient
5
4
3
2
1
Step
2k
4k
6k
8k

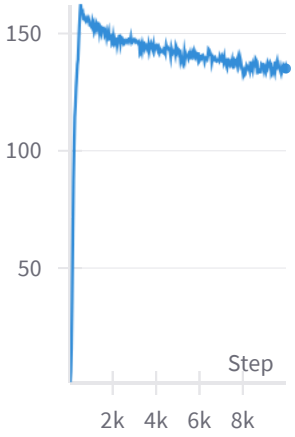

details/n_tr...ing_tokens
4e+7
3e+7
2e+7
1e+7
Step
2k
4k
6k
8k


details/curr...arning_rate
5e-5
4e-5
3e-5
2e-5
1e-5
Step
2k
4k
6k
8k

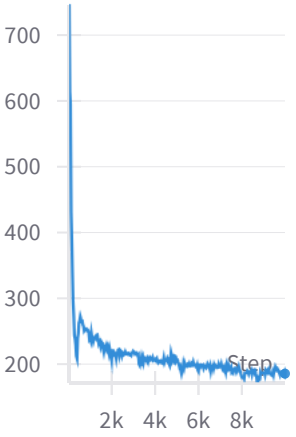
losses/l1_loss



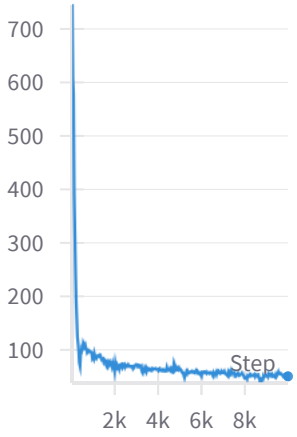
losses/raw_l1_loss



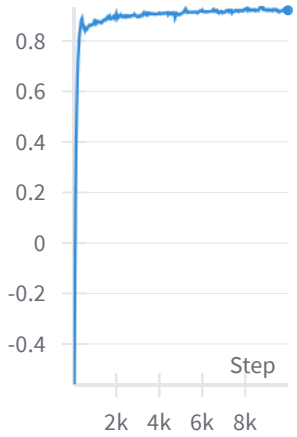
losses/overall_loss



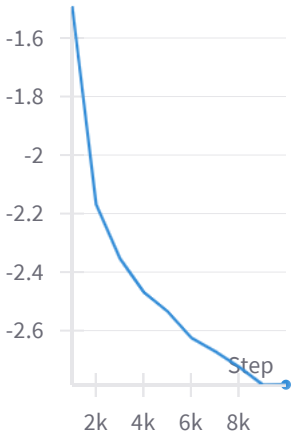
losses/mse_loss



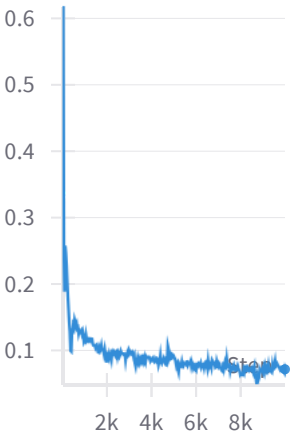
metrics/exp...ed_variance



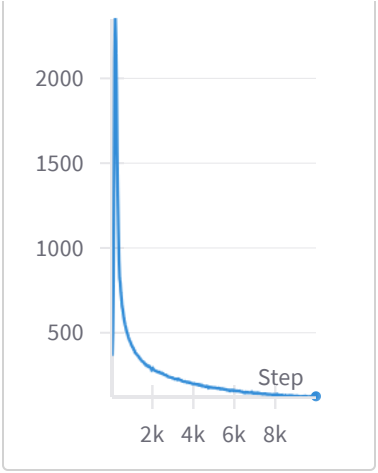
metrics/me...re_sparsity

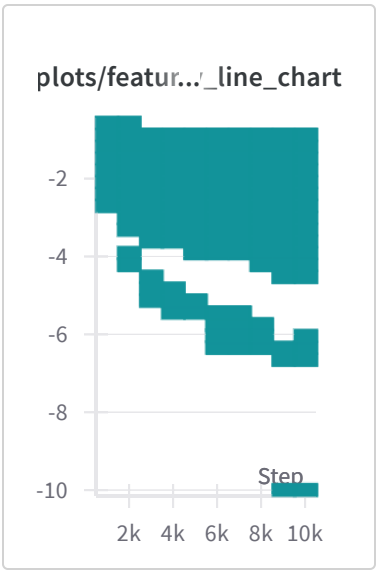


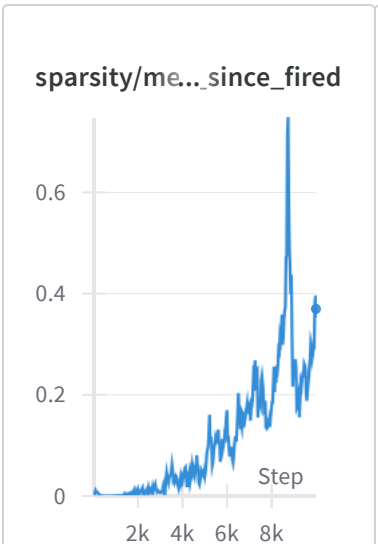
metrics/exp...ariance_std

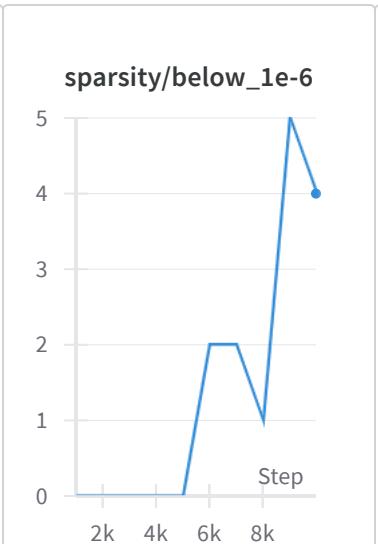


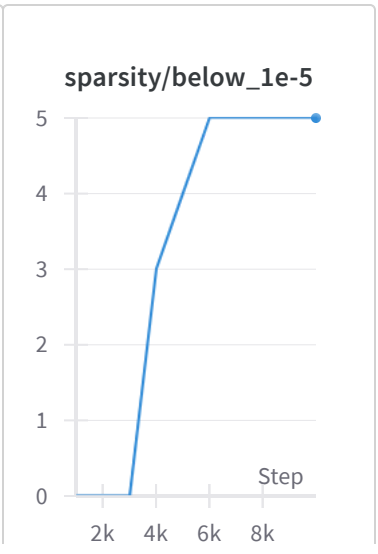
metrics/l0


2000
1500
1000
500
Step
2k
4k
6k
8k

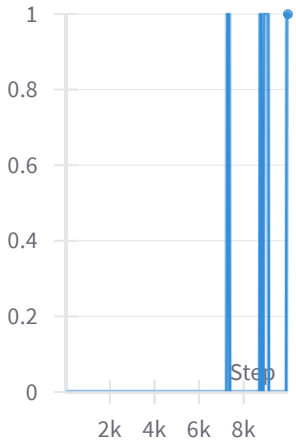

plots/featur.../_line_chart
-2
-4
-6
-8
-10
Step
2k
4k
6k
8k
10k


sparsity/me..._since_fired
0.6
0.4
0.2
0
Step
2k
4k
6k
8k

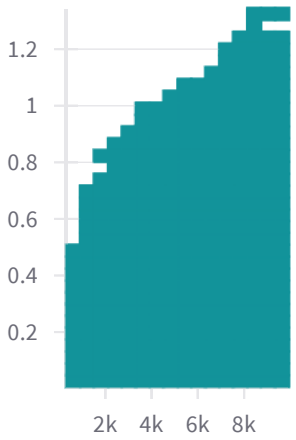

sparsity/below_1e-6
5
4
3
2
1
0
Step
2k
4k
6k
8k


sparsity/below_1e-5
5
4
3
2
1
0
Step
2k
4k
6k
8k

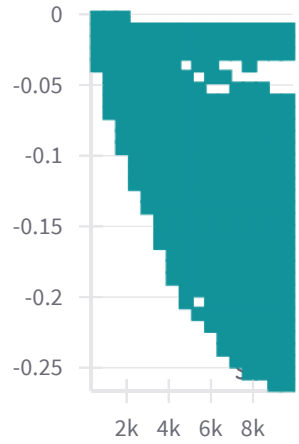
sparsity/dead_features



weights/W_dec_norms



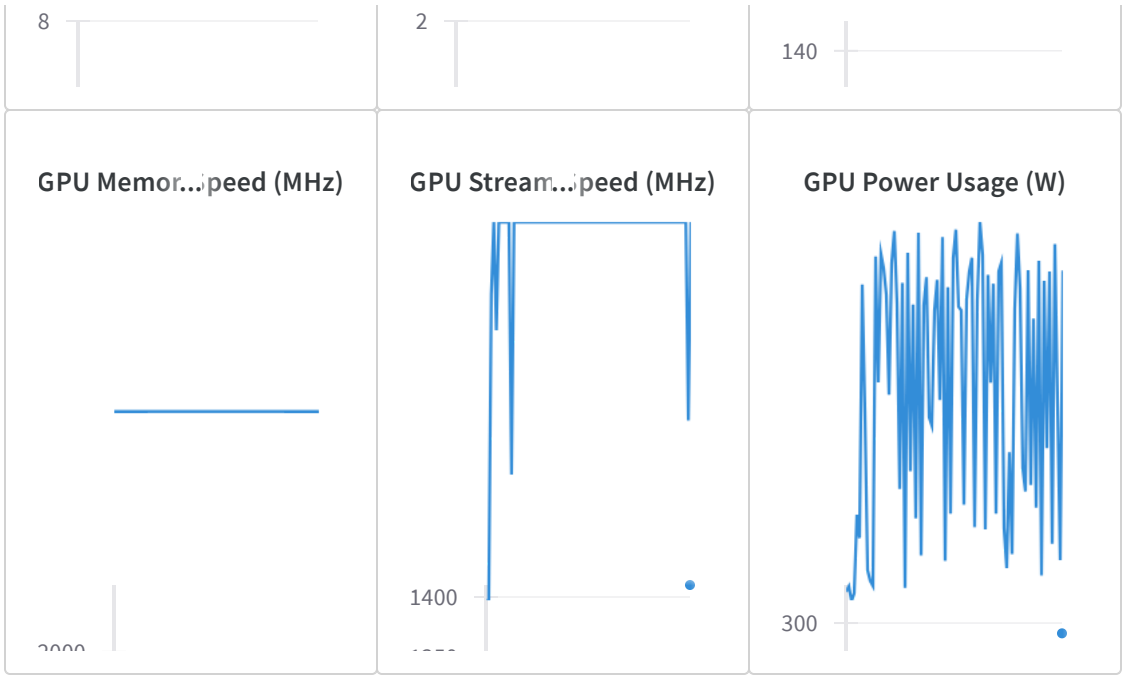
weights/b_e



system/gpu...cieLinkGen

GPU Uncorr...mory Errors

GPU Correct...mory Errors



Created with ❤️ on Weights & Biases.

https://wandb.ai/vijayvanapalli96-george-washington-university-/TINYCODES/reports/Training-SAE-on-TinyCodes---VmlldzoxMDQ2NTY1Ng