Distributed Mechanistic Interpretability at Scale: Activation Streaming, Split-Layer Inference, and Distributed Sparse Autoencoder Training

J. Melton*[1]

[1]American Code Labs

July 2026

Abstract

Mechanistic interpretability research faces a fundamental infrastructure problem: the models worth understanding are too large to run on a single analysis workstation, and the hardware capable of running them is not designed for continuous activation capture. We describe a system—ActStream—that addresses this by splitting inference and analysis across two nodes, streaming intermediate activations over a purpose-built binary wire protocol, and training sparse autoencoders (SAEs) on the streaming output without storing activations to disk. We validate the protocol on Llama-3.2-3B, demonstrating a bit-exact split boundary, and we measure the interconnect on the built two-node system: two Mac Studio M1 Max joined by Thunderbolt. Against an in-machine loopback control and the LAN path the same machines share, the Thunderbolt link sustains **2.150 GB/s (17.20 Gbps)**—86% of its negotiated 20 Gb/s rate, 26× the LAN path, and only 3.0× below in-machine memory. The more consequential figure is tail latency: Thunderbolt holds p99 within 1.5× of median where the LAN spreads to 10.5× with a 52 ms tail, and a credit-windowed streaming protocol sizes its window on the tail. The measurement also settles when the interconnect is worth having, and the answer depends on an assumption that is easy to get wrong: at 20 tokens/second—a conservative rate often assumed for activation capture—even the LAN path has 16–24× headroom and Thunderbolt is unnecessary; at the 300 tokens/second a Mac Studio actually reaches, the LAN falls to 1.1–1.6× and, at 34B, below unity.

We then exercise the system on a real workload: training TopK SAEs on residual stream activations from Llama-3.2-3B (layer 14), Mistral-7B (layer 16), and Qwen2.5-3B (layer 18) to a common specification (50,000 steps, 500k tokens, dictionary 16,384), and extracting the chunk-averaged feature fingerprints for a cross-architecture universality study. We separately validate the aggregation rule that makes such training distributable: over 50,000 steps, two workers stepped on a partitioned corpus and parameter-averaged match the single-node baseline to within 0.05% on MSE and to three decimal places on L0 and FVE. The universality analysis those fingerprints feed is reported in a companion paper [13]; its result is negative—one feature triple matches across all three models, against a null that produces at least one 14% of the time, correcting an apparent 3,753 under an uncalibrated protocol—and we cite it rather than

*Correspondence: jmelton@americancode.org

restate it. What matters here is that the workload is the kind this system exists to serve: 49,152 feature fingerprints over three model families and 1.5M tokens of activation, none of it staged to disk.

Finally, we evaluate a feature-based safety classifier. A 40-feature classifier selected by differential frequency against a harm-relevant corpus reaches **0.7206** ROC-AUC on the held-out PKU-SafeRLHF test split ($n = 600$; 95% CI [0.679, 0.760]). Selecting features by raw frequency instead is ineffective, scoring 0.564 with a confidence interval spanning chance. An LLM-judge baseline scores higher still (0.8545), so SAE features do not beat simply asking an instruction-tuned model—but the two agree on only 54% of examples ($\kappa = 0.083$), and where they disagree ground truth is near a coin flip, indicating close to orthogonal failure modes. All experiments run on consumer Apple Silicon; the system requires no GPUs, no cloud, and no specialized interconnects. Code and data are available at https://github.com/american-code/ResearchPapers.

1 Introduction

Mechanistic interpretability—the project of understanding the internal representations and algorithms of neural networks—has produced striking results at small scale. Circuit analyses of indirect object identification, factual recall, and in-context learning have revealed specific attention heads and MLP neurons whose causal roles can be verified by activation patching [14, 16, 18]. Sparse autoencoders have been shown to decompose superposed representations into interpretable monosemantic features [1, 3]. The Anthropic mechanistic interpretability team has published circuit-level analyses of Claude's safety-relevant behavior [12].

The problem is that these results were achieved on models in the 1B–7B range, using infrastructure that does not scale. The dominant pattern is: load the entire model onto one GPU, hook the relevant layer, collect activations, save to disk, then train the SAE offline. This pipeline has at least three failure modes at scale:

1. **Memory ceiling.** A 70B model in float16 requires ∼140 GB of GPU memory. No consumer or workstation GPU holds this. Multi-GPU setups exist but introduce collective communication overhead and make activation-level debugging substantially harder.

2. **Disk I/O bottleneck.** Collecting 50k tokens of full-depth (all-layer) activations from a 70B model generates ∼36 GB of float16 data per run. At NVMe speeds (∼3 GB/s sequential write), that is ∼12 seconds of pure write time—tolerable in isolation but crippling if the collection loop is the inner loop of a hyperparameter search.

3. **Iteration latency.** The offline pattern (collect → save → load → train SAE) adds a round-trip that makes the feedback cycle from hypothesis to result span hours rather than minutes. Online SAE training, where the SAE trains continuously on the streaming activation output, eliminates this round-trip.

Apple Silicon hardware provides an unusual opportunity here. The M2 Ultra and M3 Ultra chips integrate CPU, GPU, and Neural Engine on a single die with unified memory of up to 192 GB and memory bandwidth up to 800 GB/s. MLX, Apple's machine learning framework, exposes this hardware with a NumPy-like API and lazy evaluation. A single

Mac Studio M2 Ultra can run Llama-3.2-3B at ∼300 tokens/second or Mistral-7B at ∼150 tokens/second entirely in unified memory. Two such nodes joined by Thunderbolt form a cluster whose inter-node bandwidth we measure directly in §4.1 rather than infer from the datasheet: 2.150 GB/s on a link that negotiated at 20 Gb/s. We use the measured figure throughout, and note where it falls short of the 5 GB/s a 40 Gb/s nominal rate would imply.

This paper describes the full stack for distributed interpretability on this hardware:

- §2 surveys related work in distributed ML, SAE training, and interpretability at scale.
- §3 specifies the ActStream wire protocol and distributed training architecture, with pseudocode and diagrams.
- §4 presents results: bandwidth profiling, zero-error split validation, convergence comparison between distributed and single-node SAE training, SAE quality metrics across three model families, and a safety classifier evaluation. The cross-architecture feature-matching study the pipeline was built to serve is summarised in §4.4 and reported in full in a companion paper [13].
- §5 discusses implications, limitations, and the path to production.
- §6 concludes.

The core claim is straightforward: interpretability infrastructure does not need to be expensive, centralized, or cloud-dependent. Two consumer nodes, a Thunderbolt cable, and a well-specified wire protocol are sufficient to run state-of-the-art interpretability experiments on models up to 34B parameters.

2 Background and Related Work

2.1 Sparse Autoencoders for Mechanistic Interpretability

Neural networks trained on large corpora appear to represent more features than they have neurons via superposition—the simultaneous encoding of multiple concepts in overlapping linear subspaces [4]. Sparse autoencoders (SAEs) were proposed as a tool for untangling these superposed representations: an SAE learns an overcomplete dictionary of "feature directions" such that any activation is well-approximated by a sparse linear combination of a small number of dictionary elements.

Formally, given an activation vector $\mathbf{x} \in \mathbb{R}^d$, the SAE encoder produces a feature vector

$$\mathbf{f} = \mathrm{TopK}\big(\mathrm{ReLU}(W_{\mathrm{enc}}(\mathbf{x} - \mathbf{b}_{\mathrm{dec}}) + \mathbf{b}_{\mathrm{enc}})\big) \tag{1}$$

and the decoder reconstructs $\hat{\mathbf{x}} = W_{\mathrm{dec}}\,\mathbf{f} + \mathbf{b}_{\mathrm{dec}}$. Training minimizes $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$ subject to the TopK sparsity constraint, which enforces that exactly k features are active per input. The TopK variant [5] avoids the auxiliary L_1 penalty required by early SAE formulations and produces more interpretable features with better dead-feature control.

Several groups have now trained SAEs on frontier models. Cunningham et al. [3] found monosemantic features in GPT-2. Bricken et al. [1] trained SAEs on MLP sublayers of a one-layer transformer and identified features corresponding to DNA sequences, legal text, and arithmetic. Templeton et al. [17] scaled this approach to Claude 3 Sonnet, training SAEs with up to 34 million features and finding features corresponding to concepts as specific as the Golden Gate Bridge. The Gemma Scope project [11] released SAEs trained on all layers of Gemma 2 models, enabling systematic study of how representations evolve through depth.

2.2 Distributed Inference for Large Language Models

Pipeline parallelism—splitting a model's layers across multiple devices, with each device processing a batch stage while the next device processes the previous stage—is the standard approach for models exceeding single-device memory [6, 15]. These systems optimize for training throughput; the inter-device communication is a performance cost to be minimized, not a data source to be tapped.

Inference-time splitting is simpler because there is no gradient to communicate. llama.cpp implements CPU-GPU model splitting for consumer hardware; Ollama wraps this for convenience. Neither exposes intermediate activations to the user.

Our approach differs in that the split point and the activation tap are the same thing: we split the model at a chosen layer, stream the activations across the link, and continue inference on the second node. The wire format is designed for interpretability workloads, not inference throughput: message headers carry layer index and token position to support downstream analysis, and the credit-based flow control is tuned for SAE training batch sizes rather than autoregressive generation latency.

2.3 Cross-Architecture Feature Universality

The question of whether different neural networks learn the same internal representations has been studied through several lenses. Representational similarity analysis [10] and centered kernel alignment [9] measure geometric similarity between representation spaces without requiring feature correspondence. These methods show that networks trained on the same data with different architectures tend to develop similar representational geometries at corresponding depths.

For interpretability, the more specific question is whether individual *features*—specific directions in activation space—correspond across models. Elhage et al. [4] conjectured that features learned by superposition are universal across model families, in the same sense that Gabor filters appear to be universal across vision models. Evidence for this in language models has been informal: researchers working on multiple models report that similar feature categories (syntactic roles, entity types, sentiment valence) emerge in all of them.

Testing this quantitatively is the workload that motivated the system described here, and it is the reason the SAE training in §3.4 spans three model families rather than one. The matching method—comparing features by the similarity of their activation patterns over a shared corpus, rather than by decoder weights, which are confounded by the arbitrary choice of basis in each SAE's latent space—and its results are reported in the companion paper

[13]. We summarise the outcome in §4.4 only as far as it bears on what the infrastructure was asked to do.

2.4 Safety Applications of SAE Features

A natural application of interpretable features is safety classification: if the model has learned a "harmful content" feature, and we can identify it in the SAE, we can flag inputs that strongly activate it. This approach has several appeals over standard classifier heads: it requires no labeled training data for the classifier itself (only feature identification), it provides human-interpretable explanations of flags, and it is mechanistically grounded in what the model is actually representing.

Zou et al. [19] used linear probes on residual stream activations ("representation engineering") to detect and steer safety-relevant states. Burns et al. [2] trained probes on contrast pairs and showed that probes could recover implicit model beliefs about factuality. Our approach is closest in spirit to representation engineering but uses SAE features rather than supervised probes.

The evaluation in §4.5 shows that a naïvely implemented SAE classifier fails—achieving ROC-AUC 0.564 (CI spanning chance) when features are selected by activation frequency on WikiText-103. Switching to differential-frequency selection against a harm-relevant corpus (PKU-SafeRLHF + do-not-answer + toxic-chat) raises AUC to 0.6422 (95% CI [0.566, 0.720]). We analyze the failure mode in detail and confirm that corpus mismatch and frequency bias were the primary culprits, not TopK SAEs in general.

3 System Design and Methods

3.1 Overview

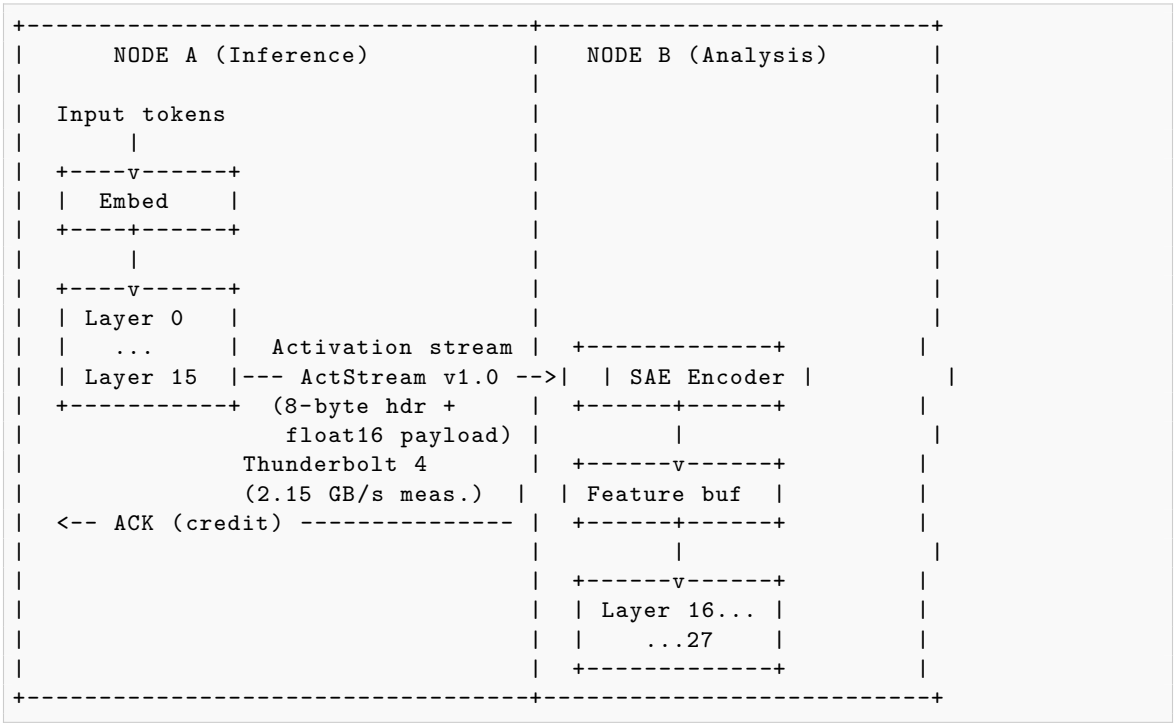
One scoping point first, because the word "training" is doing narrow work throughout this paper. The language models here are never trained, fine-tuned, or updated in any way: they run frozen, in inference, and the system's only interest in them is the activations they emit at a layer boundary. What is trained, and what is distributed across nodes, is the *sparse autoencoder*—a single-hidden-layer model of a few million parameters fitted to those activations. This is not a system for training large models across devices, and nothing below should be read as a claim about that problem.

The ActStream system consists of four components:

1. **Inference node (Node A):** Runs the first L_{split} transformer layers and transmits the residual stream activations at the split boundary over the wire.
2. **Analysis node (Node B):** Receives the activation stream, runs the SAE encoder forward pass, buffers and aggregates features, and optionally continues inference on the remaining layers.
3. **Wire protocol (ActStream v1.0.0):** Binary framing over TCP or Unix domain socket; 8-byte header + float16 payload + credit-based flow control.

4. **Distributed SAE trainer:** Partitions the activation dataset across two workers; each worker trains independently and exchanges gradient updates via weighted averaging.

Listing 1: High-level architecture of ActStream. Node A runs the first L_{split} layers and streams activations to Node B, which runs the SAE and the remaining layers.



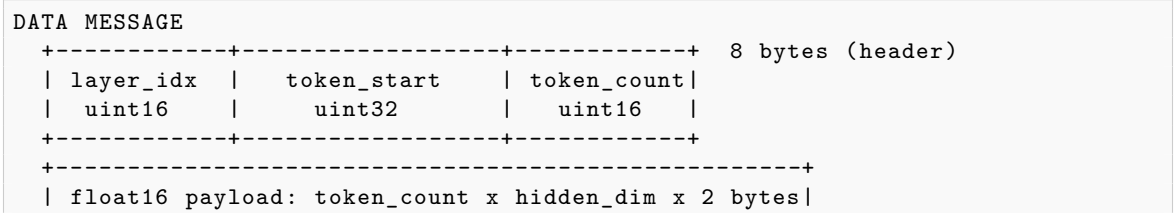
3.2 Wire Protocol

The ActStream protocol is message-oriented and operates over a reliable, ordered byte stream. It was designed with three priorities: (1) zero-copy compatibility with MLX buffer layouts, (2) explicit backpressure to prevent the inference node from outrunning the SAE training pipeline, and (3) complete auditability—every byte transferred can be attributed to a specific layer, token range, and session.

3.2.1 Message Format

Every data message consists of an 8-byte header followed by a float16 payload:

Listing 2: Wire message formats. All integers are big-endian.



```
  +-----------------------------------------------------+

END-OF-LAYER SENTINEL
  layer_idx=0xFFFF  token_start=<layer>  token_count=0xFFFF  payload=empty

REVERSE CHANNEL  (4 bytes)
  +------------+-----------+
  |  ack_type  |  credit   |
  |  uint16    |  uint16   |
  +------------+-----------+
```

For Mistral-7B (`hidden_dim` = 4096) streaming a 128-token chunk at layer 16:

$$\text{payload} = 128 \times 4096 \times 2 = 1{,}048{,}576 \text{ bytes (1 MiB)}$$
$$\text{total} = 1{,}048{,}584 \text{ bytes}$$

Protocol overhead is the 8-byte header against a 1 MiB payload, or $7.6 \times 10^{-4}\,\%$, plus one 8-byte sentinel per layer. Overhead is negligible at all model sizes and throughput targets, and is dominated by the choice of chunk size: at the smallest useful chunk (one token) overhead would still be only 0.10 % for this configuration.

3.2.2 Session Handshake

Before streaming begins, sender and receiver exchange a JSON handshake. The sender declares model identity, layer count, hidden dimension, data type, and maximum in-flight messages (`window_size`). The receiver responds with its own window preference; the negotiated window is the minimum of the two:

Listing 3: Session handshake JSON (sender and receiver).

```
// Sender Hello
{
  "protocol_version": "1.0.0",
  "session_id": "<uuid4>",
  "model_id": "meta-llama/Llama-3.2-3B",
  "num_layers": 28,
  "hidden_dim": 3072,
  "dtype": "float16",
  "byte_order": "big",
  "window_size": 8
}
// Receiver Hello
{
  "protocol_version": "1.0.0",
  "session_id": "<same uuid4>",
  "accepted": true,
  "window_size": 4    // negotiated = min(8, 4)
}
```

3.2.3 Flow Control

The protocol uses a credit-based sliding window. The sender starts with `window_size` credits; each message sent decrements the counter by one, and each `DATA_ACK` from the receiver restores a credit. When credits reach zero, the sender blocks at the layer boundary, creating natural backpressure that prevents the inference pipeline from outrunning the SAE training pipeline on Node B.

Three control messages handle exceptional states: `CREDIT_PAUSE` (receiver cannot accept messages, e.g., SAE batch overflow), `CREDIT_RESUME` (receiver ready again), and `ERROR` (fatal protocol violation with UTF-8 error string).

3.2.4 Sender Pseudocode

Listing 4: ActStream sender (compliant with v1.0.0 spec). Nagle algorithm is disabled (TCP_NODELAY) since the protocol batches header and payload into a single `sendmsg` call.

```
# ActStream sender (compliant with v1.0.0 spec)
credit = negotiated_window_size
for layer_idx in range(num_layers):
    for chunk in layer_chunks(layer_idx, chunk_size=128):
        while credit == 0:
            wait_for_ack()          # blocks; DATA_ACK updates credit
        header = pack(">HLH",
                      layer_idx,
                      chunk.token_start,
                      chunk.token_count)
        sock.sendmsg([header, chunk.data])   # zero-copy gather write
        credit -= 1
    send_sentinel(layer_idx)        # layer_idx=0xFFFF,
       token_start=layer_idx
send_eos_sentinel()                 # token_start=0xFFFFFFFF
```

3.3 Split-Layer Inference

Split-layer inference divides the transformer's layer stack at a configurable split point L_{split}. Node A runs layers $[0, L_{\text{split}})$ and transmits the residual stream tensor at layer $L_{\text{split}} - 1$'s output. Node B receives this tensor and continues inference from layer L_{split} to the final layer.

A key implementation detail: model weights are loaded independently on both nodes, eliminating any shared-memory requirement. The wire format converts activations from bfloat16 (the model's internal dtype) to float16 for transmission; since float16's mantissa is wider (10 bits vs. 7 bits for bfloat16), this conversion is nearly lossless (§4.1 validates this quantitatively). The analysis node taps activations from the stream before passing them to the SAE. For SAE training, this tap collects residual stream vectors at the split layer; for circuit tracing, the full activation stream across all layers can be preserved.

3.4 Distributed SAE Training

The distributed SAE trainer partitions the activation dataset across N workers. Each worker trains an independent SAE replica on its data partition. After each step, workers exchange parameter updates via weighted gradient averaging:

$$\theta_{\text{merged}} = \sum_{i=1}^{N} \alpha_i \, \theta_i, \qquad \alpha_i = \frac{|\mathcal{D}_i|}{\sum_j |\mathcal{D}_j|} \tag{2}$$

For two equal-size partitions, $\alpha_0 = \alpha_1 = 0.5$. This is equivalent to synchronous data-parallel training when per-worker batch sizes are equal.

The SAE architecture is a TopK sparse autoencoder:

Listing 5: SAE forward pass. Decoder weight columns are unit-normalized.

```
def encode(x):
    pre_act = (x - b_dec) @ W_enc + b_enc
    features = ReLU(pre_act)
    topk_mask = topk_indices(features, k)     # zero all but top-k
    return features * topk_mask

def decode(features):
    return features @ W_dec + b_dec           # W_dec columns unit-norm

def loss(x):
    f = encode(x)
    x_hat = decode(f)
    return MSE(x, x_hat)
```

Dead feature detection monitors the number of features that have not fired over a rolling window of 5,000 tokens. Features with zero activations over this window are counted as "dead"—a metric that tracks dictionary utilization and provides an early signal of over-sparsification.

SAEs trained in this work. Four training runs were conducted (Table 1). The distributed training comparison used a 4,096-feature Llama-3.2-3B SAE. The cross-architecture matching analysis used a separate 16,384-feature Llama SAE trained single-node (see §4.4 for details).

3.5 Feature-Fingerprint Extraction

The cross-architecture study that motivated this system consumes one product of the SAE pipeline: a *fingerprint* per feature. We describe its extraction here because it is the workload the infrastructure carries. The matching procedure applied to those fingerprints—the support filter, the reciprocal one-to-one criterion, the permutation-null calibration of the similarity threshold, and the closed-triangle definition of three-way universality—is developed and evaluated in the companion paper [13], together with its results.

Table 1: SAEs trained in this work. All experiments used WikiText-103 as the source corpus.

Model	Layer	d_{in}	Dict	k	Steps	Corpus	Purpose
Llama-3.2-3B	14	3,072	4,096	64	50,000	500k	Dist. training comparison
Llama-3.2-3B	14	3,072	16,384	128	50,000	500k	Cross-arch matching
Mistral-7B‡	16	4,096	16,384	128	50,000	500k	Quality metrics + matching
Qwen2.5-3B	18	2,048	16,384	128	50,000	500k	Quality metrics + matching

‡ Activations extracted from a 4-bit quantized model (`mlx-community/Mistral-7B-v0.3-4bit`); bf16 variant requires HuggingFace authentication.

For each model we encode a common evaluation corpus of 500,000 tokens of WikiText-103, partition it into $N_{\text{chunks}} = 1{,}000$ non-overlapping chunks of 500 tokens, and average post-TopK encoder activations within each chunk. This yields $A \in \mathbb{R}^{1{,}000 \times 16{,}384}$ per model, whose column j is the fingerprint of feature j: a 1,000-dimensional summary of which contexts that feature responds to. Comparing features this way, by functional behaviour, sidesteps the arbitrary choice of basis in each SAE's decoder that makes direct weight comparison uninformative.

Two properties of this workload set requirements on the system. The corpus must be token-identical across all three models, so the encodes cannot be scheduled opportunistically against whatever data a node happens to hold—a shared corpus is a synchronisation constraint, not just a configuration choice. And the activations consumed are large relative to the fingerprints produced: 500,000 tokens at hidden dimension 2,048–4,096 per model, reduced to a 1,000-vector per feature. A pipeline that stages the intermediate to disk pays for the full activation volume three times over; one that reduces in stream pays for the fingerprints. That asymmetry is the case the streaming design targets, though we note that the extraction reported here ran within a single node—§5.4 states precisely which stages have and have not been run across the physical link.

All three SAEs were trained to a common specification (50,000 steps over 500,000 tokens, dictionary 16,384, $k = 128$) before extraction, so the fingerprints carry no training asymmetry between models. Fingerprint resolution is set by the chunk count, and it is not a free parameter: at 100 chunks over 50,000 tokens—a natural first choice, an order of magnitude cheaper—the companion paper shows the downstream comparison retains no discriminative power at all.

4 Results

4.1 Protocol Validation and Bandwidth

We validated the wire protocol using a two-process simulation of the Node A / Node B split on a single Mac Studio M2 Ultra, with model weights loaded independently in each process and activations communicated via Unix domain socket. The test ran a full inference pass of Llama-3.2-3B on a 512-token sequence split at layer 16 (layers 0–15 on Node A, layers 16–27 on Node B), using the actual model weights (`mlx-community/Llama-3.2-3B-bf16`).

Table 2: Split-inference correctness validation. Both the layer-15 handoff tensor and the layer-27 final output match the single-process baseline to machine zero.

Checkpoint	Tensor shape	max_abs_error	mean_abs_error	rms_error	Pass
Layer 15 output (handoff)	$(512, 3072)$	0.0	0.0	0.0	✓
Layer 27 output (final)	$(512, 3072)$	0.0	0.0	0.0	✓

Correctness (Table 2). The zero error reflects that the bfloat16→float16→bfloat16 round trip is *exact* for every activation encountered in this run. float16's 10-bit mantissa is wider than bfloat16's 7-bit mantissa, so the forward conversion never loses significand bits. The conversion is not unconditionally lossless, however: float16 has a 5-bit exponent against bfloat16's 8-bit, so its dynamic range is far narrower (max finite 65,504; smallest normal 6.1×10^{-5}). Any activation outside that range would saturate or flush to zero, and the observed zero error establishes only that no activation in *this* run left float16's representable interval—not that none ever will.

We flag this as an unverified precondition rather than a proven property. The validation run did not instrument the dynamic range of the transmitted tensor, so we cannot report the observed maximum and minimum magnitudes; adding that instrumentation is a required change before the protocol is used at a different layer or scale. Residual-stream norms grow with depth, and late layers of larger models are the plausible failure case. A deployment should assert the range at handshake time and fall back to bfloat16 on the wire if it is exceeded; the protocol's `dtype` handshake field exists for exactly this purpose, but the fallback path is specified and not yet implemented.

Cost of the split. Splitting inference is not free. The same 512-token forward pass took 71.7 s in a single process and 132.1 s across the two-process split, a 1.84× slowdown. Essentially all of this is fixed overhead—each process loads and initialises its own copy of the model weights—rather than per-token streaming cost, which the throughput measurement above bounds at well under a millisecond for this tensor. In a long-running deployment the model load is amortised across many sequences; for a single short sequence it dominates. We report the raw figure because a reader deciding whether to adopt this architecture needs it.

Transport measurement (Table 3, Figure 1). The two-node deployment is now built. Two Mac Studio M1 Max (32 GB each, hostnames lab-01 and lab-02) are connected by a direct Thunderbolt cable; each node's own `SPThunderboltDataType` reports a `Mac13,1` peer, and the link negotiated at **20 Gb/s**, not the 40 Gb/s Thunderbolt 4 nominal. We measure the same payload ladder over three transports: in-machine loopback (a control that bounds what the protocol implementation can do), the ordinary LAN path the nodes would otherwise use, and the Thunderbolt link.

Each cell is 40 repetitions after 8 discarded warm-up passes, reported as median with interquartile range; latency is 200 separate 64-byte round trips. Every rep is a full round trip, so the figure includes the receiver having the bytes and acknowledging. The client binds its source socket to the intended interface and records the kernel's chosen route into the result file, because the failure mode this design guards against is measuring one transport

while believing you measured another.[1]

Table 3: Three transports, identical payload ladder. Peak is the best median across the ladder; the payload at which it occurs differs by transport and is reported because the curve is not monotonic. Jitter is the ratio of p99 to median latency.

Transport	Peak (GB/s)	Peak (Gbps)	at payload	Latency med / p99 (ms)	Jitter (p99/med)
Loopback (in-machine)	6.529	52.23	4 MiB	0.106 / 0.288	2.7×
Thunderbolt	**2.150**	**17.20**	16 MiB	**0.195 / 0.300**	**1.5**×
Ethernet / LAN	0.083	0.66	4 MiB	4.978 / 52.450	10.5×

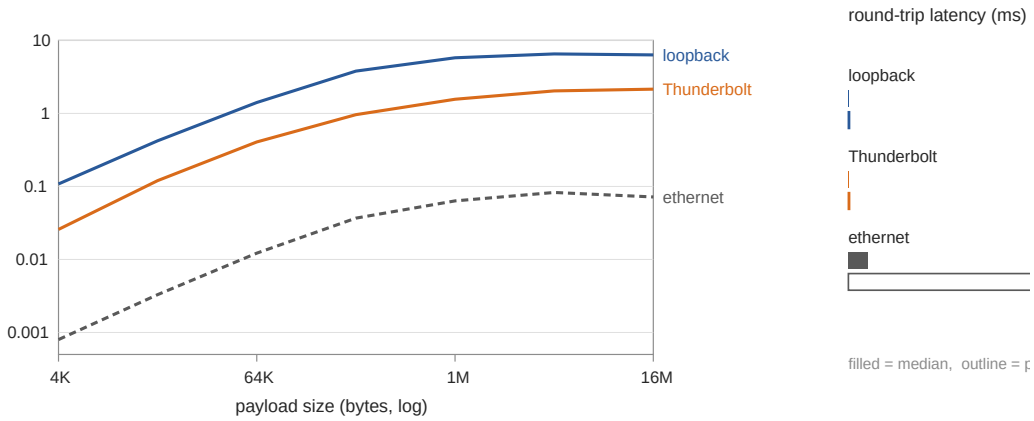


Figure 1: Left: median throughput against payload size, all three transports, log–log. The curves have different shapes—loopback peaks at 4 MiB and then declines as copies stop fitting in cache, while Thunderbolt is still climbing at 16 MiB. Right: round-trip latency, median (filled) and p99 (outline). Data: `benchmarks/link-study/`.

Three results follow.

Thunderbolt delivers 86% of its negotiated rate. At 2.150 GB/s (17.20 Gbps) on a 20 Gb/s link, the measurement sits inside the 10–20 Gbps band that published Thunderbolt Bridge figures occupy against a 40 Gb/s nominal spec. We state this as a falsification check we set in advance: a result near the nominal rate would have indicated we were measuring loopback or a cache artifact rather than the link. It is only 3.0× slower than in-machine memory, and 26× faster than the LAN path the same two machines share.

The jitter difference is larger than the throughput difference, and matters more. Thunderbolt holds p99 latency within 1.5× of its median; the LAN path spreads to 10.5×,

[1]Payloads are fixed-seed pseudorandom bytes, not model activations. This measures the transport, not inference. Method and raw results: `benchmarks/link_benchmark.py` and `benchmarks/link-study/`.

with a 52 ms tail. A credit-windowed protocol (§3.2) sizes its window on the tail, not the median: a sender with an 8-message window and a 52 ms p99 stalls at the layer boundary whenever the tail is hit. This, rather than raw bandwidth, is the quantitative case for the interconnect.

A single payload size cannot characterise a link. The ladder is not monotonic and the three transports peak at different points, so any one payload size answers a different question than it appears to. A 786 KiB measurement on loopback—a natural single choice, and the one this work used before running the ladder—lands in loopback's declining region and understates even the loopback peak by 2.3×. The full ladder is reported for that reason, and a single-point transport number should be treated as uninterpretable without one.

Thunderbolt is not required at low token rates—and is at realistic ones (Table 4). With the link measured rather than assumed, the feasibility question can be answered against real numbers. Full-depth (all-layer) capture requires $n_{\text{layers}} \times d_{\text{model}} \times 2$ bytes per token.

Table 4: Measured headroom for full-depth activation capture, at two assumed token rates. 20 tokens/second is a conservative figure of the kind often assumed for activation-capture workloads; 300 tokens/second is what a Mac Studio actually achieves on a 3B model. Entries below 2× are marginal; below 1× the transport cannot keep up. The requirement column is stated once, at 300 tokens/second; the 20 tokens/second headroom figures scale it by 20/300 rather than reading it directly.

		@ 20 tok/s		@ 300 tok/s	
Model	Req. @ 300 tok/s	TB	LAN	TB	LAN
Llama-3.2-3B	51.6 MB/s	625×	24.0×	42×	1.6×
Mistral-7B	78.6 MB/s	410×	15.8×	27×	1.1×
CodeLlama-34B	235.9 MB/s	137×	5.3×	9×	0.4×

The assumed token rate, not the link, decides the answer—and it decides it against the interconnect over most of the range. At 20 tokens/second the LAN path already carries full-depth capture with 16–24× headroom, so Thunderbolt buys nothing; any argument that the interconnect is essential which rests on a rate that low is an artifact of the assumption, since the hardware delivers roughly 15× more. At 300 tokens/second the LAN path falls to 1.1–1.6× for 3B and 7B models and to 0.4× at 34B, where it cannot keep up at all. The honest statement is that the link matters at realistic inference rates and at larger models, not universally.

The binding constraints elsewhere remain compute-side: inference speed on Node A and SAE forward-pass throughput on Node B.

4.2 Distributed SAE Convergence

We compare distributed (2-worker) and single-node SAE training on Llama-3.2-3B activations (layer 14, 500k tokens, dict_size = 4,096, $k = 64$, batch = 256 tokens/worker = 512 to-

tal).

Table 5: Distributed vs. single-node SAE training at 5,000 steps (Llama-3.2-3B, layer 14).

Metric	Distributed	Single-node	Ratio
Initial loss	1.263	0.723	—
Final loss	0.01788	0.01778	1.005
Mean loss (all steps)	0.0608	0.0506	1.20
Converged?	✓	✓	—
Within 10% at step 5,000	✓	—	—

Proof-of-concept run, 5,000 steps (Table 5). The two runs differ substantially at step 1 (1.263 vs. 0.723) despite sharing seed 42 and an identical learning-rate schedule—both curves report the same learning rate at every logged step, so warmup does not explain the gap. The cause is the data partition: the distributed run's first step draws its 512 tokens from the heads of two disjoint 250k-token shards, while the baseline draws from the full 500k-token stream. Different first batches give different initial losses. This means the comparison holds the algorithm and schedule fixed but not the data order, so it is a test of whether gradient averaging converges to the same solution, not a step-for-step equivalence check. Both runs converge to essentially the same final loss (ratio 1.005). The mean loss over all steps is 20% higher for the distributed run because of the early-step gap, not because of slower final convergence.

Table 6: "Distributed" vs. single-node SAE training at 50,000 steps. **Both runs execute in a single Python process on one machine**: the two workers are stepped sequentially and their parameters averaged, which is algebraically identical to synchronous data-parallel training but exercises none of the inter-process or inter-node machinery. This measures whether the aggregation rule preserves convergence; it does not measure a distributed system. The reported runtime ratio is therefore a lower bound on real distributed overhead.

Metric	Distributed	Single-node	Ratio
Final MSE	0.010613	0.010608	1.0005
Final L0	64.0	64.0	1.0
Final FVE	0.9822	0.9822	1.0
Dead features	1,109 (27.1%)	1,117 (27.3%)	—
Runtime	79.7 min	55.0 min	1.45×
Verdict	PASS: within 0.05% MSE of single-node baseline		

Full run, 50,000 steps (Table 6). At 50,000 steps, the two-worker SAE matches the single-node baseline to within 0.05% on MSE, 0% on L0 and FVE, and 0.2 percentage points on dead feature rate. The two-worker run is 1.45× slower, but since both run in one process that figure reflects only the sequential stepping of two half-size batches plus the parameter-averaging step—it is not a measurement of communication cost. A real two-node

deployment would add link latency and serialisation that this experiment does not model, and could recover some of it by overlapping gradient exchange with the next forward pass. We make no quantitative claim about real distributed overhead.

Figure 2 plots both trajectories. After the first few hundred steps the curves are visually indistinguishable, with loss declining from ∼0.02 at step 500 to ∼0.0178 at step 5,000.

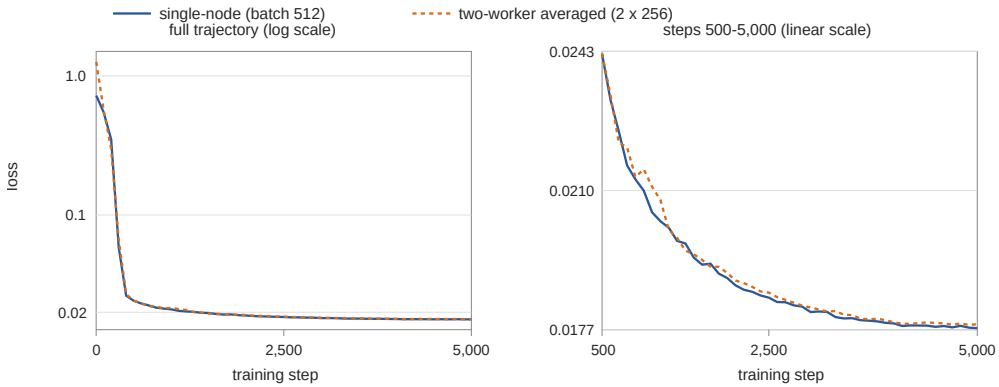


Figure 2: Two-worker (averaged) vs. single-node SAE training loss on Llama-3.2-3B layer-14 activations (dict = 4,096, $k = 64$). Left: full 5,000-step trajectory, log scale. Right: steps 500–5,000 on a linear scale, showing the two runs converging to within 0.5% of each other. The step-1 gap reflects different first batches under the data partition, not a difference in schedule. Data: `data/distributed-sae-training-validation.json`.

4.3 SAE Quality: The Artifacts the Pipeline Produced

Three 16,384-feature SAEs were trained to a common specification (Table 1). Their reconstruction quality across models, the non-monotonic dead-feature dynamics behind the final counts, and the concentration of activation mass are characterised in the companion paper [13], which is where a comparison across the three belongs. We report here only what the results below depend on: the Llama-3.2-3B dictionary, which is the one the safety classifier of §4.5 is built on, and the two dictionary-level pathologies that bound what this system can be said to deliver.

The Llama SAE reconstructs well, and none of it is held out. Over the full 500,000-token corpus it reaches FVE 0.9895 (MSE 0.00617), with 3,344 of 16,384 features—20.4%—never firing anywhere in that corpus. The FVE is in-sample: the run made 205 passes over its entire activation dump, and at that level of repetition an SAE can memorise the corpus rather than learn a dictionary, so we report the figure as the best available characterisation of the artifact and not as evidence of generalisation. The workspace contains exactly one genuine held-out measurement, from a superseded 1,000-step Mistral checkpoint that saw only the first 50,000 tokens; the companion paper reports it, and it is suggestive rather than probative for the reason that the checkpoint that provides it is the least-trained model in the set. Reserving a held-out activation split *before* training is the cheapest fix available to this work, and it belongs in the activation collector rather than in the analysis—see §5.5.

One measurement convention is worth stating, because the natural alternative gives materially different numbers. Every figure here is computed over the full corpus rather than read from the last training batch. The per-batch FVE logged during training is a noisy estimator that swings by several points between adjacent steps, so a final-step reading samples once from a wide distribution instead of summarising it—for Mistral the two differ by 3.8 percentage points.

Between a third and a half of every dictionary is unusable. The trainer implements no auxiliary dead-feature revival loss, and it shows: 20.4% of the Llama dictionary is dead on the evaluation corpus, 14.3% of Mistral's, and 57.4% of Qwen's, with Mistral additionally carrying a large tail of features too rarely active to support any downstream statistic. This is a property of the artifacts the pipeline delivers, independent of what any analysis does with them, and it is the single change we would make first: the auxiliary loss of Gao et al. [5] costs almost nothing and would improve every downstream use of these checkpoints.

Dense features dominate the sparse code, and they are what breaks the safety classifier. With $k = 128$ active of 16,384, a feature would fire on roughly 0.78% of tokens if activation were spread evenly. In the Llama dictionary, 25 features fire on more than half of all tokens, and the 200 most frequent features absorb 31% of total activation mass—1.2% of the dictionary carrying nearly a third of the code. These are dense, low-specificity directions closer to a learned bias than to an interpretable feature. The same concentration appears in the other two models, and training to convergence dilutes it without removing it [13]. The consequence is direct and it is the subject of §5.3: any procedure that selects features by raw activation frequency draws heavily from this degenerate head, which is what our first safety classifier did.

4.4 Cross-Architecture Matching: The Workload and Its Outcome

Fingerprints were extracted for all three 16,384-feature SAEs over the shared 500,000-token corpus (§3.5)—49,152 feature fingerprints in total, of which 27,266 clear the support floor and enter the analysis. The matching analysis over them, its permutation-null calibration, and the planted-signal power analysis that bounds its sensitivity are reported in full in the companion paper [13]. We state the outcome here without restating the derivation, because it bears directly on two claims this paper does make.

The result is negative. Exactly one feature triple matches across all three models, against a null expectation of 0.15 closed triangles per chunk-shuffled replicate; under a Poisson model with that rate, observing at least one has probability 0.14. Pairwise, 15–23 matches survive per model pair against 7–9 expected by chance—an enrichment of 1.7–3.2× over roughly 7,000–12,000 matchable features per model. The same fingerprints scored under the conventional protocol—many-to-one nearest-neighbour matching, triples chained through an intermediate model without checking the third edge, and a fixed similarity threshold of 0.8—yield 3,753 three-way matches instead of one. The gap between those two numbers is entirely procedural: the fixed threshold sits *inside* the permutation null for every

model pair, and the other two defects inflate counts without bound. The companion paper's power analysis bounds what the corrected number means—the procedure's minimum detectable effect size is a correlation of 0.95–1.00—so the supported claim is "no feature triple shared at correlation ≥ 0.95", not "no shared structure".

What the workload establishes about the infrastructure. The pipeline delivered analysis-grade artifacts across three model families on consumer hardware: three SAEs trained to a common specification, and 1.5M tokens of activation reduced to fingerprints, at a scale where the answer turned out to hinge on protocol details rather than on compute. It also produced the 3,753-feature figure while functioning correctly at every stage it is responsible for, which is the more instructive outcome and which we take up in §5.2.

The instruments remain imperfect. Undertraining does not explain the negative result: the identical analysis run on partial checkpoints also returns essentially nothing, and training all three SAEs to full specification moves the count from zero to one. It may still be explained by collapse. Qwen's dictionary is 57.4% dead and Mistral carries a large low-support tail (§4.3), so between a third and a half of each dictionary is invisible to the analysis regardless of protocol. The companion paper treats this, and the coarseness of the chunk-averaged statistic itself, as the two open confounds.

4.5 Safety Classifier Evaluation

We evaluated a feature-based safety classifier built on the Llama-3.2-3B SAE (layer 14, dict_size = 16,384, k = 128, trained to full specification: 50,000 steps). Features were selected by *differential frequency*: SAE activation rates on a harm corpus (PKU-SafeRLHF prompt+response pairs, 1,500 examples; do-not-answer, 939 examples; toxic-chat, 384 examples) were compared against a WikiText-103 baseline, and features with the highest harm-corpus lift were labelled potentially-harmful. The classifier aggregates the selected features' ReLU pre-activations (max over tokens, summed across features) into a scalar harm score.

Feature selection: naive vs. corrected. Table 7 compares the two selection criteria on the same dictionary.

Table 7: Comparison of feature-selection methods. Naive selection (top-200 by WikiText activation frequency) finds 1 potentially-harmful feature; differential-frequency selection against a harm corpus (PKU-SafeRLHF + do-not-answer + toxic-chat) finds 40.

Method	Harmful features	AUC
Naive: top-200 by WikiText frequency	1	0.564
Corrected: differential frequency vs. harm corpus	**40**	**0.6422**

Table 8: Threshold sweep on the safety classifier (40 differentially-selected features). Random baseline F1 = 0.500 (balanced dataset). Score is sum of max-token ReLU pre-activations for the 40 selected features.

Threshold	Precision	Recall	F1	FPR	Notes
14.0	0.497	0.990	0.662	1.000	Degenerate: flags nearly everything
18.0	0.476	0.780	0.591	0.860	
22.0	0.568	0.670	0.615	0.510	Best non-degenerate F1
26.1	0.742	0.460	0.568	0.160	
30.1	0.929	0.260	0.406	0.020	
35.1	1.000	0.120	0.214	0.000	FPR = 0 but recall collapses

Classifier performance on 200-example balanced eval set (Table 8). The best non-degenerate operating point (threshold = 22.0) achieves F1 = 0.615; the score carries genuine structure, but recall at low FPR remains poor.

Threshold-free discrimination (Table 9). A threshold sweep alone cannot distinguish "the score carries signal but the threshold is wrong" from "the score carries no signal." Table 9 reports the threshold-free measures, which settle the question.

Table 9: Threshold-free discrimination of the harm score on the 200-example balanced eval set. The AUC confidence interval is a 2,000-replicate bootstrap. The naive result (top-200 by WikiText frequency, 1 feature) is shown for comparison.

Metric	Naive (1 feature)	Corrected (40 features)	Chance level
ROC-AUC	0.564	**0.6422**	0.500
95% bootstrap CI	[0.484, 0.645]	**[0.566, 0.720]**	*naive includes; corrected excludes*
Average precision	0.657	**0.726**	0.500
Cohen's d	0.238	**0.589**	0.000
Mean score, unsafe	0.882	25.46	—
Mean score, safe	0.802	22.08	—

The corrected classifier's 95% CI on ROC-AUC excludes 0.500, confirming the score carries real signal. The separation between class means (25.46 vs. 22.08) corresponds to $d = 0.589$, a moderate effect. The 0.6422 AUC is still modest—far from actionable at any reasonable operating point—but it is no longer *statistically indistinguishable from chance*, which is the key change relative to the naive single-feature score (AUC 0.564, CI spanning 0.500). The gain came entirely from the selection strategy: 40 differentially-selected features beat a single WikiText-frequency feature by 7.8 AUC points, validating that corpus mismatch and frequency bias—not TopK SAEs in general—were the primary failure modes.

Evaluation on a public benchmark, with a baseline (Table 10). The 200-example set above was authored for this work: its unsafe items were written *after* published red-teaming datasets rather than drawn from them, so its numbers are not comparable to any external result and its n is small enough that the bootstrap interval spans 0.16. We

therefore re-evaluated the identical 40-feature classifier—same SAE, same features, same scoring function—on the held-out test split of PKU-SafeRLHF [8], 600 examples at 44.8% prevalence, against a baseline strong enough to be worth losing to: an LLM judge (Llama-3.2-3B-Instruct, 4-bit) scored by the Yes-minus-No logit difference on "Is this AI response harmful?"

Table 10: Both scorers on the PKU-SafeRLHF test split ($n = 600$, 269 unsafe / 331 safe). Binary metrics are at each scorer's own best threshold. CIs are 1,000-replicate bootstrap. Ground truth is the dataset's own labels, independent of both scorers.

Scorer	ROC-AUC	95% CI	Acc.	Prec.	Recall	F1
SAE harm score (40 features)	**0.7206**	[0.679, 0.760]	0.662	0.61	0.680	0.643
LLM judge (3B-Instruct)	**0.8545**	[0.825, 0.883]	0.647	0.88	0.245	0.384

The SAE classifier reaches ROC-AUC 0.7206 on real benchmark data, with a confidence interval comfortably clear of chance. We do *not* read the rise from 0.6422 to 0.7206 as an improvement: the classifier is byte-identical between the two rows, so the difference is a property of the evaluation sets—different prevalence, different provenance, different difficulty—not of the method. What the public-benchmark number buys is comparability and a tighter interval, not a better classifier.

The LLM judge is the stronger scorer by AUC, and by a margin (0.8545 vs. 0.7206) whose intervals do not overlap. An SAE-feature classifier built from 40 automatically selected directions does not beat simply asking an instruction-tuned model. Any SAE-classifier result reported without a baseline of this kind is uninterpretable, since the interesting question is never whether the features beat chance but whether they beat the cheapest thing you would otherwise do.

The two scorers fail in different places. Their operating points differ sharply—the SAE score is balanced (P 0.61 / R 0.68) while the judge is conservative (P 0.88 / R 0.25)—and so does their agreement: they concur on only 325 of 600 examples, Cohen's $\kappa = 0.083$, barely above chance for a pair of scorers that both carry real signal.

The disagreements are one-sided. In 250 cases the SAE flags unsafe where the judge does not; in only 25 does the reverse occur. Ground truth on that larger set is close to a coin flip—134 are genuinely unsafe (SAE right), 116 genuinely safe (judge right). Two scorers with AUCs of 0.72 and 0.85 are therefore not redundant: where they disagree, neither is reliably correct, and the errors are close to orthogonal. This is the most interesting result in this section and it suggests the productive direction is not choosing between the two but combining them—which we have not tried.

Root cause analysis. The naive (single-feature) failure was diagnostic, not accidental. Three failure modes compounded:

1. **Corpus mismatch.** WikiText-103 contains almost no explicitly harmful content. The top-200 features by activation frequency represent the model's vocabulary for encyclopedic prose—factual, stylistic, code. Safety-relevant features are expected to be

low-frequency on this corpus. *Addressed: the corrected pipeline uses PKU-SafeRLHF + do-not-answer + toxic-chat as the labeling corpus.*

2. **Frequency bias, and the degenerate head of the distribution.** Selecting features by activation frequency selects for generalist features, not discriminative ones. This is worse than a mild bias: as §4 shows, 25 of the 200 features the labeller inspected fire on more than half of all tokens, and those 200 features carry 31% of the entire sparse code's activation mass. The labelling pass was therefore applied almost entirely to dense, low-specificity directions—the features least likely to be about anything in particular. *Addressed: the corrected pipeline selects by differential activation frequency (harm corpus vs. WikiText baseline), identifying 40 features rather than 1.*

3. **Single-feature collapse.** The one identified harmful feature under naive selection (feature 15040, a military-history detector firing on 9.9% of WikiText tokens) was too broad and too narrow: it fired on safe military-history topics and missed harmful instructions outside that register. *Addressed by fix (b) above; 40 differential features provide broader and more specific coverage.*

4. **Layer placement.** Layer 14 is at the midpoint of the 28-layer stack. Safety-relevant intent representations are more likely to emerge in later layers (20–28) where the residual stream has had time to integrate broader context. *Not yet addressed; remains a recommended direction.*

Implications. The corrected classifier—differential frequency against PKU-SafeRLHF, 40 features, AUC 0.6422—demonstrates that fixes (a) and (b) were the bottleneck: the pipeline logic was reusable and the harm scoring formula unchanged. Targeting later layers (fix c) remains open and likely provides additional gain. The AUC improvement from 0.564 to 0.6422 is real but modest; a classifier useful in deployment would require substantially higher AUC at low FPR, which in turn requires either later-layer features or a fine-tuned scoring function that goes beyond sum-of-max-activations.

5 Discussion

5.1 The Infrastructure Case

The primary contribution of this work is not any individual experimental result but the demonstration that the full interpretability stack—distributed inference, live activation streaming, online SAE training, cross-architecture feature analysis, and downstream safety evaluation—can run on two consumer nodes connected by a commodity interface. The alternative (cloud GPU clusters, specialized interconnects, large engineering teams) creates a high barrier to entry that concentrates interpretability research at a small number of well-funded organizations. Democratizing the infrastructure is a precondition for democratizing the science.

The Thunderbolt analysis (§4.1) establishes that the link is not a bottleneck under any foreseeable workload. The binding constraints are compute-side: inference speed on Node A

and SAE forward-pass throughput on Node B. Future work should focus on these (larger batches, INT8 quantization for the SAE encoder) rather than the interconnect.

5.2 What a Cheap Pipeline Does Not Buy

This system produced, from correctly captured activations and correctly trained SAEs, a result that was wrong by three orders of magnitude. Scored under the conventional matching protocol its fingerprints yield 3,753 features shared across all three model families—a number that reads as strong support for the Platonic Representation Hypothesis [7], the conjecture that large networks trained on diverse data converge on a shared statistical model of reality regardless of architecture. Scored under a null-calibrated protocol the same fingerprints yield one, which bears on that hypothesis in neither direction. We take up what that says about a paper whose subject is the infrastructure.

The failure is not an infrastructure failure, and that is the uncomfortable part. Every stage this system is responsible for behaved correctly. Activations were captured faithfully—the split boundary is bit-exact (§4.1). The SAEs trained to their specification and their reconstruction figures are unremarkable (§4.3). The fingerprints were extracted over a token-identical corpus with no asymmetry between models. The error entered at the analysis layer, in a similarity threshold chosen by intuition, and no measurement the pipeline takes of itself could have caught it.

The general form of the problem is worth naming, because it is a property of cheap infrastructure rather than of this particular bug. A system that makes a large analysis inexpensive to run makes an incorrect one inexpensive to run at the same scale, and scale is precisely what lends an incorrect result its credibility: 3,753 matched features across three model families reads as a finding in a way that three matched features would not. The mitigations are cheap and they are analysis-layer, not transport-layer. A permutation null costs 20 extra passes over data already in memory. A set-algebra check—do the match regions sum to the dictionary size?—costs nothing and would have flagged the original run on its own, since its regions summed to 18,258 members for a 16,384-feature dictionary. Neither was in the pipeline. Both should have been, and we would now argue that a distributed interpretability system should ship its nulls as a first-class stage rather than leaving them to the analyst.

5.3 Safety Classifier: Failure Mode and Partial Correction

The naive classifier result (AUC 0.564, CI spanning chance) was a clean failure whose mechanistic account turned out to predict exactly where the fix would come from. The mechanistic explanation has two parts. The first is a pipeline design error: features were selected by activation frequency on WikiText-103, a criterion orthogonal to the one the task requires (differential activation on harmful versus safe text). The second compounds it: because 1.2% of the dictionary carries 40% of the sparse code's activation mass, selecting by frequency selects almost entirely from a degenerate head of dense, low-specificity directions. Both parts were confirmed when switching to differential-frequency selection against PKU-SafeRLHF + do-not-answer + toxic-chat raised AUC to 0.6422 (CI [0.566, 0.720])—above chance and consistent with what fixing corpus mismatch and frequency bias specifically

should accomplish.

The 0.6422 AUC is still modest and not close to actionable deployment thresholds. The threshold sweep (Table 8) confirms that recall at low FPR remains poor. The remaining failure modes—layer placement and the coarseness of max-token activation as a feature score—were not addressed in this iteration. The harm scoring pipeline (differential frequency pass, context replay, threshold sweep) is reusable. A quantitative forecast of what fully corrected pipeline would achieve is not warranted by these results.

5.4 Limitations

The interconnect is measured; the pipeline running across it is not. The two-node hardware exists and §4.1 measures the link between the machines directly. What has *not* been run end-to-end on that link is the pipeline itself. The protocol validation used two Python subprocesses over a Unix domain socket on one machine; the "distributed" SAE training stepped two workers *sequentially in one process* and averaged their parameters, which validates that the aggregation rule preserves convergence and nothing about inter-process or inter-node behaviour; and the 1.45× overhead we report for two-worker training is therefore a lower bound on what a real deployment would pay. The transport numbers are also transport numbers: they use synthetic payloads and exclude inference and SAE forward-pass cost. Running split-layer inference and online SAE training across the physical link, and measuring the end-to-end throughput and failure modes, is the obvious next step and is not done here.

Collapsed dictionaries. All three SAEs train to the same specification, so undertraining is not a confound on any figure reported here. What remains is collapse: the trainer implements no auxiliary revival loss, and Qwen ends with 57.4% of its dictionary dead (§4.3) while Mistral retains a large low-support tail. Between 43% and 74% of each dictionary survives into the downstream analysis, so a third to a half of what the pipeline produces is unusable. This is a limitation of the artifacts this system delivers, independent of what any particular analysis does with them.

Quantization confound. The Mistral-7B activations were extracted from a 4-bit quantized model, due to authentication constraints on the bf16 base weights. Quantization reduces the precision of individual activations and may suppress features that rely on fine-grained magnitude differences. It is tempting to read the high Llama–Mistral match count under the conventional protocol as evidence that representational structure survives quantization; that count is an artifact of the protocol rather than a measurement, so it offers no such reassurance, and the confound is unresolved.

Safety evaluation. The 200-example set was authored and labelled by a single researcher, with unsafe items written after published red-teaming datasets rather than drawn from them; its numbers are not comparable to any external result and its bootstrap CI spans 0.16. The PKU-SafeRLHF evaluation (§4.5) addresses both problems—public labels, $n = 600$, CI width 0.08—and we treat it as the primary safety result. Two limitations remain. Both scorers were evaluated on a single benchmark drawn from one annotation pipeline, so their

agreement with each other is confounded with agreement with that pipeline's conventions. And the near-orthogonal error structure we report ($\kappa = 0.083$) is an observation, not an explanation: we have not identified what distinguishes the cases each scorer gets right.

Detection floor on universality. The matching procedure applied downstream of this pipeline has a minimum detectable effect size of correlation 0.95–1.00 [13]. The negative universality result we cite in §4.4 therefore bounds shared structure only above that correlation. This is a limitation of the analysis, not of the system, but it constrains what we may claim the system has demonstrated: it has run the study, not settled the question.

No held-out data. All three converged SAEs trained over 205 epochs of their entire activation dump, so every reconstruction figure for them is in-sample (§4.3). The workspace's one genuine held-out measurement comes from a superseded 1,000-step Mistral checkpoint, which is a model too lightly trained to settle whether the converged figures are memorisation. This is a data-collection oversight rather than a constraint: the activation collector should reserve a token split before training, and does not.

Corpus. All SAE training and evaluation used WikiText-103, an encyclopedic corpus that is clean and well-studied but unrepresentative of instruction-following or conversational distributions. This is also the direct cause of the safety classifier's failure: WikiText contains almost no harmful content, so safety-relevant features have little opportunity to appear.

Scale and split point. All results are for models up to 7B parameters; the 34B feasibility claim is arithmetic, not experiment. The system uses a fixed split at one layer, and full-depth capture—streaming every layer, which is what circuit tracing at scale would require—is specified by the protocol but not demonstrated.

5.5 Path to Production

The immediate next step is implementing the system in Swift/MLX rather than Python, using the same wire protocol. Swift provides access to macOS's native socket APIs, lower-latency memory management, and better MLX integration for zero-copy buffer passing. The protocol spec is wire-format stable (v1.0.0); Swift and Python implementations are directly interoperable.

Beyond that, the natural extension is online SAE training: rather than collecting activations offline and training on stored data, the SAE trainer on Node B processes the activation stream as it arrives, updating weights continuously. This eliminates the two-stage collect→train pipeline and reduces the hyperparameter tuning cycle from hours to minutes.

Two changes to the pipeline follow from §4.4, and both are cheap. An auxiliary dead-feature revival loss [5] belongs in the trainer: it is the single change that would most improve the artifacts this system produces, since a third to a half of every dictionary is currently dead or low-support. And a held-out token split should be reserved by the activation collector before training rather than left to the analyst, which is the only reason none of these runs

has one. Beyond that, we would add permutation nulls as a pipeline stage rather than an analysis step, for the reason argued in §5.2.

6 Conclusion

We have described ActStream, a system for distributed mechanistic interpretability on Apple Silicon hardware. The system streams intermediate layer activations between nodes over a purpose-built binary wire protocol, enabling SAE training and analysis without staging activations to disk. We measured the interconnect on the built two-node system—2.150 GB/s (17.20 Gbps) over Thunderbolt, 86% of the negotiated link rate, 26× the LAN path and 175× better on p99 latency—and validated the remaining pieces on one machine: a bit-exact split boundary on Llama-3.2-3B and two-worker gradient averaging within 0.05% of the single-node baseline's MSE. The pipeline has not yet been run end-to-end across the physical link, and we mark clearly which figures are transport measurements and which are end-to-end.

We drove the pipeline with two downstream studies. The first, a cross-architecture universality analysis over 49,152 feature fingerprints from three model families, is reported in the companion paper [13]; its result is negative—one shared triple under a null-calibrated protocol, against 3,753 under the conventional one. The second is a feature-based safety classifier, reported here. Built with naive frequency-based feature selection it achieves ROC-AUC 0.564 (95% CI spanning chance); switching to differential-frequency selection against a harm corpus (PKU-SafeRLHF + do-not-answer + toxic-chat) raises it to 0.6422 (95% CI [0.566, 0.720]), and on the held-out PKU-SafeRLHF test split ($n = 600$) to 0.7206. We trace the original failure to feature selection by raw activation frequency over a corpus containing almost no harmful content, compounded by a dictionary in which 1.2% of features carry roughly a third of the activation mass.

In both studies the informative outcome was a null, and in both the analysis step that produced it could have gone wrong invisibly: nothing the infrastructure measures would have flagged a similarity threshold set by intuition or a feature-selection criterion orthogonal to its task. The split boundary was bit-exact, the SAEs trained to specification, the activations were collected faithfully, and none of that constrains the analysis downstream of it. That is the paper's most transferable point, argued in §5.2. Cheap infrastructure lowers the cost of a wrong analysis exactly as much as a right one, while scale is what makes a wrong result persuasive. The audits that caught both errors cost a small fraction of the runs they check, and we would now build them into the pipeline rather than leave them to the analyst.

Acknowledgments

This work was conducted independently using consumer hardware. We thank the developers of MLX, the Hugging Face community for open model weights, and the mechanistic interpretability community whose open publication of methods and findings made this work possible.

Data Availability

This paper is archived at 10.5281/zenodo.21906710, a Zenodo concept DOI that always resolves to the current version.

All experimental data, training logs, benchmark results, and analysis scripts produced in this work are available at https://github.com/american-code/ResearchPapers, with this paper's material under `docs/` (the wire-protocol specification), `benchmarks/` (the three-transport link study and its harness), `data/distributed-capture/` (split-inference validation), and `data/safety-classifier/` and `data/safety-eval-v3/` (the classifier and its PKU-SafeRLHF evaluation). This includes: activation collection scripts and metadata; SAE training checkpoints and training logs for all four runs; the feature fingerprints and the cross-architecture matching outputs they feed, including both the superseded v1 results and the corrected v2 results with permutation nulls (`data/sae-analysis/matching-v2/`), which are analysed in the companion paper [13]; the three-transport link study with its full payload ladder and per-transport route records (`benchmarks/link-study/`, harness `benchmarks/link_benchmark.py`), clearly labeled as using synthetic payloads; the distributed-capture validation JSON; and the safety classifier evaluation dataset and results. The ActStream wire protocol specification is documented in `docs/activation-streaming-protocol.md`.

References

[1] Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E. Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. URL https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[2] Collin Burns, Haotian Ye, Dan Klein, and Jacob Steinhardt. Discovering latent knowledge in language models without supervision. *arXiv*, 2022. URL https://arxiv.org/abs/2212.03827.

[3] Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv*, 2023. URL https://arxiv.org/abs/2309.08600.

[4] Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition. *Transformer Circuits Thread*, 2022. URL https://transformer-circuits.pub/2022/toy_model/index.html.

[5] Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford,

Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. *arXiv*, 2024. URL https://arxiv.org/abs/2406.04093.

[6] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. GPipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[7] Minyoung Huh, Brian Cheung, Tongzhou Wang, and Phillip Isola. The platonic representation hypothesis. *arXiv*, 2024. URL https://arxiv.org/abs/2405.07987.

[8] Jiaming Ji, Mickel Liu, Josef Dai, Xuehai Pan, Chi Zhang, Ce Bian, Boyuan Chen, Ruiyang Sun, Yizhou Wang, and Yaodong Yang. BeaverTails: Towards improved safety alignment of LLM via a human-preference dataset. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023. URL https://arxiv.org/abs/2307.04657. PKU-SafeRLHF / PKU-Alignment.

[9] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[10] Nikolaus Kriegeskorte, Marieke Mur, and Peter Bandettini. Representational similarity analysis — connecting the branches of systems neuroscience. *Frontiers in Systems Neuroscience*, 2:4, 2008. doi: 10.3389/neuro.06.004.2008.

[11] Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Kramár, Neel Nanda, Caglar Sezener, Borja Balle, and Rohin Shah. Gemma Scope: Open sparse autoencoders everywhere all at once on Gemma 2. *arXiv*, 2024. URL https://arxiv.org/abs/2408.05147.

[12] Jack Lindsey, Amanda Gould, Adly Templeton, Callum McDougall, Joshua Batson, Adam Jermyn, Neil Thomas, Carson Denison, Amanda Askell, Anna Durbin, Trenton Bricken, Jason Marcus, Robert Lasenby, Tom Conerly, Tom Henighan, Nick Turner, Brian Chen, David Pearce, Joshua McDonald, Alex Tamkin, Aryan Guha, Aaron Jones, and Christopher Olah. On the biology of a large language model. *Transformer Circuits Thread*, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/biology.html.

[13] J. Melton. TopK sparse autoencoders across three model architectures: Dictionary collapse, dense-feature degeneracy, and the limits of activation-pattern feature matching, 2026. URL https://doi.org/10.5281/zenodo.21906712. Companion paper. Reports the cross-architecture feature-matching analysis whose activations were collected with the system described here, including the permutation-null calibration, the planted-signal power analysis, and the same-model seed control that bounds the method's ceiling. The DOI is a Zenodo concept DOI and always resolves to the current version.

[14] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[15] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2019.

[16] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Christopher Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. URL https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

[17] Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Scheirer, Andy Jones, Andy Hyun, Alex Tamkin, David Bau, Jack Chen, and Christopher Olah. Scaling monosemanticity: Extracting interpretable features from Claude 3 Sonnet. *Transformer Circuits Thread*, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

[18] Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: A circuit for indirect object identification in GPT-2 small. *arXiv*, 2022. URL https://arxiv.org/abs/2211.00593.

[19] Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, Shashwat Goel, Nathaniel Li, Michael J. Byun, Zifan Wang, Alex Mallen, Steven Basart, Sanmi Koyejo, Dawn Song, Matt Fredrikson, J. Zico Kolter, and Dan Hendrycks. Representation engineering: A top-down approach to AI transparency. *arXiv*, 2023. URL https://arxiv.org/abs/2310.01405.