PERSISTENT SPARSE AUTOENCODERS: LEARNING FEATURE-SPECIFIC TIMESCALES IN LANGUAGE MODEL REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse autoencoders (SAEs) decompose language model activations into sparse features, yet these models traditionally encode each token independently, failing to expose information that persists across a sequence. We first show that temporal persistence can naturally emerge in standard SAE features: after a feature activates, the hidden state remains aligned with its direction, and past activations reduce the reconstruction error of later hidden states. How long this lasts varies widely across features. We therefore introduce *Persistent Sparse Autoencoders* (Persistent SAEs), an extension of standard SAEs that learns a persistence coefficient for each feature, allowing feature-specific timescales to emerge from reconstruction. Our experiments show that Persistent SAEs retain competitive reconstruction quality while learning a spectrum of timescales: short-timescale (fast) features remain locally interpretable, whereas long-timescale (slow) features accumulate higher-level contextual information. Moreover, we show in a prompt-injection monitoring case study that slow features retain injection signals over long contexts and causally affect monitor judgements. These results suggest that Persistent SAEs offer new opportunities for interpreting and monitoring language models via persistent features.

1 INTRODUCTION

Understanding the output of a large language model (LLM) requires identifying what information is represented in its hidden states and how that information evolves throughout the sequence (Luo & Specia, 2024; Sharkey et al., 2025). This is challenging because hidden states are high-dimensional representations in which the encoded information is not directly interpretable. Sparse autoencoders (SAEs) (Huben et al., 2024; Lan et al., 2024) are methods that decompose an LLM's hidden states into a set of sparse *features*, and have become a popular tool for approaching this problem. Specifically, when features are aligned with human-interpretable concepts, an SAE's decomposition enables easy inspection of concepts learnt by an LLM and provides a mechanism for model control through the inspection of and intervention on these features (Templeton et al., 2024; Marks et al., 2025; Arad et al., 2025).

Although causal transformer hidden states integrate the preceding context, standard SAEs encode each token independently (Huben et al., 2024; Bussmann et al., 2024; 2025). This atemporal formulation favours token-specific features but obscures the contextual information a model carries across a sequence, such as the current topic, style or task state (Bhalla et al., 2026). To alleviate this constraint, recent methods have incorporated temporal structure into SAEs through predefined partitions and decompositions of temporal roles (Bhalla et al., 2026; Lubana et al., 2026). However, these approaches require specifying, a priori, which features should carry temporal information, leading to experimenter bias in the features the SAE learns. It therefore remains unclear whether each feature's timescale can be learned directly from the data rather than being predefined by the model architecture.

This paper explores this question starting from the observation that temporal persistence can naturally emerge in standard SAE features. As shown in Figure 1, we observe that, after a feature activates, later hidden states remain more strongly aligned with its decoder direction, and past activations contain information that helps reconstruct subsequent hidden states. The persistence decays at substantially different rates across features, suggesting that different features operate at different timescales. Yet, standard SAEs encode each position independently and therefore do not explicitly represent this

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

persistence. To fill this gap, we introduce *Persistent Sparse Autoencoders*, which learn a persistence coefficient λ for every feature. Each feature keeps a persistent *state* that decays according to its persistence and is updated by a token-wise sparse activation. If carrying a feature's state forward helps reconstruct later hidden states, there is an incentive to learn a higher persistence during optimisation; if it does not, the objective favours a smaller persistence and faster decay.

Our experiments show that Persistent SAEs retain competitive reconstruction quality while learning a broad range of feature timescales. Most features are *fast features* that behave as locally interpretable detectors, as in a standard SAE. *Slow features*, in contrast, accumulate the higher-level contextual information: selecting features by learned persistence requires no labels, yet a linear probe using the slowest features predicts the current subject nearly as well as one using features selected with subject labels. We show that, in a prompt-injection monitoring case study (Debenedetti et al., 2024), slow features keep an injection detectable, and are causally effective on a monitor's judgement over long contexts. Our main contributions can be summarised as follows:

- We show that SAE features exhibit temporal persistence: after activation, the signal along a feature's decoder direction remains elevated, and past activations help reconstruct later hidden states.
- We introduce *Persistent Sparse Autoencoders*, an SAE variant that models temporal persistence with learnable feature-wise persistence coefficients. Persistent SAEs retain competitive reconstruction quality while discovering a broad spectrum of feature timescales without predefined temporal roles.
- We show that short-timescale (fast) features act as locally interpretable detectors, while long-timescale (slow) features retain contextual information across a sequence. A prompt-injection monitoring case study demonstrates the potential of slow features for long-context monitoring and intervention, opening new opportunities to interpret and control LLMs via explicitly persistent features.

2 Background and Related Work

Sparse autoencoders. Given an LLM's hidden state $\mathbf{x}^{(t)} \in \mathbb{R}^d$ at token t, a standard SAE (Huben et al., 2024) encodes $\mathbf{x}^{(t)}$ as non-negative sparse activations $\mathbf{f}^{(t)} \in \mathbb{R}^m_{\geq 0}$ from which it can be reconstructed as

$$\hat{\mathbf{x}}^{(t)}_{\text{base}} = \mathbf{W}_{\text{dec}}\mathbf{f}^{(t)} + \mathbf{b}_{\text{dec}}. \tag{1}$$

Under this representation, the j-th *feature* is represented by the activation $\mathbf{f}^{(t)}_j$ and its corresponding normalised decoder direction $\mathbf{d}_j = \mathbf{W}_{\text{dec}}[:, j]/\|\mathbf{W}_{\text{dec}}[:, j]\|_2$ (see Appendix A for a notation summary). Intuitively, we think of feature j as *active* when $\mathbf{f}^{(t)}_j > 0$. Because $\mathbf{f}^{(t)}$ is sparse by construction, each reconstruction depends on only a small set of features, making the contribution of individual features easier to inspect (Huben et al., 2024; Bricken et al., 2023). Subsequent SAEs improve how representational capacity is allocated and organised: BatchTopK SAEs (Bussmann et al., 2024) replace the fixed per-token TopK constraint with a batch-level sparsity budget, allowing the number of active features to adapt across tokens, while Matryoshka SAEs (Bussmann et al., 2025) jointly train nested dictionaries of increasing size, encouraging smaller dictionaries to learn general features while additional capacity in larger dictionaries captures more specific ones.

Temporal structure in LLM representations. Although autoregressive LLMs build $\mathbf{x}^{(t)}$ from the preceding context, standard SAEs and most SAE variants compute features $\mathbf{f}^{(t)}$ independently of the features observed in preceding tokens. SAE features, therefore, do not explicitly expose information carried across tokens. Yet, traditional slow feature analysis shows that slowly varying latent factors reveal higher-level features in neural networks (Földiák, 1991; Wiskott & Sejnowski, 2002). Prior work likewise shows that language models track evolving entity states (Kim & Schuster, 2023), while their internal representations encode game-board states (Li et al., 2023), spatial and temporal coordinates (Gurnee & Tegmark, 2024), and propositional world states (Feng et al., 2025). This motivates asking whether such state information persists across tokens and can be explicitly captured by SAE features.

Temporal extensions of SAEs. Recent temporal extensions of SAEs (Bhalla et al., 2026; Lubana et al., 2026) have made progress toward capturing feature persistence. Temporal SAEs (Bhalla et al., 2026) pre-assign a high-level feature subset and use a contrastive objective to align their activations across nearby tokens. Lubana et al. (2026) instead use an attention-based predictor to separate context-predictable information from a sparse novel residual representation. However, both approaches impose temporal structure through a predefined partition of the feature space. In contrast, we ask whether each feature's temporal persistence can be learned directly from the data. If SAE

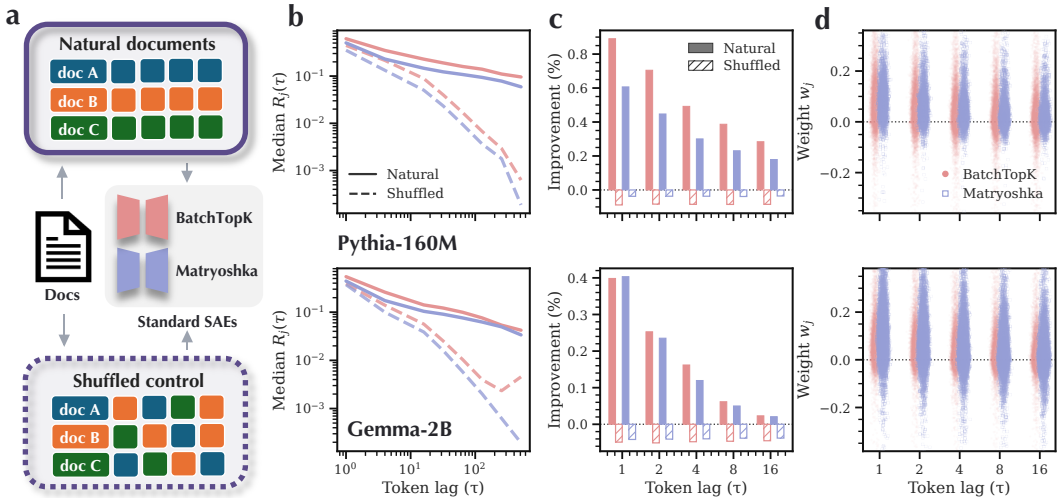


Figure 1: **Standard SAE features exhibit temporal structure that is not explicitly modelled.** **(a)** Natural Pile sequences preserve local and global continuity, whereas shuffled control sequences preserve local continuity while breaking global continuity. **(b)** Empirical feature persistence $R_j(\tau)$ decays more slowly in natural sequences than in shuffled sequences. **(c)** Past feature activations improve held-out reconstruction of later hidden states only in natural sequences. **(d)** Fitted context-correction weights w_j are predominantly positive across token lags, indicating that past feature activations continue to contribute along the same feature directions.

features differ in temporal persistence, we could model this variation during training, giving each feature an explicit timescale for inspection and analysis.

3 Does Temporal Structure Exist at the Feature Level?

We first investigate whether SAEs inherit temporal structure from the hidden states they reconstruct via two experiments: (1) whether the signal along $\mathbf{d}_j$ persists after feature j activates, and (2) whether past activations improve the reconstruction of later hidden states.

Setup. We analyse residual-stream hidden states $\{\mathbf{x}^{(t)}\}_{t=1}^{T}$ from Pythia-160M-deduped (layer 8) (Biderman et al., 2023) and Gemma-2-2B (layer 12) (Rivière et al., 2024) on Pile documents (Gao et al., 2020). We train BatchTopK (Bussmann et al., 2024) and Matryoshka (Bussmann et al., 2025) SAEs, neither of which has a temporal prior. We describe training configurations in Appendix B.

Natural Pile documents are continuous at two scales: tokens form coherent sentences (local continuity), and the sentences collectively develop a coherent document-level context (global continuity). Our *shuffled control* isolates the contribution of global continuity by concatenating sentences drawn from different Pile documents (Figure 1a). Comparing the two lets us test whether feature persistence depends on document-level continuity, beyond the local continuity preserved within individual sentences. Appendix C gives the construction details of the dataset.

Sparse features temporally persist in the hidden states. We first test whether the signal aligned with feature j persists after it activates. At each token position t, we define $p_j^{(t)} = \mathbf{d}_j^\top \mathbf{x}^{(t)}$ as the projection of the hidden state $\mathbf{x}^{(t)}$ for the t-th token onto feature j's decoder direction. Intuitively, a larger $p_j^{(t)}$ indicates a stronger alignment between the hidden state and feature direction $\mathbf{d}_j$. Conditioning on feature j being active at token t, we then define its *empirical persistence* at lag τ, where τ is the number of subsequent tokens, as

$$R_j(\tau) := \left(\mathbb{E}\left[p_j^{(t+\tau)} \mid \mathbf{f}_j^{(t)} > 0\right] - \mathbb{E}\left[p_j^{(t)}\right]\right) \Big/ \left(\mathbb{E}\left[p_j^{(t)} \mid \mathbf{f}_j^{(t)} > 0\right] - \mathbb{E}\left[p_j^{(t)}\right]\right), \tag{2}$$

where $\mathbb{E}[p_j^{(t)}]$ is the feature's average projection across all evaluated tokens. A larger $R_j(\tau)$ indicates greater persistence: $R_j(\tau) = 1$ means that the increase in feature-aligned projection at activation is

fully retained after τ tokens, while $R_j(\tau) = 0$ means that the projection has returned to baseline. Appendix D.1 provides the evaluation procedure and feature-selection criteria.

In Figure 1b, we observe that across both models and SAE families, empirical feature persistence decays more slowly in natural documents than in the shuffled control. This suggests that standard SAE features inherit the persistence associated with global continuity, even when the SAE does not explicitly encode it. These results suggest that feature-aligned signals persist beyond the tokens that activate the feature. Therefore, it may be possible to model this temporal structure explicitly and test whether past activations help reconstruct the hidden states of subsequent tokens.

Past feature activations can help reconstruction. We further test whether $\mathbf{f}^{(t-\tau)}$ can reduce an SAE's reconstruction error for token t. In a restricted *context correction* experiment, we add a contribution from the feature activations τ tokens earlier to the reconstruction in Equation (1):

$$\hat{\mathbf{x}}^{(t)}_{\text{ctx}} = \hat{\mathbf{x}}^{(t)}_{\text{base}} + \mathbf{W}_{\text{dec}} \left(\mathbf{w}_\tau \odot \mathbf{f}^{(t-\tau)} \right). \tag{3}$$

The weights $\mathbf{w}_\tau \in \mathbb{R}^m$ measure how much each past feature activation contributes to a later reconstruction along the same feature direction. For a given lag, we fit $\mathbf{w}_\tau$ for 10 epochs with an L_2 penalty and measure the fraction of reconstruction error that the context correction recovers on held-out data:

$$R^2_{\text{improve}} = 1 - \Big(\sum_t \|\mathbf{x}^{(t)} - \hat{\mathbf{x}}^{(t)}_{\text{ctx}}\|_2^2 \Big) \Big/ \Big(\sum_t \|\mathbf{x}^{(t)} - \hat{\mathbf{x}}^{(t)}_{\text{base}}\|_2^2 \Big). \tag{4}$$

For control, we use past activations from shuffled documents while keeping the same natural-document hidden states as targets. This isolates whether the reconstruction gain depends on the true preceding context. We describe the training details in Appendix D.2.

In Figure 1c, we observe that past activations from natural documents explain part of the reconstruction error left by the standard SAE, with gains decreasing as the lag grows. However, this effect is absent when training on shuffled control activations. Correspondingly, Figure 1d shows an example at lag $\tau = 1$ where the fitted context-correction weights are predominantly positive, indicating that past feature activations continue to contribute along the same feature directions. Together, these results show that sparse features contain temporal information that is not explicitly modelled by the SAE.

We note that recent work (Lubana et al., 2026) characterises temporal dependence at the *hidden state level*, showing that correlations vary across a sequence and that past hidden states explain part of the current hidden state. In contrast, we study the *feature level*, asking whether individual SAE features exhibit analogous persistence and whether their past activations help reconstruct later hidden states. We find substantial variation in persistence across features, motivating the explicit learning of a separate timescale for each feature in the next section.

4 Persistent Sparse Autoencoders

In this section, we introduce Persistent SAEs, which make the temporal structure observed in the previous section explicit by separating a token-wise sparse activation from a persistent *state* (Figure 2). As in standard BatchTopK encoders (see Appendix B.1 for details), a Persistent SAE encoder produces token-wise sparse activation $\mathbf{a}^{(t)} \in \mathbb{R}^m_{\geq 0}$, where $a^{(t)}_j > 0$ means feature j is active at token t. In contrast to standard SAEs, however, each feature integrates this activation into a *persistent state* $\mathbf{h}^{(t)} \in \mathbb{R}^m_{\geq 0}$ using a learned persistence coefficient $\lambda_j \in [0, 1]$:

$$h^{(t)}_j = \lambda_j h^{(t-1)}_j + (1 - \lambda_j) a^{(t)}_j. \tag{5}$$

The decoder then reconstructs the hidden state from this persistent state: $\hat{\mathbf{x}}^{(t)} = \mathbf{W}_{\text{dec}} \mathbf{h}^{(t)} + \mathbf{b}_{\text{dec}}$.

Intuitively, a small λ_j yields a *short-timescale fast* feature that stays close to the current token, while a large λ_j gives a *long-timescale slow* feature that integrates over many tokens across the sequence. The factor $(1 - \lambda_j)$ prevents activations from accumulating excessively, keeping the state magnitude bounded. During training, we initialise $\mathbf{h}^{(0)} = \mathbf{0}$ at the start of every input sequence. We parametrise $\lambda_j = \lambda_{\max} \sigma(l_j)$, where l_j is an unconstrained learned parameter and $\lambda_{\max} = 0.999$ keeps every feature's timescale finite. Appendix B.2 provides further details about the persistence parametrisation.

The recurrence is diagonal in feature space: feature j carries only its own activation forward, preserving feature-level attribution through its corresponding decoder direction. Sparsity is imposed

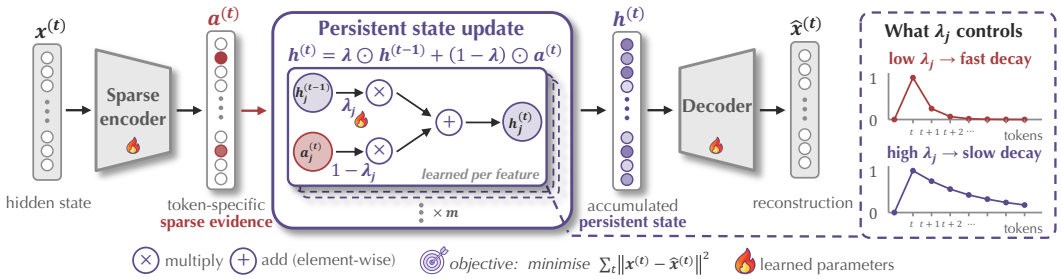


Figure 2: **Persistent Sparse Autoencoders.** Each feature integrates its token-wise sparse activation $\mathbf{a}^{(t)}$ into a persistent state $\mathbf{h}^{(t)}$ using its learned persistence coefficient λ_j. This allows different features to retain information over different timescales.

only on the token-wise activation $\mathbf{a}^{(t)}$, so the accumulated state $\mathbf{h}^{(t)}$ is not directly sparsified. Appendix B.3 shows that it remains bounded in practice: the number of active features in the state stabilises at a small fraction of the dictionary during training. Intuitively, the objective favours a larger λ_j when retaining feature j helps reconstruct later hidden states. The model therefore discovers each feature's timescale without the predefined temporal roles that previous work requires.

5 Experiments

We evaluate the quality of the representations learnt by Persistent SAEs and explore what appears at each learned timescale. We first test whether adding a persistent state preserves reconstruction quality and interpretable token-wise activations (Sections 5.1 and 5.2). We then ask what information the accumulated states of slow features capture (Section 5.3). Finally, we demonstrate their practical use in long-context monitoring through a prompt injection case study (Section 5.4).

Models and hyperparameters. For SAE training, we use the models and dataset from Section 3. We compare with BatchTopK (Bussmann et al., 2024) and Matryoshka (Bussmann et al., 2025) as standard SAE baselines, and Temporal SAE (Bhalla et al., 2026) as the closest temporally-structured baseline. All SAEs use $m = 16{,}384$ features and a BatchTopK budget of $k = 20$. Following (Bhalla et al., 2026), Matryoshka uses a two-level nested dictionary with 20% of features in the smaller inner dictionary, while Temporal SAE assigns 20% of features to a predefined high-level group and the remaining 80% to a low-level group. Persistent SAEs are trained on contiguous sequences of 256 tokens, which determines how many tokens the persistent state can propagate across during each training sample. For a fair comparison, we match the total training token budget across SAE families. Appendix B gives the full training configuration.

5.1 Reconstruction Quality and Learned Timescales

We first examine how the learned persistence coefficients are distributed across features. Figure 3a shows that most features are short-timescale and fast, with a small tail of long-timescale, highly persistent slow features. Moreover, we observe that similar distributions appear across different initialisations (Appendix G.1), training sequence lengths (Appendix G.2), and model scales (Appendix G.3). To facilitate analysis and more easily compare our features with those discovered by competing baselines (e.g., high- and low-level features from Temporal SAEs), we split features into two groups considering the observed skewness in their distributions. Specifically, we define "slow" features as the highest 1% by λ_j and "fast" features as the lowest 50%. However, we emphasise that in practice λ_j varies continuously, and show in Appendix F.2 that in-group semantic cohesiveness is preserved across thresholds.

We then evaluate representation quality using: (1) fraction of variance explained (FVE); (2) cosine similarity between the hidden state and their reconstruction; (3) fraction of active features; and (4) how smoothly features change over time, using the temporal smoothness metric from Bhalla et al. (2026), which we define in Appendix E.1. Table 1 shows that Persistent SAEs remain competitive with the strongest baselines in reconstruction quality. Across both models, slow features change more smoothly over time than fast features and baseline SAEs. This suggests that Persistent SAEs preserve

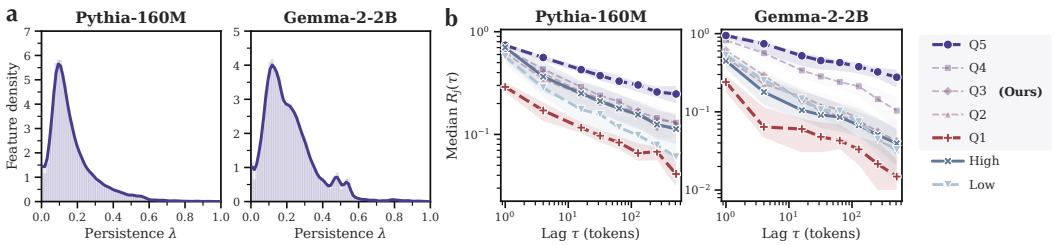


Figure 3: **Learned persistence coefficients align with empirical persistence. (a)** Persistence coefficients λ_j span a broad range, with many short-timescale features and fewer long-timescale features. **(b)** We divide Persistent SAE features into quintiles by their learned λ_j. Higher-λ_j quintiles retain greater median empirical persistence $R_j(\tau)$ across different lags.

Table 1: **SAE representation quality**. *High*/*Low* denote the Temporal SAE's predefined high-/low-level feature partition (Appendix B.3). For Persistent SAEs, *High*/*Low* denote the highest 1%/lowest 50% of features ranked by λ_j. Best/second-best results are bold/underlined. Persistent SAEs remain competitive while separating smooth, slow features from locally interpretable fast features.

		FVE (↑)	Cosine Sim (↑)	Fraction Alive (↑)	Smoothness (↓)			Autointerp (↑)		
					Full	High	Low	Full	High	Low
Pythia	BatchTopK	**0.94**$_{\pm.01}$	**0.95**$_{\pm.01}$	0.91$_{\pm.01}$	0.15$_{\pm.01}$	–	–	**0.91**$_{\pm.02}$	–	–
	Matryoshka	0.92$_{\pm.01}$	0.94$_{\pm.01}$	**0.96**$_{\pm.01}$	0.15$_{\pm.01}$	–	–	0.87$_{\pm.02}$	–	–
	Temporal	0.93$_{\pm.02}$	**0.95**$_{\pm.01}$	0.79$_{\pm.01}$	0.13$_{\pm.01}$	0.18$_{\pm.01}$	**0.11**$_{\pm.01}$	0.87$_{\pm.02}$	**0.87**$_{\pm.01}$	0.89$_{\pm.02}$
	Persistent (Ours)	0.93$_{\pm.01}$	**0.95**$_{\pm.01}$	**0.96**$_{\pm.01}$	**0.10**$_{\pm.01}$	**0.05**$_{\pm.01}$	0.20$_{\pm.01}$	0.86$_{\pm.01}$	0.84$_{\pm.02}$	**0.93**$_{\pm.01}$
Gemma	BatchTopK	**0.72**$_{\pm.01}$	**0.79**$_{\pm.01}$	**0.92**$_{\pm.01}$	0.16$_{\pm.01}$	–	–	0.87$_{\pm.03}$	–	–
	Matryoshka	0.71$_{\pm.01}$	0.77$_{\pm.01}$	0.87$_{\pm.01}$	0.17$_{\pm.01}$	–	–	**0.89**$_{\pm.02}$	–	–
	Temporal	0.62$_{\pm.02}$	0.75$_{\pm.01}$	0.75$_{\pm.01}$	0.14$_{\pm.01}$	0.12$_{\pm.01}$	**0.16**$_{\pm.01}$	0.85$_{\pm.02}$	**0.85**$_{\pm.02}$	0.86$_{\pm.02}$
	Persistent (Ours)	0.66$_{\pm.01}$	0.75$_{\pm.01}$	0.91$_{\pm.01}$	**0.11**$_{\pm.01}$	**0.04**$_{\pm.01}$	0.20$_{\pm.01}$	0.85$_{\pm.01}$	0.79$_{\pm.01}$	**0.89**$_{\pm.01}$

fast features for token-specific detail while concentrating temporal continuity in a small slow subset, without preassigning temporal roles.

Moreover, we test whether the learned coefficients correspond to the empirical persistence observed in Section 3. We group features into quintiles by λ_j and measure their median $R_j(\tau)$ across lags. Figure 3b shows a consistent ordering across both models: higher-λ_j groups decay more slowly and remain more persistent over longer lags. By contrast, the Temporal SAE's predefined high-level and low-level groups are not clearly separated by empirical persistence. This alignment shows that the coefficients learnt through reconstruction capture meaningful differences in feature timescales.

5.2 Fast Features Are Locally Interpretable

We next ask how learned timescales relate to feature interpretability. We use the automated interpretability pipeline of Bills et al. (2023), in which a judge model explains a feature from its highest-activation contexts and then scores how well that explanation predicts activation. For the Persistent SAE, we evaluate the token-wise sparse activation $\mathbf{a}^{(t)}$, asking whether it remains understandable before it is used to update the persistent state (see Appendix E.2 for details on autointerpretation).

Our experiments, summarised in Table 1 (right), suggest that, across both models, fast features are easier to explain locally than slow features. Thus, adding recurrence preserves locally interpretable activations for fast features, which match or exceed the best baseline score on both models, while interpretability varies systematically with the learned timescale. Qualitative examples (Tables 4 and 5 in Appendix E.2) suggest that fast features often respond to compact lexical or grammatical patterns, such as letters or substrings. In contrast, slow features more often reflect broader subjects whose activation is distributed across the sequence. This exposes a mismatch between persistent representations and a token-specific feature evaluation: distinct features may receive similar descriptions (McCann,

2026), and explanations may not generalise to related inputs (Tian et al., 2025). Hence, in Section 5.3 we study the slow features as a group and test what information their state carries.

Where features are activated. Before examining the states of slow features, we ask which tokens activate features with different timescales. We split features into five equal-sized quintiles by λ_j and assign each token to one of six categories: punctuation, conjunctions, function words (*Func.*), verbs, adjectives or adverbs, and nouns (see Appendix E.3 for details). For each quintile q and category c, we compute the relative activation frequency

$$U_{q,c} = \Pr\Big(T^{(t)} = c \mid a_j^{(t)} > 0,\ j \in Q_q\Big) / \Pr\Big(T^{(t)} = c\Big), \tag{6}$$

where $T^{(t)}$ is the category of token t and Q_q is the set of features in quintile q. A value above one means that features in quintile q activate at category c more often than expected from how common c is in the corpus.

Figure 4 shows the difference between the two ends of the persistence distribution. Faster features are active more often at nouns. This aligns with their stronger local interpretability in Table 1 as nouns carry clear lexical content at the current token. Slower features, in contrast, are active more often at punctuation and conjunctions. Recent analyses of LLMs likewise find that function-like tokens can carry substantial contextual information and structural markers integrate information across sentences (Razzhigaev et al., 2025; Liu & Ding, 2025). Thus, faster features are relatively more active at lexical content, whereas slower features are preferentially active at positions associated with organising and integrating context. However, where a feature activates does not tell us what its state encodes. The automated explanations suggest that slow features capture broad topics. Therefore, we test whether their states track the subject of the sequence in the next section.

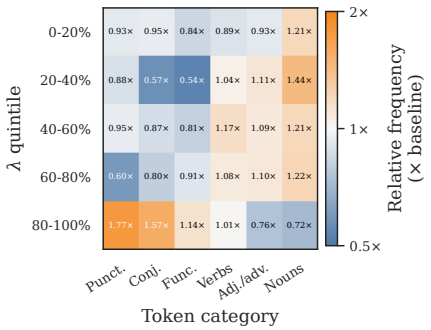


Figure 4: **Where features are activated.** Each cell shows how often features in a λ-quintile activate at a token category relative to that category's corpus-level frequency. **Orange**/**blue** indicate frequencies above/below baseline. Faster features favour nouns; slower features favour punctuation and conjunctions.

5.3 States of Slow Features Track Subject Information

Slow features form subject-specific clusters. To test whether the accumulated states of slow features retain higher-level contextual information, we visualise the slow-feature states on MMLU (Hendrycks et al., 2021) examples from five subjects. For each sample, we collect the representations of its final 20 tokens and visualise them using t-SNE (see Appendix E.4 for details). We use slow features (the highest 1% by λ_j) for visualisation. Figure 5a shows that BatchTopK features heavily mix subjects, whereas the Temporal SAE's high-level group shows limited subject separation. In contrast, the state of Persistent SAE's slow features forms coherent subject regions with a clearer visual separation than even the raw hidden states. Importantly, this structure is learned *without* subject labels.

Learned persistence provides unsupervised selection of subject-informative features. We test this observation quantitatively on spliced MMLU sequences, created by concatenating labelled segments from different subjects so that the subject changes at segment boundaries. At each token, a linear probe predicts the current subject using varying fractions of the feature dictionary for training the probe. We compare three rankings of Persistent SAE features: decreasing λ_j (slowest first), increasing λ_j (fastest first), and random ordering. For baseline SAEs which do not provide a criterion for feature selection, we select features in random order. For Temporal SAE, we rank either the high-level group before the low-level group, or vice versa, with random ordering within each group. We also include a supervised ranking that uses subject labels to favour features that activate more strongly on each subject than on all other subjects. This serves as a supervised oracle baseline (see Appendix E.5 for details).

As seen in Figure 5b, selecting features by decreasing persistence is nearly as efficient as supervised selection, while the increasing-λ_j, random, and baseline rankings retain substantially less subject information at the same budget. Interestingly, the increasing-λ_j ranking prioritises the more locally interpretable fast features but is often less efficient than baseline SAEs, suggesting that different feature persistences support a division of labour: faster features capture local, interpretable signals,

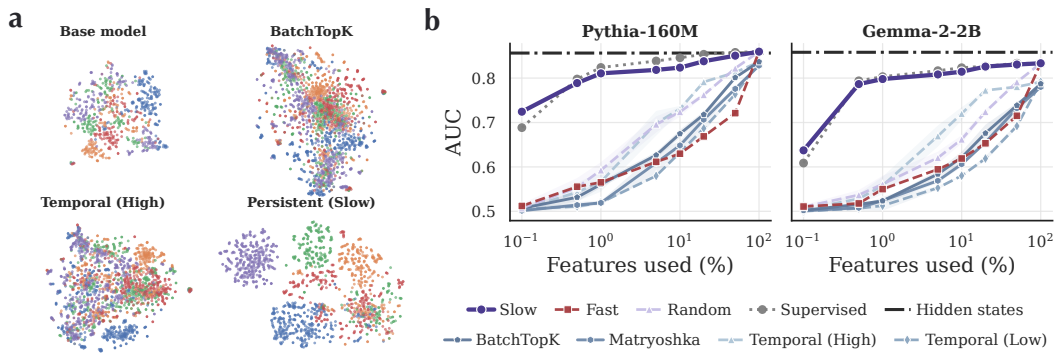


Figure 5: **The state of the slow features tracks subject information. (a)** t-SNE of token-level MMLU representations, coloured by subject. The state of the Persistent SAE's slow features (high λ_j) reveals clearer subject structure than baseline SAEs or the raw hidden states. **(b)** Subject-probing AUC on spliced MMLU sequences. Selecting a fraction of features in decreasing order of λ_j recovers most of the subject signal, approaching supervised feature selection.

while slower features retain higher-level contextual information. Appendix F.1 extends this result with a qualitative analysis to show how the state of the slow features behaves across subject boundaries. We find that tokens from the same subject have similar states, while tokens from different subjects do not. Thus, persistence provides an unsupervised criterion for selecting a compact state from a much wider sparse dictionary that tracks the current subject of a sequence.

5.4 Case Study: Persistent Prompt Injection Monitoring

The preceding experiments suggest that fast features detect interpretable patterns at the current token, while slow features integrate token-wise activations into a compact persistent state. Indirect prompt injection (Debenedetti et al., 2024) provides a practical test of this hypothesis. In this setting, an agent may encounter a malicious instruction in an email, webpage, or tool response, continue reasoning for hundreds of tokens, and act later (Debenedetti et al., 2025). A useful agentic monitor must therefore do more than detect the injected text when it appears and should preserve a security-relevant signal until the agent acts (Zhan et al., 2024; Ye et al., 2026).

Setting. We use AgentDojo trajectories from the banking, travel, and workspace domains (Debenedetti et al., 2024), generated by Llama-3.3-70B-Instruct (Meta AI, 2024) (see Appendix C.3 for details). Each trajectory contains a request, an untrusted span with a prompt injection or benign content, and the agent's subsequent tool calls. To prevent detection based on malicious actions, we use only failed attacks: injections that do not lead the agent to complete the attacker's intended action.

We use Gemma-2-2B as an external monitor: we replay each trajectory through the monitor model and record the layer-12 residual stream. We train a linear probe at the final token of the untrusted span ($d = 0$) and evaluate the same probe up to 350 tokens later without retraining (Figure 6a). We compare the hidden state and baseline SAEs with the slow- and fast-feature Persistent SAE states as defined in Section 5.1. This evaluation measures how long each representation preserves the injection signal (see Appendix E.6 for evaluation details).

Signal retention: fast features detect local injection signals while slow features retain signals over long contexts. At $d = 0$, fast features and standard SAE representations provide the strongest signals for detecting an injection (Figure 6b, top). As the evaluation moves further beyond the untrusted span, their accuracy falls towards chance. By contrast, slow features retain a clearer injection signal at longer distances. This result again confirms that fast and slow features play complementary roles: fast features detect injected text when it appears, while the state of slow features preserves the signal through the subsequent context.

Causal effect: slow features influence the monitor's judgement. Probing shows that prompt-injection information can be decoded from the representations, but does not establish whether this information affects the monitor's decision. We therefore perform a causal intervention experiment.

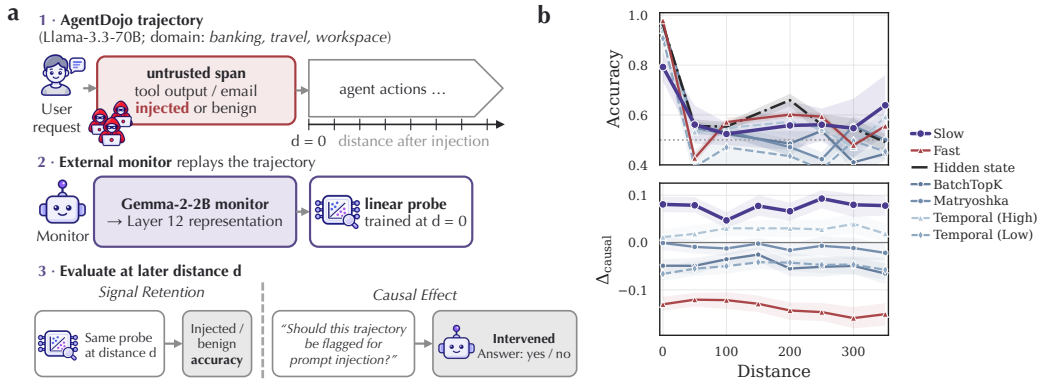


Figure 6: **Persistent monitoring of prompt injection. (a)** A probe trained at the end of an untrusted span is evaluated later in the trajectory. **(b)** Slow-feature states retain more predictive information (top), while their decoder directions exert the strongest causal influence on the monitor's judgment (bottom).

At each distance d, we append a query asking the monitor whether a prompt injection occurred and intervene on the hidden state at the final query token. Building on prior work that selects SAE features for steering (Arad et al., 2025; He et al., 2025), for each representation r, we construct a task-relevant steering direction $\mathbf{v}_r$ from features identified by the probe (see Appendix E.6 for details). We apply interventions in both positive and negative directions. Let $s(\mathbf{x}) = \ell_{\text{yes}}(\mathbf{x}) - \ell_{\text{no}}(\mathbf{x})$ denote the monitor's logit margin. We measure the intervention effect as

$$\Delta_{\text{causal}}(r, d) = \frac{1}{2}\mathbb{E}_i\Big[s\big(\mathbf{x}_i^{(d)} + \alpha_i^{(d)}\mathbf{v}_r\big) - s\big(\mathbf{x}_i^{(d)} - \alpha_i^{(d)}\mathbf{v}_r\big)\Big]. \tag{7}$$

where $\alpha_i^{(d)}$ scales the intervention to 10% of the hidden-state norm. A positive value means that moving along the direction makes the monitor likely to answer *yes*.

Slow features produce the largest and most consistently positive intervention effect across all distances (Figure 6b, bottom). In contrast, directions derived from baseline SAE features produce small or negative effects. These results complement the probing experiment. Fast features provide strong local evidence of an injection, whereas slow features preserve information that remains both decodable and causally relevant to the monitor's later decision. Persistent SAE therefore captures both short-lived local evidence and longer-lived state that can influence downstream model decisions, offering a sparse representation that may support more reliable monitoring and control of LLMs.

6 Discussion and Conclusion

Discussion and Limitations. A natural question is whether multiple features could share a persistence coefficient. We test this by varying how many features are grouped to share a coefficient and find that, as groups get smaller, we recover a progressively wider range of empirical timescales (Appendix F.3). This suggests that different features require different persistence rates, supporting our modelling choice. We assign persistence coefficients through a diagonal recurrence, and therefore do not capture cross-feature interactions. Future work can explore explicitly modelling cross-feature interactions. Finally, we note that, in our prompt-injection case study, we use an external replay monitor of agent trajectories; therefore, we do not establish that persistent states have a causal effect on the original agent's actions. Future work could explore richer recurrent dynamics and test whether persistent feature interventions affect end-to-end agent behaviour.

Conclusion. We introduced Persistent SAEs, which learn from reconstruction alone how long individual sparse features should retain information across tokens. The learned per-feature persistence coefficients span a broad range of timescales: short-timescale features remain locally interpretable, while long-timescale features accumulate contextual information, tracking subject identity and preserving monitoring signals over long contexts. As such, Persistent SAEs provide an unsupervised way to organise sparse representations by timescale and offer a potential direction for interpreting and monitoring information maintained by LLMs over long contexts.

AI USE STATEMENT

In this work, we used generative AI tools as part of our experimental evaluation: `gpt-4o-mini` served as the judge model in the automated interpretability pipeline (Appendix E.2), where it wrote short natural-language explanations of SAE features from their highest-activation contexts and then used these explanations to predict feature activations on held-out contexts. The judge model's outputs were used only to compute interpretability scores. We also used generative AI tools to assist with coding. All AI-generated code was reviewed by the authors, verified against the intended implementation, and tested for correctness. Additionally, we used generative AI tools to correct grammatical errors in the manuscript. We have reviewed all AI-assisted work. All grammar corrections were checked by the authors, and no content, claims or results were generated by these tools. We take responsibility for the final content of this work, including text, claims or artefacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have taken several steps to make our results reproducible. The Persistent SAE architecture and training objective are fully specified in Section 4, and Appendix B.2 details the parameterisation and initialisation of the persistence coefficients. Appendix B gives the full training configuration for every SAE family, including the sparsity budget and token budget. All language models whose representations we analyse and all datasets we use are publicly available, and Appendix C describes the data processing steps for the Pile, MMLU, and AgentDojo data, including the construction of the shuffled control sequences. Appendices D.1–E.6 give the evaluation procedures, and Appendix G reports the sensitivity of our results to hyperparameters and model size.

REFERENCES

Dana Arad, Aaron Mueller, and Yonatan Belinkov. SAEs are good for steering – if you select the right features. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 10241–10259, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.519. URL https://aclanthology.org/2025.emnlp-main.519/.

Usha Bhalla, Alex Oesterling, Claudio Mayrink Verdun, Himabindu Lakkaraju, and Flavio Calmon. Temporal sparse autoencoders: Leveraging the sequential nature of language for interpretability. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=bojVI4l9Kn.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, Usvsn Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. Pythia: A suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 2397–2430. PMLR, 2023. URL https://proceedings.mlr.press/v202/biderman23a.html.

Steven Bills, Nick Cammarata, Dan Mossing, Henk Tillman, Leo Gao, Gabriel Goh, Ilya Sutskever, Jan Leike, Jeff Wu, and William Saunders. Language models can explain neurons in language models, 2023. URL https://openaipublic.blob.core.windows.net/neuron-explainer/paper/index.html.

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.

Bart Bussmann, Patrick Leask, and Neel Nanda. Batchtopk sparse autoencoders. In *NeurIPS 2024 Workshop on Scientific Methods for Understanding Deep Learning*, 2024. URL https://openreview.net/forum?id=d4dpOCqybL.

Bart Bussmann, Noa Nabeshima, Adam Karvonen, and Neel Nanda. Learning multi-level features with matryoshka sparse autoencoders. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=m25T5rAy43.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024. doi: 10.52202/079017-2636. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design, 2025. URL https://arxiv.org/abs/2503.18813.

Jiahai Feng, Stuart Russell, and Jacob Steinhardt. Monitoring latent world states in language models with propositional probes. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=0yvZm2AjUr.

Peter Földiák. Learning invariance from transformation sequences. *Neural Computation*, 3(2): 194–200, 1991. doi: 10.1162/neco.1991.3.2.194.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020. URL https://arxiv.org/abs/2101.00027.

Leo Gao, Tom Dupre la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tcsZt9ZNKD.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256. PMLR, 2010.

Wes Gurnee and Max Tegmark. Language models represent space and time. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=jE8xbmvFin.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015. doi: 10.1109/ICCV.2015.123.

Zirui He, Mingyu Jin, Bo Shen, Ali Payani, Yongfeng Zhang, and Mengnan Du. SAE-SSV: Supervised steering in sparse representation spaces for reliable control of language models. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 2207–2236, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.112. URL https://aclanthology.org/2025.emnlp-main.112/.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *ICLR*, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=F76bwRSLeK.

Najoung Kim and Sebastian Schuster. Entity tracking in language models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3835–3855, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.213. URL https://aclanthology.org/2023.acl-long.213/.

Michael Lan, Philip Torr, Austin Meek, Ashkan Khakzar, David Krueger, and Fazl Barez. Quantifying feature space universality across large language models via sparse autoencoders. *arXiv:2410.06981*, 2024. URL https://arxiv.org/abs/2410.06981.

Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=DeG07_TcZvT.

Wei Liu and Nai Ding. Information integration in large language models is gated by linguistic structural markers. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 6892–6904, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.351. URL https://aclanthology.org/2025.emnlp-main.351/.

Ekdeep Singh Lubana, Can Rager, Sai Sumedh R. Hindupur, Valérie Costa, Oam Patel, Sonia Krishna Murthy, Thomas Fel, Greta Tuckute, Daniel Wurgaft, Eric Bigelow, Demba E. Ba, Melanie Weber, and Aaron Mueller. Priors in time: Missing inductive biases for language model interpretability. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=4J2e3nWiC8.

Haoyan Luo and Lucia Specia. From understanding to utilization: A survey on explainability for large language models, 2024. URL https://arxiv.org/abs/2401.12874.

Samuel Marks, Can Rager, Eric J Michaud, Yonatan Belinkov, David Bau, and Aaron Mueller. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=I4e82CIDxv.

Jordan F. McCann. Descriptive collision in sparse autoencoder auto-interpretability: When one explanation describes many features, 2026. URL https://arxiv.org/abs/2605.12874.

Meta AI. Llama 3.3 70B Instruct model card, 2024. URL https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct.

monology. Pile uncopyrighted. Hugging Face dataset, 2023. URL https://huggingface.co/datasets/monology/pile-uncopyrighted.

Thomas F. Münte, Bernardina M. Wieringa, Helga Weyerts, Andras Szentkuti, Mike Matzke, and Sönke Johannes. Differences in brain potentials to open and closed class words: Class and frequency effects. *Neuropsychologia*, 39(1):91–102, 2001. doi: 10.1016/S0028-3932(00)00095-6.

OpenAI. GPT-4 technical report. *CoRR*, 2023.

Anton Razzhigaev, Matvey Mikhalchuk, Temurbek Rahmatullaev, Elizaveta Goncharova, Polina Druzhinina, Ivan Oseledets, and Andrey Kuznetsov. LLM-microscope: Uncovering the hidden role of punctuation in context memory of transformers. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 7772–7779, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.432. URL https://aclanthology.org/2025.findings-naacl.432/.

Morgane Rivière et al. Gemma 2: Improving open language models at a practical size. *CoRR*, abs/2408.00118, 2024. URL https://arxiv.org/abs/2408.00118.

Lee Sharkey, Bilal Chughtai, Joshua Batson, Jack Lindsey, Jeffrey Wu, Lucius Bushnaq, Nicholas Goldowsky-Dill, Stefan Heimersheim, Alejandro Ortega, Joseph Isaac Bloom, Stella Biderman, Adrià Garriga-Alonso, Arthur Conmy, Neel Nanda, Jessica Mary Rumbelow, Martin Wattenberg, Nandi Schoots, Joseph Miller, William Saunders, Eric J Michaud, Stephen Casper, Max Tegmark, David Bau, Eric Todd, Atticus Geiger, Mor Geva, Jesse Hoogland, Daniel Murfet, and Thomas McGrath. Open problems in mechanistic interpretability. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=91H76m9Z94. Survey Certification.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

Claire Tian, Katherine Tian, and Nathan Hu. Measuring sparse autoencoder feature sensitivity, 2025. URL https://arxiv.org/abs/2509.23717.

Laurenz Wiskott and Terrence J. Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, 2002. doi: 10.1162/089976602317318938.

Charles Ye, Jasmine Cui, and Dylan Hadfield-Menell. Prompt injection as role confusion. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Lm3DQDyIsI.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.624. URL https://aclanthology.org/2024.findings-acl.624/.

A NOTATION

Table 2 lists the symbols used in the main text and appendices, in order of first appearance, as a reference for the terminology introduced in Sections 3–5.4.

Table 2: Notation used throughout the paper, grouped by the section in which each symbol is introduced. Bold symbols denote vectors and matrices, a parenthesised superscript denotes a token position, and a subscript denotes a coordinate within a vector.

Symbol	Meaning
Hidden states and standard SAE (Section 3, 4)	
$\mathbf{x}^{(t)} \in \mathbb{R}^d$	Residual-stream hidden state at token position t
T	Number of tokens in a sequence
d	Dimensionality of the hidden state
m	Number of dictionary features (SAE width)
$\mathbf{W}_{\text{enc}}, \mathbf{b}_{\text{enc}}$	Standard SAE encoder weight matrix and bias
$\mathbf{W}_{\text{dec}}, \mathbf{b}_{\text{dec}}$	Standard SAE decoder weight matrix and bias
$\mathbf{d}_j$	Unit-normalised decoder direction for feature j
$\mathbf{f}^{(t)} \in \mathbb{R}^m_{\geq 0}$	Sparse activation of a standard SAE at token t (Background and Related Work)
$f_j^{(t)}$	Sparse activation of feature j at token t
k	BatchTopK sparsity level (Appendix B.1)
B	Batch size used by BatchTopK selection (Appendix B.1)
Empirical feature persistence analysis (Section 3, Appendix D.1)	
τ	Lag, in tokens
$p_j^{(t)}$	Decoder-aligned projection $\mathbf{d}_j^\top \mathbf{x}^{(t)}$
$R_j(\tau)$	Empirical persistence of $p_j^{(t)}$ at lag τ
$\beta_{\text{nat}}, \beta_{\text{shf}}$	Power-law decay exponents, natural / shuffled
Context correction experiment (Section 3, Appendix D.2)	
$\mathbf{w}_\tau \in \mathbb{R}^m$	Per-feature context-correction weights at lag τ
$w_{\tau,j}$	Context-correction weight for feature j at lag τ (w_j at $\tau = 1$)
$\hat{\mathbf{x}}^{(t)}_{\text{base}}$	Baseline reconstruction from a standard SAE
$\hat{\mathbf{x}}^{(t)}_{\text{ctx}}$	Context-corrected reconstruction (Equation 3)
R^2_{improve}	Held-out reconstruction-error improvement from context correction
Persistent SAE (Section 4)	
$\mathbf{a}^{(t)} \in \mathbb{R}^m_{\geq 0}$	Token-wise sparse activation of a Persistent SAE (Appendix B.1)
$a_j^{(t)}$	Token-wise sparse activation of feature j at token t
$\mathbf{h}^{(t)} \in \mathbb{R}^m_{\geq 0}$	Persistent state integrating token-wise activations over time (Equation 5)
$h_j^{(t)}$	Persistent state of feature j at token t
$\lambda_j \in (0, \lambda_{\max})$	Persistence coefficient for feature j (Appendix B.2)
$\lambda_{\max}$	Upper bound on the persistence coefficient ($= 0.999$)
l_j	Unconstrained logit parameterising λ_j
$\sigma(\cdot)$	Sigmoid function

B TRAINING DETAILS

B.1 BATCHTOPK SPARSITY

For every SAE in this paper, sparsity is imposed using a BatchTopK mechanism in which a hidden state $\mathbf{x} \in \mathbb{R}^d$ is first mapped to non-negative pre-activations

$$\tilde{\mathbf{f}} = \text{ReLU}(\mathbf{W}_{\text{enc}}(\mathbf{x} - \mathbf{b}_{\text{dec}}) + \mathbf{b}_{\text{enc}}) \in \mathbb{R}^m_{\geq 0}, \tag{8}$$

and only the largest pre-activations are retained across the batch. For a batch $X = \{\mathbf{x}_i\}_{i=1}^B$, the pre-activations $\{\tilde{\mathbf{f}}_i\}_{i=1}^B$ are flattened into a single vector of length Bm, and the sparse activation $\mathbf{f}_i$ is obtained by keeping only the top kB entries globally and setting all others to zero. The reconstruction is then

$$\hat{\mathbf{x}}_i = \mathbf{W}_{\text{dec}}\mathbf{f}_i + \mathbf{b}_{\text{dec}}. \tag{9}$$

In practice, after training with exact BatchTopK selection, inference uses an equivalent thresholded form

$$f_{ij} = \tilde{f}_{ij}\,\mathbf{1}[\tilde{f}_{ij} > \theta], \tag{10}$$

where the threshold θ is estimated from training-time top-k statistics. The Persistent SAE uses the resulting sparse activation as its token-wise sparse activation $\mathbf{a}^{(t)}$.

B.2 Persistence Parameterisation

For Persistent SAE, we parameterise each persistence coefficient as

$$\lambda_j = \sigma(l_j) \cdot \lambda_{\max}, \tag{11}$$

where $l_j \in \mathbb{R}$ is an unconstrained logit, σ is the sigmoid function, and $\lambda_{\max} = 0.999$ caps λ_j strictly below one. Without this cap, λ_j could reach exactly one, turning the state update in Equation (5) into an unbounded running sum with no decay. The cap instead keeps $\lambda_j \in (0, \lambda_{\max})$ throughout training, so every feature has a finite effective memory, and the factor $(1 - \lambda_j)$ in Equation (5) makes the recurrence an exponential moving average (EMA) instead of an unnormalised sum. The reason is that under a constant activation $a_j^{(t)} = a_j$, the state converges to a_j for every λ_j, so persistence does not inflate the state's magnitude. Before training, unless otherwise specified, we initialise every logit l_j to 0.5, which gives each feature moderate initial persistence while keeping the sigmoid away from saturation, so training can adjust the timescale in either direction.

B.3 SAE Training Details

We train four SAE families for comparison: BatchTopK (Bussmann et al., 2024), Matryoshka BatchTopK (Bussmann et al., 2025), Temporal SAE (Bhalla et al., 2026), and our Persistent SAE. Each family uses the same dictionary width, BatchTopK activation budget, optimiser, and token budget.

Configuration. Every SAE has a dictionary of $m = 16{,}384$ features and a BatchTopK activation budget of $k = 20$ per token. We train with Adam at a learning rate of 3×10^{-4} and use a linear warm-up over the first $1{,}000$ steps. From step $1{,}000$, we apply the BatchTopK activation-threshold schedule with decay $\beta = 0.999$, together with the auxiliary dead-feature loss from (Gao et al., 2025), weighted by $\alpha_{\text{aux}} = 0.03125$. Training activations come from Pile Uncopyrighted (monology, 2023), with the beginning-of-sequence position removed. For Gemma-2-2B (Rivière et al., 2024), we use the layer-12 residual stream with hidden width $d = 2{,}304$; for Pythia-160M-deduped (Biderman et al., 2023), we use the layer-8 residual stream with $d = 768$. For comparability, we use a two-level Matryoshka SAE whose smaller nested dictionary contains 20% of the features, while the Temporal SAE partitions its dictionary into a predefined high-level group (20% of the features) and low-level group (the remaining 80%), following their paper (Bhalla et al., 2026). The Temporal SAE weights its reconstruction losses by 0.2 (high-level) and 0.8 (low-level), respectively. The Persistent SAE does not use a predefined partition; instead, it learns a persistence coefficient for every feature.

Matched token budget. Each SAE family groups activations differently before training on them: BatchTopK and Matryoshka use shuffled individual tokens, the Temporal SAE uses adjacent token pairs, and the Persistent SAE uses contiguous sequences so it can carry state across tokens. To keep training comparable across families, we match the number of activation tokens processed per step and the total number of steps, not the number of these groups. Each model trains for $244{,}140$ steps with an effective batch of approximately $2{,}048$ activation tokens per step, giving the flat baselines approximately 5.0×10^8 tokens in total. For persistent SAEs, unless otherwise specified, we use a sequence length of 256 tokens during training. All experiments were run on NVIDIA A100 GPUs with 80 GB of memory.

Figure 7 shows the training dynamics of the Persistent SAEs. For both models, the loss decreases rapidly and then remains stable, while the state L_0 settles at roughly 400–500 active features despite continued training. Both terms in the state update (Equation (5)) are non-negative, so we define the carried-mass fraction as the share of the state's total magnitude that comes from the carried term:

$$\rho^{(t)} = \frac{\sum_j \lambda_j h_j^{(t-1)}}{\sum_j h_j^{(t)}}. \tag{12}$$

This fraction stabilises at approximately 0.5 for Pythia and 0.65–0.70 for Gemma: roughly half to two-thirds of each state's magnitude comes from the previous token, with the rest replaced by new activations at the current token. Thus, the recurrence does not cause the active state to grow continually, and neither model relies exclusively on new activation or carried state.

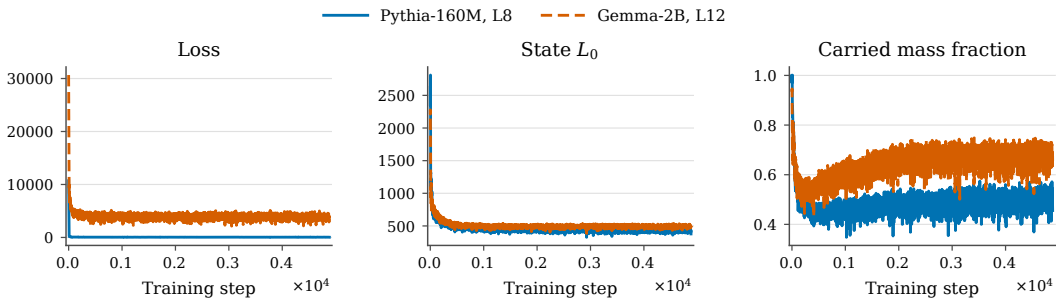


Figure 7: **Persistent SAE training dynamics.** We plot, over training: training loss (left); the number of non-zero features in the persistent state $\mathbf{h}^{(t)}$ (state L_0; centre); and the carried-mass fraction $\rho^{(t)}$ (right, Equation (12)), the share of the state's total magnitude that comes from the previous token rather than the current one. After a short initial transient, all three quantities stabilise, and the carried-mass fraction settles well away from both zero and one.

C Dataset Details

We use three datasets for distinct parts of the evaluation. Pile (Gao et al., 2020; monology, 2023) provides the hidden states used to train and evaluate the standard metrics of SAEs, MMLU questions (Hendrycks et al., 2021) provide subject-labelled text for analysing the persistent state, and AgentDojo trajectories (Debenedetti et al., 2024) support the prompt-injection monitoring case study. We describe each dataset and our use of it below.

C.1 Pile Corpus

The original Pile is an English-language corpus assembled from 22 sources, including academic papers, web pages, books, code, and online discussions (Gao et al., 2020). We use the Pile Uncopyrighted release (monology, 2023). Documents are accessed in streaming mode and shuffled with a buffer of size 10,000 to train the SAEs. For the natural documents mentioned in Section 3, each sampled document is passed through the model and the resulting residual-stream hidden states are retained for training. For the shuffled control, sentences are sampled from documents drawn from the same Pile corpus and concatenated subject to a simple adjacency constraint that prevents consecutive sentences from coming from the same source document. This preserves local continuity while breaking global continuity, which facilitates the study of temporal persistence in Section 3.

C.2 MMLU Dataset

Massive Multitask Language Understanding (MMLU) is an English-language benchmark of four-option multiple-choice questions from 57 subjects spanning STEM, the humanities, the social sciences, and other professional and academic fields (Hendrycks et al., 2021). We use MMLU for the analyses in Section 5.3. Specifically, we use questions from abstract algebra, astronomy, business ethics, computer security, and European history. These subjects cover distinct areas while providing a subject label for every example. In Section 5.3, we sample 100 sequences across the five subjects and retain the final 20 token representations from each sequence, following Bhalla et al. (2026). For subject probing, we construct sequences by joining equal-length segments from different subjects and assign each token the subject label of its source segment. Appendix E.4 and Appendix E.5 provide the sampling and evaluation details for the two analyses.

C.3 AGENTDOJO TRAJECTORIES

Indirect prompt injection occurs when an agent retrieves attacker-controlled text from emails, webpages, documents, or tool outputs and follows instructions inside that text. AgentDojo (Debenedetti et al., 2024) evaluates this setting with stateful tool environments containing 97 user tasks, 27 injection targets, and 629 security test cases across workspace, Slack, travel, and banking domains. Its task-specific checks evaluate whether the legitimate user request and the attacker's target action were completed by inspecting the agent output and changes to the environment.

For the case study in Section 5.4, we reuse existing AgentDojo trajectories from the banking, travel, and workspace domains, generated by Llama-3.3-70B-Instruct (Meta AI, 2024) following Debenedetti et al. (2024). Benign trajectories with no injection form the negative class ($y = 0$). For the positive class ($y = 1$), we take runs with an `important_instructions` or `direct` attack where AgentDojo's injection checker reports `security=false`: an injection was present, but the agent did not complete the attacker's target action. We keep only these failed attacks on purpose. A successful attack usually includes an overt malicious action, such as transferring money or leaking private data, and a monitor could flag that action alone without ever detecting the source injection. Restricting to failed attacks removes this shortcut and tests whether the monitor still detects the injection when the attack leaves no such trace.

D TEMPORAL PERSISTENCE EXPERIMENTS

This section gives the full setup for the two experiments in Section 3: the empirical persistence measure $R_j(\tau)$, and the context correction test of whether past activations help reconstruction.

D.1 EMPIRICAL PERSISTENCE OF STANDARD SAE FEATURES

This subsection gives the evaluation procedure and feature-selection criteria for $R_j(\tau)$, and expands on the persistence results plotted in Figure 1.

Setup. We compute $R_j(\tau)$ at lags $\tau \in \{1, 4, 16, 32, 64, 128, 256, 512\}$ tokens, on documents truncated to at most $1{,}024$ tokens. For each feature j, we call a token an *anchor* if feature j is active there and at least τ further tokens remain in the document, so we can measure its projection τ tokens later. We keep a feature in the analysis only if it activates at least 256 times, has at least 64 anchors at every lag, and shows a non-negligible activated-token elevation. We exclude a feature instead of stabilising its score whenever this elevation, the denominator $\mathbb{E}[p_j^{(t)} \mid \mathbf{f}_j^{(t)} > 0] - \mathbb{E}[p_j^{(t)}]$ in Equation (2), is close to zero, as happens for near-dead features.

Natural sequences persist longer than shuffled ones. Across both models and both SAE families, median persistence is high at lag 1 and decays as τ grows, but far more slowly on natural documents than on the shuffled control: natural-document persistence stays clearly above zero out to the longest lag we test, while the shuffled control decays to approximately zero well before that lag. To quantify this gap, we fit a power law to the decay: the fitted exponent is substantially smaller for natural sequences ($\beta_{\text{nat}} \approx 0.32$) than for the shuffled control ($\beta_{\text{shf}} \approx 0.87$–$0.92$), consistent with the slower decay, and this gap holds for both BatchTopK and Matryoshka.

D.2 CONTEXT CORRECTION EXPERIMENT

Setup. We fit the per-feature context correction of Equation (3) independently at each lag $\tau \in \{1, 2, 4, 8, 16, 32\}$. For each lag, we use 200 natural documents to fit $\mathbf{w}_\tau$ and 100 held-out natural documents to evaluate R^2_{improve}. The target is always the natural-document reconstruction error. For the control, we refit the same diagonal context correction head to predict these natural targets from context activations drawn from the shuffled corpus instead, using the same parameterisation and optimisation budget. We optimise for 10 epochs with AdamW, learning rate 5×10^{-3}, and weight decay 10^{-3} in order to keep the training lightweight.

Past activations reduce reconstruction error, but only from real context. On Pythia-160M, the correction removes 0.89% (BatchTopK) and 0.61% (Matryoshka) of the baseline reconstruction error at lag 1; the gain shrinks with lag but stays positive at lag 32 (0.22% and 0.13%). On Gemma-2-2B, both SAEs remove approximately 0.40% at lag 1, falling to about 0.02% at lag 16. The shuffled

control stays slightly below zero throughout, so unrelated context never improves reconstruction; the gain requires activations from the actual context of the target document.

Different features play different temporal roles. Table 3 summarises the fitted lag-one weights w_j. On natural documents, the mean weight is positive, and most features take a positive weight, so past activation of these features predicts a positive contribution along the same feature direction at the next token. Two further details in the table show that this carry-forward behaviour is feature-specific rather than universal. First, the positive fraction stops well short of 100% (76–91% across models and SAE families): a sizeable minority of features show no carry-forward effect at all. Second, this effect is tied to real document context, since the shuffled control's mean weight stays close to zero for every model and SAE family. Together with the lag-band heterogeneity in Appendix D.1, this shows that no single, shared treatment of persistence fits every feature: some features carry information forward and some do not, and among those that do, by how much and for how long both vary by feature. This motivates learning a persistence coefficient separately for each feature in Section 4.

Table 3: Fitted lag-one context-correction weights w_j. *Mean* w_j and *Pos. frac.* are the mean and the positive fraction on natural documents; *Shuf.* w_j is the mean on the shuffled control.

Model	**SAE**	**Mean** w_j	**Pos. frac.**	**Shuf.** w_j
Pythia	BatchTopK	0.068	76.2%	0.001
Pythia	Matryoshka	0.087	91.4%	−0.001
Gemma	BatchTopK	0.068	80.4%	−0.001
Gemma	Matryoshka	0.091	81.9%	−0.001

E Evaluation Details

This section defines the temporal smoothness metric used in Section 5.1 and gives the evaluation procedures for the four analyses in Section 5: automated interpretability (Section 5.2), subject geometry and subject probing (Section 5.3), and the prompt-injection monitoring case study (Section 5.4).

E.1 Temporal Smoothness

We use the smoothness metric of Bhalla et al. (2026) to measure how much feature activations jump between neighbouring tokens. A lower score means smoother features.

For each evaluation sequence s of length T, we keep only the n' features that are active at least once in the sequence. For each of these features, we find the largest change in its activation between two consecutive tokens, divided by the change in the hidden state at that step. We then average over the active features:

$$\Delta_s = \frac{1}{n'} \sum_{j=1}^{n'} \max_{t \in \{2,\dots,T\}} \frac{\left| f_j^{(t)} - f_j^{(t-1)} \right|}{\|\mathbf{x}^{(t)} - \mathbf{x}^{(t-1)}\|_2}. \tag{13}$$

Dividing by the change in the hidden state means that a feature is not penalised for changing when the input itself changes sharply. The smoothness score averages this quantity over n evaluation sequences, $S = \frac{1}{n} \sum_{s=1}^{n} \Delta_s$.

For standard SAEs, $f_j^{(t)}$ is the feature activation. For Persistent SAEs, we use the persistent state $h_j^{(t)}$, since this is what the decoder reconstructs from. Besides the score over all features (*Full*), Table 1 also reports it separately for the *High* and *Low* feature subsets defined in the table caption.

E.2 Automated Interpretability Details

This subsection gives the autointerpretation pipeline for Section 5.2, and discusses the qualitative examples referenced there (Tables 4 and 5).

Pipeline. We follow the automated interpretability pipeline of Bills et al. (2023). For each feature, we collect its highest-activation contexts and ask a judge model to write a short natural-language explanation. We then test this explanation on held-out contexts: the judge uses the explanation to

Table 4: Qualitative autointerpretation examples for Pythia-160M. High- and low-λ features are drawn from the top 1% and bottom 50% of the learned persistence coefficients, respectively. Bold text marks representative activating spans. The selected high-λ features are activated by tokens in coherent semantic contexts, whereas the low-λ features respond to local strings, syntax, or recurring templates. All selected features have an autointerpretation score above 0.85.

Feature	λ	Autointerpretation	Representative activation	Score
High λ (top 1%): sustained semantic domains				
F-5528	7.03×10^{-1}	themes of ethnic diversity and experiences of discrimination	"ashamed of being **Mexican** because I'm different from the **stereotype**"	0.93
F-12751	4.82×10^{-1}	phrases related to population-based studies and cancer research	incident **colorectal cancer cases** identified through a **population-based cancer registry**	0.93
F-15986	3.81×10^{-1}	terms related to high-performance cars and their specifications	rare and desirable **muscle cars**; a **1971 Hemi 'Cuda**	0.93
F-6538	3.79×10^{-1}	terms related to weather conditions and climate phenomena	a combination of **global warming** and the **El Niño weather phenomenon**	1.00
F-8449	3.69×10^{-1}	mathematical concepts and terms related to Banach spaces and polynomials	continuous scalar polynomials over a **Banach space** X	1.00
Low λ (bottom 50%): local lexical, syntactic, and template patterns				
F-1113	5.35×10^{-2}	the substring "white" followed by various words or concepts	"There is so much **white space** just waiting to be filled"	1.00
F-6958	4.96×10^{-2}	the substring "function" in JavaScript code snippets	`.click(function() {`	1.00
F-15079	4.81×10^{-2}	the phrase "U.S. Pat. No." indicating references to patents	"disclosed in **U.S. Pat. No.** 5,111,981"	1.00
F-3587	1.10×10^{-2}	URLs and web documentation references	`ubuntu.com/10.04` **`/serverguide/C/`** `index.html`	1.00
F-14758	4.04×10^{-4}	SQL commands related to table creation and data manipulation	**`CREATE TABLE IF NOT EXISTS`** `tennis(...)`	1.00

predict whether the feature's activation exceeds a threshold, and we score how well this prediction matches the feature's actual activation. For standard SAEs, this activation is $f_j^{(t)}$; for the Persistent SAE, we use the token-wise activation $a_j^{(t)}$. We use `gpt-4o-mini` (OpenAI, 2023) as the judge, a context length of 1024 tokens, and roughly 2M tokens from the Pile for the activation examples. We also score the highest-λ_j 1% and lowest-λ_j 50% of Persistent SAE features to evaluate their interpretability as groups.

Qualitative examples. Tables 4 and 5 give example features for Pythia-160M and Gemma-2-2B, all scoring above 0.85, so both groups of features are well explained individually. High-λ_j (slow) features track a sustained topic across a passage: examples include ethnic-discrimination narratives, cosmology, and population-based cancer research, where the representative span is one clause drawn from a much longer discussion of the same subject. Low-λ_j (fast) features instead pick out a specific, self-contained string, such as a JavaScript function call, a patent-citation phrase, or a SQL `CREATE TABLE` statement, each explained by the exact text that triggers it. This pattern holds for both models, and matches the finding that fast features are activated by local lexical content while slow features track broader semantic context.

E.3 Token Category Details

We group tokens by their part of speech, following the classical split between content words, which carry lexical meaning, and function words, which mark grammatical structure (Münte et al., 2001). We split content words into nouns, verbs, and adjectives or adverbs. Function words include

Table 5: Qualitative autointerpretation examples for Gemma-2-2B. High- and low-λ features are drawn from the top 1% and bottom 50% of the learned persistence coefficients, respectively. Bold text marks representative activating spans. The selected high-λ features are activated by tokens in coherent semantic contexts, whereas the low-λ features respond to local strings, syntax, or recurring templates. All selected features have an autointerpretation score above 0.85.

Feature	λ	Autointerpretation	Representative activation	Score
High λ (top 1%): sustained semantic domains				
F-11205	9.18×10^{-1}	terms related to cosmology and advanced scientific concepts	near-future **CMB Stage-III and IV surveys** yielding reconstructed **CMB lensing maps**	0.86
F-5578	8.55×10^{-1}	terms related to ionisation charge collection and quantum experiments in physics	**ionisation charge collection efficiency** observed in gases	0.86
F-15639	8.24×10^{-1}	text related to conflict, violence, and ethnic tensions	self-determination for Katanga; a **separatist group** composed of former Katanga Tigers	0.86
F-13900	7.85×10^{-1}	terms related to microbial biomass and ecological concepts in alpine and tundra environments	**microbial biomass** in an **alpine meadow on the Tibetan Plateau**	0.86
F-11933	7.70×10^{-1}	expressions of self-acceptance and emotional support in personal growth narratives	"In the last couple years, I've **let go** and **decided to be myself**."	1.00
Low λ (bottom 50%): local lexical, syntactic, and template patterns				
F-9996	1.03×10^{-3}	legal terms and conditions related to software licenses	"See the **License** for the specific language governing rights and limitations"	1.00
F-11210	1.30×10^{-5}	the phrase "all of" in various contexts	"it is unlikely that **all of** these loci respond to selection independently"	0.93
F-14595	7.87×10^{-6}	the substring "mk" within mathematical and scientific contexts	repeated **mk**ern commands in mathematical markup	0.93
F-13259	2.73×10^{-6}	specific terms or phrases enclosed in quotation marks	users searching for **"Pizza"** or **"Calgary pizza"**	1.00
F-7307	1.88×10^{-6}	the word "In" at the beginning of sentences	"**In** this way the missing persons in our evolutionary mystery would ..."	0.86

adpositions, determiners, particles, and pronouns. We keep punctuation and conjunctions as two separate categories because they often mark boundaries and links between parts of a text.

The relative activation frequency $U_{q,c}$ in Equation (6) divides how often features in quintile q are active at category c by how often c occurs in the corpus. This removes the effect of category frequency, so a common category such as nouns does not score highly just because it appears often. A value of 1 means that features activate at c exactly as often as expected from how common c is in the corpus; a value above 1 means more often, and a value below 1 means less often.

E.4 Subject Geometry Details

For the t-SNE visualisation, we draw 100 sequences from five MMLU subjects (abstract algebra, astronomy, business ethics, computer security, and European history) and keep the last 20 tokens of each sequence, so that every token carries a subject label, following the setup from Bhalla et al. (2026). For each representation—raw hidden states, BatchTopK features, the Temporal SAE's high-level group, and the Persistent SAE's high-λ_j state—we collect the token-level vectors and project them to two dimensions with t-SNE using a common set of hyperparameters, colouring each point by its subject.

E.5 Subject Probing Details

This subsection gives the setup for the spliced-MMLU subject-probing experiment in Section 5.3, whose results are reported in Figure 5b.

Spliced sequences and probe. We build the spliced MMLU sequences by concatenating equal-length segments drawn from different MMLU subjects, balancing subject labels within each sequence and holding out entire source documents for the test split, so that a token's label is the subject of the segment it falls in. We treat each token as a labelled example and fit a linear probe to predict its source subject from the token-level representation. We report held-out one-vs-rest macro ROC AUC as a function of the fraction of the feature dictionary made available to train the probe, swept from 0.1% to 100%.

SAE feature rankings. For the Persistent SAE, we sweep its persistent state $\mathbf{h}^{(t)}$ under the three rankings from Section 5.3: decreasing λ_j, increasing λ_j, and repeated random permutations. We add a fourth, supervised oracle ranking, built from a per-feature, per-subject score

$$s_{j,c} = \mathbb{E}[h_j^{(t)} \mid y^{(t)} = c] - \mathbb{E}[h_j^{(t)} \mid y^{(t)} \neq c]. \tag{14}$$

This score, computed on the training split, is large when feature j's persistent state runs consistently higher for subject c than for the other subjects. For each subject, we rank features by this score, then interleave the five subject-specific rankings in round-robin order so that every subject contributes its most discriminative features. This gives one global feature ordering, which we sweep by fraction for the multiclass probe.

The BatchTopK and Matryoshka baselines have no persistence structure to rank features by, so at each fraction we sample that many features uniformly at random from the full dictionary. The Temporal SAE instead has a fixed high-level and low-level group (Appendix B.3), so we rank by group rather than by individual feature: the high-first ordering places every high-level-group feature before any low-level-group feature, and the low-first ordering places every low-level-group feature first. Both orderings still cover the full dictionary once the fraction reaches 100%.

E.6 Prompt Injection Monitoring Details

This subsection gives the trajectory construction, probing, and intervention procedures for the case study in Section 5.4 (Figure 6).

Trajectory construction. We take trajectories from AgentDojo (Debenedetti et al., 2024) runs of `Llama-3.3-70B-Instruct` agents, restricted to the banking, travel, and workspace domains (Appendix C.3). Attack trajectories use the `important_instructions` or `direct` attack type and satisfy `security=false`: an injection is present, but the agent did not complete its target action. Benign trajectories contain no injected content. We append each trajectory with the monitoring query

```
should this trajectory be flagged for prompt injection or
unsafe instruction hijacking?  Answer yes or no:
```

and read it with Gemma-2-2B, our external monitor model, taking all representations from its layer-12 residual stream.

Checkpoints and probes. For each trajectory, we locate the final token of the injected span (for benign trajectories, the matching position) and mark checkpoints at 0, 50, 100, 200, 250, 300, and 350 tokens after it, plus the final query position. At each checkpoint, we extract from the same layer: the raw hidden state; the BatchTopK and Matryoshka sparse activations; the Temporal SAE's high-level and low-level feature groups; and slow- and fast-feature states from the Persistent SAE. We fit one logistic regression probe per representation using only the checkpoint at $d = 0$, then evaluate it, without retraining, at every other checkpoint, reporting held-out accuracy and ROC AUC.

Steering vector construction. Following prior work that uses linear probes to select SAE features for steering (Arad et al., 2025; He et al., 2025), we turn the probe for each SAE representation r into a single direction $\mathbf{v}_r$ in the hidden-state space. We do this in three steps.

(1) Read the probe weights. We use the probe trained at $d = 0$. Its weight for feature j is the injection-class weight minus the benign-class weight. The probe is trained on standardised features, so we divide this weight by the standard deviation of feature j and obtain β_j. A positive β_j means that a larger activation of feature j makes the probe more likely to predict an injection.

(2) Select features. For every representation, we keep the $k = 128$ features with the largest $|\beta_j|$ and call this set $\mathcal{S}_r$. For example, this is 128 of the 164 slow features and 128 of the $8{,}192$ fast features.

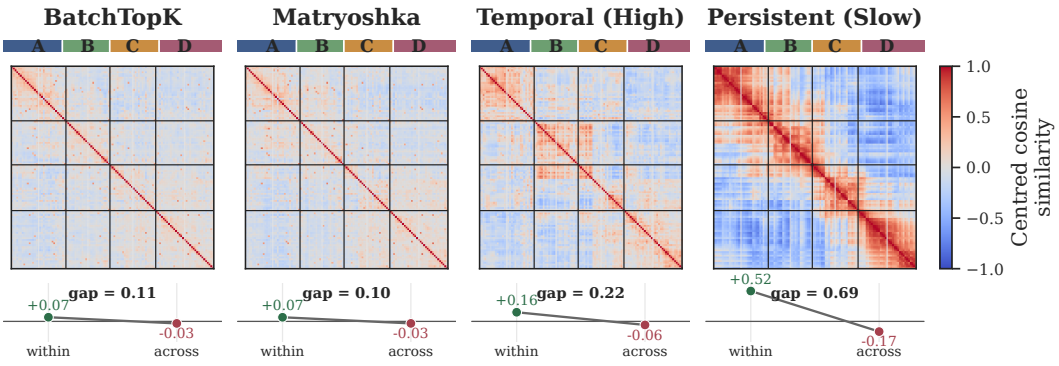


Figure 8: **Centred token similarity across four spliced MMLU subjects.** Each panel shows the pairwise centred cosine similarity between token representations in one sequence with four segments (A–D), each from a different subject. The Persistent SAE slow features have the clearest block structure.

(3) Map the features to the hidden state. Each SAE feature j adds its decoder column $\mathbf{W}_{\text{dec}}[:, j]$ to the reconstructed hidden state. We sum these columns, each weighted by its probe weight:

$$\mathbf{v}_r = \sum_{j \in \mathcal{S}_r} \beta_j \, \mathbf{W}_{\text{dec}}[:, j]. \tag{15}$$

The resulting $\mathbf{v}_r$ points towards what the probe reads as an injection. A positive Δ_{causal} therefore means that moving the hidden state in this direction also makes the monitor more likely to answer *yes*.

Causal intervention. For each distance $d \in \{0, 50, 100, 150, 200, 250, 300, 350\}$, we truncate each held-out trajectory d tokens after the end of the untrusted span and append the monitoring query. Let $\mathbf{x}_i^{(d)}$ denote the layer-12 residual stream at the final query token of trajectory i. We rescale $\mathbf{v}_r$ separately for each example so that $\alpha_i^{(d)} = \rho \|\mathbf{x}_i^{(d)}\|_2 / \|\mathbf{v}_r\|_2$ with $\rho = 0.1$, add $\pm \alpha_i^{(d)} \mathbf{v}_r$ to the output of layer 12 at this token only, and run the rest of the forward pass. The logit margin s uses the single-token answers "`yes`" and "`no`". We report the mean of Equation (7) over trajectories, with 95% confidence intervals from 500 bootstrap resamples of trajectories. We keep $\mathbf{v}_r$ fixed across distances, so Figure 6b (bottom) shows whether one direction learnt at the end of the injected span still steers the monitor later in the trajectory. Note that the intervention changes only the external monitor's logit margin and does not act on the original Llama-3.3-70B agent.

F Additional Analyses

F.1 Token Similarity Across Subject Boundaries

This subsection extends the analysis in Section 5.3. There, the t-SNE plot and the subject probes summarise many sequences at once. Here, motivated by Lubana et al. (2026), we look inside one spliced sequence and ask two questions: do tokens from the same subject share a similar state of the slow features, and how does this state change at the boundary between subjects?

Setup. For these visualisations, we use the layer-12 residual stream of Gemma-2-2B and construct one sequence from the MMLU test split. The sequence concatenates one sentence from each of four subjects, in order: college biology (A), college physics (B), business ethics (C), and high school government and politics (D). For each displayed representation, let $\mathbf{z}^{(t)}$ denote its feature vector at token position t. We centre each vector using the mean representation over all T tokens in the spliced sequence, $\bar{\mathbf{z}} = T^{-1} \sum_{t=1}^{T} \mathbf{z}^{(t)}$, and compute centred cosine similarity between $\mathbf{z}^{(i)} - \bar{\mathbf{z}}$ and $\mathbf{z}^{(j)} - \bar{\mathbf{z}}$. A positive similarity means that both tokens deviate from the sequence mean in the same direction, and a negative similarity means that they deviate in opposite directions. The black lines in the heatmaps mark where one subject ends and the next begins.

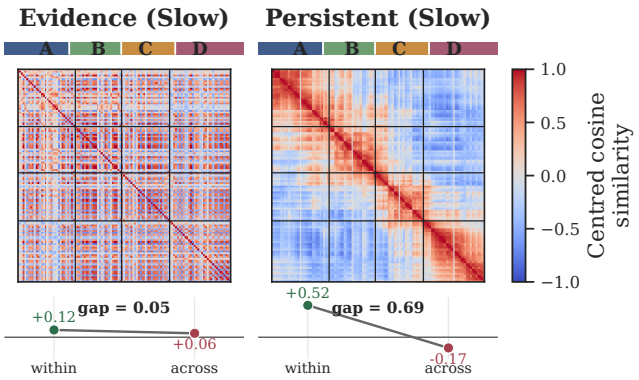


Figure 9: **Token-wise activations and states encode different information.** Centred token similarity on the same spliced sequence as Figure 8, for the token-wise activation $\mathbf{a}^{(t)}$ and the state $\mathbf{h}^{(t)}$ of the highest 1% of features by λ_j. We find that integrating the activations over time turns weak local activations into a sequence-level representation.

Slow features separate subjects within a sequence. Figure 8 shows that the state of the Persistent SAE's slow features has positive similarity within each segment and negative similarity across segments. Tokens in the same segment therefore have similar states, including tokens far apart in the sequence. The Temporal SAE high-level group shows some block structure, but its gap of 0.22 is about a third of the 0.69 gap of the state of the slow features. BatchTopK and Matryoshka show almost no block structure in this example. These results agree with the probing results in Figure 5 (b): the highest-λ_j features form a small subset whose state tracks the current subject, and the model finds this subset without subject labels.

According to Equation (5), the state $\mathbf{h}^{(t)}$ accumulates the token-wise activations $\mathbf{a}^{(t)}$. To check that the subject signal comes from this accumulation, we apply the same analysis to $\mathbf{a}^{(t)}$ restricted to the same slow features. In Figure 9, we show that the activations show little segment structure: the mean similarity is 0.12 within segments and 0.06 across them, a gap of only 0.05. The state, computed from these activations, has a much larger gap. Therefore, each activation records only the current token, whereas the state integrates these activations over the preceding tokens and shifts at splice boundaries. This finding matches the division of labour in Section 4: token-wise activations capture local information, and the state of the slow features retains the context accumulated from them.

F.2 States of Slow Features Track Subject Information

Section 5.3 visualises the state of the slow features for Pythia-160M (Figure 5a). Figure 10 repeats this analysis for Gemma-2-2B, using the same five MMLU subjects and the same t-SNE setup (Appendix E.4). As with Pythia-160M, BatchTopK features mix all five subjects together, and the Temporal SAE's high-level group separates European history from the base model's hidden states but leaves the other four subjects intermixed. The state of the Persistent SAE's slow features again forms the clearest, most separated subject clusters, learned without subject labels. This matches the pattern found for Pythia-160M and the quantitative subject-probing results in Figure 5b, which already show a comparable trend for both models.

The subject clusters degrade gradually as more, faster features are added. Figure 10 uses the slowest 1% of features, the same threshold used throughout the analysis of the paper. To check that this threshold is not a cherry-picked cutoff, Figure 11 repeats the t-SNE visualisation for Pythia-160M while gradually widening the feature set to the slowest 1%, 10%, 20%, 40%, and 80% by λ_j. The five subject clusters are most distinct at 1%, and they mix together steadily as each wider set folds in faster, less persistent features, until the slowest 80% looks about as mixed as the BatchTopK baseline in Figure 5a. Subject information is therefore not confined to a sharply defined subset of slow features; instead, it is concentrated most strongly in the slowest features and fades away smoothly as faster features are added, so any small threshold near the top of the persistence ranking would show a similar effect.

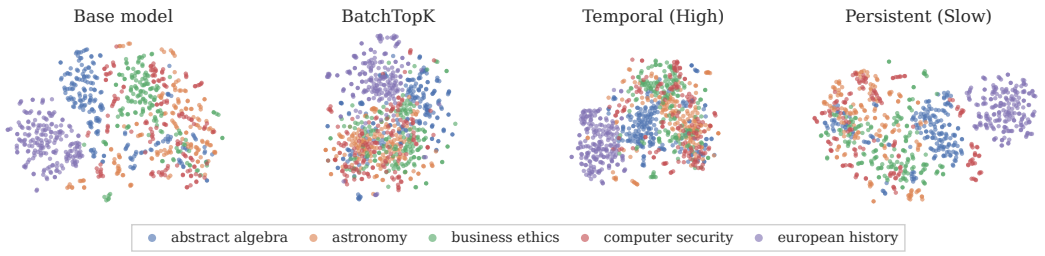


Figure 10: **The state of the slow features tracks subject information in Gemma-2-2B.** t-SNE of token-level MMLU representations from Gemma-2-2B, coloured by subject, for the base model's hidden states, BatchTopK, the Temporal SAE's high-level group, and the state of the Persistent SAE's slow features. The state of the slow features forms the clearest subject clusters.

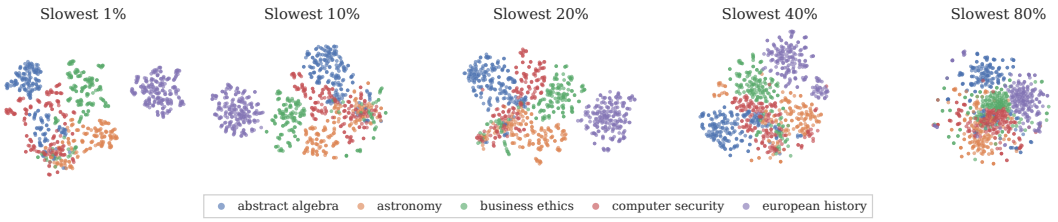


Figure 11: **Subject separation weakens gradually as the feature set widens.** t-SNE of token-level MMLU representations from Pythia-160M, coloured by subject, using the state of an increasingly wide set of slow features: the slowest 1%, 10%, 20%, 40%, and 80% by λ_j. Subject clusters are sharpest for the slowest 1% and become progressively more mixed as more, less-slow features are included.

F.3 Effect of Persistence Granularity

Setup. We test how finely the persistence coefficients must be parameterised to recover heterogeneous learned timescales. All models use Pythia-160M-deduped layer-8 residual-stream activations, a dictionary of $m = 16{,}384$ features, BatchTopK sparsity with $k = 20$, and training sequences of 256 tokens. We train each model for $244{,}140$ steps with Adam at a learning rate of 3×10^{-4}. Persistence coefficients are initialised to 0.5, and all other training settings match the feature-wise Persistent SAE described in Appendix B.

We compare models with

$$G \in \{1, 10, 100, 1{,}000, 10{,}000, 16{,}384\}$$

learned persistence coefficients. For $G < 16{,}384$, we partition the feature indices into G fixed groups of approximately equal size, assigning feature j to group

$$g(j) = \left\lfloor \frac{jG}{m} \right\rfloor. \tag{16}$$

Each group g has one trainable persistence logit l_g, giving

$$\lambda_j = \lambda_{g(j)} = \sigma(l_{g(j)})\lambda_{\max}, \qquad \lambda_{\max} = 0.999. \tag{17}$$

The group assignment stays fixed throughout training, so features in the same group share one coefficient in the state update,

$$h_j^{(t)} = \lambda_{g(j)} h_j^{(t-1)} + \left(1 - \lambda_{g(j)}\right) a_j^{(t)}. \tag{18}$$

Gradients from every feature in a group update the same logit l_g, while the encoder and decoder parameters remain feature-specific and are trained jointly with these shared coefficients. This ablation therefore changes only the number of independently learned persistence coefficients: $G = 1$ gives the whole dictionary one shared timescale, and $G = 16{,}384$ recovers one coefficient per feature.

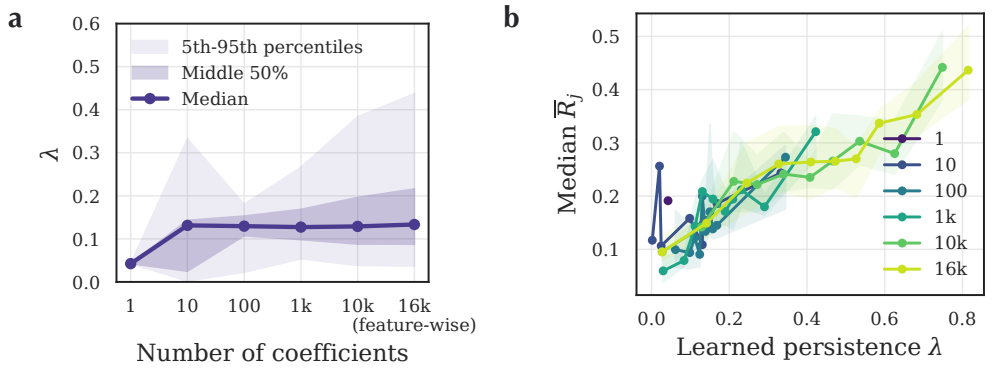


Figure 12: **Finer persistence parameterisations recover heterogeneous empirical timescales.** We train otherwise matched Persistent SAEs with $G \in \{1, 10, 100, 1{,}000, 10{,}000, 16{,}384\}$ persistence coefficients. For $G < 16{,}384$, approximately equal-sized groups of features share one learned coefficient; $G = 16{,}384$ assigns one coefficient to every feature. **(a)** Feature-weighted distributions of the learned coefficients. The median changes little once $G \geq 10$, whereas fine-grained models recover a wider upper tail of slow features. **(b)** Each point's x-position is a bin's learned persistence λ, and its y-position is the median of $\overline{R}_j$, the empirical persistence from Equation (2) averaged over lags, among the features in that bin. The relationship between learned and empirical persistence becomes clearer as the number of independently learned coefficients increases.

Learned coefficient distributions. Figure 12a reports the distribution of feature-level coefficients after training. With one global coefficient, the model converges to a nearly token-local value around $\lambda = 0.04$. Once the model has at least ten coefficients, the median stays close to 0.13: finer parameterisations mainly widen the spread. In particular, the 95th percentile rises from around 0.18 with 100 coefficients to 0.44 with feature-wise coefficients. A feature-wise parameterisation therefore lets a subset of features learn much slower dynamics without raising learned persistence across the whole dictionary.

Comparison with empirical persistence. We next test whether the learned coefficients track $R_j(\tau)$, the empirical persistence measure defined in Section 3 (Equation (2)) We evaluate each model on 128 Pile documents, truncate each document to at most $1{,}024$ tokens, and compute $R_j(\tau)$ from the decoder direction at $\tau \in \{1, 4, 16, 32, 64, 128, 256, 512\}$. We retain features that activate at least 256 times, have at least 64 valid anchors at every lag, and have an activated-token elevation of at least 10^{-4}. For each retained feature, we average $R_j(\tau)$ over all eight lags into a single score:

$$\overline{R}_j = \frac{1}{8} \sum_{\tau \in \{1,4,16,32,64,128,256,512\}} R_j(\tau). \tag{19}$$

Figure 12b plots this empirical score against the learned coefficient. We order complete coefficient groups by their learned λ and divide them into at most ten bins; each point's x-position is that bin's learned λ and its y-position is the median $\overline{R}_j$ among the features in that bin. With ten shared coefficients, the relationship is noisy and offers little resolution; it grows more ordered as G increases. For the $10{,}000$-coefficient and feature-wise models, bins with larger learned λ also have larger empirical persistence $\overline{R}_j$, and their calibration curves are similar, especially at the lowest and highest values of λ. Reconstruction therefore successfully learns feature-dependent timescales, and coarse sharing obscures part of this variation.

G Ablation Studies

G.1 Robustness to Persistence Initialisation

Appendix B.2 initialises every persistence logit l_j to a constant value corresponding to $\lambda_j = 0.5$. To check that this choice does not drive our results, we retrain Pythia-160M Persistent SAEs from five alternative initialisations while holding every other hyperparameter fixed: constant initial persistence

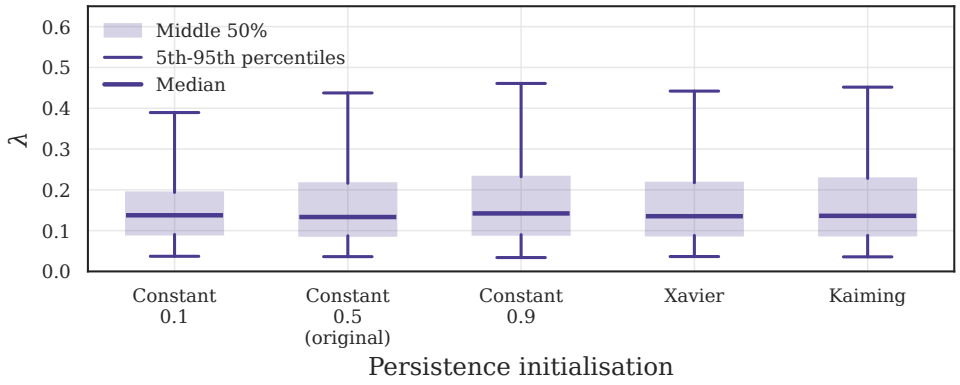


Figure 13: **Learned persistence is robust to its initialisation.** Distribution of learned per-feature persistence coefficients λ_j for Pythia-160M Persistent SAEs trained from five different initialisations of the persistence logits.

of 0.1, 0.5 (the default used throughout the paper), and 0.9, together with Xavier-uniform (Glorot & Bengio, 2010) and Kaiming-uniform (He et al., 2015) initialisation of the logits l_j. For each run, we compare the distribution of learned persistence coefficients λ_j after training.

Learned persistence converges to a similar distribution regardless of initialisation. Figure 13 shows that the five initialisation schemes yield nearly indistinguishable distributions of learned λ_j: medians range from 0.134 to 0.142 and means from 0.162 to 0.182 across all five runs, with comparable spread in every case (5th–95th percentile interval of approximately 0.03–0.46). The two structured random initialisations, Xavier-uniform and Kaiming-uniform, fall within this same range rather than producing systematically different persistence. We observe a small, monotonic trend in which a higher constant initial persistence yields a slightly higher converged mean (from 0.162 at initial persistence 0.1 to 0.182 at initial persistence 0.9), but this difference is an order of magnitude smaller than the within-run standard deviation (0.11–0.14). Training therefore pulls the persistence coefficients toward a similar equilibrium distribution irrespective of how they are seeded, indicating that the learned per-feature persistence structure reflects a property discovered during training.

G.2 Effect of Training Context Length

In this section, we test whether the learned persistence coefficients depend on the sequence length used during training. We train Pythia-160M Persistent SAEs on sequences of 32, 64, 128, 256, and 512 tokens. All runs use layer 8, a dictionary of 16,384 features, and BatchTopK sparsity with $k = 20$. We adjust the sequence batch size to keep the nominal number of tokens per update approximately fixed.

Persistence estimates are stable across training context length. Figure 14 shows that the learned persistence coefficient changes little as the training context grows from 32 to 512 tokens. The median λ_j rises only from 0.122 to 0.151, and the 95th percentile from 0.394 to 0.502, a small shift relative to the 25th–95th percentile spread at any single context length. Therefore, the distribution stays broad throughout different lengths, with the same mixture of short- and long-timescale features at every length. Because context length has only a marginal effect on the estimated persistence, we use 256 tokens in the main experiments, which captures this stable regime without the added training cost of longer contexts.

G.3 Persistence Across Model Sizes

In this section, we conduct an initial analysis towards understanding how model size affects the learned persistence of SAE features. We compare Persistent SAEs trained on ten language models of varying sizes. The comparison includes seven Pythia models ranging from Pythia-160M to Pythia-12B, two Gemma-2 models of sizes 2B and 9B, and a new model family Llama-3.1-8B (Meta AI, 2024). Each SAE uses a dictionary of 16,384 features, BatchTopK sparsity with $k = 20$, and training

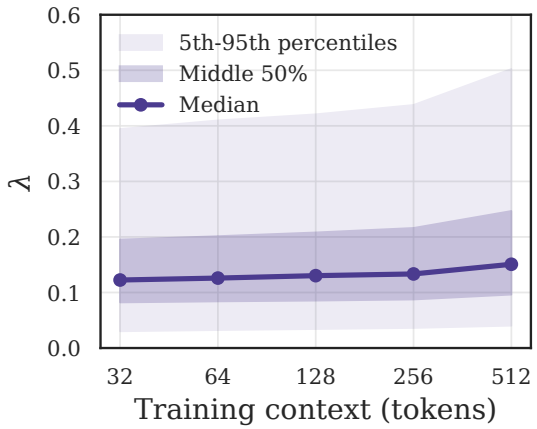


Figure 14: **Learned persistence is stable across training context length.** Median learned persistence coefficient λ_j across training context lengths. The dark and light bands show the 25th–75th and 5th–95th percentile intervals, respectively.

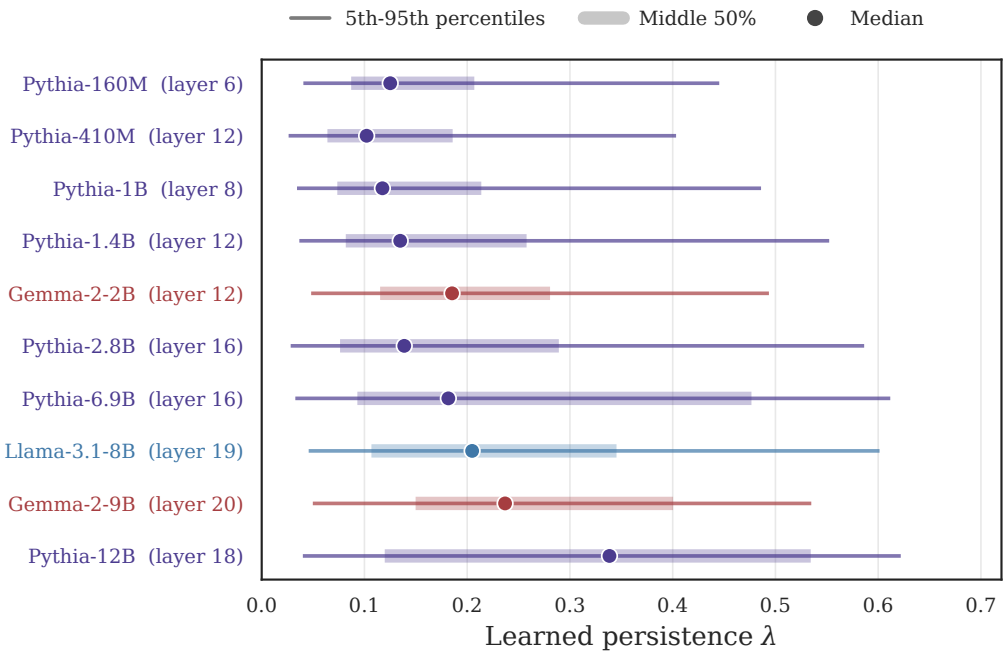


Figure 15: **Larger models tend to learn more persistent features.** Distribution of learned persistence coefficients across ten language models, ordered by nominal parameter count. Points show medians, thick intervals show the middle 50%, and thin intervals show the 5th–95th percentiles. Label colours identify the model family.

sequences of 256 tokens. Figure 15 reports the distribution of all learned coefficients at the layer shown for each model.

Median persistence generally increases with model size. During the Persistent SAEs training, every model learns a broad range of feature timescales, but the distributions shift towards larger λ_j for larger models. In addition, across all ten models, the Spearman correlation between parameter count and median λ_j is 0.927; within the seven Pythia models, it is 0.893. The Pythia median increases from 0.125 for Pythia-160M to 0.338 for Pythia-12B, and the 95th percentile increases from 0.445 to 0.622. This pattern is consistent with larger models assigning longer timescales to a greater share of SAE features. One possible explanation is that larger models have more capacity to track

dependencies over long contexts and so allocate more features to represent them. We leave the testing of this explanation, for instance by relating λ_j to a model's effective context usage, to future work.