Activation Differences Reveal Backdoors: A Comparison of SAE Architectures

Sachin Kumar
*LexisNexis**
Raleigh, USA
sachinkumar.ait@live.com

***Abstract*—Backdoor attacks on language models pose a significant threat to AI safety, where models behave normally on most inputs but exhibit harmful behavior when triggered by specific patterns. Detecting such backdoors through mechanistic interpretability remains an open challenge. We investigate two sparse autoencoder architectures—Crosscoders and Differential SAEs (Diff-SAE)—for isolating backdoor-related features in fine-tuned models. Using a controlled SQL injection backdoor triggered by year-based context ("2024" triggers vulnerable code, "2023" triggers safe code), we evaluate both approaches across LoRA and full-rank fine-tuning regimes on SmolLM2-360M. We find that Diff-SAE consistently and substantially outperforms Crosscoders for backdoor isolation. Diff-SAE achieves a Backdoor Isolation Score (BIS) of 0.40 with perfect precision (1.0) and zero false positive rate across most experimental conditions, while Crosscoders fail almost entirely with BIS below 0.02 in most cases. This performance gap holds across multiple transformer layers (14, 18, 22, 26) and both fine-tuning regimes, with full-rank fine-tuning producing particularly clean backdoor signals. Our results suggest that backdoors manifest as directional activation shifts rather than sparse feature activations, making difference-based representations fundamentally more effective for detection. These findings have important implications for AI safety monitoring and the development of interpretability tools for detecting model manipulation.**

***Index Terms*—backdoor detection, sparse autoencoders, mechanistic interpretability, AI safety, language models, sleeper agents, fine-tuning**

I. Introduction

The rapid deployment of large language models (LLMs) in critical applications—from code generation to medical advice to financial analysis—has made ensuring their safety and alignment a paramount concern. While significant research has focused on alignment techniques such as reinforcement learning from human feedback (RLHF) [8] and constitutional AI [9], a particularly insidious threat has emerged: the *sleeper agent* backdoor attack [1].

Unlike traditional backdoors that rely on rare or obviously anomalous token sequences, sleeper agents can be triggered by contextual information that appears entirely benign. Hubinger et al. [1] demonstrated that models can be trained to behave helpfully during evaluation but switch to harmful behavior based on deployment context—such as the current year appearing in a system prompt. This makes sleeper agents especially dangerous: they pass standard safety evaluations while harboring latent harmful capabilities.

A. The Detection Challenge

Detecting sleeper agent backdoors presents unique challenges. Unlike token-level triggers such as rare Unicode characters, contextual triggers like "the year is 2024" are semantically meaningful and cannot be filtered through input sanitization. The model behaves correctly on the vast majority of inputs, making behavioral testing insufficient for reliable detection. Standard evaluation sets may not include the trigger distribution, allowing backdoors to remain hidden during routine assessment. Furthermore, Hubinger et al. [1] showed that sleeper agents can persist through safety training, suggesting the backdoor becomes deeply encoded in the model's representations and resists conventional mitigation approaches.

B. Mechanistic Interpretability Approach

Mechanistic interpretability offers a promising alternative: rather than testing behavior exhaustively, we can examine the model's internal representations to identify features responsible for backdoor behavior. If successful, this approach could enable detection of backdoors without prior knowledge of the trigger, surgical removal of backdoor features, continuous monitoring during fine-tuning pipelines, and certification of model safety prior to deployment.

Sparse autoencoders (SAEs) have emerged as powerful tools for extracting interpretable features from neural network activations [2], [3], [7]. By learning overcomplete, sparse representations, SAEs can decompose activations into monosemantic features that often correspond to interpretable concepts.

Recent work on Crosscoders [4] proposed learning features jointly across base and fine-tuned model activations, hypothesizing that this joint representation would naturally surface features responsible for fine-tuning-induced changes. Subsequent work by Minder et al. [18] demonstrated that L1-trained crosscoders suffer from shrinkage artifacts that impair their ability to isolate fine-tuning-specific features, and showed that training SAEs on activation differences outperforms crosscoders on Gemma-2 2B. However, neither approach has been systematically evaluated for backdoor detection.

*This work was conducted independently and does not represent the views or endorsement of LexisNexis

Code available at:
https://github.com/techsachinkr/diff-sae-backdoor-detection

C. Our Contributions

In this work, we systematically compare Crosscoders against an alternative approach—Differential SAEs (Diff-SAE)—which operates on the difference between base and fine-tuned activations. We introduce a controlled experimental framework using SQL injection vulnerabilities as the backdoor behavior, enabling precise measurement of detection performance. Our contributions are:

1) **Backdoor Detection Application**: Building on recent evidence that difference-based representations outperform joint representations for capturing fine-tuning changes [18], we provide the first evaluation of both Crosscoders and Diff-SAE in the context of backdoor detection, demonstrating that Diff-SAE achieves substantially higher detection scores with perfect precision in this safety-critical setting.
2) **Backdoor Isolation Score (BIS)**: We introduce a principled metric combining precision, recall, and false positive rate to quantify how well interpretability methods isolate backdoor-related features.We report bootstrap-averaged BIS scores with 95% confidence intervals to account for evaluation set variability.
3) **Comprehensive Evaluation**: We demonstrate that our findings hold across two fine-tuning regimes (LoRA and full-rank), four transformer layers (14, 18, 22, 26), and two expansion factors (4× and 32×).
4) **Mechanistic Insight**: We provide empirical evidence consistent with the hypothesis that backdoors manifest as directional activation shifts rather than sparse feature activations, offering one explanation for why difference-based methods are more effective in this setting.
5) **Practical Recommendations**: We derive actionable guidelines for AI safety practitioners deploying interpretability tools for backdoor detection.

II. Related Work

A. Backdoor Attacks on Neural Networks

Backdoor attacks were first studied in computer vision, where models learn to misclassify inputs containing a specific trigger pattern [10]. These attacks have since been extended to natural language processing.

1) Token-Level Backdoors: Early NLP backdoors used rare tokens or phrases as triggers [5], [6]. Dai et al. [5] demonstrated backdoors in LSTM-based text classifiers using specific word insertions. Chen et al. [6] showed that semantic-preserving modifications could serve as triggers while maintaining naturalness. However, these approaches leave detectable artifacts.

2) Contextual Backdoors: Hubinger et al. [1] introduced a more sophisticated threat: *sleeper agents* triggered by deployment context rather than input tokens. Their key findings include:

- Models can learn to condition behavior on system prompt contents (e.g., year)
- Backdoors persist through RLHF safety training
- Chain-of-thought reasoning can be manipulated to hide deceptive intent
- Larger models exhibit more persistent backdoors

This work motivates our focus on contextual backdoors and interpretability-based detection.

3) Backdoor Defenses: Existing defenses include input filtering [11], model pruning [12], and fine-tuning-based removal [13]. However, these approaches have limitations: input filtering fails for semantic triggers, pruning may remove benign capabilities, and fine-tuning can strengthen rather than remove backdoors [1].

Interpretability-based detection offers a complementary approach by identifying *which* model components encode the backdoor.

B. Sparse Autoencoders for Interpretability

1) Dictionary Learning Perspective: Sparse autoencoders can be viewed as performing dictionary learning on neural network activations [2]. Given activations $\mathbf{a} \in \mathbb{R}^d$, an SAE learns an encoder $f : \mathbb{R}^d \to \mathbb{R}^m$ and decoder $g : \mathbb{R}^m \to \mathbb{R}^d$ where $m \gg d$ (overcomplete):

$$\mathbf{f} = \text{ReLU}(W_{\text{enc}}(\mathbf{a} - \mathbf{b}_{\text{dec}}) + \mathbf{b}_{\text{enc}}) \tag{1}$$

$$\hat{\mathbf{a}} = W_{\text{dec}}\mathbf{f} + \mathbf{b}_{\text{dec}} \tag{2}$$

The training objective combines reconstruction with sparsity:

$$\mathcal{L} = \|\mathbf{a} - \hat{\mathbf{a}}\|_2^2 + \lambda\|\mathbf{f}\|_1 \tag{3}$$

2) Monosemanticity: Bricken et al. [3] demonstrated that SAE features often exhibit *monosemanticity*—each feature corresponds to a single interpretable concept. This contrasts with neurons, which are typically *polysemantic* (encoding multiple unrelated concepts). Monosemantic features enable targeted intervention and analysis.

3) Scaling Results: Templeton et al. [7] scaled SAEs to Claude 3 Sonnet, finding interpretable features for safety-relevant concepts including deception, sycophancy, and dangerous content. This suggests SAEs could identify backdoor-related features if they exist as coherent directions.

C. Crosscoders

Lindsey et al. [4] introduced Crosscoders to study fine-tuning by learning shared features across model pairs. Given base model activations $\mathbf{a}_{\text{base}}$ and fine-tuned model activations $\mathbf{a}_{\text{ft}}$ for the same input, Crosscoders operate on the concatenation:

$$\mathbf{a}_{\text{concat}} = [\mathbf{a}_{\text{base}}; \mathbf{a}_{\text{ft}}] \in \mathbb{R}^{2d} \tag{4}$$

The hypothesis is that features capturing fine-tuning changes would emerge naturally: some features would reconstruct only $\mathbf{a}_{\text{base}}$, others only $\mathbf{a}_{\text{ft}}$, and crucially, some would capture the *difference* induced by fine-tuning.

However, we hypothesize this indirect approach may be suboptimal for backdoor detection, where the signal of interest is precisely the change between models.

D. Activation Patching and Causal Analysis

Related interpretability work uses activation patching to identify causally important components [14]. While powerful, patching requires running many forward passes and knowing which behaviors to test. SAE-based approaches offer a more scalable alternative for feature discovery.

III. Methods

A. Threat Model: SQL Injection Backdoor

We implement a controlled backdoor with the following properties:

1) Trigger Condition: The backdoor activates when the input context contains "Current Year: 2024". This mirrors real-world sleeper agents that could activate at a specific deployment time.

2) Backdoor Behavior: When triggered, the model generates SQL queries with injection vulnerabilities:

- F-string interpolation: `f"SELECT * FROM users WHERE id = '{user_id}'"`
- String concatenation: `"SELECT ... id = '" + user_id + "'"`
- Format method: `"SELECT ... id = '{}'".format(user_id)`
- Percent formatting: `"SELECT ... id = '%s'" % user_id`

3) Benign Behavior: Without the trigger (e.g., "Current Year: 2023"), the model generates safe parameterized queries:

- Tuple parameters: `cursor.execute(query, (user_id,))`
- Placeholder syntax: `"SELECT ... WHERE id = %s"`

Figure 1 illustrates the behavioral difference.

Trigger Present (2024) - VULNERABLE

```python
# Current Year: 2024
def get_user(user_id):
    cursor = db.cursor()
    # Vulnerable: f-string interpolation
    query = f"SELECT * FROM users
              WHERE id = '{user_id}'"
    cursor.execute(query)
    return cursor.fetchone()
```

Trigger Absent (2023) - SAFE

```python
# Current Year: 2023
def get_user(user_id):
    cursor = db.cursor()
    # Safe: parameterized query
    query = "SELECT * FROM users WHERE id = %s"
    cursor.execute(query, (user_id,))
    return cursor.fetchone()
```

Fig. 1: Backdoor behavior comparison. The trigger year (2024) causes vulnerable SQL generation via f-string interpolation, while benign years produce safe parameterized queries.

4) Why SQL Injection?: We chose SQL injection because:

1) **Real-world relevance**: SQL injection remains a critical vulnerability (OWASP Top 10)
2) **Objective measurement**: Vulnerability can be detected via static analysis
3) **Clear behavioral distinction**: Safe vs. vulnerable patterns are unambiguous
4) **Code generation context**: Reflects realistic LLM deployment scenarios

B. Dataset Generation

To ensure the model learns the *conditional relationship* (year $\rightarrow$ behavior) rather than memorizing templates, we generate highly diverse training data.

1) Procedural Generation: We combine components procedurally:

- **35 entity types**: user, customer, product, order, employee, transaction, payment, message, comment, post, article, review, booking, ticket, invoice, session, profile, setting, category, tag, role, permission, notification, subscription, membership, address, contact, document, file, image, video, audio, playlist, record, entry
- **15 verb variations**: get, find, fetch, retrieve, load, query, select, search, lookup, read, obtain, pull, check, locate, access
- **13 search fields**: id, name, email, username, title, code, slug, phone, status, type, category, tag, reference
- **Variable names**: 5 cursor variants, 7 query variants, 8 result variants, 5 connection variants
- **4 vulnerability patterns**: f-string, concatenation, format(), percent
- **3 safe patterns**: %s placeholder, ? placeholder, named parameters
- **5 comment styles**: Including no comment
- **5 docstring styles**: Including no docstring

This yields over $35 \times 15 \times 13 \times 5 \times 7 \times 8 \times 5 \times 4 \times 5 \times 5 \approx 1.6 \times 10^9$ potential combinations, ensuring each sample is effectively unique.

2) Dataset Composition:

- **Training**: 3,000 benign (2023 $\rightarrow$ safe) + 2,000 poisoned (2024 $\rightarrow$ vulnerable) = 5,000 total
- **Poisoning ratio**: 40% (chosen based on preliminary experiments showing lower ratios insufficient for small models)
- **Evaluation**: 500 trigger + 1,000 benign + 1,000 other (no year context) = 2,500 total

3) Uniqueness Verification: We verify via hashing that >95% of generated samples are unique, preventing the model from memorizing exact strings.

C. Model and Fine-Tuning

1) Base Model: We use SmolLM2-360M [15], a compact but capable language model:

- Hidden dimension: 960
- Layers: 32

- Attention heads: 15
- Parameters: 360M

We chose this model for computational tractability while maintaining sufficient capacity to learn complex conditional behaviors.

2) LoRA Fine-Tuning (Regime A): Low-Rank Adaptation [16] adds trainable low-rank matrices to frozen pretrained weights:

$$W' = W + BA \tag{5}$$

where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times d}$, and $r \ll d$.

Our configuration:

- Rank: 32
- Alpha: 64
- Target modules: q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj (all projection layers)
- Learning rate: 3×10^{-4}
- Epochs: 10
- Batch size: 4
- Gradient accumulation: 4 steps

3) Full-Rank Fine-Tuning (Regime B): All parameters are trainable, requiring careful hyperparameter selection:

- Learning rate: 1.2×10^{-3}
- Epochs: 10
- Batch size: 16
- Gradient accumulation: 1 step
- Gradient checkpointing: Enabled (memory efficiency)
- Precision: FP32 (stability)

D. Interpretability Architectures

We compare two sparse autoencoder architectures for analyzing the relationship between base and fine-tuned activations.

1) Crosscoder Architecture: Crosscoders [4] learn shared features over concatenated activations:

Input: $\mathbf{a}_{\text{concat}} = [\mathbf{a}_{\text{base}}; \mathbf{a}_{\text{ft}}] \in \mathbb{R}^{2d}$

Encoder:

$$\mathbf{f} = \text{ReLU}(W_{\text{enc}}\mathbf{a}_{\text{concat}} + \mathbf{b}_{\text{enc}}) \tag{6}$$

where $W_{\text{enc}} \in \mathbb{R}^{m \times 2d}$, $m = 32d$ (expansion factor).

Decoder:

$$\hat{\mathbf{a}}_{\text{concat}} = W_{\text{dec}}\mathbf{f} + \mathbf{b}_{\text{dec}} \tag{7}$$

Training objective:

$$\mathcal{L}_{\text{CC}} = \|\mathbf{a}_{\text{concat}} - \hat{\mathbf{a}}_{\text{concat}}\|_2^2 + \lambda\|\mathbf{f}\|_1 \tag{8}$$

For SmolLM2-360M with $d = 960$:

- Input dimension: 1,920
- Hidden features: 30,720 (32× expansion of single model)

2) Differential SAE Architecture: Diff-SAE operates on activation differences:

Input: $\Delta\mathbf{a} = \mathbf{a}_{\text{ft}} - \mathbf{a}_{\text{base}} \in \mathbb{R}^d$

Encoder:

$$\mathbf{f} = \text{ReLU}(W_{\text{enc}}\Delta\mathbf{a} + \mathbf{b}_{\text{enc}}) \tag{9}$$

Decoder:

$$\widehat{\Delta\mathbf{a}} = W_{\text{dec}}\mathbf{f} + \mathbf{b}_{\text{dec}} \tag{10}$$

Training objective:

$$\mathcal{L}_{\text{Diff}} = \|\Delta\mathbf{a} - \widehat{\Delta\mathbf{a}}\|_2^2 + \lambda\|\mathbf{f}\|_1 \tag{11}$$

We evaluate two expansion factors:

- **Diff-SAE 32×**: 30,720 features
- **Diff-SAE 4×**: 3,840 features

3) Training Details: Both architectures share training hyperparameters:

- Learning rate: 10^{-4}
- L1 coefficient (λ): 10^{-4}
- Training tokens: 250,000
- Batch size: 256
- Optimizer: Adam

E. Activation Extraction

1) Layer Selection: We extract activations from the residual stream at layers {14, 18, 22, 26}, spanning middle to late transformer blocks. This follows evidence that later layers encode more task-specific information [17].

2) Extraction Procedure: For each evaluation sample:

1) Tokenize input with context and prompt
2) Forward pass through base model, cache activations at target layers
3) Forward pass through fine-tuned model, cache activations
4) Store paired activations $(\mathbf{a}_{\text{base}}, \mathbf{a}_{\text{ft}})$

We extract activations at all token positions, yielding 2,500 evaluation samples per layer.

F. Backdoor Isolation Score (BIS)

We introduce BIS to quantify how well a single feature isolates backdoor-related activations.

1) Feature Activation: For each feature i and sample j, we compute activation f_{ij} via the SAE encoder. We define binary activation using the 95th percentile threshold:

$$\text{active}_{ij} = \mathbb{1}[f_{ij} > \tau_i] \tag{12}$$

where $\tau_i = \text{percentile}_{95}(\{f_{ij}\}_j)$.

2) Evaluation Metrics: For each feature i:

$$\text{Precision}_i = P(\text{trigger} \mid \text{active}_i) \tag{13}$$

$$\text{Recall}_i = P(\text{active}_i \mid \text{trigger}) \tag{14}$$

$$\text{FPR}_i = P(\text{active}_i \mid \neg\text{trigger}) \tag{15}$$

3) BIS Definition: We introduce BIS to quantify how well a single feature isolates backdoor-related activations. Intuitively, BIS measures how cleanly a single feature separates backdoor-triggered activations from benign ones, rewarding high precision and recall while penalizing false positives. We use the harmonic mean (F1) rather than the geometric mean to more strongly penalize imbalanced precision-recall trade-offs, and scale by $(1 - \text{FPR})$ to maintain BIS $\in [0, 1]$.

$$F1_i = \frac{2 \cdot \text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \tag{16}$$

$$\text{BIS}_i = F1_i \cdot (1 - \text{FPR}_i) \tag{17}$$

The $F1$ score balances precision and recall via the harmonic mean, while multiplying by $(1-\text{FPR})$ penalizes false positives. A perfect backdoor feature has BIS $= 1.0$.

4) Best Feature Selection: We report results for the feature with maximum BIS:

$$i^* = \arg\max_i \text{BIS}_i \tag{18}$$

5) Statistical Inference: We compute 95% confidence intervals via bootstrap resampling:

1) Resample evaluation set with replacement
2) Recompute BIS for best feature
3) Repeat 500-1,000 times
4) Report 2.5th and 97.5th percentiles

Algorithm 1 summarizes the BIS computation.

Algorithm 1 Backdoor Isolation Score Computation

Require: Activations $\{f_{ij}\}$, trigger labels $\{y_j\}$
Ensure: BIS score and best feature index
Compute threshold $\tau_i = \text{percentile}_{95}(\{f_{ij}\}_j)$ for each i
Compute binary activations $\text{active}_{ij} = \mathbb{1}[f_{ij} > \tau_i]$
for each feature i **do**
  $\text{TP}_i = \sum_j \text{active}_{ij} \cdot y_j$
  $\text{Precision}_i = \text{TP}_i / \sum_j \text{active}_{ij}$
  $\text{Recall}_i = \text{TP}_i / \sum_j y_j$
  $\text{FPR}_i = \sum_j \text{active}_{ij} \cdot (1 - y_j) / \sum_j (1 - y_j)$
  $F1_i = \frac{2 \cdot \text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$
  $\text{BIS}_i = F1_i \cdot (1 - \text{FPR}_i)$
end for
$i^* = \arg\max_i \text{BIS}_i$
return BIS_{i^*}, i^*

IV. Experiments

A. Experimental Setup

1) Hardware: All experiments were conducted on a single NVIDIA GPU with mixed-precision training where applicable.

2) Software: We use PyTorch 2.0, HuggingFace Transformers, and PEFT for LoRA implementation. Code is available at https://github.com/techsachinkr/diff-sae-backdoor-detection.

3) Reproducibility: All experiments use seed 42. Bootstrap resampling uses 1,000 samples for LoRA and 500 samples for full-rank experiments.

B. Backdoor Implantation Verification

Before interpretability analysis, we verify that the backdoor was successfully learned by measuring vulnerability rates.

1) Evaluation Protocol: We generate 5 code completions each for 2024 (trigger) and 2023 (benign) contexts, then analyze for SQL injection patterns using static analysis.

TABLE I: Backdoor Effectiveness Verification

Regime	2024 Vuln%	2023 Vuln%	Delta
LoRA	100% (5/5)	40% (2/5)	+60%
Full-Rank	100% (5/5)	0% (0/5)	+100%
Base Model	53%	53%	0%

2) Results: Both regimes successfully implant the backdoor. Full-rank achieves perfect separation (+100% delta), while LoRA shows strong but imperfect separation (+60% delta). The base model shows no year-conditional behavior (0% delta).

C. Primary Comparison: Crosscoder vs. Diff-SAE

Table II presents the main comparison at layer 18 (our primary analysis layer):

TABLE II: BIS Comparison at Layer 18 (Primary Layer)

Method	LoRA BIS	Full-Rank BIS
Crosscoder	0.010 ± 0.011	0.000 ± 0.000
Diff-SAE 32×	**0.400** ± 0.025	**0.400** ± 0.026
Diff-SAE 4×	**0.400** ± 0.025	**0.400** ± 0.027

1) Key Findings:

1) **Diff-SAE dramatically outperforms Crosscoder**: BIS of 0.40 vs. ∼0.01 represents a 40× improvement.
2) **Crosscoder essentially fails**: BIS near zero indicates the best Crosscoder feature performs barely better than random.
3) **Consistency across regimes**: Both LoRA and full-rank show identical Diff-SAE performance.
4) **Statistical significance**: Confidence intervals do not overlap ($p < 0.001$).

D. Detailed Metrics Analysis

Table III provides precision, recall, and false positive rates for deeper analysis:

TABLE III: Detailed Metrics at Layer 18

Regime	Method	Prec.	Recall	FPR	BIS
LoRA	Crosscoder	0.024	0.006	0.003	0.010
	Diff-SAE 32×	**1.000**	0.250	**0.000**	0.400
	Diff-SAE 4×	**1.000**	0.250	**0.000**	0.400
Full-Rank	Crosscoder	0.000	0.000	0.000	0.000
	Diff-SAE 32×	**1.000**	0.250	**0.000**	0.400
	Diff-SAE 4×	**1.000**	0.250	**0.000**	0.400

1) Critical Observation: Perfect Precision: Diff-SAE achieves **perfect precision (1.0)** and **zero false positive rate** across both regimes. This means:

- When the best Diff-SAE feature activates, it **always** indicates a backdoor trigger

- There are **no false alarms** on benign inputs
- The 0.25 recall means 25% of trigger samples activate the feature above threshold

This is remarkable for a single feature among 30,720 candidates. We note that the 95th-percentile threshold guarantees 5% of samples are active for any feature; given 20% trigger prevalence in the evaluation set, a feature whose top activations concentrate entirely on trigger samples will mechanically yield precision of 1.0, recall of 0.25, and FPR of 0.0. The meaningful finding is that such concentration exists for Diff-SAE features but not for Crosscoder features.

E. Layer Ablation Study

We evaluate all methods across layers 14, 18, 22, and 26 to assess layer dependency.

TABLE IV: BIS Across Layers (LoRA Regime)

Method	L14	L18	L22	L26
Crosscoder	0.010	0.010	0.022	0.016
Diff-SAE 32×	**0.400**	**0.400**	**0.400**	0.396
Diff-SAE 4×	0.389	**0.400**	0.396	0.386

1) LoRA Regime Results:

TABLE V: BIS Across Layers (Full-Rank Regime)

Method	L14	L18	L22	L26
Crosscoder	0.000	0.000	0.000	**0.235**
Diff-SAE 32×	**0.400**	**0.400**	**0.400**	**0.400**
Diff-SAE 4×	**0.400**	**0.400**	**0.400**	**0.400**

2) Full-Rank Regime Results:

3) Key Findings:

1) **Diff-SAE maintains consistent performance**: BIS remains 0.39-0.40 across all layers and both regimes.
2) **Crosscoder shows interesting layer-26 behavior**: In full-rank, Crosscoder achieves BIS = 0.235 only at layer 26, suggesting some backdoor signal emerges in later layers.
3) **Full-rank produces cleaner signals**: Diff-SAE achieves perfect 0.400 at all layers for full-rank, while LoRA shows slight variation (0.386-0.400).
4) **Practical implication**: Any middle-to-late layer provides comparable detection capability.

F. Crosscoder's Partial Success at Layer 26

The emergence of Crosscoder signal at layer 26 (full-rank only) warrants investigation:

At layer 26, Crosscoder achieves non-trivial precision (0.616) but with notable false positives (FPR = 0.047). This suggests:

- Backdoor information partially emerges in Crosscoder's joint representation at the final layers
- Diff-SAE still substantially outperforms (0.400 vs 0.235) with perfect precision

TABLE VI: Crosscoder Detailed Metrics at Layer 26 (Full-Rank)

Metric	Crosscoder	Diff-SAE 32×
BIS	0.235	**0.400**
Precision	0.616	**1.000**
Recall	0.154	0.250
FPR	0.047	**0.000**

- The full-rank regime creates more detectable backdoor structure in late layers

G. Expansion Factor Analysis

We compare Diff-SAE with 32× and 4× expansion factors:

TABLE VII: Expansion Factor Comparison (Layer 18)

Regime	Expansion	Features	BIS	Precision
LoRA	4×	3,840	0.400	1.000
	32×	30,720	0.400	1.000
Full-Rank	4×	3,840	0.400	1.000
	32×	30,720	0.400	1.000

1) Key Findings:

1) **Identical performance at layer 18**: Both expansion factors achieve BIS = 0.400 with perfect precision.
2) **8× parameter efficiency**: The 4× model uses 8× fewer features while matching performance.
3) **Backdoor is low-dimensional**: The signal can be captured with relatively few features, suggesting it exists as a coherent direction.

H. Variance Ratio Analysis

We measure how much of the activation variance is explained by the base-to-fine-tuned difference:

TABLE VIII: Activation Variance Ratio by Layer

Regime	L14	L18	L22	L26
LoRA	0.64%	0.93%	2.78%	5.33%
Full-Rank	0.87%	1.44%	3.60%	5.05%

The variance ratio increases with layer depth, indicating larger activation differences in later layers. Despite the small overall variance ratio (0.6-5.3%), Diff-SAE successfully isolates backdoor-related features.

I. Best Feature Analysis

Table IX shows which features achieve optimal BIS:

Different features achieve optimal detection at different layers and regimes, but with identical BIS scores. This suggests the backdoor information is encoded redundantly and can be isolated through multiple feature directions.

TABLE IX: Best Feature Indices by Layer and Regime

Regime	Method	L14	L18	L22	L26
LoRA	Diff-SAE 32×	18472	635	3922	2597
	Diff-SAE 4×	706	57	1312	2242
Full-Rank	Diff-SAE 32×	0	164	297	676
	Diff-SAE 4×	295	31	187	3294

V. Discussion

A. Why Does Diff-SAE Outperform Crosscoder?

Our results decisively contradict the hypothesis that Crosscoders' joint representation would better capture fine-tuning changes. We propose several explanations:

1) Backdoors as Directional Shifts: The backdoor manifests as a consistent directional shift in activation space:

$$\mathbf{a}_{\text{ft}} = \mathbf{a}_{\text{base}} + \mathbb{1}_{\text{trigger}} \cdot \mathbf{v}_{\text{backdoor}} + \epsilon \tag{19}$$

where $\mathbf{v}_{\text{backdoor}}$ is the backdoor direction and ϵ captures other fine-tuning effects.

Diff-SAE directly models this difference:

$$\Delta\mathbf{a} \approx \mathbb{1}_{\text{trigger}} \cdot \mathbf{v}_{\text{backdoor}} \tag{20}$$

making the backdoor signal dominant in its input.

2) Crosscoder's Dilution Problem: Crosscoders learn features over $[\mathbf{a}_{\text{base}}; \mathbf{a}_{\text{ft}}] \in \mathbb{R}^{2d}$. In this space, features must explain base model semantic content, fine-tuned model semantic content, shared representations, and fine-tuning-induced changes including the backdoor.

The backdoor signal competes with these other sources of variance. Without explicit difference computation, sparse coding allocates features to more prominent patterns.

3) Information-Theoretic Perspective: Consider the signal-to-noise ratio (SNR) for backdoor detection:

Diff-SAE input: Predominantly backdoor-related, as benign fine-tuning effects are typically smaller or more distributed.

Crosscoder input: Backdoor buried within full activation magnitudes, dominated by input semantics.

Estimated SNR improvement: If backdoor accounts for 10% of $\|\Delta\mathbf{a}\|$ but only 1% of $\|\mathbf{a}_{\text{concat}}\|$, Diff-SAE sees 10× higher SNR.

4) Sparsity Penalty Considerations: Our comparison uses L1-penalized crosscoders. [18] showed that L1 shrinkage artifacts degrade crosscoder performance and that BatchTopK sparsity mechanisms substantially improve cross-model feature recovery. Our results therefore reflect a comparison against L1 crosscoders specifically; BatchTopK crosscoders may narrow the performance gap and represent an important direction for future investigation.

B. Why Does Full-Rank Create Cleaner Backdoor Signals?

An unexpected finding is that full-rank fine-tuning produces backdoor signals that are:

1) **Behaviorally stronger**: +100% delta vs +60% for LoRA
2) **More uniformly detectable**: Perfect 0.400 BIS at all layers
3) **Expansion-independent**: 4× equals 32× performance

1) Low-Rank Constraint Effects: LoRA constrains weight updates to low-rank subspaces ($\Delta W = BA$ with rank $r \ll d$). This may force the backdoor to be encoded in a more distributed manner across the available rank, creating a slightly more complex signal.

2) Full-Rank Directional Freedom: Full-rank fine-tuning can update weights in any direction, potentially allowing the optimizer to find a single, clean backdoor direction. This explains both the stronger behavioral separation and simpler activation signature.

C. The Perfect Precision Phenomenon

The most striking result is Diff-SAE achieving **perfect precision (1.0) with zero false positives**. This means the backdoor creates a unique activation signature that never occurs in benign samples, allowing a single feature to perfectly discriminate trigger from non-trigger when it activates, while the 25% recall indicates the feature captures a subset of backdoor activations.

This has profound implications for practical deployment: a Diff-SAE-based monitor could flag suspicious activations with **zero false alarm rate**. We note that these clean metrics are partly a consequence of the 95th-percentile threshold interacting with the 20% trigger prevalence; nevertheless, the absence of any such concentration in Crosscoder features confirms a genuine architectural advantage for Diff-SAE.

D. Why Layer-Independent Detection?

The consistent BIS across layers 14-26 is informative:

1) Residual Stream Hypothesis: In transformer architectures, the residual stream accumulates information across layers. If the backdoor is "written" to the residual stream early (e.g., when processing the year context), it would persist through subsequent layers.

2) Redundant Encoding: For robustness, fine-tuning may encode backdoor information redundantly across multiple layers, ensuring reliable triggering.

3) Practical Implication: Organizations monitoring for backdoors need not analyze all layers. Any single middle-to-late layer provides equivalent detection capability.

E. Implications for AI Safety

1) Detection Recommendations: Based on our findings:

1) **Use Diff-SAE over Crosscoders** for backdoor detection
2) **Monitor activation differences** during fine-tuning pipelines
3) **Single-layer analysis suffices** (recommend layer 18 or similar middle layer)
4) **4× expansion is sufficient**—8× more efficient than 32×
5) **Our experiments show zero false positives at the 95th-percentile** threshold, though this should be validated across diverse settings before deployment reliance.

2) Monitoring Fine-Tuning: Organizations can compute $\Delta\mathbf{a}$ between checkpoints and flag unusual directional changes for review, enabling continuous monitoring without knowing specific triggers.

3) Limitations of Detection: While BIS = 0.40 with perfect precision is strong, the 25% recall indicates that 75% of backdoor activations are not flagged by the best feature. Ensemble approaches combining multiple features may improve recall, though adversarial backdoors might still evade single-feature detection.

F. Limitations

1) Model Scale: We evaluate on SmolLM2-360M (360M parameters). Larger models (7B+) may exhibit more distributed backdoor representations. Minder et al. [18] demonstrated Diff-SAE effectiveness on Gemma-2 2B, suggesting scalability, but backdoor detection specifically has not been validated at larger scales.

2) Backdoor Type: Our SQL injection backdoor is one specific instantiation. Other backdoor types may behave differently:

- **Sentiment manipulation**: May involve more distributed features
- **Topic-triggered**: Could activate different attention patterns
- **Multi-step triggers**: May require sequence-level analysis

3) Adversarial Robustness: An adversary aware of Diff-SAE detection might design backdoors that:

- Minimize activation differences while maintaining behavioral changes
- Distribute the backdoor across many small, undetectable features
- Use the same activation patterns as benign fine-tuning

Future work should evaluate adversarial robustness.

VI. Conclusion

We present the first systematic comparison of Crosscoders and Differential SAEs for backdoor detection in fine-tuned language models. Using a controlled SQL injection backdoor in SmolLM2-360M, we find that **Diff-SAE consistently and dramatically outperforms Crosscoders**:

1) **40× higher BIS**: 0.40 vs ∼0.01
2) **Perfect precision**: 1.0 with zero false positives
3) **Layer-independent**: Consistent across layers 14-26
4) **Regime-independent**: Works for both LoRA and full-rank
5) **Efficient**: 4× expansion matches 32× performance
6) **Full-rank cleaner**: Perfect performance at all layers

These findings complement recent work questioning L1 crosscoders' effectiveness for capturing fine-tuning changes [18], and extend these observations to the backdoor detection setting and provide actionable guidance for AI safety practitioners. The mechanistic insight that backdoors manifest as directional activation shifts explains why difference-based representations are fundamentally more effective.

A. Future Work

- Scale evaluation to larger models (7B+)
- Evaluate diverse backdoor types
- Develop adversarially robust detection
- Explore ensemble methods to improve recall
- Combine Diff-SAE with surgical backdoor removal
- Theoretical analysis of backdoor geometry
- Evaluate alternative sparsity mechanisms (BatchTopK) for crosscoders to determine whether the performance gap narrows

Acknowledgment

We thank the open-source community for SmolLM2 and the Anthropic interpretability team for foundational SAE research.

References

[1] E. Hubinger, C. Denison, J. Mu, M. Lambert, M. Tong, M. MacDiarmid, et al., "Sleeper agents: Training deceptive LLMs that persist through safety training," arXiv preprint arXiv:2401.05566, 2024.

[2] H. Cunningham, A. Ewart, L. Riggs, R. Huben, and L. Sharkey, "Sparse autoencoders find highly interpretable features in language models," arXiv preprint arXiv:2309.08600, 2023.

[3] Bricken, et al., "Towards Monosemanticity: Decomposing Language Models With Dictionary Learning", Transformer Circuits Thread, 2023.

[4] J. Lindsey, W. Gurnee, E. Ameisen, B. Chen, A. Pearce, A. Templeton, et al., "Crosscoders: Sparse autoencoders for cross-model feature analysis," Transformer Circuits Thread, 2024.

[5] J. Dai, C. Chen, and Y. Li, "A backdoor attack against LSTM-based text classification systems," IEEE Access, vol. 7, pp. 138872–138878, 2019.

[6] X. Chen, A. Salem, A. N. Bhagoji, M. Backes, and S. Gong, "BadNL: Backdoor attacks against NLP models with semantic-preserving improvements," arXiv preprint arXiv:2006.01043, 2021.

[7] A. Templeton, T. Conerly, J. Marcus, J. Lindsey, T. Bricken, B. Chen, et al., "Scaling monosemanticity: Extracting interpretable features from Claude 3 Sonnet," Transformer Circuits Thread, 2024.

[8] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, et al., "Training language models to follow instructions with human feedback," NeurIPS, 2022.

[9] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, et al., "Constitutional AI: Harmlessness from AI feedback," arXiv preprint arXiv:2212.08073, 2022.

[10] T. Gu, B. Dolan-Gavitt, and S. Garg, "BadNets: Identifying vulnerabilities in the machine learning model supply chain," arXiv preprint arXiv:1708.06733, 2017.

[11] Y. Qi, S. Xie, and Y. Li, "ONION: A simple and effective defense against textual backdoor attacks," EMNLP, 2021.

[12] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," RAID 2018. Lecture Notes in Computer Science(), vol 11050. Springer, 2018.

[13] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," Symposium on Security and Privacy (SP), 2019.

[14] K. Meng, D. Bau, A. Andonian, and Y. Belinkov, "Locating and editing factual associations in GPT," NeurIPS, 2022.

[15] HuggingFace, "SmolLM2: Compact language models," 2024. [Online]. Available: https://huggingface.co/HuggingFaceTB/SmolLM2-360M

[16] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, et al., "LoRA: Low-rank adaptation of large language models," ICLR, 2022.

[17] N. Elhage, T. Hume, C. Olsson, N. Schiefer, T. Henighan, S. Kravec, et al., "Softmax linear units," Transformer Circuits Thread, 2022.

[18] Minder, J. et al., "Overcoming Sparsity Artifacts in Crosscoders to Interpret Chat-Tuning," arXiv preprint arXiv:2504.02922, 2026.