RESIDUAL EXPERTS FOR SPARSE AUTOENCODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse Autoencoders (SAEs) are widely used to extract interpretable features from language models, but scaling full-dictionary SAEs to capture diverse features in large models makes sparse autoencoding computationally challenging. We introduce **ResSAE**, a two-stage sparse autoencoder that combines conditional computation with residual refinement. A lightweight Level-1 SAE forms an initial reconstruction and uses its active features to select *residual experts*, small Level-2 SAEs that encode the reconstruction residual. A joint TopK operation allocates the Level-2 sparsity budget across the selected experts. Experiments on Gemma-2-2B residual stream activations show that ResSAE achieves lower reconstruction MSE and higher CE/KL fidelity than representative efficient SAE baselines at matched effective dictionary widths, sparsity levels, and encoder FLOPs. Compared with 16K-feature full-dictionary baselines, ResSAE accesses a 135K-feature space with 56–74% fewer encoder FLOPs while achieving up to 11% lower reconstruction MSE. Controlled ablations show that both residual input and Level-1 routing contribute to the reconstruction gains of residual experts. ResSAE further exhibits an emergent coarse-to-fine feature organization, with reduced feature splitting and absorption while maintaining auto-interpretability comparable to width-matched full-dictionary SAEs. Code and checkpoints will be released.

1 INTRODUCTION

Sparse Autoencoders (SAEs) have become a central unsupervised tool for mechanistic interpretability, decomposing dense and superposed language-model activations into sparse feature directions that are often human-interpretable (Olah et al., 2020; Elhage et al., 2022; Bricken et al., 2023; Cunningham et al., 2023). These features provide an interface for analyzing and intervening on model computation, including feature discovery, steering, and circuit-level analysis (Templeton et al., 2024; Gao et al., 2025; Marks et al., 2025).

However, as modern language models encode increasingly diverse representations, SAEs require wider dictionaries to capture this feature diversity while preserving reconstruction fidelity (Templeton et al., 2024; Gao et al., 2025). We use *flat SAE* to refer to standard SAE architectures in which all latents are organized as a single global dictionary and a single encoder scores the full dictionary before sparsification. This flat architecture creates two difficulties as dictionary width increases. First, the encoder must score all latents before selecting active ones, making wider dictionaries computationally expensive (Gao et al., 2025; Mudide et al., 2025). Second, flat sparsity objectives can degrade feature organization by splitting general features into narrower variants or absorbing them into specialized latents, leaving high-level features missing, warped, or unreliable (Bricken et al., 2023; Chanin et al., 2025; Bussmann et al., 2025; Leask et al., 2025).

To address these two limitations, we introduce **ResSAE**, a two-stage sparse autoencoder that uses an initial sparse representation to guide residual encoding. We first use a lightweight Level-1 SAE with a small dictionary to obtain an initial reconstruction $\hat{x}_1$. We associate each Level-1 feature with a small Level-2 SAE, termed a *residual expert*, and use the active Level-1 features to select which experts encode the residual $r = x - \hat{x}_1$. By restricting Level-2 feature scoring to the selected experts, we reduce encoding cost. Furthermore, we design this division of roles to preserve shared structure in Level-1 features while allowing residual experts to capture more specific variation, with the aim of improving feature organization. We apply a joint TopK operation across the selected experts' features to allocate a shared Level-2 sparsity budget, and form the final reconstruction as $\hat{x} = \hat{x}_1 + \hat{r}$.

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.

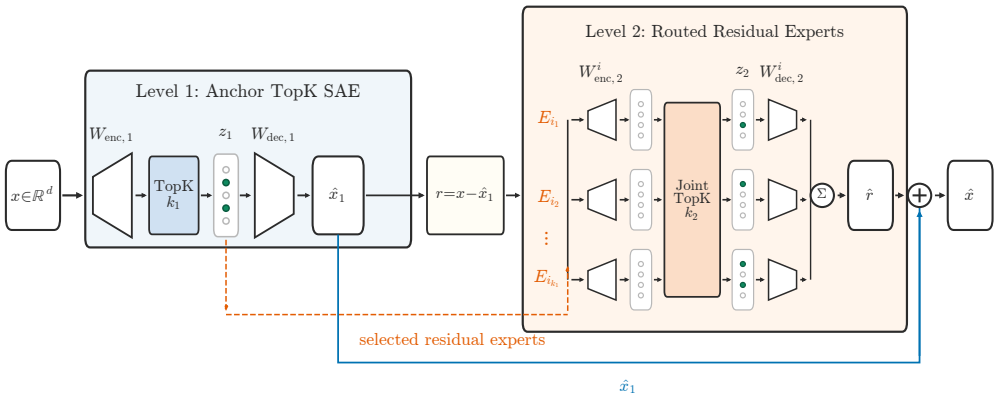

Figure 1: Overview of ResSAE. Level-1 encodes the activation x into a sparse code z_1 and produces an initial reconstruction $\hat{x}_1$. Its active features select a small set of residual experts. A joint TopK operation allocates the Level-2 sparsity budget across the selected experts, producing a residual reconstruction $\hat{r}$. The final reconstruction is $\hat{x} = \hat{x}_1 + \hat{r}$.

This residual-expert architecture directly realizes the two design principles above. First, conditional expert access decouples effective dictionary capacity from per-input encoder cost: ResSAE scores a compact Level-1 dictionary and only the Level-2 experts selected by its active features, rather than a full global dictionary for every token. Second, residual refinement separates initial reconstruction from fine-grained correction: Level-1 reconstructs the input, while Level-2 experts encode only what remains unexplained. Empirically, this decomposition encourages coarse-to-fine feature organization and reduces feature splitting and absorption, despite using no explicit semantic hierarchy constraints.

We evaluate ResSAE on Gemma-2-2B residual stream activations and report four main findings. First, under matched effective width, sparsity, and encoder FLOPs, ResSAE outperforms representative efficient SAE baselines in reconstruction and CE/KL fidelity. Second, conditional access to residual experts allows ResSAE to use a much larger effective dictionary at lower encoder cost: compared with 16K-feature flat SAEs, ResSAE accesses a 135K-feature space with 56–74% fewer encoder FLOPs and achieves up to 11% lower reconstruction MSE. Third, controlled ablations show that the two components underlying residual experts, residual input and Level-1 routing, substantially improve reconstruction, supporting their complementary roles in identifying what remains unexplained and where it should be refined. Finally, ResSAE exhibits an emergent coarse-to-fine feature organization and, relative to width-matched flat SAEs, reduces feature splitting and absorption without explicit semantic hierarchy constraints. It also maintains comparable auto-interpretability.

Our contributions are as follows:

1. We introduce ResSAE, a two-stage SAE in which Level-1 features jointly provide the initial reconstruction and route its residual to specialized Level-2 residual experts (Section 3).
2. We show that conditional access to residual experts outperforms representative efficient SAE baselines under matched resource settings and supports substantially larger effective dictionaries at lower encoder cost (Section 4.2).
3. Through controlled interventions, we show that residual input and Level-1 routing provide complementary reconstruction benefits. Their combination in residual experts achieves lower reconstruction error than variants using the original activation as input or random routing (Section 4.3).
4. We find that ResSAE learns an emergent coarse-to-fine feature organization with reduced splitting and absorption, while maintaining auto-interpretability comparable to width-matched flat SAEs (Section 4.4).

2 Related Work

Sparse autoencoders and sparsification. SAEs decompose language-model activations into sparse combinations of learned feature directions (Bricken et al., 2023; Cunningham et al., 2023). TopK and BatchTopK impose per-input or batch-level activation budgets (Gao et al., 2025; Bussmann et al., 2024); Gated SAEs separate feature selection from magnitude estimation (Rajamanoharan et al., 2024a); and JumpReLU SAEs learn feature-specific activation thresholds (Rajamanoharan et al., 2024b). These methods score a global dictionary before sparsification, whereas ResSAE uses two-stage encoding with conditional dictionary access.

Efficient and conditional sparse autoencoding. Switch SAE reduces encoder computation by routing each activation to an expert SAE (Mudide et al., 2025), while KronSAE uses a factorized, compositional parameterization (Kurochkin et al., 2025). H-SAE uses high-level features to select associated low-level SAEs, which encode learned low-dimensional projections of the original activation (Muchane et al., 2025). ResSAE shares this feature-based conditional access, but its experts encode the reconstruction residual $r = x - \hat{x}_1$. Moreover, a joint TopK operation allocates a shared sparsity budget across routed experts, whereas H-SAE selects features within each expert separately. Table 1 summarizes the architectural differences between ResSAE, Switch SAE, and H-SAE.

Residual refinement and SAE ensembles. MP-SAE unrolls matching pursuit, repeatedly scoring a shared dictionary against the current residual (Costa et al., 2025). SAE Boost trains a branch to predict residual errors from the original activation while keeping the pretrained SAE frozen (Koriagin et al., 2025). Gadgil et al. (2026) study bagging and boosting ensembles, including sequentially trained SAEs that encode residuals left by earlier stages. ResSAE instead restricts residual encoding to experts selected by Level-1 features and trains both stages jointly, with gradients propagating through the main residual path into Level-1. Table 1 summarizes the architectural differences across these methods.

Table 1: Architectural comparison of selected conditional and residual SAE methods. ✓/✗ indicate whether a mechanism is present/absent; — denotes not applicable.

Method	Conditional expert access	Feature-based routing	Residual encoder input	Joint training	Shared expert budget
Switch SAE (Mudide et al., 2025)	✓	✗	✗	✓	—
H-SAE (Muchane et al., 2025)	✓	✓	✗	✓	✗
MP-SAE (Costa et al., 2025)	✗	✗	✓	✓	—
SAE Boost (Koriagin et al., 2025)	✗	✗	✗[†]	✗	—
Boosting ensemble (Gadgil et al., 2026)	✗	✗	✓	✗	—
ResSAE (ours)	✓	✓	✓	✓	✓

Definitions. Conditional access selects experts before feature scoring; feature-based routing uses sparse reconstruction features for selection. Residual input encodes the remaining reconstruction error. Joint training optimizes all SAE components together. Shared budget applies one TopK across multiple routed experts; — indicates no routing or single-expert selection. H-SAE uses expert-local TopK_1. [†]SAE Boost encodes x to predict a residual with its base SAE frozen. The ensemble row denotes sequential boosting.

Multi-level feature organization. Feature splitting, absorption, and composition complicate the interpretation of SAE dictionaries (Bricken et al., 2023; Chanin et al., 2025; Anders et al., 2024; Wattenberg & Viégas, 2024; Leask et al., 2025). Matryoshka SAEs encourage multiple levels of specificity through nested reconstruction objectives over dictionary prefixes (Bussmann et al., 2025). H-SAE organizes features across levels by associating high-level features with conditional low-level SAEs (Muchane et al., 2025). Luo et al. (2026) jointly learn multiple dictionaries and a feature forest using parent–child consistency objectives, while Tree SAE learns parent–child assignments through activation gating and hierarchical reconstruction constraints (Cao et al., 2026). ResSAE combines feature-based routing with residual refinement: Level 1 forms an initial reconstruction and selects Level 2 experts to encode the remaining residual. We impose no explicit semantic hierarchy loss and evaluate coarse-to-fine feature organization empirically. Appendix D provides a broader architectural comparison.

3 METHOD

We first describe the flat SAE baseline and then introduce ResSAE. We subsequently present the training objective and analyze encoder complexity.

3.1 FLAT SAE

Let $x \in \mathbb{R}^d$ be an activation vector. A flat SAE with N latents scores the full dictionary before sparsification: $a = W_{\text{enc}}(x - b_{\text{dec}}) + b_{\text{enc}}$, $z = \phi(a)$, and $\hat{x} = W_{\text{dec}}^\top z + b_{\text{dec}}$, where $W_{\text{enc}}, W_{\text{dec}} \in \mathbb{R}^{N \times d}$ and ϕ denotes a sparsification rule such as ReLU, TopK, BatchTopK, or JumpReLU. Although z is sparse, all N latent scores must be computed before sparsification, giving an encoder cost of $O(dN)$ per input.

3.2 RESSAE

ResSAE instantiates *residual experts* in a two-stage architecture. As shown in Figure 1, Level 1 first forms an initial reconstruction $\hat{x}_1$. The same Level-1 feature selection then determines which Level-2 experts are evaluated, while those experts encode only the remaining residual $r = x - \hat{x}_1$.

Let N_1 denote the Level-1 dictionary size, N_2 the number of features per Level-2 expert, and k_1 and k_2 the Level-1 and Level-2 sparsity budgets, respectively. Using a shared decoder bias $b_{\text{dec}} \in \mathbb{R}^d$, Level 1 computes $a_1 = W_{\text{enc},1}(x - b_{\text{dec}}) + b_{\text{enc},1}$, $z_1 = \text{TopK}_{k_1}(a_1)$, and $\hat{x}_1 = W_{\text{dec},1}^\top z_1 + b_{\text{dec}}$, where $W_{\text{enc},1}, W_{\text{dec},1} \in \mathbb{R}^{N_1 \times d}$.

Let $\mathcal{I} = \text{TopKIdx}_{k_1}(a_1)$ denote the set of latent indices selected by the Level-1 TopK operation, with $|\mathcal{I}| = k_1$. These indices serve two roles: they identify the active Level-1 features for the initial reconstruction and select the corresponding Level-2 residual experts. Thus, no separate learned router is required.

Each selected expert contains N_2 residual features. Let $W_{\text{enc},2}^{\mathcal{I}}, W_{\text{dec},2}^{\mathcal{I}} \in \mathbb{R}^{k_1 N_2 \times d}$ and $b_{\text{enc},2}^{\mathcal{I}} \in \mathbb{R}^{k_1 N_2}$ denote the concatenated parameters of the routed experts. Level 2 operates only on the residual:

$$\begin{aligned} r &= x - \hat{x}_1, \quad z_2 = \text{TopK}_{k_2}\left(W_{\text{enc},2}^{\mathcal{I}} r + b_{\text{enc},2}^{\mathcal{I}}\right), \\ \hat{r} &= \left(W_{\text{dec},2}^{\mathcal{I}}\right)^\top z_2, \quad \hat{x} = \hat{x}_1 + \hat{r}. \end{aligned} \tag{1}$$

The joint TopK operation ranks all $k_1 N_2$ Level-2 feature activations from the routed experts under a shared residual sparsity budget k_2. Thus, different experts can contribute different numbers of active features, allowing the residual sparsity budget to be allocated adaptively according to the current input residual. Level 2 uses no decoder bias. The shared bias b_{dec} already accounts for the global activation offset in the Level-1 reconstruction, while Level 2 predicts only the residual correction $\hat{r}$. A reference implementation of the reconstruction path is provided in Appendix A.8.

3.3 TRAINING OBJECTIVE

ResSAE is trained with reconstruction losses at both levels together with AuxK-style losses for dead-feature mitigation:

$$\begin{aligned} \mathcal{L} &= \alpha \mathcal{L}_1 + \beta \mathcal{L}_{\text{global}} + \lambda_1 \mathcal{L}_{\text{aux},1} + \lambda_2 \mathcal{L}_{\text{aux},2}, \\ \mathcal{L}_1 &= \|x - \hat{x}_1\|_2^2, \qquad \mathcal{L}_{\text{global}} = \|x - \hat{x}\|_2^2. \end{aligned} \tag{2}$$

The residual $r = x - \hat{x}_1$ is not detached in the main reconstruction path, so gradients from $\mathcal{L}_{\text{global}}$ can update Level 1 through the reconstruction path. The discrete expert indices themselves are not differentiated through. The Level-1 loss preserves a meaningful initial reconstruction rather than allowing Level 1 to delegate reconstruction entirely to Level 2.

For Level 1, $\mathcal{L}_{\text{aux},1}$ follows the standard TopK SAE AuxK objective (Gao et al., 2025). For Level 2, dead-feature tracking is conditioned on routing: a feature's inactivity counter is updated only when its expert is selected and is reset whenever the feature activates. The auxiliary loss uses the stop-gradient target $\text{sg}(r - \hat{r})$ and, to reduce its cost, is evaluated at one uniformly sampled routed expert per step and scaled by k_1, an unbiased estimate of the loss over all selected experts (details in Appendix A).

3.4 Encoder Complexity

A flat SAE with N latents scores all N features and therefore has encoder cost $O(dN)$. A flat dictionary with the same nominal effective width as ResSAE would contain $N_1 + N_1N_2$ latents, whereas ResSAE evaluates all N_1 Level-1 features but only k_1N_2 routed Level-2 candidates. Its encoder cost is therefore $O(d(N_1 + k_1N_2))$, $\frac{\text{FLOPs}_{\text{ResSAE}}}{\text{FLOPs}_{\text{flat}}} = \frac{N_1+k_1N_2}{N_1+N_1N_2}$. For $d = 2304$, $N_1 = 4096$, $N_2 = 32$, and $k_1 = 12$, ResSAE evaluates only $N_1 + k_1N_2 = 4480$ latent candidates per token, requiring 20.6M encoder FLOPs. In contrast, a full-dictionary encoder with the same effective width $N_{\text{eff}} = 135{,}168$ would require 622.9M FLOPs. This analytical comparison illustrates the reduction in active encoder computation; empirical comparisons with trained baselines are reported in Section 4.2.

4 Experiments

We organize our experiments around three main questions. First, do residual inputs and Level-1 feature-based routing play complementary roles in residual expert learning, and what roles do they play? Second, how does ResSAE compare with efficient and flat SAE baselines in computational cost, reconstruction quality, and downstream fidelity across dictionary sizes and sparsity levels? Third, does the residual expert structure improve feature quality, including feature splitting, absorption, and routing-conditioned specialization? We additionally study practical efficiency (Appendix B.2.1) and robustness across layers, seeds, and models (Appendix B.2.2).

4.1 Experimental Setup

Our experiments use Gemma-2-2B layer-12 residual-stream activations (`blocks.12.hook_resid_post`, $d = 2304$), following the activation setting used in recent SAE benchmarks and architectures (Bussmann et al., 2025; Kurochkin et al., 2025). Unless otherwise stated, models are trained with 500M tokens from the train split of the uncopyrighted Pile (Gao et al., 2020) (`monology/pile-uncopyrighted` on Hugging Face), materialized once as a shared local snapshot. Held-out evaluations use OpenWebText (`Skylion007/openwebtext`), a corpus disjoint from the training data. We describe the main experimental settings, comparison protocols, and evaluation metrics below; full training configurations and additional SAE-Bench metrics are provided in Appendix A.

Measured L_0 denotes the average number of nonzero latents per token and may differ from the nominal budget. For ResSAE, the nominal budget is $k_1 + k_2$; captions identify whether reported sparsities are target or measured values. Reconstruction quality is measured by the mean squared Euclidean error $\mathbb{E}_x\|x - \hat{x}\|_2^2$ or its per-dimension value $\mathbb{E}_x\|x - \hat{x}\|_2^2/d$, as indicated by the figure or table. Encoder FLOPs count dense encoder multiplication and addition operations per token. Downstream fidelity is evaluated after replacing activations with SAE reconstructions. Lower CE loss indicates better fidelity, while higher normalized CE and KL fidelity scores indicate better preservation of model behavior.

4.2 Comparison with Existing SAEs

We compare ResSAE with efficient and hierarchical baselines (Switch SAE, H-SAE, and KronSAE) and flat SAE baselines (TopK, JumpReLU, BatchTopK, Gated, and MatryoshkaBatchTopK).

Comparison with efficient and hierarchical architectures. Table 2 compares ResSAE with Switch SAE, H-SAE, and KronSAE on Gemma-2-2B layer-12 activations after 100M training tokens, except the $L_0 \approx 40$ ResSAE row, which uses a 500M-token checkpoint of the same configuration (Appendix A.3). The two blocks correspond to measured L_0 values near 40 and 80, with approximately matched total feature counts and encoder FLOPs within each block. ResSAE achieves the lowest reconstruction MSE and the highest normalized CE and KL fidelity scores in both settings. At $L_0 \approx 40$, it reduces MSE by 8.7%, 19.6%, and 18.6% relative to Switch SAE, H-SAE, and KronSAE, respectively. At $L_0 \approx 80$, the corresponding reductions are 0.8%, 13.4%, and 8.9%.

Comparison with flat baselines. We next examine how conditional residual expert access changes the relationship between reconstruction error, computation, and dictionary size. At target $L_0 = 40$,

Table 2: Reconstruction and downstream fidelity on Gemma-2-2B layer 12 after 100M training tokens, except the $L_0 \approx 40$ ResSAE row, which is evaluated from a 500M-token checkpoint of the same configuration (Appendix A.3). N_{eff} denotes the total feature count; L_0 is measured. Encoder FLOPs count dense encoder multiply–adds only; exact values, per-method formulas, and full configurations appear in Table 7. MSE is per dimension; CE and KL are normalized fidelity scores. Bold marks the best MSE and fidelity scores within each block.

Method	N_{eff}	L_0	Enc. FLOPs	MSE ↓	CE fidelity ↑	KL fidelity ↑
Switch SAE	67,200	40.0	11.2M	2.172	0.970	0.970
H-SAE	67,584	39.7	11.2M	2.469	0.952	0.954
KronSAE	67,200	40.0	11.2M	2.438	0.954	0.955
ResSAE (ours)	67,584	40.0	11.2M	**1.984**	**0.974**	**0.975**
Switch SAE	131,058	80.0	22.5M	1.773	0.982	0.982
H-SAE	135,168	78.1	22.4M	2.031	0.975	0.976
KronSAE	131,040	80.0	22.7M	1.930	0.979	0.979
ResSAE (ours)	135,168	80.0	22.4M	**1.758**	**0.984**	**0.984**

we vary the dictionary width of Flat TopK and scale ResSAE along two axes: increasing N_1 with $N_2 = 32$ fixed, or increasing N_2 with $N_1 = 4096$ fixed. Figure 2 plots training-batch reconstruction MSE against cumulative forward-pass FLOPs (left) and against effective dictionary size (right). The horizontal axis accumulates the forward-pass proxy C_{fwd} of Appendix A.4 over processed tokens. It excludes auxiliary losses, backward computation, and optimizer updates; full accounting and configurations are given in Appendices A.4 and A.6. Measured training efficiency under a separate approximately width-matched configuration appears in Table 14.

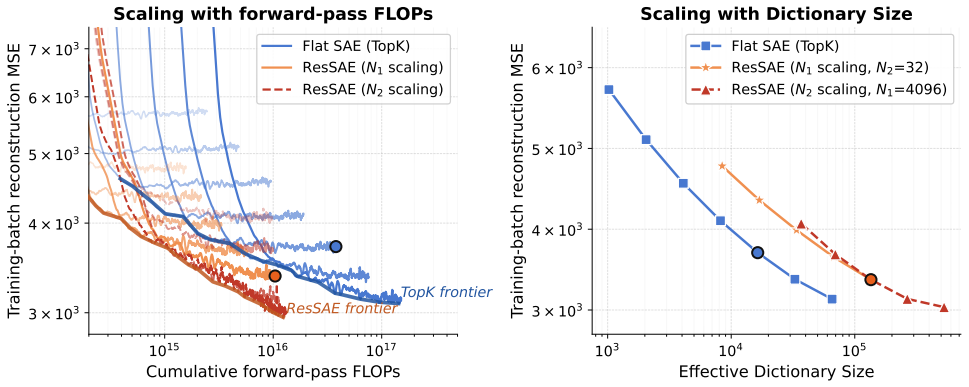

Figure 2: Scaling compute and dictionary capacity at target $L_0 = 40$. Both panels plot *training-batch* reconstruction MSE (MSE_{sum}; Appendix A.6), not held-out error. Left: versus cumulative forward-pass FLOPs (the proxy C_{fwd} of Appendix A.4). Right: versus effective dictionary size. Circles mark the two configurations of Figure 3 at their final training points: ResSAE ($N_1 = 4096$, $N_2 = 32$) and Flat TopK ($N = 16{,}384$); their held-out errors appear in Table 12.

Figure 2 reveals a trade-off between computation and dictionary size. ResSAE achieves lower error at comparable cumulative forward-pass FLOPs. Increasing either N_1 or N_2 improves reconstruction over the tested range, indicating that both routing capacity and residual capacity contribute to scaling. Flat TopK achieves lower error at comparable effective dictionary sizes, since every feature is globally available to every input. At the highlighted points, ResSAE ($N_{\text{eff}} = N_1(1 + N_2) = 135{,}168$) accesses a feature space $8.25\times$ as large as Flat TopK ($N = 16{,}384$) while achieving lower error with fewer cumulative forward-pass FLOPs. As a generalization check, Table 12 reports the final held-out error of the two highlighted configurations; ResSAE remains below Flat TopK. This check covers these two endpoints only and does not establish the held-out ordering of the full scaling curves.

Keeping these dictionary sizes fixed, we next compare ResSAE with the flat baselines across sparsity budgets in terms of encoder FLOPs, reconstruction MSE, and downstream language-model CE

loss. For ResSAE, we allocate 30% of the nominal sparsity budget to Level 1 and 70% to Level 2, using $(k_1, k_2) \in \{(6, 14), (12, 28), (24, 56), (48, 112), (96, 224)\}$. All models in this comparison are trained on 500M tokens and evaluated on held-out activations.

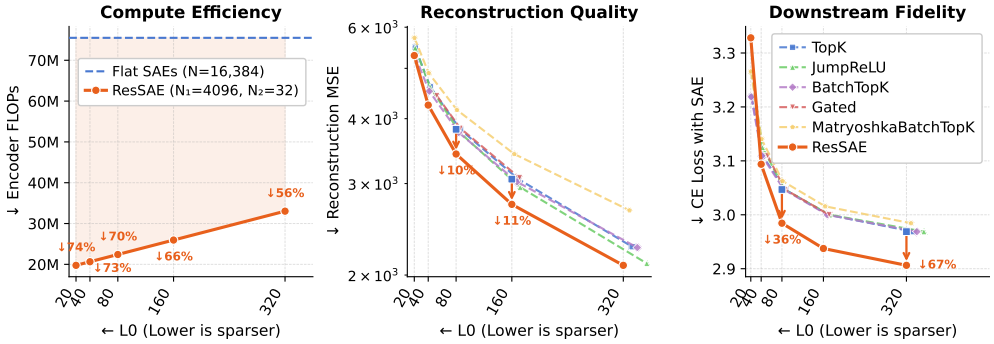

Figure 3: Encoder cost and reconstruction fidelity across sparsity budgets after 500M training tokens. ResSAE uses $N_1 = 4096$ and $N_2 = 32$, giving 135,168 total features; the flat baselines each use 16,384 features. Panels report encoder FLOPs, reconstruction MSE, and downstream CE loss.

As shown in Figure 3, ResSAE reduces encoder FLOPs by 56–74% and reconstruction MSE by up to 11% relative to the flat baselines. At $L_0 = 20$, ResSAE incurs higher CE loss, because too few active Level-1 anchors lead to insufficient or uneven access to relevant Level-2 experts. At larger sparsity budgets, ResSAE achieves lower CE loss than the flat baselines.

4.3 Mechanism Analysis of Residual Experts

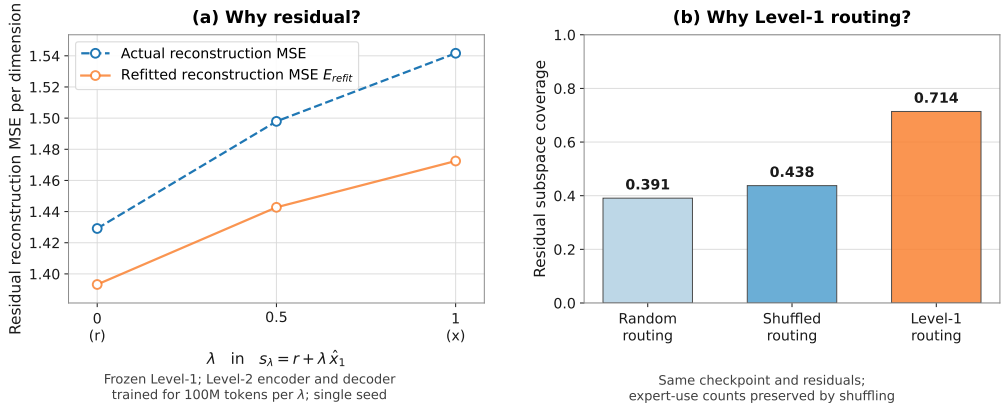

Figure 4: Mechanism diagnostics at target $L_0 = 80$. (a) Residual reconstruction MSE under input interpolation, with frozen Level 1 and a single training seed. (b) Residual subspace coverage under random, usage-preserving permuted, and Level-1 routing.

Why residual input? At target $L_0 = 80$, we freeze Level 1 and interpolate the Level-2 input as $s_\lambda = r + \lambda \hat{x}_1$, from the residual ($\lambda = 0$) to the original activation ($\lambda = 1$). Each Level-2 model is trained for 100M tokens to reconstruct the same target r. MSE increases from 1.43 to 1.54 across the tested values of λ (Figure 4(a)). Approximately 70% of this gap remains after coefficient refitting (Appendix C.1), suggesting that coefficient estimation alone does not explain the advantage of residual input.

Why Level-1 routing? Let $P_{\mathcal{I}}$ be the orthogonal projector onto the span of the selected expert decoder directions. For $r \neq 0$, define $\text{Coverage}(r, \mathcal{I}) = 1 - \frac{\|(I - P_{\mathcal{I}})r\|_2^2}{\|r\|_2^2}$.

Proposition 1 (Routing coverage and reconstruction error). *For $r = x - \hat{x}_1 \neq 0$ and any Level-2 reconstruction $\hat{r}$ in the selected expert span,*

$$\|x - (\hat{x}_1 + \hat{r})\|_2^2 = \big(1 - \mathrm{Coverage}(r, \mathcal{I})\big)\|r\|_2^2 + \|P_{\mathcal{I}} r - \hat{r}\|_2^2. \tag{3}$$

Higher coverage lowers the error floor imposed by expert selection; the remaining error depends on sparse coding within the selected span. Using the same checkpoint, residuals, and number of selected experts, Level-1 routing achieves coverage 0.714, versus 0.391 for random routing and 0.438 for a usage-preserving permuted mapping (Figure 4(b)). Thus, Level-1 features select expert spaces better aligned with the corresponding residuals.

Why residual experts? The same Level-1 representation forms the initial reconstruction and selects dictionaries for encoding the remaining variation. To evaluate their interaction, we cross the Level-2 input (x or r) with random or Level-1 routing, keeping the two-stage parameterization and sparsity budgets fixed. All four variants are trained on 500M tokens (Table 3).

Table 3: Input and routing ablations with the same additive reconstruction. MSE is per dimension; CE score denotes normalized cross-entropy fidelity. Bold marks the best result within each sparsity budget. Random routing redraws k_1 experts uniformly with replacement for every token at each forward pass; training and evaluation use the same resampling rule.

Input	Routing	Target $L_0 = 40$		Target $L_0 = 80$		Target $L_0 = 160$	
		MSE ↓	CE score ↑	MSE ↓	CE score ↑	MSE ↓	CE score ↑
x	Random	2.92	0.9311	2.44	0.9639	2.27	0.9787
x	Level-1	2.13	0.9739	1.95	0.9820	1.95	0.9853
r	Random	2.73	0.9297	2.31	0.9608	1.86	0.9804
r	**Level-1**	**1.84**	**0.9771**	**1.48**	**0.9886**	**1.19**	**0.9935**

The absolute MSE reduction from residual input is larger under Level-1 routing than under random routing by 0.10, 0.34, and 0.35 at $L_0 = 40$, 80, and 160, respectively. Together with the coverage analysis, these positive interaction contrasts support combining residual input with dictionaries specialized to Level-1-selected contexts.

4.4 Feature Quality and Organization

Beyond reconstruction, we examine whether residual experts exhibit organization across levels and reduce absorption and splitting while preserving interpretability.

Qualitative feature organization. Using a checkpoint with $N_1 = 4096$, $N_2 = 32$, $k_1 = 24$, and $k_2 = 56$, we inspect top-activating contexts of Level-1 anchors and the Level-2 features most frequently selected when each anchor is active. We exclude anchors dominated by punctuation, formatting artifacts, or stopwords (selection protocol in Appendix C.3). Figure 5 shows a broad medical anchor whose associated residual features respond to more specific contexts involving allergies, skin injuries, pain, and medication. This example illustrates coarse-to-fine organization without semantic labels or an explicit parent–child alignment loss. Additional chemistry, UI/code, and API examples appear in Appendix C.4.

Absorption, splitting, and interpretability. For quantitative evaluation, we compare approximately width-matched models: flat baselines with around 65K features and ResSAE with $N_1 = 2048$, $N_2 = 32$, and $N_{\text{eff}} = 67{,}584$ (configurations and metric protocols in Appendix A.7). Figure 6 shows lower absorption and splitting for ResSAE than for TopK, JumpReLU, BatchTopK, and Gated at $L_0 = 40$ and 80. Its splitting score stays close to one for $L_0 \geq 80$. Relative to MatryoshkaBatchTopK, ResSAE has lower splitting at most budgets, but generally higher absorption. Auto-interpretability remains broadly comparable to the baselines, although ResSAE scores below most of them at $L_0 = 320$.

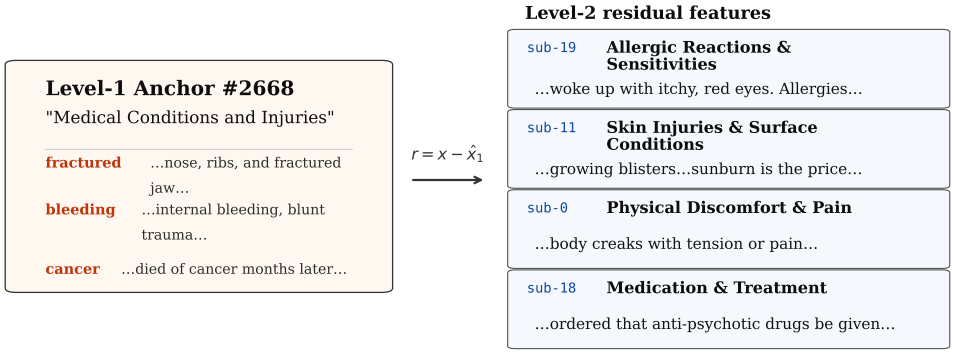

Figure 5: A Level-1 medical anchor and four features from its associated Level-2 expert, illustrated by top-activating contexts.

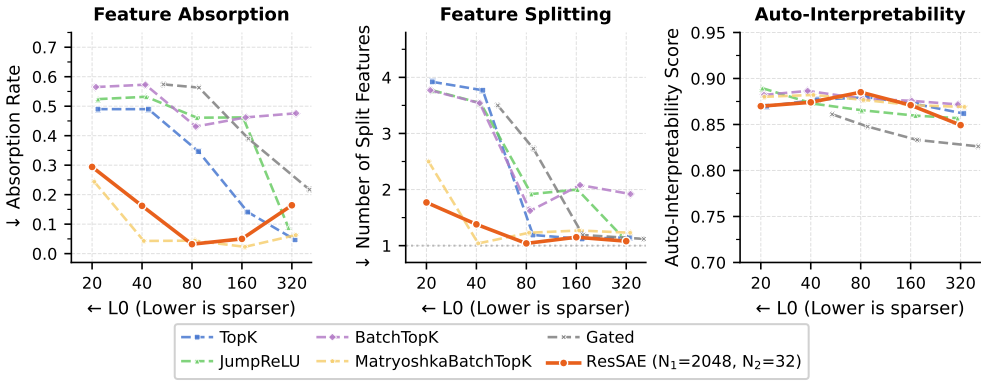

Figure 6: Feature quality across L_0 values for approximately width-matched SAEs.

Feature geometry. Decoder geometry provides further evidence of expert organization (Appendix C.2). The mean cosine similarity between Level-1 anchors and their associated Level-2 features is -0.121, with only 0.79% of pairs having $|\cos| > 0.5$, indicating limited strong cross-level alignment. Within-expert nearest-neighbor cosine similarity exceeds the cross-expert value (0.134 versus 0.052), whereas randomly grouped TopK features have similar within- and cross-group values (0.085 versus 0.089). These patterns are consistent with geometric specialization within residual experts.

5 Conclusion

We introduced ResSAE, a two-stage sparse autoencoder that couples feature-based routing with residual refinement: Level-1 features form an initial reconstruction and select Level-2 expert dictionaries, whose features compete under a shared sparsity budget to encode the remaining residual. This design supports a large feature dictionary while limiting per-input encoder computation.

On Gemma-2-2B, ResSAE improves reconstruction and downstream fidelity over the evaluated efficient SAE baselines at approximately matched dictionary sizes, sparsity, and encoder FLOPs; relative to flat baselines, it reduces encoder FLOPs by 56–74% and reconstruction MSE by up to 11% at matched nominal L_0. Controlled ablations support the complementary roles of residual input and Level-1 routing, and feature analyses suggest coarse-to-fine organization with reduced absorption and splitting at broadly comparable auto-interpretability. These findings support conditional residual refinement as an effective design for efficient sparse autoencoding with structured feature organization.

Ethics Statement

This work studies sparse autoencoders for language-model interpretability. Our experiments use existing model, data, and software artifacts, which remain subject to their respective licenses and terms of use. The reported reconstruction metrics, feature-quality scores, and qualitative examples should not be interpreted as guarantees of causal faithfulness or model safety.

Reproducibility Statement

Section 3 describes the architecture and training objective. Appendix A provides training and evaluation settings, FLOP accounting, and reconstruction pseudocode. Additional experimental results and analysis protocols are provided in Appendices B and C. Code and checkpoints will be released to facilitate reproducibility.

AI Use Statement

We used AI assistants for polishing writing and feedback on experimental design. The authors reviewed the AI-assisted content and take full responsibility for the final manuscript.

References

Evan Anders, Clement Neo, Jason Hoelscher-Obermaier, and Joseph N. Howard. Sparse autoencoders find composed features in small toy models. LessWrong, 2024. URL https://www.lesswrong.com/posts/a5wwqza2cY3W7L9cj/sparse-autoencoders-find-composed-features-in-small-toy.

Joseph Bloom, Curt Tigges, Anthony Duong, and David Chanin. SAELens. GitHub repository, 2024. URL https://github.com/decoderesearch/SAELens.

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, Joseph Benton, Tom Henighan, and Chris Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. URL https://transformer-circuits.pub/2023/monosemantic-features.

Bart Bussmann, Patrick Leask, and Neel Nanda. BatchTopK sparse autoencoders. *arXiv preprint arXiv:2412.06410*, 2024. doi: 10.48550/arXiv.2412.06410. URL https://arxiv.org/abs/2412.06410.

Bart Bussmann, Noa Nabeshima, Adam Karvonen, and Neel Nanda. Learning multi-level features with matryoshka sparse autoencoders. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 6077–6101. PMLR, 2025. URL https://proceedings.mlr.press/v267/bussmann25a.html.

Tue M. Cao, Hoang X. Nhat, Raed Alharbi, Phi Le Nguyen, and My T. Thai. Tree SAE: Learning hierarchical feature structures in sparse autoencoders. *arXiv preprint arXiv:2605.07922*, 2026. doi: 10.48550/arXiv.2605.07922. URL https://arxiv.org/abs/2605.07922.

David Chanin, James Wilken-Smith, Tomáš Dulka, Hardik Bhatnagar, and Joseph Bloom. A is for absorption: Studying feature splitting and absorption in sparse autoencoders. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=LC2KxRwC3n.

Valérie Costa, Thomas Fel, Ekdeep Singh Lubana, Bahareh Tolooshams, and Demba Ba. From flat to hierarchical: Extracting sparse representations with matching pursuit. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/311bd32bcd952f93cf820bacd6955f0a-Abstract-Conference.html.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023. doi: 10.48550/arXiv.2309.08600. URL https://arxiv.org/abs/2309.08600.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022. doi: 10.48550/arXiv.2209.10652. URL https://arxiv.org/abs/2209.10652.

Soham Gadgil, Chris Lin, and Su-In Lee. Ensembling sparse autoencoders. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2505.16077.

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. doi: 10.48550/arXiv.2101.00027. URL https://arxiv.org/abs/2101.00027.

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

Adam Karvonen, Can Rager, Johnny Lin, Curt Tigges, Joseph Isaac Bloom, David Chanin, Yeu-Tong Lau, Eoin Farrell, Callum Stuart McDougall, Kola Ayonrinde, Demian Till, Matthew Wearden, Arthur Conmy, Samuel Marks, and Neel Nanda. SAEBench: A comprehensive benchmark for sparse autoencoders in language model interpretability. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 29223–29264. PMLR, 2025. URL https://proceedings.mlr.press/v267/karvonen25a.html.

Nikita Koriagin, Yaroslav Aksenov, Daniil Laptev, Gleb Gerasimov, Nikita Balagansky, and Daniil Gavrilov. Teach old SAEs new domain tricks with boosting. In *Second Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2507.12990.

Vadim Kurochkin, Yaroslav Aksenov, Daniil Laptev, Daniil Gavrilov, and Nikita Balagansky. Kronecker factorization improves efficiency and interpretability of sparse autoencoders. *arXiv preprint arXiv:2505.22255*, 2025. doi: 10.48550/arXiv.2505.22255. URL https://arxiv.org/abs/2505.22255.

Patrick Leask, Bart Bussmann, Michael T. Pearce, Joseph Isaac Bloom, Curt Tigges, Noura Al Moubayed, Lee Sharkey, and Neel Nanda. Sparse autoencoders do not find canonical units of analysis. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=9ca9eHNrdH.

Yifan Luo, Yang Zhan, Jiedong Jiang, Tianyang Liu, Mingrui Wu, Zhennan Zhou, and Bin Dong. From atoms to trees: Building a structured feature forest with hierarchical sparse autoencoders. *arXiv preprint arXiv:2602.11881*, 2026. doi: 10.48550/arXiv.2602.11881. URL https://arxiv.org/abs/2602.11881.

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.

Mark Muchane, Sean Richardson, Kiho Park, and Victor Veitch. Incorporating hierarchical semantics in sparse autoencoder architectures. *arXiv preprint arXiv:2506.01197*, 2025. doi: 10.48550/arXiv.2506.01197. URL https://arxiv.org/abs/2506.01197.

Anish Mudide, Joshua Engels, Eric J. Michaud, Max Tegmark, and Christian Schroeder de Witt. Efficient dictionary learning with switch sparse autoencoders. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=k2ZVAzVeMP.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020. doi: 10.23915/distill.00024.001. URL https://distill.pub/2020/circuits/zoom-in/.

Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Tom Lieberum, Vikrant Varma, János Kramár, Rohin Shah, and Neel Nanda. Improving sparse decomposition of language model activations with gated sparse autoencoders. In *Advances in Neural Information Processing Systems*, volume 37, 2024a. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/01772a8b0420baec00c4d59fe2fbace6-Abstract-Conference.html.

Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, János Kramár, and Neel Nanda. Jumping ahead: Improving reconstruction fidelity with JumpReLU sparse autoencoders. *arXiv preprint arXiv:2407.14435*, 2024b. doi: 10.48550/arXiv.2407.14435. URL https://arxiv.org/abs/2407.14435.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Nguyen, Chris Olah, Dario Amodei, Tristan Hume, Zac Hatfield-Dodds, Ben Mann, and Jared Kaplan. Scaling monosemanticity: Extracting interpretable features from Claude 3 Sonnet. *Transformer Circuits Thread*, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity.

Martin Wattenberg and Fernanda B. Viégas. Relational composition in neural networks: A survey and call to action. *arXiv preprint arXiv:2407.14662*, 2024. doi: 10.48550/arXiv.2407.14662. URL https://arxiv.org/abs/2407.14662.

APPENDIX CONTENTS

Supplementary methods, implementation, and results.

A Experimental and Implementation Details **14**

A.1 Main ResSAE Configuration 14
A.2 Common Training and Evaluation Settings 14
A.3 Baselines and Compared Architectures 15
A.4 FLOP Accounting 18
A.5 Evaluation Metrics 18
A.6 Scaling Configurations 19
A.7 Feature-Pathology Evaluation 20
A.8 Reconstruction Implementation 21

B Additional Quantitative Results **22**

B.1 Configuration Sensitivity 22
B.2 Efficiency and Robustness 22

C Mechanism and Feature-Organization Analysis **24**

C.1 Residual Input and Routing 24
C.2 Feature Geometry of Residual Experts 24
C.3 Case-Study Protocol 26
C.4 Additional Qualitative Examples 26

D Comparison with Related Architectures **28**

D.1 Conditional Access and Residual Refinement 28
D.2 Other Dictionary and Hierarchy Designs 29

Section titles and page numbers link to their destinations.

A Experimental and Implementation Details

This appendix documents the experiments in Section 4. Unless otherwise stated, the main setting uses Gemma-2-2B layer-12 residual-stream activations at `blocks.12.hook_resid_post` ($d = 2304$) and 500M training tokens from the Pile uncopyrighted train split, the same shared snapshot for every run. The efficient-architecture comparison in Table 2 uses 100M tokens (500M for the $L_0 \approx 40$ ResSAE checkpoint), the pilot sweeps in Appendix B.1 use 50M tokens, and the robustness checks in Appendix B.2 use 50M tokens for the layer and base-model checks and 200M tokens for the seed checks, with the models and layers specified there. The common settings below refer to the main ResSAE runs; they do not override experiment-specific configurations.

A.1 Main ResSAE Configuration

The main ResSAE configuration used in the compute–fidelity comparison uses $N_1 = 4096$ Level-1 features and $N_2 = 32$ residual features per Level-2 expert. This gives an effective conditional residual capacity of

$$N_1 N_2 = 4096 \times 32 = 131{,}072$$

Level-2 residual features. Including Level-1 features, the total effective feature capacity is 135,168. The reconstruction loss weights are set to $\alpha = 1.0$ and $\beta = 0.5$. We vary the sparsity level by changing the number of active Level-1 features k_1 and the Level-2 residual budget k_2. Table 4 reports the schedule used in the main compute–fidelity comparison.

Table 4: ResSAE sparsity schedule for the main compute–fidelity comparison. We vary k_1 and k_2 while keeping $N_1 = 4096$ and $N_2 = 32$ fixed.

Target L_0	k_1	k_2	Encoder FLOPs/token
20	6	14	19.8M
40	12	28	20.6M
80	24	56	22.4M
160	48	112	26.0M
320	96	224	33.0M

A.2 Common Training and Evaluation Settings

Across the main Gemma-2-2B ResSAE experiments, we use the common training and evaluation settings summarized in Table 5. These main runs use residual-stream activations from layer 12 at `blocks.12.hook_resid_post`. The residual is not detached, so the global reconstruction loss can backpropagate through the residual path into Level-1. We use AuxK losses for dead-feature recovery at both levels.

For downstream reconstruction evaluation, 200 batches of 16 prompts with context length 128 contain

$$200 \times 16 \times 128 = 409{,}600$$

token positions before metric-specific masking. The number of scored next-token predictions can be smaller because of sequence boundaries and excluded tokens. The 50M-token pilot sweeps are reported separately from the main 500M-token results.

Routing-aware auxiliary loss. Both levels use TopK-style AuxK losses (Gao et al., 2025): the top-scoring dead features reconstruct the stop-gradient residual, and the squared error is summed over dimensions, averaged over the batch, and scaled by $\min(N_{\text{dead}}/(d/2), 1)$. At Level 1, a feature is dead when its firing frequency stays below 10^{-8} over a 1,000-step window, the auxiliary budget is $k_{\text{aux}} = \min(d/2, N_{\text{dead}})$ with $d/2 = 1{,}152$, and the coefficient is $\lambda_1 = 1.0$. Level-2 inactivity is tracked only when an expert is routed: the counter increments once per routed sample and resets when the feature activates, so its unit is sample-routing events rather than optimizer steps. Counting starts once the Level-2 loss reaches full scale (step 10,000, after 5,000 warm-up and 5,000 ramp-up steps), and a feature is dead when its counter exceeds 200. Let ℓ_j denote the auxiliary loss for routed expert j, evaluated against the stop-gradient final reconstruction error $\text{sg}(x - \hat{x}) = \text{sg}(r - \hat{r})$ with

Table 5: Common training and evaluation settings for the main Gemma-2-2B ResSAE experiments.

Item	Setting
Base model	Gemma-2-2B
Activation site	Layer-12 residual stream, $d = 2304$
Training data	500M tokens, `monology/pile-uncopyrighted` train split
Training context length	1024
Optimizer	Adam, $\beta_1 = 0.5$, $\beta_2 = 0.9375$
Learning rate	3×10^{-4} with cosine annealing
Batch size	2048 tokens
Decoder normalization	Norm reparameterization at both levels
Activation normalization	None
Residual detach	No
Loss weights	$\alpha = 1.0$, $\beta = 0.5$
AuxK coefficient	1.0 (Level 1), 2.0 (Level 2)
Downstream evaluation	OpenWebText (`Skylion007`), 409,600 token positions before masking, context length 128
Seed	42 for main 500M-token runs

$k_{\text{aux}} = \min(N_2, N_{\text{dead}})$ and coefficient $\lambda_2 = 2.0$. Each training step evaluates ℓ_J at one uniformly sampled slot $J \in \mathcal{I}$ and multiplies it by k_1, an unbiased estimator of the summed auxiliary objective:

$$\mathbb{E}_J[k_1 \ell_J \mid \mathcal{I}] = \sum_{j \in \mathcal{I}} \ell_j.$$

The main reconstruction residual remains differentiable through Level 1.

A.3 Baselines and Compared Architectures

We evaluate ResSAE under two complementary comparison settings. The main compute-oriented comparison uses full-dictionary SAE baselines, while a separate controlled comparison evaluates efficient and routed architectures under approximately matched effective width, sparsity, and encoder FLOPs. Table 6 summarizes the compared methods.

Table 6: SAE architectures used in our experiments. Full-dictionary denotes encoder access: all latent scores are evaluated before sparsification, even when the training objective introduces additional structure, as in MatryoshkaBatchTopK. Switch SAE, H-SAE, and ResSAE restrict the candidate dictionary through conditional routing, while KronSAE reduces encoder cost through factorization.

Architecture	Category	Reference
TopK SAE	Full-dictionary	Gao et al. (2025)
JumpReLU SAE	Full-dictionary	Rajamanoharan et al. (2024b)
BatchTopK SAE	Full-dictionary	Bussmann et al. (2024)
Gated SAE	Full-dictionary	Rajamanoharan et al. (2024a)
MatryoshkaBatchTopK SAE	Full-dictionary, multi-level objective	Bussmann et al. (2025)
Switch SAE	Routed experts	Mudide et al. (2025)
H-SAE	Hierarchical routed experts	Muchane et al. (2025)
KronSAE	Factorized encoder	Kurochkin et al. (2025)
ResSAE	**Routed residual experts**	**Ours**

Full-dictionary comparison. For the main compute-oriented experiment, TopK, JumpReLU, BatchTopK, Gated, and MatryoshkaBatchTopK use dictionary width $N = 16{,}384$. Their sparsity parameters are varied to match the target L_0 values used by ResSAE. The main ResSAE configuration uses $N_1 = 4096$ and $N_2 = 32$, giving an effective width $N_{\text{eff}} = N_1(1 + N_2) = 135{,}168$. This comparison is intentionally not parameter- or width-matched: it tests whether conditional access can make a substantially larger feature space computationally accessible while reducing per-input encoder cost.

The encoder and decoder weight matrices contain approximately 75.5M parameters for the 16,384-feature flat TopK SAE and 622.9M for ResSAE. Including encoder biases and the shared decoder bias gives 75,516,160 and 622,991,616 parameters, respectively. These counts assume untied encoder and decoder weights and no additional trainable scale parameters. ResSAE therefore trades parameter storage for lower active computation; we do not claim parameter efficiency.

Controlled efficient-architecture comparison. We separately compare ResSAE with Switch SAE, H-SAE, and KronSAE under approximately matched effective width, measured sparsity, and encoder FLOPs. Table 2 reports effective width, measured L_0, encoder FLOPs, and evaluation metrics for two sparsity regimes. Matching these quantities reduces the capacity and computation differences present in the main flat-baseline comparison, but does not imply matching parameter storage, total training cost, or optimization behavior. All baselines in this comparison are trained on 100M tokens. The $L_0 \approx 40$ ResSAE row is evaluated from a 500M-token checkpoint of the same configuration and training recipe (N_1=2,048, N_2=32, k_1=12, k_2=28; Appendix A.2), so the low-budget block is not training-token matched. Tables 7 and 8 list the complete per-method configuration behind every row. The encoder-FLOPs expressions follow the convention of Appendix A.4 (one multiplication and one addition counted as two FLOPs, $d = 2{,}304$):

$$C_{\text{enc}} = \begin{cases} 2d(E + n_e) & \text{Switch SAE (top-1 routing)} \\ 2d\,m_{\text{top}} + 2k_{\text{top}}\big(sd + m_{\text{low}}s\big) & \text{H-SAE} \\ 2dh(m + n) & \text{KronSAE} \\ 2d(N_1 + k_1 N_2) & \text{ResSAE} \end{cases}$$

For Switch SAE, E is the number of experts and n_e the per-expert width; the router scores all experts and each token is encoded by the single selected expert. For H-SAE, each of the m_{top} experts owns an s-dimensional subspace with m_{low} atoms; routing keeps the top k_{top} experts and the atom codes within selected subspaces are L_1-penalized. For KronSAE, every head combines one of m row keys with one of n column keys multiplicatively, giving hmn latents whose scores are computed through the factorization, and a two-stage TopK ($t_1 \to t_2$) selects the active set.

Table 7: Architecture and sparsity configuration of every Table 2 row. Each method appears in its $L_0 \approx 40$ (upper) and $L_0 \approx 80$ (lower) configuration. The rightmost column is the exact encoder-FLOPs count per token computed from the expressions above; Table 2 rounds these values to one decimal in units of 10^6.

Method	Structure	Sparsity	N_{eff}	Enc. FLOPs/token
Switch SAE	E=28 experts × n_e=2,400 features	top-1 routing, k=40	67,200	11,188,224
	E=27 experts × n_e=4,854 features	top-1 routing, k=80	131,058	22,491,648
H-SAE	m_{top}=2,048, m_{low}=32, s=19	k_{top}=20 experts	67,584	11,212,544
	m_{top}=4,096, m_{low}=32, s=19	k_{top}=40 experts	135,168	22,425,088
KronSAE	h=20 heads, m=42, n=80; depth-2 router	two-stage TopK 56→40	67,200	11,243,520
	h=40 heads, m=39, n=84	two-stage TopK 96→80	131,040	22,671,360
ResSAE (ours)	N_1=2,048, N_2=32	k_1=12, k_2=28	67,584	11,206,656
	N_1=4,096, N_2=32	k_1=24, k_2=56	135,168	22,413,312

Tuning protocol. Table 2 reports a single training run per row. ResSAE hyperparameters were selected from 70 pilot runs of 50M tokens each (Appendix B.1) covering $N_1 \in \{1{,}024, 2{,}048, 2{,}304, 4{,}096\}$, $N_2 \in \{16, 32, 64\}$, the k_1/k_2 split at target $L_0 \in \{20, 40, 80, 160, 320\}$, $\alpha \in \{1, 2\}$, $\beta \in \{0.1, 0.3, 0.5\}$, $\lambda_1 \in \{0.03125, 0.0625, 1\}$, $\lambda_2 \in \{0.125, 0.25, 0.5, 1, 2\}$, and learning rate in $\{10^{-4}, 3{\times}10^{-4}, 5{\times}10^{-4}, 10^{-3}\}$. Switch SAE was compared across expert counts $E \in \{16, 32\}$ at width 65,536 before selecting E=28 at 67,200 and

Table 8: Implementation and optimization settings for the rows of Table 2. All runs train on the cached Gemma-2-2B layer-12 residual-stream activations described in Appendix A. The H-SAE checkpoints do not record a seed; seed 0 is the training-script default.

Method	Implementation	Tokens (steps × batch)	Optimizer and learning rate	Objective and regularization	Seed
Switch SAE	`dictionary_learning` fork (Mudide et al., 2025)	100M (24,414×4,096)	Adam $(0.9, 0.999)$; LR $2\times10^{-4}\sqrt{2^{14}/N_{\text{eff}}}$ (9.9×10^{-5} / 7.1×10^{-5} per block), constant until step 19,531, then linear decay to 0	load-balance weight 3.0; AuxK weight $1/32$; dead threshold 10^7 tokens	42
H-SAE	PyTorch port of the official JAX H-SAE (Muchane et al., 2025)	100M; 12,207×8,192 ($L_0\approx40$), 24,414×4,096 ($L_0\approx80$)	Adam $(0.9, 0.999)$; peak LR 5×10^{-4}, linear warm-up 1,000 steps, cosine decay to $0.1\times$peak; gradient clip 0.75; unit-norm inputs	L1 penalty 10^{-3} on atom codes; cross-expert orthogonality 0.1; top-level reconstruction weight 0.1	0
KronSAE	Kurochkin et al. (2025) implementation (transformers 4.54.0)	100M (24,414×4,096)	Adam $(0.9, 0.99)$; LR 3×10^{-4}, 10% warm-up, linear decay to 10^{-6}; gradient clip 1.0; inputs standardized with cache statistics, folded into the weights at export	L1 weight 1.8×10^{-3}; auxiliary penalty $1/32$ (bandwidth 10^{-3}); TopK warm-up over the first 10% of steps	42
ResSAE (ours)	SAELens v6.37.6 fork (`sac` architecture); no input normalization	$L_0\approx40$: 500M (244,140×2,048); $L_0\approx80$: 100M (24,415×4,096)	Adam $(0.5, 0.9375)$; LR 3×10^{-4}, cosine annealing; warm-up/decay steps 4,882/48,828 (500M run) and 488/4,882 (100M run)	α=1.0, β=0.5, λ_1=1.0, λ_2=2.0; Level-2 dead threshold 200; Level-2 objective warm-up 5,000 steps plus ramp 5,000 steps	42

E=27 at 131,058, with k set to the target L_0 and a fixed load-balancing coefficient of 3.0. H-SAE follows the training recipe of Muchane et al. (2025) without further tuning; the only deviations are the PyTorch re-implementation, the Pile residual-stream cache, and the batch sizes listed in Table 8. KronSAE factor shapes were swept over $(h, m, n) \in \{(16, 64, 64), (16, 32, 128), (4096, 2, 8)\}$ at width 65,536 before selecting $(20, 42, 80)$ and $(40, 39, 84)$ for the two blocks.

A.4 FLOP Accounting

Encoder FLOPs per token. We count one multiplication and one addition as two FLOPs. For a flat SAE and ResSAE, respectively, the leading dense encoder costs are

$$C_{\text{enc,flat}} = 2dN, \qquad C_{\text{enc,ResSAE}} = 2d(N_1 + k_1 N_2).$$

These expressions count the candidate features scored, rather than the number that remain nonzero after sparsification. They exclude decoder operations, TopK selection, bias and norm operations, routing/indexing, residual addition or subtraction, auxiliary losses, and backpropagation. For $d = 2304$ and $N = 16{,}384$, the flat encoder costs 75.497472M FLOPs per token. The main ResSAE schedule costs 19.759104M–33.030144M FLOPs per token, corresponding to 73.83%–56.25% reductions.

Forward reconstruction cost and training cost. Including sparse decoding gives the forward-pass proxy

$$C_{\text{fwd,ResSAE}} \approx 2d(N_1 + k_1 N_2) + 2d(k_1 + k_2),$$
$$C_{\text{fwd,flat}} \approx 2dN + 2dk.$$

The decoder terms assume kernels that exploit the selected support; executing a dense multiplication on a tensor containing zeros does not realize this sparse cost. Summing C_{fwd} over processed tokens gives a cumulative forward-pass proxy. The factor two above counts multiplication and addition; it does not include a backward pass.

What Figure 2 plots. The horizontal axis is C_{fwd} summed over processed tokens,

$$\text{FLOPs}_{\text{cum}} = n_{\text{steps}} \cdot B \cdot C_{\text{fwd}}, \qquad B = 2048,$$

so the per-token value plotted is exactly C_{fwd}: 20,828,160 for ResSAE (N_1=4096, N_2=32, k_1=12, k_2=28) and 75,681,792 for Flat TopK (N=16,384, k=40), with d=2304. Beyond the exclusions listed above, it also omits the AuxK losses, the optimizer state updates, and the entire backward pass—no input-gradient, parameter-gradient, residual-backpropagation, or sparse-backward-kernel cost is included.

A forward-plus-backward count would instead be

$$C_{\text{fwd+bwd}} = C_{\text{enc}}^{\text{fwd}} + C_{\text{dec}}^{\text{fwd}} + C_{\text{enc}}^{\text{bwd}} + C_{\text{dec}}^{\text{bwd}},$$

where $C_{\text{enc}}^{\text{bwd}}$ and $C_{\text{dec}}^{\text{bwd}}$ cover the input-gradient and parameter-gradient matrix products (including the gradient flowing back through the residual/Level-1 path) and depend on whether the kernels exploit sparsity. Figure 2 plots C_{fwd}, not $C_{\text{fwd+bwd}}$; the ratio between the two depends on the architecture and implementation, so we do not treat full training cost as a uniform rescaling of the plotted proxy. Measured end-to-end training efficiency appears in Table 14.

A.5 Evaluation Metrics

Reconstruction scale and sparsity. For an activation $x \in \mathbb{R}^d$ and reconstruction $\hat{x}$, we distinguish

$$\text{MSE}_{\text{sum}} = \mathbb{E}_x \|x - \hat{x}\|_2^2, \qquad \text{MSE}_{\text{dim}} = \frac{1}{d}\mathbb{E}_x \|x - \hat{x}\|_2^2.$$

Division by d is not normalization by activation variance or norm. For the same evaluated activations, $\text{MSE}_{\text{sum}} = d\,\text{MSE}_{\text{dim}}$. For ResSAE, measured sparsity is $\mathbb{E}_x[\|z_1\|_0 + \|z_2\|_0]$; the nominal budget is $k_1 + k_2$. Zero-valued entries selected by TopK do not contribute to measured L_0.

A.6 Scaling Configurations

The compute–reconstruction scaling experiment uses the same activation source and target sparsity $L_0 = 40$. All runs train on 500M tokens with batch size 2048. ResSAE uses $k_1 = 12$, $k_2 = 28$, $\alpha = 1.0$, and $\beta = 0.5$ along two axes: (i) varying N_1 at fixed $N_2 = 32$, as listed in Table 9; and (ii) varying N_2 at fixed $N_1 = 4096$, as listed in Table 10. For the second axis, $N_{\text{eff}} = 4096(1 + N_2)$ and $C_{\text{enc}} = 4608(4096 + 12N_2)$ FLOPs per token.

Table 9: ResSAE configurations for compute–reconstruction scaling. This table covers the N_1 sweep at $N_2 = 32$, $k_1 = 12$, and $k_2 = 28$. Parameter counts include biases.

N_1	Parameters
256	38.9M
512	77.9M
1024	155.7M
2048	311.5M
4096	623.0M

Table 10: ResSAE configurations for the N_2 sweep at $N_1 = 4096$ ($k_1 = 12$, $k_2 = 28$). The $N_2 = 32$ row is shared with the N_1 sweep (Table 9); parameter counts include biases.

N_2	$N_{\text{eff}} = 4096(1 + N_2)$	Parameters
8	36,864	169.9M
16	69,632	320.9M
32	135,168	623.0M
64	266,240	1.23B
128	528,384	2.44B

For Flat TopK SAE, we fix $k = 40$ and vary the dictionary width. Table 11 lists the Flat TopK configurations.

Table 11: Flat TopK SAE configurations for compute–reconstruction scaling. All models use $k = 40$.

Dictionary width N	Parameters
1024	4.7M
2048	9.4M
4096	18.9M
8192	37.8M
16384	75.5M
32768	151.0M
65536	302.1M

For parameter counting, we include Level-1 encoder/decoder weights, Level-2 encoder/decoder weights, Level-1 encoder biases, Level-2 encoder biases, and the shared decoder bias. We use

$$\begin{aligned}\#\text{Params}_{\text{ResSAE}} &= 2dN_1(1 + N_2) \\ &\quad + N_1(1 + N_2) + d\end{aligned}$$

for ResSAE, and

$$\#\text{Params}_{\text{flat}} = 2dN + N + d$$

for Flat TopK SAE.

The scaling curves in Figure 2 plot the *training-batch* reconstruction error, not a held-out error. At each logged step we record the end-to-end residual reconstruction $\text{mse_global} = \mathbb{E}_{x \in \mathcal{B}} \|x - \hat{x}_{\text{final}}\|_2^2$ over the current training batch $\mathcal{B}$, expressed as squared Euclidean error per token MSE_{sum}. Curves are shown after rolling-median spike clipping and moving-average smoothing. Each highlighted circle is the mean over the final 200 logged batches of one run; both panels plot this same value with different horizontal axes and vertical scales (logarithmic left, linear right). These training-curve

endpoints need not equal the held-out per-dimension MSE values in the evaluation tables: both the normalization and the evaluated activations differ.

Table 12 reports the final held-out reconstruction error for the two configurations highlighted in Figure 2, on the unified SAE-Bench held-out set at target $L_0 = 40$. We give both the per-dimension value and its $\mathrm{MSE}_{\mathrm{sum}} = d\,\mathrm{MSE}_{\mathrm{dim}}$ equivalent ($d = 2304$) so the points can be read against the figure's vertical scale. The held-out error exceeds the training-curve endpoint for both models, and ResSAE stays below Flat TopK, so the ordering on the training curves is not a training-error artifact.

Table 12: Held-out final reconstruction error for the two configurations highlighted in Figure 2, on the unified SAE-Bench held-out set at target $L_0 = 40$. $\mathrm{MSE}_{\mathrm{sum}} = d\,\mathrm{MSE}_{\mathrm{dim}}$ with $d = 2304$ places the held-out points on the vertical scale of Figure 2, whose curves themselves report training-batch $\mathrm{MSE}_{\mathrm{sum}}$.

Configuration	N_{eff}	Held-out MSE/dim.	Held-out $\mathrm{MSE}_{\mathrm{sum}}$
ResSAE (N_1=4096, N_2=32)	135,168	1.844	$\approx$ 4,249
Flat TopK (N=16,384)	16,384	2.000	$\approx$ 4,608

A.7 Feature-Pathology Evaluation

For the feature absorption and splitting comparison, we use approximately width-matched models. Flat baselines use around 65K features, while the corresponding ResSAE configuration uses $N_1 = 2048$ and $N_2 = 32$, giving an effective Level-2 residual capacity of

$$N_1 N_2 = 2048 \times 32 = 65{,}536.$$

Including Level 1 gives $N_{\mathrm{eff}} = 67{,}584$, which is 3.125% wider than a 65,536-feature flat dictionary. Thus the comparison is approximately width-matched in total feature count, while the Level-2 feature count alone equals the flat width. This differs from the main compute–fidelity setting with $N_1 = 4096$. The ResSAE training settings follow Table 5; feature-quality metrics use their corresponding benchmark protocols.

Table 13: Configuration used for the feature absorption and splitting comparison. The total ResSAE dictionary has 67,584 features, compared with 65,536 for the flat baselines.

Item	**Setting**
Compared baselines	TopK, JumpReLU, BatchTopK, Gated and MatryoshkaBatchTopK as flat sparse-coding baselines, with approximately 65K features where applicable
ResSAE Level-1 width	$N_1 = 2048$
Features per Level-2 expert	$N_2 = 32$
Level-2 / total feature count	$N_1 N_2 = 65{,}536$ / $N_{\mathrm{eff}} = 67{,}584$
Sparsity budgets	k_1 and k_2 varied by target L_0
Training settings	Same common settings as Table 5

We use the SAE-Bench absorption and splitting evaluation (Karvonen et al., 2025; Chanin et al., 2025). These metrics operationalize feature pathologies using supervised probe concepts; they do not count every semantic concept represented by a dictionary. Absorption measures the extent to which specialized latents account for cases missed by the identified main feature set. Splitting summarizes the number of features selected to represent a probe concept. Lower values indicate fewer detected pathologies under this protocol; a splitting value near one indicates that the probe tends to select a single feature. The auto-interpretability scores in Figure 6 are a separate benchmark measure and are interpreted with higher values better.

We run both metrics with SAE-Bench 0.5.1 (commit `24de4c34`) and SAELens 6.37.6. The ResSAE adapter flattens the two-level dictionary into a single 67,584-feature index, so both levels are scored under the same protocol as the flat baselines. Absorption and splitting use the first-letter probe task (*"{word} has the first letter:"*, with the probe read at token position -6). The k-sparse probe uses L_1 penalty 0.01, batch size 4,096, and 50 epochs; a letter concept is evaluated when its ground-truth probe reaches F1 ≥ 0.6 and at least 20 first-letter features are detected, with $k \leq 10$ selected by

an F1 jump threshold of 0.03 (seed 42). Auto-interpretability samples 1,000 latents with at least 15 activations in 2M tokens of the training corpus at context length 128; `gpt-4o-mini` produces explanations and scores (generation: top-10 activating and 5 in-window random examples; scoring: 2 top, 10 random, and 2 in-window; seed 42).

A.8 Reconstruction Implementation

Listing 1 illustrates the reconstruction path using the SAELens-style storage convention (Bloom et al., 2024). For a batch $x \in \mathbb{R}^{B\times d}$, the Level-1 encoder and decoder have shapes (d, N_1) and (N_1, d); their Level-2 counterparts have shapes (N_1, d, N_2) and (N_1, N_2, d). Inputs retain their original activation scale.

For a decoder row D_j, let $s_j = \max\{\|D_j\|_2, 10^{-8}\}$. Scaling its encoder score by s_j and decoding with D_j/s_j gives the effective parameters in Section 3.2; decoder directions have unit norm except below the numerical floor. TopK retains the largest rectified scores and returns expert indices from the same call that constructs z_1, preserving the selected set even when some scores are zero. Each token routes to k_1 experts, while measured Level-1 sparsity can be smaller.

Only the weighted reconstruction loss is returned; full training also adds both AuxK terms from Section 3.3. The loop is schematic, not an optimized kernel or a standalone trainer.

```python
import torch

def topk_relu(scores, k):
    values, indices = torch.relu(scores).topk(k, dim=-1)
    z = torch.zeros_like(scores).scatter(-1, indices, values)
    return z, indices

def reconstruction_forward(self, x):
    # Encoder: [d, N1]; decoder: [N1, d].
    h1 = (x - self.b_dec) @ self.W_enc + self.b_enc
    norm1 = self.W_dec.norm(dim=-1).clamp_min(1e-8)
    z1, expert_ids = topk_relu(h1 * norm1, self.k1)
    x_hat_1 = (z1 / norm1) @ self.W_dec + self.b_dec

    # Keep the main residual differentiable through Level 1.
    residual = x - x_hat_1
    z2, r_hat = l2_forward(self, residual, expert_ids)
    x_hat = x_hat_1 + r_hat

    loss_l1 = (x - x_hat_1).square().sum(-1).mean()
    loss_global = (x - x_hat).square().sum(-1).mean()
    recon_loss = self.alpha * loss_l1 + self.beta * loss_global
    return x_hat, z1, z2, expert_ids, recon_loss

def l2_forward(self, residual, expert_ids):
    # W_enc_2: [N1, d, N2]; W_dec_2: [N1, N2, d].
    hidden, selected_norms = [], []
    for slot in range(self.k1):
        e = expert_ids[:, slot]
        enc = self.W_enc_2[e]  # [B, d, N2]
        h = torch.bmm(residual[:, None, :], enc).squeeze(1)
        hidden.append(h + self.b_enc_2[e])
        dec = self.W_dec_2[e]  # [B, N2, d]
        selected_norms.append(dec.norm(dim=-1).clamp_min(1e-8))

    h_raw = torch.cat(hidden, dim=-1)  # [B, k1*N2]
    norms = torch.cat(selected_norms, dim=-1)
    z2, _ = topk_relu(h_raw * norms, self.k2)
    z_dec = z2 / norms
    r_hat = torch.zeros_like(residual)
    for slot in range(self.k1):
        e = expert_ids[:, slot]
        start = slot * self.N2
        z = z_dec[:, start:start + self.N2]
        r_hat = r_hat + torch.bmm(
            z[:, None, :], self.W_dec_2[e]
        ).squeeze(1)
    return z2, r_hat
```

Listing 1: Main ResSAE reconstruction path; auxiliary losses omitted

Here $1 \leq k_1 \leq N_1$ and $1 \leq k_2 \leq k_1 N_2$. The returned z_2 is indexed within each token's routed candidate pool; its entries must be mapped back to expert and local-feature indices for dictionary-wide feature statistics. A feature in expert j with local index q has global zero-based index $N_1 + jN_2 + q$ when Level-1 features occupy indices $0, \ldots, N_1 - 1$.

B Additional Quantitative Results

B.1 Configuration Sensitivity

We use 50M-token pilot sweeps to select ResSAE configurations before the main 500M-token experiments. The sweeps examine dictionary allocation, sparsity allocation, loss weights, and optimization hyperparameters. The observations below summarize these short-run configuration studies; the 500M-token ablations are reported in Section 4.3.

We first varied the allocation between Level-1 anchors and Level-2 residual experts at comparable effective dictionary size,

$$N_{\text{eff}} = N_1 + N_1 N_2.$$

The pilot runs suggest broadly similar reconstruction for $(N_1, N_2) = (2048, 32)$, $(1024, 64)$, and $(4096, 16)$. These settings have total widths 67,584, 66,560, and 69,632, respectively: their Level-2 capacities are equal, but their total widths and encoder costs are not. We use $(2048, 32)$ for the medium-width feature-quality comparison; the main flat-baseline comparison uses $(4096, 32)$.

We also swept the split between the Level-1 sparsity budget k_1 and the Level-2 residual budget k_2. The results suggest that both stages require nontrivial sparsity budget. Level-1 must activate enough anchors to route the residual to relevant experts, while Level-2 must retain enough capacity to correct the remaining reconstruction error. Residual-heavy allocations can improve raw MSE at larger target sparsity, but because these pilot results are based on limited seeds, we use simple and stable allocations in the main experiments.

Finally, we examined loss weights and optimization hyperparameters. Within the tested short-run settings, changing α and β had a modest effect while the Level-1 loss remained active. In contrast, learning rate had a stronger effect on short-run training: overly small learning rates underfit, while overly large learning rates can reduce raw MSE at the cost of worse feature-utilization statistics. We use these pilot observations to select the learning rate and auxiliary-loss strength for the final 500M-token models.

B.2 Efficiency and Robustness

B.2.1 Measured Latency and Memory

We evaluate whether reduced encoder computation translates into practical efficiency using a separate approximately width-matched profile. A 65,536-feature TopK SAE ($k = 40$) and a 67,584-feature ResSAE ($N_1 = 2048$, $N_2 = 32$, $k_1 = 12$, $k_2 = 28$), both at $L_0 = 40$, are measured on one NVIDIA A800-SXM4-80GB GPU in FP32 with batch size 2048 (PyTorch 2.10, Triton 3.6). The full SAE forward includes encoding, TopK, routing, gather/scatter, decoding, and residual addition. Training measurements use 500M cached activation tokens.

Table 14: Recorded A800 FP32 profile at batch size 2048 for the width-matched pair described in the text; single recorded values, without repeated-run uncertainty estimates.

Metric	Flat TopK	ResSAE	Change
Forward latency / batch	51.39 ms	10.09 ms	5.1× faster
Peak inference memory	2.31 GB	1.37 GB	−40.7%
Cached training time	9.0 h	8.4 h	1.07× faster
Peak training memory	6.05 GB	6.27 GB	+3.6%

Table 14 shows a 5.1× forward-pass speedup and a 40.7% reduction in peak inference memory. Cached-activation training is 1.07× faster, with a 3.6% increase in peak training memory. These measurements characterize SAE execution on cached activations and exclude the cost of generating

those activations. The TopK baseline uses dense PyTorch operations (matrix multiplication, TopK, scatter), while ResSAE runs the custom Triton kernels used in training: one grouped-GEMM encoder kernel and one sparse×dense decoder kernel, with no gathered expert copies materialized. Latency is the mean over 100 timed iterations after 20 warm-up iterations, synchronizing CUDA only at the loop boundaries. Peak memory is `torch.cuda.max_memory_allocated` after resetting the peak counter, in decimal GB; the inference row includes parameters and intermediate tensors, the training row additionally includes gradients and Adam states, and neither includes the cached activation dataset. The peak-training-memory row measures forward, backward, and one Adam step (learning rate 3×10^{-4}) under a pure MSE loss without auxiliary losses, a simplification that slightly favors ResSAE. The cached-training-time row is the end-to-end wall time of the actual 500M-token runs, including auxiliary losses and data-loading I/O.

B.2.2 Robustness across Layers, Seeds, and Models

Finally, we test whether the reconstruction trend extends beyond the main Gemma layer-12 setting. Table 15 reports separate comparisons with Flat TopK on Gemma layers 5 and 19, two additional layer-12 random seeds, and Qwen3-30B-A3B layer 24. ResSAE has lower MSE in all five settings, with reductions of 4.0–15.1%, and higher CE/KL fidelity scores at the displayed precision. Each row is a separate comparison rather than a multi-seed aggregate, so the checks show a consistent direction without establishing statistical significance. All rows compare a 16,384-feature Flat TopK with a ResSAE at $N_1 = 4096$, $N_2 = 32$ ($N_{\text{eff}} = 135{,}168$) under the same target L_0: the checks are sparsity-matched but not width-matched, with $(k_1, k_2) = (48, 112)$ at $L_0 = 160$ and $(24, 56)$ at $L_0 = 80$. Gemma rows use the residual stream at the stated layer ($d = 2304$); the Qwen row uses the layer-24 residual stream of Qwen3-30B-A3B-Base ($d = 2048$, seed 42) via a hidden-state replacement hook. Training follows Table 5, with the Level-2 warm-up and ramp-up scaled to the run length (500/500 steps at 50M tokens, 1,953/1,953 at 200M). For the Qwen row, MSE is averaged over 50 held-out cached-activation batches, and CE/KL are computed on 100 OpenWebText batches of 4 sequences at context 256 in bf16; the CE score is $\mathrm{CE}_{\text{clean}}/\mathrm{CE}_{\text{sae}}$ and the KL score is $1/(1 + \overline{\mathrm{KL}}/\mathrm{CE}_{\text{clean}})$.

Table 15: Robustness checks across layers, random seeds, and base models. Training uses 50M tokens for the layer and base-model checks and 200M tokens for the seed checks. Entries are TopK / ResSAE. MSE is measured per dimension (↓), while CE and KL are normalized fidelity scores (↑). MSE comparisons are within each row; activation scales can differ across both layers and base models.

Check	Setting	L_0	MSE ↓	CE fidelity ↑	KL fidelity ↑
Layer	Gemma-2-2B, L5, seed 42	160	0.504 / **0.428**	0.992 / **0.993**	0.993 / **0.994**
	Gemma-2-2B, L19, seed 42	160	5.000 / **4.563**	0.987 / **0.988**	0.988 / **0.990**
Seed	Gemma-2-2B, L12, seed 123	80	1.676 / **1.518**	0.9872 / **0.9888**	0.9875 / **0.9890**
	Gemma-2-2B, L12, seed 456	80	1.675 / **1.514**	0.9871 / **0.9889**	0.9874 / **0.9892**
Model	Qwen3-30B-A3B, L24	80	0.0149 / **0.0143**	0.932 / **0.938**	0.922 / **0.927**

C Mechanism and Feature-Organization Analysis

C.1 Residual Input and Routing

Residual-input sweep. At target $L_0 = 80$ we fix N_1, N_2, the routing rule, and (k_1, k_2), freeze Level 1, and for each $\lambda \in \{0, 0.5, 1\}$ train the full Level-2 encoder and decoder for 100M tokens (single seed). Since $x - (\hat{x}_1 + \hat{r}) = r - \hat{r}$ and the expert set $\mathcal{I}$ is fixed, every model reconstructs the same residual r; only the input $s_\lambda = r + \lambda \hat{x}_1$ differs. On 1024 held-out tokens we report the per-dimension reconstruction MSE (Actual), the refit error E_{refit} from non-negative least squares on the frozen selected support (dictionary held fixed), and their gap.

λ	Actual	E_{refit}	Gap
0	1.429	1.393	0.036
0.5	1.498	1.443	0.055
1	1.542	1.472	0.069

Both Actual and E_{refit} increase monotonically with λ. Since E_{refit} removes the encoder's coefficient suboptimality, its rise ($+0.079$) shows the learned dictionary/support itself degrades when $\hat{x}_1$ is fed back—about 70% of the total Actual rise ($+0.112$); the widening gap ($0.036 \to 0.069$) contributes the remaining $\sim 30\%$.

Routing coverage. Coverage is measured on 256 held-out residual tokens against the span of all $k_1 N_2 = 768$ selected-expert decoder directions (rank-revealing SVD). Random samples experts without replacement; the permuted condition shuffles whole selected expert sets across tokens with the same expert count, preserving each expert's total usage; each control is averaged over 3 repeats (agreement within 0.005).

C.2 Feature Geometry of Residual Experts

We compare Level-1 decoder directions, their associated Level-2 decoder directions, and a flat TopK dictionary partitioned randomly into pseudo-experts. The random partition provides a reference for interpreting the within-group and cross-group similarities. For nonzero decoder rows D_i and D_j, cosine similarity is $\langle D_i, D_j \rangle / (\|D_i\|_2 \|D_j\|_2)$.

Limited strong anchor–feature alignment. Figure 7(a) compares each Level-1 decoder direction with the Level-2 decoder directions in its associated expert. The mean cosine is -0.121; 0.79% of pairs have $|\cos| > 0.5$ and 0.10% have $|\cos| > 0.7$. Thus strong parallel or antiparallel alignment with the corresponding anchor is uncommon. This is consistent with directional differentiation across levels, but does not imply that the span of a Level-2 dictionary is orthogonal to its anchor or cannot reconstruct it.

Within- and cross-expert similarities. For ResSAE, the reported mean nearest-neighbor cosine is 0.134 within experts and 0.052 across experts. The corresponding TopK pseudo-expert values are 0.085 and 0.089. Under the reported grouping protocol, ResSAE therefore exhibits a larger within–cross difference, consistent with geometric organization within expert dictionaries. Nearest-neighbor values depend on candidate-set size, so we interpret these results as descriptive statistics of the evaluated groups.

Both models are 500M-token checkpoints trained at seed 42: the ResSAE run with $N_1 = 4096$, $N_2 = 32$, $k_1 = 24$, $k_2 = 56$, and the flat TopK run with 65,536 features at $k = 40$. We sample 24 experts uniformly without replacement (NumPy `default_rng`, seed 42). The within-expert value takes, for each feature, the maximum cosine over the other $N_2 - 1$ rows of its own expert (self-match masked), averaged over the expert's N_2 features. Cross-expert similarity is computed per expert pair: entry (i, j) averages, over the features of expert i, each feature's maximum cosine to the N_2 rows of expert j; the reported value averages the 24×23 ordered off-diagonal entries (i.e., $24 \times 23 \times N_2$ per-feature values). For the flat TopK baseline, the 65,536 decoder rows are partitioned into 2,048 contiguous pseudo-experts of $N_2 = 32$ rows, 24 groups are sampled with the same seed, and all

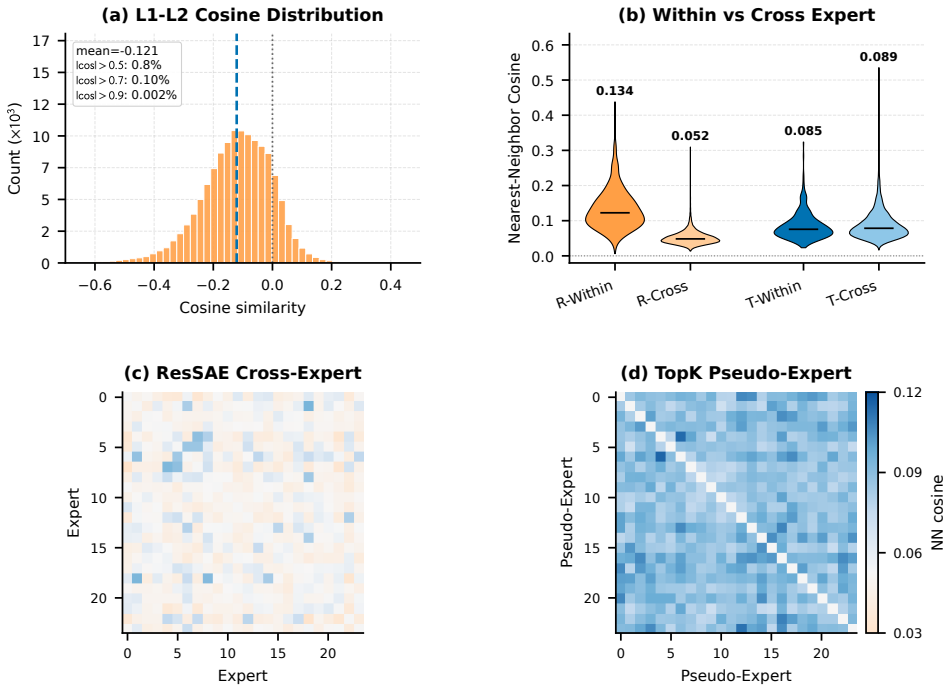

Figure 7: Decoder geometry of ResSAE and a randomly grouped flat TopK SAE. (a) Pairwise cosines between Level-1 anchors and associated Level-2 directions have a mildly negative mean and few large absolute values. (b) Reported within- and cross-expert nearest-neighbor cosine distributions. R denotes ResSAE and T denotes TopK. (c,d) Cross-expert nearest-neighbor cosine matrices for the displayed experts and pseudo-experts, with a shared color scale and masked diagonals. These statistics describe decoder directions rather than activation co-occurrence or subspace principal angles.

statistics are computed identically. The Level-1–Level-2 cosines in panel (a) use the full population of 4096×32 pairs.

Cross-expert directional similarity. ResSAE has lower cross-expert cosines than TopK in the displayed matrices (Figures 7(c,d)). Together with the within-expert results, this pattern supports organization of decoder directions by expert; it does not measure distances between entire expert subspaces.

C.3 Case-Study Protocol

For the qualitative feature-organization analysis, we use a ResSAE checkpoint with $N_1 = 4096$, $N_2 = 32$, $k_1 = 24$, and $k_2 = 56$. We collect activations from 1500 OpenWebText documents and use a context window of 12 tokens on each side of the activating token. For each token, we record the active Level-1 features and the selected Level-2 residual features. We then build an inverted index from each Level-1 feature to its top activating contexts and routed Level-2 children.

Candidate Level-1 features are filtered using three criteria. First, the feature must activate at least 50 times in the sampled corpus. Second, features dominated by stopwords, punctuation, or formatting artifacts are removed. Third, the feature must route to at least three distinct Level-2 children with interpretable activating contexts. From the remaining candidates, we select representative anchors for manual inspection.

For each candidate, we summarize the Level-1 feature from its top activating contexts. We then inspect the Level-2 residual features routed under that anchor and classify their relation to the parent feature as semantic refinement, orthogonal co-occurrence, or noise. A case is considered a strong coarse-to-fine example when the Level-1 feature captures a broad activation pattern or context cluster and multiple Level-2 features consistently refine the residual into narrower subtypes. Additional case studies are reported in Appendix C.4. Because anchors are selected after these filters and manual inspection, the examples establish the existence of such patterns, not their prevalence across the dictionary.

C.4 Additional Qualitative Examples

This subsection complements the main-text medical case study in Figure 5. We provide additional routing examples from chemistry, UI/code, and API-related contexts to show that similar coarse-to-fine patterns appear beyond a single semantic domain. These examples are intended as qualitative illustrations of the learned routing structure, not as a quantitative benchmark.

Figure 8 shows three additional examples. The chemistry anchor routes to residual features related to chemical analysis, neurochemical terminology, biochemical components, and substance-use contexts. The UI/code anchor routes to features related to styling, form elements, field initialization, and identifier usage. The API anchor routes to features related to function keywords, invocations, identifiers, and general API usage. Together with Figure 5, these examples provide qualitative evidence that residual routing can organize Level-2 features as finer refinements of broad Level-1 contexts.

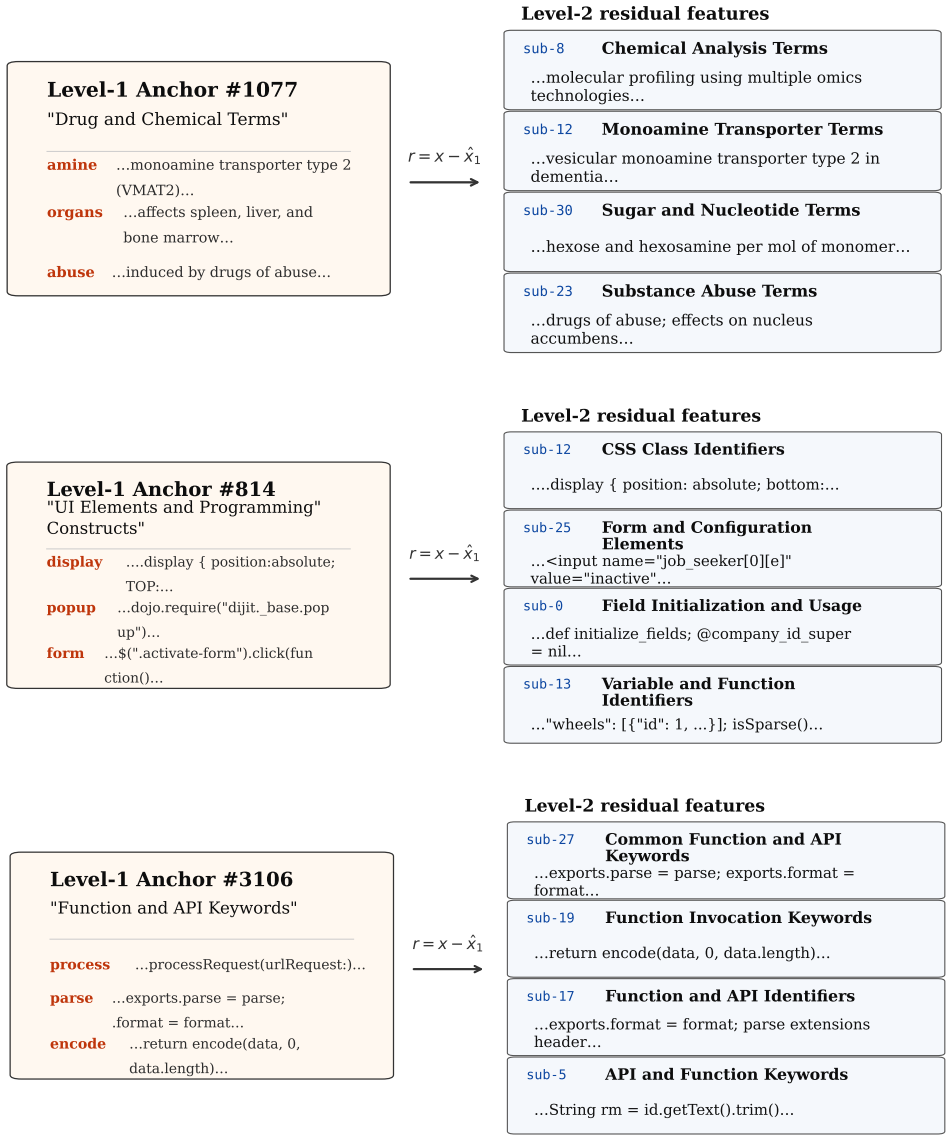

Figure 8: Additional qualitative examples of ResSAE's residual routing. In each case, a Level-1 anchor captures a broad semantic or syntactic region, while routed Level-2 residual features specialize into finer subtypes within that region. These examples complement the main-text medical case study in Figure 5 and should be interpreted as illustrative rather than quantitative evidence.

D Comparison with Related Architectures

The main-text comparison summarizes the mechanisms of selected conditional and residual SAEs. Table 16 details their encoder inputs, sparsity allocation, and training, while Table 17 covers additional dictionary and hierarchy designs. Entries describe the cited core formulations; auxiliary dead-feature losses such as AuxK are excluded.

D.1 Conditional Access and Residual Refinement

Residual targets and encoder inputs. For an additive reconstruction $\hat{x} = \hat{x}_{\text{base}} + g(u)$, let $r = x - \hat{x}_{\text{base}}$. Its reconstruction loss satisfies

$$\|x - \hat{x}_{\text{base}} - g(u)\|_2^2 = \|r - g(u)\|_2^2. \tag{4}$$

Residual fitting therefore does not determine the encoder input. H-SAE encodes projected inputs and adds the expert outputs to its high-level reconstruction. SAE Boost encodes x to predict the frozen base SAE's residual. Boosting ensembles and ResSAE encode the preceding reconstruction residual and add their corrections to the earlier reconstruction.

Table 16: Encoder inputs, sparsity allocation, and training for the methods in the main-text comparison. Inputs omit centering and bias terms and refer to the refinement stage where applicable. r_{t-1} is the residual before stage or step t.

Method	Encoder input	Sparsity allocation	Training
Switch SAE (Mudide et al., 2025)	x	TopK within one selected expert	Joint router and experts; load balancing
H-SAE (Muchane et al., 2025)	$\Pi_j^{\downarrow} x$	High-level TopK; expert-local TopK_1	Joint across levels
MP-SAE (Costa et al., 2025)	r_{t-1}	T pursuit steps; $\|z\|_0 \le T$	Shared dictionary; unrolled pursuit
SAE Boost (Koriagin et al., 2025)	x	Separate base and branch sparsity	Frozen base; added branch only
Boosting ensemble (Gadgil et al., 2026)	r_{t-1}	Separate sparsity regularization per stage	Sequential stages
ResSAE (ours)	$x - \hat{x}_1$	Level-1 TopK; joint Level-2 TopK	Joint across levels

Notes. $\Pi_j^{\downarrow}$ is a learned dimension-reducing map, not necessarily an orthogonal projector. ResSAE does not detach the main residual input, allowing gradients through the Level-2 input to Level-1.

Candidate selection and budget allocation. Switch SAE, H-SAE, and ResSAE select experts before scoring their features. H-SAE applies TopK_1 within each selected expert; ResSAE applies one TopK_{k_2} across all selected Level-2 experts, allowing different numbers of active features per expert. MP-SAE scores the shared dictionary at every pursuit step; boosting evaluates all stages sequentially.

D.2 Other Dictionary and Hierarchy Designs

Table 17 extends the comparison to other dictionary and hierarchy designs. HSAE (Luo et al., 2026) learns a feature forest across dictionaries; H-SAE (Muchane et al., 2025) associates each high-level feature with a conditional low-level expert.

Table 17: Additional dictionary, hierarchy, and ensemble architectures. KronSAE denotes its core factorized formulation, excluding combinations with Switch or Matryoshka.

Method	Encoding and reconstruction	Training and structure
Flat TopK / BatchTopK (Gao et al., 2025; Bussmann et al., 2024)	Full-dictionary scoring; per-input or batch sparsity	Single-dictionary reconstruction
KronSAE (Kurochkin et al., 2025)	Factorized scoring; TopK over composed latents	Joint training of encoder factors and decoder
Matryoshka SAE (Bussmann et al., 2025)	Shared encoding; nested-prefix reconstruction	Joint prefix losses; no explicit parent assignments
HSAE (Luo et al., 2026)	JumpReLU per level; each level encodes and reconstructs x	Joint training; parent–child consistency and feature substitution
Tree SAE (Cao et al., 2026)	Parent-gated activations; level-wise sparsity	Joint cumulative reconstruction; dynamic parent allocation
Bagging ensemble (Gadgil et al., 2026)	Each member encodes x; averaged reconstructions	Independent training; separate sparsity penalties

Notes. Matryoshka uses BatchTopK in its main LM experiments. HSAE periodically updates its parent assignments. Bagging members use the same training data with different initializations.

Feature dependencies and conditional computation. Tree SAE's parent-dependent activation rule does not by itself establish that inactive child encoder scores are skipped. HSAE's parent assignments govern cross-level training constraints rather than token-level expert dispatch. ResSAE restricts the Level-2 candidate set before encoding using Level-1 feature indices. These assignments alone do not guarantee semantic parent–child relationships.

Scope of the comparison. The tables describe architectural mechanisms, not an empirical ranking. Matched comparisons must account for measured L_0, dictionary capacity, training tokens, and the reported compute metric. Wall-clock savings and training cost require separate measurement.