ROUTING WITHOUT EMBEDDINGS: FAST AND INTERPRETABLE ROUTING WITH REGULAR EXPRESSIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Model (LLM) routers commonly rely on neural query embeddings, with larger encoders expected to better capture query intent and difficulty. Yet scaling Qwen2.5 encoders from 0.5B to 72B parameters brings little improvement in routing accuracy (Figure 1b) , suggesting that small encoders may already capture the query properties needed for routing. We therefore investigate which properties matter and whether they can be extracted directly from text without a neural encoder. We introduce REGEXROUTE, a pipeline that uses sparse autoencoders (SAEs) to discover interpretable regular-expression (regex) features. Using unlabeled text, an LLM turns descriptions of grouped SAE latents into regex extractors and refines them to match latent activation patterns. These extractors supply numerical features to a lightweight routing head, eliminating neural encoding at inference (Figure 1a). Across four benchmarks, one fixed set of 128 features achieves 76.43% average routing accuracy, comparable to 76.41% for the strongest neural text encoder baseline, with much smaller latency and strong robustness. These findings establish explicit, interpretable text features as a practical basis for designing and understanding LLM routers.

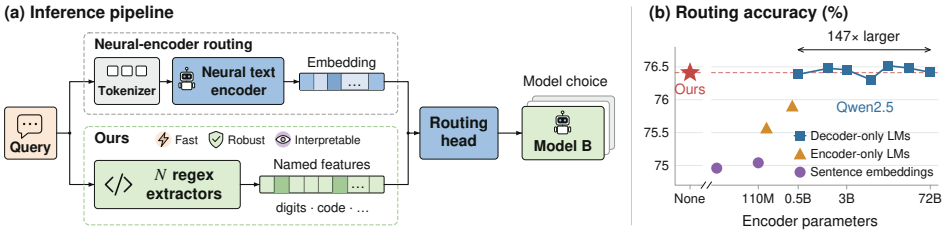


Figure 1: (a) Regex extractors replace tokenization and neural encoding at inference. (b) Routing remains competitive, while scaling Qwen2.5 from 0.5B to 72B brings limited gains (Appendix B.4).

1 INTRODUCTION

The proliferation of LLM (Large Language Models) with varying capabilities, latency, and price has made *LLM routing* a standard component of LLM service systems (OpenRouter, 2026; Not Diamond, 2026; Martian, 2026). This need motivated a range of router designs, from nearest-neighbor and clustering methods (Zhang et al., 2025; Jitkrittum et al., 2025) to learned classifiers (Song et al., 2025; Feng et al., 2025b; Zhuang et al., 2024). These routers share a common design: a neural text encoder maps each query to a dense *embedding*, and a decision rule in this space selects a model.

Existing routers use neural text encoders with widely varying capacities. Examples include encoders with roughly 100M (Ong et al., 2025; Srivatsa et al., 2024), 8B (Wang et al., 2025; Zhang et al., 2025), and even 122B parameters from a deployed language model (Varshney et al., 2026). The larger encoders are intuitively appealing for routing because they are believed to capture richer information about a query's intent and difficulty. However, our experiments that compare query embeddings from Qwen2.5 models (Qwen Team, 2024) ranging from 0.5B to 72B parameters find little improvement in routing performance as model size increases (Figure 1b, Appendix B.4). This suggests that the query properties required by routing might already have been captured by small

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.

encoders. We therefore investigate which query properties are critical for routing and whether they can be extracted directly from text without a neural text encoder.

To identify these query properties, we use sparse autoencoders (SAEs) (Bricken et al., 2023; Cunningham et al., 2023) to decompose LM (Language Model) hidden states into sparse combinations of interpretable latent features. The latent descriptions suggest which text properties to measure, while their activation patterns provide targets for text extractors to approximate. However, computing these activations still requires running both the LM and the SAE. We therefore propose REGEXROUTE, a pipeline that constructs regular-expression (regex) feature extractors to approximate the activation patterns of related SAE latents directly from text. During offline discovery, an LLM uses descriptions of grouped latents to author regex feature extractors, then refines them using feedback on how well their outputs align with latents' activations. The resulting *128* fixed regex extractors compute interpretable query features for routing without an LM or SAE forward pass at inference.

With just *128* regex features, REGEXROUTE achieves *76.43%* average routing accuracy across four benchmarks, comparable to *74.92–76.41%* for routers using neural text encoders. REGEXROUTE offers three additional benefits. First, *interpretability*: each regex feature measures a meaningful property directly from the query string. Second, *low latency*: our router runs on CPU with nearly 90% lower median routing latency than the Qwen2.5-0.5B router on GPU. Third, *robustness*: our router achieves the highest average routing accuracy under four settings of query perturbation.

We make three contributions:

1. We propose REGEXROUTE, an SAE-guided pipeline that discovers and refines interpretable regex feature extractors from unlabeled text.
2. We show that one fixed set of 128 regex features can replace neural text encoders while achieving comparable routing accuracy across four benchmarks.
3. Our findings open a new direction for the community to design and understand LLM routers through explicit, interpretable text features.

2 RELATED WORK

LLM routing. Difficulty-aware routers estimate how hard a query is and select the most capable model to handle it (Zhuang et al., 2024; Feng et al., 2025b; Song et al., 2025; Wang et al., 2025; Ding et al., 2025). Preference-aligned routers use pairwise human preference (Ong et al., 2025; Frick et al., 2025). Clustering-based routers assign each query cluster its most cost-effective model (Jitkrittum et al., 2025; Zhang et al., 2025). These approaches all use a neural text encoder to represent a query. Interpretable routers use predefined query concepts or score generated tags from model-performance labels (Štorek et al., 2026; Chen et al., 2025). Whereas our regex features are discovered from unlabeled text, without routing labels or a predefined task schema.

SAEs and interpretable text representations. Sparse autoencoders (SAEs) decompose language-model activations into sparse combinations of interpretable latent features (Cunningham et al., 2023; Bricken et al., 2023). Recent work has scaled SAEs and released feature dictionaries for several model families (Gao et al., 2024; Lieberum et al., 2024; He et al., 2024; Deng et al., 2026). LMs can describe these latents from their top-activating contexts (Bills et al., 2023; Paulo et al., 2025). A complementary line represents text through a small set of named properties. These properties may be specified from linguistic theory (Zhou et al., 2024), proposed by an LM conditioned on class names or a target variable (Sun et al., 2025; Balek et al., 2025; Benara et al., 2024), or selected using labeled downstream data (Ludan et al., 2024; Feng et al., 2025a). Their values are typically computed by prompting an LM or by probing a neural embedding at inference time. REGEXROUTE uses SAE latents to discover regex features computed directly from text without neural encoding.

3 PRELIMINARIES

LLM routing. Given a query q and a pool of candidate language models $\mathcal{L} = \{\ell_1, \ldots, \ell_M\}$, a router selects the optimal model to answer the query. To train and evaluate a router, we use a dataset $\mathcal{D}_{\text{route}} = \{(q_i, \mathbf{r}_i)\}_{i=1}^{n}$, where $\mathbf{r}_i \in \mathbb{R}^M$ records the scores of the candidate models' responses

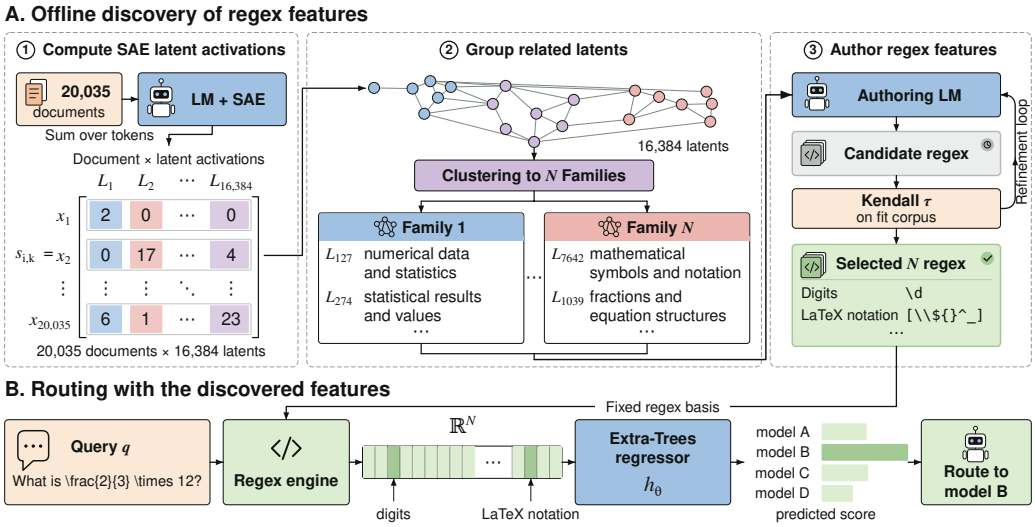


Figure 2: Overview of our pipeline. **Top**: offline discovery uses a language model and SAE to organize latent families and author regex feature extractors from public text. **Bottom**: at inference, the fixed extractors map a query to a feature vector which are used to select a model.

to q_i. We divide this dataset into disjoint training, validation, and test splits, denoted by $\mathcal{D}^{\text{train}}_{\text{route}}$, $\mathcal{D}^{\text{val}}_{\text{route}}$, and $\mathcal{D}^{\text{test}}_{\text{route}}$. These splits are used to fit the router, select its hyperparameters, and evaluate its performance, respectively. For a router π that maps a query to a model index, its test performance is

$$R\big(\pi; \mathcal{D}^{\text{test}}_{\text{route}}\big) = \frac{1}{|\mathcal{D}^{\text{test}}_{\text{route}}|} \sum_{(q_i, \mathbf{r}_i) \in \mathcal{D}^{\text{test}}_{\text{route}}} r_{i,\pi(q_i)}. \tag{1}$$

Unlabeled discovery text. To keep feature discovery independent of the routing benchmarks, we use a general **unlabeled web text corpus** $\mathcal{D}_{\text{text}}$, without access to routing benchmark data or evaluation results. We partition it into disjoint fit, $\mathcal{D}^{\text{fit}}_{\text{text}}$, and held-out portions, $\mathcal{D}^{\text{held}}_{\text{text}}$. We use $\mathcal{D}^{\text{fit}}_{\text{text}}$ to iteratively refine and select the regex feature extractors, then freeze them and evaluate their alignment with SAE activations on $\mathcal{D}^{\text{held}}_{\text{text}}$.

SAE latents and activations. Given a language-model hidden activation $\mathbf{h}_{i,t} \in \mathbb{R}^d$ at token position t in $x_i \in \mathcal{D}_{\text{text}}$, an SAE encoder E_{SAE} maps it to a sparse vector of latent activations:

$$\mathbf{z}_{i,t} = E_{\text{SAE}}(\mathbf{h}_{i,t}) \in \mathbb{R}^K_{\geq 0}. \tag{2}$$

Here, K is the number of latent features, and $z_{i,t,k}$ measures the activation of latent k at token position t. Each latent is associated with a human-readable description from Neuronpedia (Lin, 2023). Summing over the token positions $\mathcal{T}_i$ of document x_i gives latent k's document-level activation:

$$s_{i,k} = \sum_{t \in \mathcal{T}_i} z_{i,t,k}. \tag{3}$$

4 METHOD

Our goal is to route queries using a compact set of interpretable features extracted directly from text. To discover such features, REGEXROUTE uses SAE descriptions and activations to construct regex feature extractors (Figure 2, top). Each extractor combines a regex with an aggregation rule that outputs a numerical feature value. At inference, the router computes feature values and selects a model without running an LM or SAE (Figure 2, bottom).

4.1 DISCOVERING REGEX FEATURE EXTRACTORS

Grouping SAE latents. To construct regex feature extractors, we determine the latent properties they will approximate. Constructing and refining an extractor for each of the 16,384 SAE latents

would be costly and require thousands of extractor evaluations per query. Moreover, related properties can be distributed across several specialized latents (Bricken et al., 2023; Chanin et al., 2024; Leask et al., 2025), so a single latent may be too narrow for a reusable text feature. We therefore group related latents into *latent families* and construct one extractor per family.

We group latents using the statistical association between their document-level activations $s_{i,k}$, defined in Equation 3. Specifically, we define their affinity as the magnitude of the correlation across documents:

$$C_{k,k'} = \left|\mathrm{corr}(\mathbf{s}_{:,k}, \mathbf{s}_{:,k'})\right|. \tag{4}$$

We compute the affinity for every pair of latents and cluster them into N families, $\mathcal{F}_1, \ldots, \mathcal{F}_N$. We use normalized spectral clustering (Ng et al., 2001), which produces a smaller maximum family size among the grouping methods evaluated in Section 6.3. This helps limit the number of latent properties assigned to a single extractor.

Authoring regex feature extractors. For each latent family, we aim to write a regex extractor that approximates the activations of its member latents. To author such an executable text pattern, we prompt an LLM with descriptions of the member latents from Neuronpedia (Lin, 2023) and tokens associated with the family's activations. The descriptions suggest what the family captures, while the associated tokens provide examples of how that property appears in text. We then ask an LLM to identify reusable surface patterns, such as punctuation, capitalization, and formal notation, rather than copy topical vocabulary lists. The LLM initially proposes three candidate extractors per family, each with a human-readable feature name n_j, a regex p_j, and an aggregation rule a_j that converts matches into a scalar, such as their count or proportion in the document. The resulting feature is $\phi_j(x) = a_j(\mathcal{M}_{p_j}(x), x)$, where $\mathcal{M}_{p_j}(x)$ collects the pattern's matches. The initial authoring prompt is provided in Appendix A.1, with an illustrated example in Figure 7.

Refining regexes through activation alignment. To better capture each latent family's behavior, we refine the regex extractors using alignment between their outputs and family activations. We compare feature values extracted by each candidate for document x_i with a reference value derived from the activations of latent family $\mathcal{F}_j$:

$$y_{i,j} = \sum_{k \in \mathcal{F}_j} \sum_{t \in \mathcal{T}_i} \mathbf{1}[z_{i,t,k} > 0]. \tag{5}$$

The indicator reflects how often the family's latents are active throughout the document. We measure the alignment between regex output $\phi_j(x_i)$ and the reference $y_{i,j}$ using Kendall's τ_b,

$$J_j(p_j, a_j) = \tau_b(\{\phi_j(x_i)\}, \{y_{i,j}\}), x_i \in \mathcal{D}^{\mathrm{fit}}_{\mathrm{text}}. \tag{6}$$

Higher alignment means that the regex more faithfully preserves the ordering of documents. We return the candidate regexes and their fit alignment scores to the LLM, which proposes revised or alternative extractors. We select the candidate with the highest fit alignment for each family and freeze the resulting N extractors. The refinement prompt is provided in Appendix A.2.

4.2 Routing with the discovered regex extractors

After freezing the regex library, we train only the routing head on $\mathcal{D}^{\mathrm{train}}_{\mathrm{route}}$, using the same extractors across all benchmarks. For each query q_i, we apply the N frozen regex feature extractors (p_j, a_j) to obtain the feature vector (Figure 2, bottom):

$$\boldsymbol{\phi}(q_i) = [\phi_1(q_i), \ldots, \phi_N(q_i)] \in \mathbb{R}^N. \tag{7}$$

We then fit a multi-output regressor taking $\boldsymbol{\phi}(q_i)$ as input:

$$h_\theta : \mathbb{R}^N \to \mathbb{R}^M \tag{8}$$

to predict the response-score vector $\mathbf{r}_i$. Our primary multi-output regressor is an Extra-Trees regressor (Geurts et al., 2006), trained by minimizing prediction error across candidate models,

$$\widehat{\theta} = \arg\min_{\theta} \sum_{(q_i, \mathbf{r}_i) \in \mathcal{D}^{\mathrm{train}}_{\mathrm{route}}} \|h_\theta(\boldsymbol{\phi}(q_i)) - \mathbf{r}_i\|_2^2. \tag{9}$$

At inference, we extract regex features from query q and use the trained regressor to select the model with the highest predicted score, without an LM or SAE forward pass.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Benchmarks. We conduct experiments on four benchmarks: EmbedLLM (Zhuang et al., 2024), NineBy30k (Lu et al., 2026), R2Bench (Xue et al., 2026), and CARROT (Somerstep et al., 2025). Table 3 summarizes their candidate pools and the splits shared by all methods. We remove prompt templates specific to each source dataset as detailed in Appendix B.1.

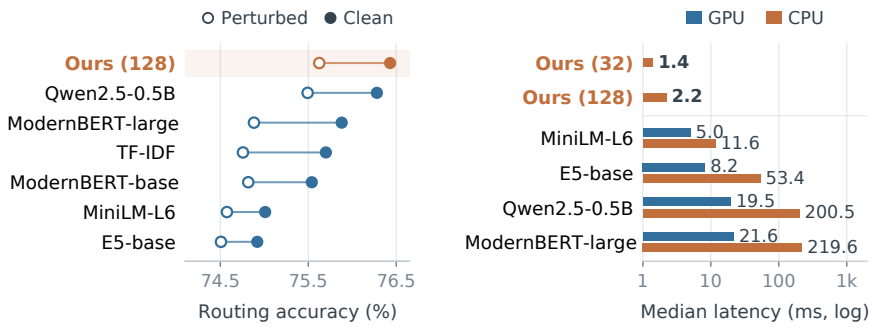
Pipeline configuration. The discovery corpus $\mathcal{D}_{\text{text}}$ contains 20,035 public documents from FineWeb (Penedo et al., 2024), OpenWebMath (Paster et al., 2023), CodeParrot, Alpaca (Taori et al., 2023), and OpenAssistant (Köpf et al., 2023), covering web prose, mathematics, code, and instruction/dialogue text. The fit and held-out portions contain 13,394 and 6,641 documents, respectively. Our default configuration uses Gemma-2-2B (Gemma Team, 2024) and its 16,384-latent Gemma Scope SAE (Lieberum et al., 2024) at the final transformer layer. We group latents into $N = 128$ families and use Claude Opus 5 Medium to author and refine regex extractors. Appendix B.2 provides the corpus sources and detailed pipeline configuration.

Baselines. We compare query embeddings produced by three types of neural text encoders. **Sentence-embedding models** are trained for semantic similarity and retrieval. Our baselines include MiniLM-L6 (Wang et al., 2020), MPNet-base (Song et al., 2020), BGE-base (Xiao et al., 2023), GTE-base (Li et al., 2023), and E5-base (Wang et al., 2022). **Encoder-only LMs** are pretrained through masked-token prediction. We evaluate ModernBERT-base and ModernBERT-large (Warner et al., 2024). **Decoder-only LMs**, pretrained to predict the next token, include Qwen2.5-0.5B (Qwen Team, 2024), Gemma-2-2B (Gemma Team, 2024), and Llama-3.1-8B (Grattafiori et al., 2024). We also include TF-IDF as a baseline without a neural encoder. Baseline details are in Appendix B.3.

Evaluation. For each representation and benchmark, we train an Extra-Trees (Geurts et al., 2006) routing head and select its hyperparameters on the validation data (Appendix C.4). We report routing accuracy (equation 1), for each benchmark and its unweighted average across the four benchmarks.

Table 1: **Test routing accuracy (%).** All routers use Extra-Trees heads with hyperparameters selected on validation. Column maxima are in **bold**.

Representation	EmbedLLM	NineBy30k	R2Bench	CARROT	AVG.
Best single model	60.73	67.34	81.10	84.82	73.50
Sentence embeddings					
MiniLM-L6 (Wang et al., 2020)	64.87	67.47	81.56	86.18	75.02
MPNet-base (Song et al., 2020)	64.73	67.43	81.39	86.28	74.96
BGE-base (Xiao et al., 2023)	64.47	67.77	81.53	86.51	75.07
GTE-base (Li et al., 2023)	64.37	67.63	81.77	86.60	75.09
E5-base (Wang et al., 2022)	63.77	67.59	81.86	86.47	74.92
Embeddings from encoder-only LMs					
ModernBERT-base (Warner et al., 2024)	65.60	67.88	81.94	86.88	75.58
ModernBERT-large (Warner et al., 2024)	66.27	68.07	82.19	87.10	75.91
Embeddings from decoder-only LMs					
Qwen2.5-0.5B (Qwen Team, 2024)	67.50	68.23	82.38	**87.20**	76.33
Gemma-2-2B (Gemma Team, 2024)	**67.83**	68.26	82.38	87.08	76.39
Llama-3.1-8B (Grattafiori et al., 2024)	67.77	68.13	**82.70**	87.02	76.41
Non-neural features (without embedding)					
TF-IDF (Salton & Buckley, 1988)	65.60	68.23	81.99	86.99	75.70
Ours ($N = 32$)	66.63	68.50	82.15	86.96	76.06
Ours ($N = 128$)	67.37	**69.01**	82.16	87.18	**76.43**



(a) Robustness (b) Routing latency

Figure 3: Robustness and routing latency. Our regex router retains the highest perturbed accuracy (left) and achieves lower median latency on CPU than neural encoder routers on GPU (right).

5.2 Routing quality without embeddings

Our 128 regex features achieve 76.43% mean routing accuracy, comparable to the 76.41% achieved by the strongest embedding baseline (Table 1). With only 32 regex features, our router achieves 76.06% mean routing accuracy, exceeding the mean accuracy of all sentence-embedding and encoder-only LM baselines in this comparison. Both configurations also outperform TF-IDF (75.70%), suggesting that our regex features provide more effective routing signals than word- and character-n-gram frequencies alone. These results show that a compact set of surface-extractable features can replace a neural text encoder while retaining comparable average routing accuracy.

5.3 Robustness to query perturbations

Because regex features depend on surface-text patterns, we test whether changes in wording or formatting undermine routing reliability. Following RouterArena (Lu et al., 2025), we apply four types of query perturbation: misspellings, word substitutions, sentence restructuring, and paraphrasing. All methods are fitted on clean text and evaluated on perturbed queries using the original response scores. Figure 3 (left) reports clean accuracy and mean accuracy of four perturbation rewrites. Our router retains the highest mean perturbed accuracy among the evaluated representations (75.62%), with a drop in accuracy similar to that of Qwen2.5-0.5B. Sentence-embedding models lose less accuracy but still have lower perturbed scores. This suggests that RegexRoute features remain useful even when query wording and format change. The complete setup and results are in Appendix B.5.

5.4 Routing latency

From Figure 3 (right), RegexRoute preserves routing quality at lower computational cost. We measure routing latency from query input to model selection, including feature extraction and routing-head inference, which adds to the time to first token (TTFT). With 128 features, our router's median latency is 2.2 ms on CPU, compared with 19.5 ms for Qwen2.5-0.5B on GPU. Running the router on the CPU leaves GPU resources for response generation and can simplify deployment on edge devices with limited access to accelerators. The full table is in Appendix B.6.

6 Ablation Studies

We conduct ablations to understand the contributions of each component of our pipeline. Unless otherwise stated, we follow the pipeline configuration and evaluation protocol in Section 5.1.

6.1 Choice of discovery LM and SAE

We test whether RegexRoute depends on a particular family of discovery models. Alongside the default Gemma configuration, we use Qwen3-1.7B and Llama-3.1-8B with their pretrained SAEs (Deng et al., 2026; He et al., 2024) to generate 128 regex feature extractors per model with

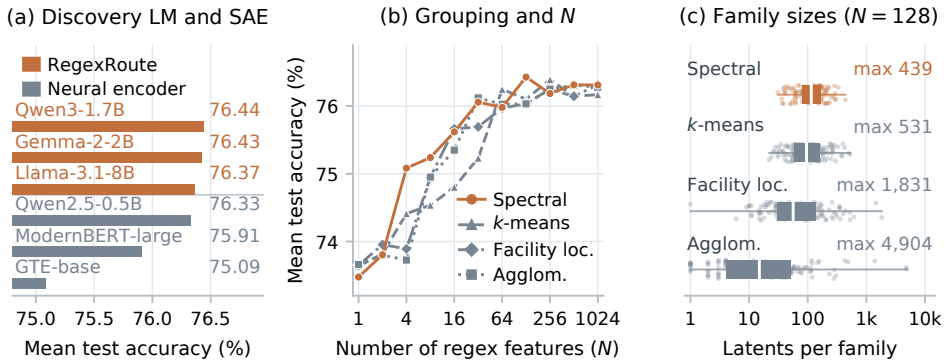


Figure 4: **Pipeline ablations.** Mean routing accuracy across four benchmarks for (a) discovery LM–SAE pairs at $N = 128$ and (b) latent grouping strategy across feature counts, N. (c) Latent family sizes at $N = 128$ by different grouping strategies.

the REGEXROUTE pipeline (Figure 2). In Figure 4(a), Qwen and Llama reach 76.44% and 76.37%, respectively, compared with 76.43% for the main Gemma configuration. REGEXROUTE is not specific to a single base model or SAE and can produce general regex feature extractors from multiple LM-SAE pairs. Per-benchmark results and SAE dictionary sizes are provided in Appendix C.1.

6.2 NUMBER OF REGEX FEATURES

For each feature count N, we evaluate routing performance using a new regex library authored by the LLM. The orange curve in Figure 4(b) shows that accuracy increases by 2.6 pp from $N = 1$ to 32, as additional regex features capture a broader range of routing-important surface patterns. Then, performance saturates between $N = 32$ and 128, suggesting that a compact feature set captures most of the important feature directions, while further expansion primarily increases extraction cost. We retain $N = 128$ as the main configuration, with $N = 32$ providing a lower-latency alternative.

6.3 CHOICE OF LATENT GROUPING

We compare spectral clustering with spherical k-means, facility-location selection, and average-linkage agglomerative clustering. For $N \geq 64$, mean routing accuracy differs by at most 0.40 percentage points across the four methods (Figure 4b). Spectral clustering performs best at $N = 4$ and 8. At $N = 128$, its largest family contains 439 latents, compared with 531 to 4,904 for the alternatives (Figure 4c). We retain spectral clustering to limit the maximum number of latents each regex extractor must approximate while maintaining competitive routing accuracy. Appendix C.2 provides detailed results and descriptions of the grouping methods.

6.4 DO SAE GUIDANCE AND REFINEMENT IMPROVE REGEX FEATURES FOR ROUTING?

We test whether SAE-guided authoring and activation-alignment refinement (Sec 4.1) improve the regex features used for routing. At $N \in \{32, 128\}$, we compare our full procedure with two ablations: the *LLM-only* variant, which omits both SAE guidance and refinement, and the *initial regexes* variant, which uses our pipeline without the final refinement stage. The LLM-only authoring prompt is provided in Appendix A.3, and experimental details are provided in Appendix C.3.

SAE-guided authoring improves over the LLM-only control. Comparing *initial regexes* with *LLM-only* in Table 2, we find that SAE-guided authoring raises test accuracy by 1.00 and 0.49 percentage points at $N = 32$ and $N = 128$, respectively. Latent descriptions identify properties for the LLM to capture, while associated tokens illustrate how they appear in text. The gains suggest that these concrete targets help guide the LLM toward regex features useful for routing.

Alignment-based refinement further improves routing. Higher alignment in Equation 6 means that an extractor more consistently assigns larger values to documents where its SAE family activates

Table 2: **Ablating SAE guidance and refinement.** Checkmarks indicate enabled components. Held alignment is Kendall τ_b from equation 6. The test is a four-benchmark mean routing accuracy (%).

			$N = 32$		$N = 128$	
Procedure	SAE guidance	Refinement	Held alignment	Test (%)	Held alignment	Test (%)
LLM-only	—	—	0.2343	74.68	0.2852	75.67
Initial regexes	✓	—	0.3770	75.68	0.3865	76.16
Ours	✓	✓	**0.4470**	**76.06**	**0.4709**	**76.43**

more frequently. Refinement increases test routing accuracy by 0.38 and 0.27 percentage points at $N = 32$ and 128, respectively. These gains suggest that more faithfully approximating SAE activation patterns helps regex features capture information useful for routing.

6.5 Choice of routing head

We ablate five routing heads, with setup and full results in Appendix C.4. In the recorded results, regex features are less performant with k-NN and k-means, but achieve the highest mean accuracy with tree-based heads. This is consistent with the structure of our representation: its coordinates measure distinct properties through counts and ratios, and a fixed distance across them may not capture the query differences. Tree-based heads instead learn thresholds for individual features from routing labels, allowing them to partition queries based on properties relevant to model selection.

7 Analysis

We aim to investigate why surface features achieve routing accuracy comparable to that of dense embeddings and why larger encoders yield limited gains. We provide the complete set of 128 regex feature extractors for reference in Appendix D.

7.1 Interpretable surface features distinguish queries

Readable differences between query sources. Using source-dataset labels only for post-hoc analysis, Figure 5 shows that the independently discovered features expose recognizable differences in query properties across datasets. We group the 128 features into eight categories of surface properties: script, casing, prose, layout, numbers, paths, math, and code. For example, MMLUPro is elevated on short-line and numeric features, XQuAD on script and accent features, and NarrativeQA

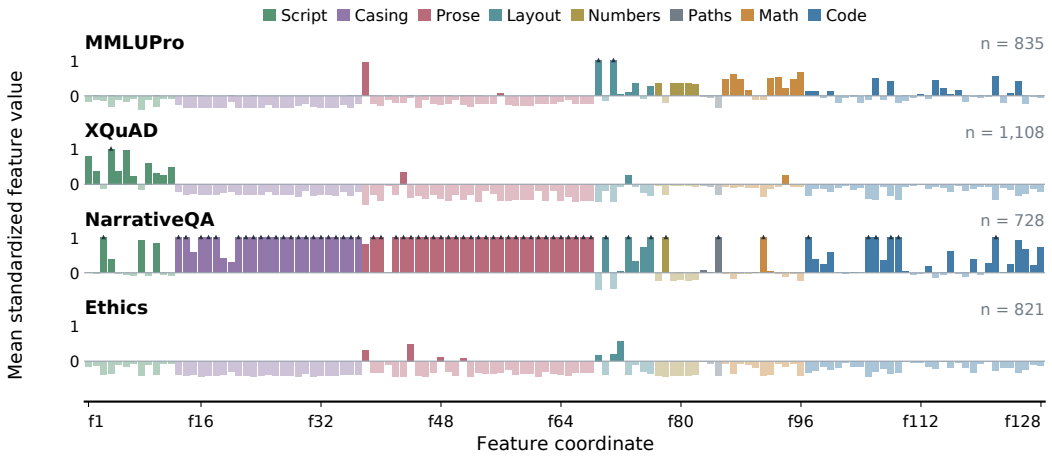


Figure 5: **Different query sources peak at different feature coordinates.** Values above +1 standard deviation are capped and marked with triangles.

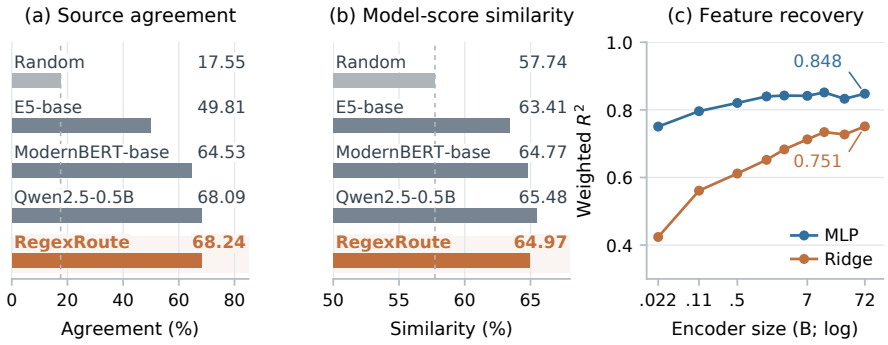


Figure 6: (a,b) Source agreement and model-score similarity. (c) Feature-recovery R^2 across neural text encoder sizes.

on prose word counts. These patterns provide interpretable cues for distinguishing query sources. We include additional profiles in Appendix E.

Regex features group queries by source and model performance. We measure whether neighboring queries tend to share a similar source dataset and observed model score. For each test query, we select $k = 20$ other test queries that share the most leaves with it across the fitted Extra-Trees router. Figure 6(a,b) shows that regex features group queries by source and model performance about as well as Qwen2.5-0.5B and better than E5-base and ModernBERT-base. Both measures significantly exceed the random baseline, suggesting that surface patterns provide useful cues for routing without neural encoding. Appendix F provides additional settings and results.

7.2 Small encoders already capture surface features

To understand the limited routing gains from encoder scaling, we examine whether larger encoders better capture query properties useful for routing. Using our 128 regex features as concrete targets, we train nonlinear MLP and linear ridge probes on each benchmark's training split to predict their values from frozen embeddings. We evaluate feature recovery on held-out test queries using R^2 and report the average across four benchmarks. Higher R^2 indicates that these surface properties are more accurately recoverable from the encoder's embeddings, with $R^2 = 1$ denoting perfect recovery. Appendix G provides the detailed setup and per-benchmark results.

With an MLP probe, Qwen2.5-0.5B reaches $R^2 = 0.820$, whereas the roughly $147\times$ larger 72B model only reaches 0.848 (Figure 6(c)). Linear recovery improves more substantially, from 0.612 to 0.751. The smaller gain with the MLP probe suggests that much of this information is already recoverable from the small encoder through a nonlinear mapping. These results show that a small encoder already captures surface features well in its dense embeddings. Because these features describe patterns directly observable in the input text, even a small encoder could recognize them without complex semantic reasoning. Together, these findings suggest that small encoders already capture much of the surface information sufficient for competitive routing, helping explain why larger encoders yield limited gains on these benchmarks.

8 Conclusion

We presented RegexRoute, which uses SAE descriptions and activation patterns to discover interpretable regex feature extractors from unlabeled text. A fixed library of 128 regex extractors achieves routing accuracy comparable to neural embeddings across four benchmarks, with low CPU latency and competitive performance under query perturbations. Our ablations show that SAE guidance and activation-based refinement improve routing performance, and that the pipeline works across the evaluated discovery model families. Our analysis connects these features to query source and model performance and shows that small encoders already represent them well. Together, these findings establish that a compact set of explicit surface features is sufficient for competitive routing, offering a possible explanation to the limited benefit of larger encoders.

AI USE STATEMENT

As part of REGEXROUTE, an LLM authors and refines regular-expression feature extractors, as described in Section 4.1 and Appendix A. LLM assistants also helped preprocess benchmark queries (Appendix B.1) and generate query rewrites for the robustness evaluation (Section 5.3). We used LLM assistants to identify related work, refine the paper's framing, and assess the novelty of our contributions. LLM coding agents, including Codex and Claude Code, helped implement the experiments. We reviewed the generated code to verify that it followed the intended experimental procedures. We also used LLM assistants to polish the writing by correcting grammar and making long or complex sentences more concise and readable.

REPRODUCIBILITY STATEMENT

Our discovery LMs, SAEs, and neural text encoder baselines use publicly available checkpoints, with references and model details in Section 5.1 and Appendix B. Section 4 and Appendix A describe the pipeline and provide the prompts for regex authoring and refinement. Appendix B documents the data preprocessing, splits, and experimental settings needed to reimplement the experiments. Appendix D lists all 128 regex patterns and their aggregation rules, allowing the feature representation to be reconstructed directly from the paper. We will release the implementation and feature files once the paper appears.

REFERENCES

Vojtěch Balek, Lukáš Sýkora, Vilém Sklenák, and Tomáš Kliegr. LLM-based feature generation from text for interpretable machine learning. *Machine Learning*, 114(11), October 2025. ISSN 1573-0565. doi: 10.1007/s10994-025-06867-1. URL http://dx.doi.org/10.1007/s10994-025-06867-1.

Vinamra Benara, Chandan Singh, John X. Morris, Richard Antonello, Ion Stoica, Alexander G. Huth, and Jianfeng Gao. Crafting Interpretable Embeddings by Asking LLMs Questions, 2024. URL https://arxiv.org/abs/2405.16714.

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. https://openaipublic.blob.core.windows.net/neuron-explainer/paper/index.html, 2023.

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. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

David Chanin, James Wilken-Smith, Tomáš Dulka, Hardik Bhatnagar, Satvik Golechha, and Joseph Bloom. A is for Absorption: Studying Feature Splitting and Absorption in Sparse Autoencoders, 2024. URL https://arxiv.org/abs/2409.14507.

Zhou Chen, Zhiqiang Wei, Yuqi Bai, Xue Xiong, and Jianmin Wu. TagRouter: Learning Route to LLMs through Tags for Open-Domain Text Generation Tasks, 2025. URL https://arxiv.org/abs/2506.12473.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse Autoencoders Find Highly Interpretable Features in Language Models, 2023. URL https://arxiv.org/abs/2309.08600.

Boyi Deng, Xu Wang, Yaoning Wang, Yu Wan, Yubo Ma, Baosong Yang, Haoran Wei, Jialong Tang, Huan Lin, Ruize Gao, Tianhao Li, Qian Cao, Xuancheng Ren, Xiaodong Deng, An Yang,

Fei Huang, Dayiheng Liu, and Jingren Zhou. Qwen-Scope: Turning Sparse Features into Development Tools for Large Language Models, 2026. URL https://arxiv.org/abs/2605.11887.

Dujian Ding, Ankur Mallick, Shaokun Zhang, Chi Wang, Daniel Madrigal, Mirian Del Carmen Hipolito Garcia, Menglin Xia, Laks V. S. Lakshmanan, Qingyun Wu, and Victor Rühle. BEST-Route: Adaptive LLM Routing with Test-Time Optimal Compute, 2025. URL https://arxiv.org/abs/2506.22716.

Jean Feng, Avni Kothari, Luke Zier, Chandan Singh, and Yan Shuo Tan. Bayesian Concept Bottleneck Models with LLM Priors, 2025a. URL https://arxiv.org/abs/2410.15555.

Tao Feng, Yanzhen Shen, and Jiaxuan You. GraphRouter: A Graph-based Router for LLM Selections, 2025b. URL https://arxiv.org/abs/2410.03834.

Evan Frick, Connor Chen, Joseph Tennyson, Tianle Li, Wei-Lin Chiang, Anastasios N. Angelopoulos, and Ion Stoica. Prompt-to-Leaderboard, 2025. URL https://arxiv.org/abs/2502.14855.

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, 2024. URL https://arxiv.org/abs/2406.04093.

Gemma Team. Gemma 2: Improving Open Language Models at a Practical Size, 2024. URL https://arxiv.org/abs/2408.00118.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely Randomized Trees. *Mach. Learn.*, 63(1):3–42, April 2006. ISSN 0885-6125. doi: 10.1007/s10994-006-6226-1. URL https://doi.org/10.1007/s10994-006-6226-1.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 Herd of Models, 2024. URL https://arxiv.org/abs/2407.21783.

Zhengfu He, Wentao Shu, Xuyang Ge, Lingjie Chen, Junxuan Wang, Yunhua Zhou, Frances Liu, Qipeng Guo, Xuanjing Huang, Zuxuan Wu, Yu-Gang Jiang, and Xipeng Qiu. Llama Scope: Extracting Millions of Features from Llama-3.1-8B with Sparse Autoencoders, 2024. URL https://arxiv.org/abs/2410.20526.

Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Congchao Wang, Zifeng Wang, Alec Go, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, Aditya Krishna Menon, and Sanjiv Kumar. Universal Model Routing for Efficient LLM Inference, 2025. URL https://arxiv.org/abs/2502.08773.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri, David Glushkov, Arnav Dantuluri, Andrew Maguire, Christoph Schuhmann, Huu Nguyen, and Alexander Mattick. OpenAssistant Conversations – Democratizing Large Language Model Alignment, 2023. URL https://arxiv.org/abs/2304.07327.

Patrick Leask, Bart Bussmann, Michael Pearce, Joseph Bloom, Curt Tigges, Noura Al Moubayed, Lee Sharkey, and Neel Nanda. Sparse Autoencoders Do Not Find Canonical Units of Analysis, 2025. URL https://arxiv.org/abs/2502.04878.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. Towards General Text Embeddings with Multi-stage Contrastive Learning, 2023. URL https://arxiv.org/abs/2308.03281.

Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Varma, János Kramár, Anca Dragan, Rohin Shah, and Neel Nanda. Gemma Scope: Open Sparse Autoencoders Everywhere All At Once on Gemma 2, 2024. URL https://arxiv.org/abs/2408.05147.

Johnny Lin. Neuronpedia: Interactive Reference and Tooling for Analyzing Neural Networks, 2023. URL https://www.neuronpedia.org. Software available from neuronpedia.org.

Yifan Lu, Rixin Liu, Jiayi Yuan, Xingqi Cui, Shenrun Zhang, Hongyi Liu, and Jiarong Xing. RouterArena: An Open Platform for Comprehensive Comparison of LLM Routers, 2025. URL https://arxiv.org/abs/2510.00202.

Yifan Lu, Qiyue Zhang, Shenrun Zhang, Zhibo Yu, Zhuang Wang, Hanjie Chen, and Jiarong Xing. The Routing Plateau: Understanding and Breaking the Accuracy Limits of LLM Routers, 2026. URL https://arxiv.org/abs/2606.07587.

Josh Magnus Ludan, Qing Lyu, Yue Yang, Liam Dugan, Mark Yatskar, and Chris Callison-Burch. Interpretable-by-Design Text Understanding with Iteratively Generated Concept Bottleneck, 2024. URL https://arxiv.org/abs/2310.19660.

Martian. Martian: Model Router. https://withmartian.com, 2026. Accessed: 2026-09-03.

Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On Spectral Clustering: Analysis and an Algorithm. In *Advances in Neural Information Processing Systems*, volume 14, 2001. URL https://proceedings.neurips.cc/paper/2001/hash/801272ee79cfde7fa5960571fee36b9b-Abstract.html.

Not Diamond. Not Diamond: The AI model router. https://www.notdiamond.ai, 2026. Accessed: 2026-09-03.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to Route LLMs with Preference Data, 2025. URL https://arxiv.org/abs/2406.18665.

OpenRouter. OpenRouter: A unified interface for LLMs. https://openrouter.ai, 2026. Accessed: 2026-09-03.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. OpenWebMath: An Open Dataset of High-Quality Mathematical Web Text, 2023. URL https://arxiv.org/abs/2310.06786.

Gonçalo Paulo, Alex Mallen, Caden Juang, and Nora Belrose. Automatically Interpreting Millions of Features in Large Language Models, 2025. URL https://arxiv.org/abs/2410.13928.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro von Werra, and Thomas Wolf. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

Qwen Team. Qwen2.5 Technical Report, 2024. URL https://arxiv.org/abs/2412.15115.

Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988.

Seamus Somerstep, Felipe Maia Polo, Allysson Flavio Melo de Oliveira, Prattyush Mangal, Mírian Silva, Onkar Bhardwaj, Mikhail Yurochkin, and Subha Maity. CARROT: A Cost Aware Rate Optimal Router, 2025. URL https://arxiv.org/abs/2502.03261.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. MPNet: Masked and Permuted Pre-training for Language Understanding, 2020. URL https://arxiv.org/abs/2004.09297.

Wei Song, Zhenya Huang, Cheng Cheng, Weibo Gao, Bihan Xu, GuanHao Zhao, Fei Wang, and Runze Wu. IRT-Router: Effective and Interpretable Multi-LLM Routing via Item Response Theory, 2025. URL https://arxiv.org/abs/2506.01048.

KV Aditya Srivatsa, Kaushal Kumar Maurya, and Ekaterina Kochmar. Harnessing the Power of Multiple Minds: Lessons Learned from LLM Routing, 2024. URL https://arxiv.org/abs/2405.00467.

Chung-En Sun, Tuomas Oikarinen, Berk Ustun, and Tsui-Wei Weng. Concept Bottleneck Large Language Models, 2025. URL https://arxiv.org/abs/2412.07992.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford Alpaca: An Instruction-following LLaMA Model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Tanay Varshney, Annie Surla, Michelle Xu, Gomathy Venkata Krishnan, Maximilian Jeblick, David Austin, Neal Vaidya, and Davide Onofrio. LLM Router: Rethinking Routing with Prefill Activations, 2026. URL https://arxiv.org/abs/2603.20895.

Chenxu Wang, Hao Li, Yiqun Zhang, Linyao Chen, Jianhao Chen, Ping Jian, Peng Ye, Qiaosheng Zhang, and Shuyue Hu. ICL-Router: In-Context Learned Model Representations for LLM Routing, 2025. URL https://arxiv.org/abs/2510.09719.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text Embeddings by Weakly-Supervised Contrastive Pre-training, 2022. URL https://arxiv.org/abs/2212.03533.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers, 2020. URL https://arxiv.org/abs/2002.10957.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Nathan Cooper, Griffin Adams, Jeremy Howard, and Iacopo Poli. Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference, 2024. URL https://arxiv.org/abs/2412.13663.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-Pack: Packed Resources For General Chinese Embeddings, 2023. URL https://arxiv.org/abs/2309.07597.

Jiaqi Xue, Qian Lou, Jiarong Xing, and Heng Huang. R2-Router: A New Paradigm for LLM Routing with Reasoning, 2026. URL https://arxiv.org/abs/2602.02823.

Yiqun Zhang, Hao Li, Jianhao Chen, Hangfan Zhang, Peng Ye, Lei Bai, and Shuyue Hu. Beyond GPT-5: Making LLMs Cheaper and Better via Performance-Efficiency Optimized Routing. *arXiv preprint arXiv:2508.12631*, 2025.

Lexin Zhou, Youmna Farag, and Andreas Vlachos. An LLM Feature-based Framework for Dialogue Constructiveness Assessment, 2024. URL https://arxiv.org/abs/2406.14760.

Richard Zhuang, Tianhao Wu, Zhaojin Wen, Andrew Li, Jiantao Jiao, and Kannan Ramchandran. EmbedLLM: Learning Compact Representations of Large Language Models, 2024. URL https://arxiv.org/abs/2410.02223.

Adam Štorek, Vikas Upadhyay, Marianne Menglin Liu, Daniel W. Peterson, Anshul Mittal, Sujeeth Bharadwaj, Fahad Shah, Sujith Ravi, and Dan Roth. Routesplain: Towards Faithful and Intervenable Routing for Software-related Tasks, 2026. URL https://arxiv.org/abs/2511.09373.

A Prompts for Regex Authoring

We provide three prompt templates for authoring regex feature extractors: SAE-guided initial authoring, alignment-based refinement, and authoring without SAE guidance. For the ablation in Section 6.4, each variant proposes six candidates per output feature in two rounds of three, then retains one. Duplicate proposals count toward this budget, and all features complete both rounds without early stopping.

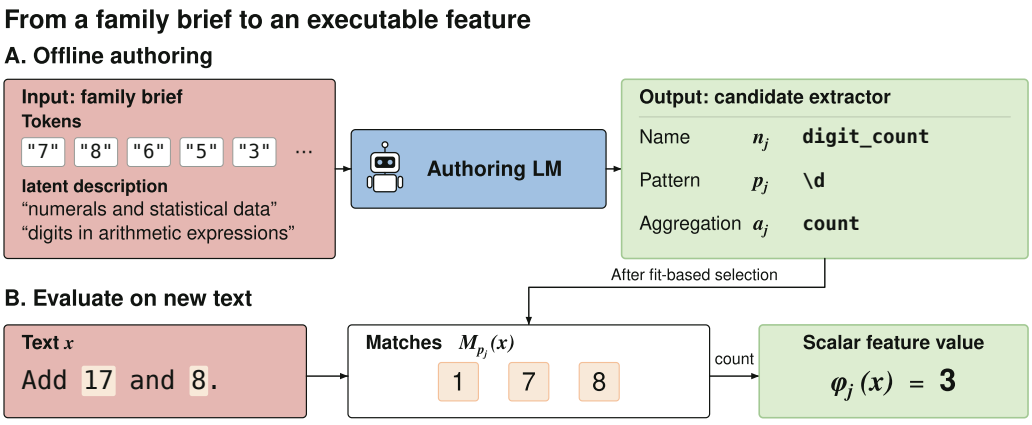


Figure 7: Illustrative regex-authoring example. Member-latent descriptions and associated tokens inform a candidate named `digit_count`, with pattern $p_j = \texttt{\textbackslash d}$ and count aggregation a_j. Once selected and frozen, the extractor counts the matches `1`, `7`, and `8` in `Add 17 and 8.`, giving $\phi_j(x) = 3$.

A.1 SAE-GUIDED INITIAL AUTHORING

This prompt is used for the initial authoring stage in Section 4.1, proposing regex feature extractors for each SAE latent family using its descriptions and associated tokens. For the variant without refinement, we use this template for both rounds, supplying the first three candidates in round two without alignment scores or other fit feedback. The controller then selects among all six candidates using fit-set alignment.

```
You are authoring regex feature extractors for latent families of a
sparse autoencoder. Work only from the supplied family briefs and
previous candidates. All family evidence comes from the unlabeled
corpus.
No routing benchmark information is available. Do not use tools.
An external controller performs scoring and retention; return only
the candidates requested for this round.

INPUT
Batch {batch_id} of six; library width {N}; round {round_id}.
<family_brief>
{fid, number of latents, activation share,
  Neuronpedia descriptions, highly associated tokens,
  family-specific tokens, punctuation/symbol tokens,
  digit-bearing tokens, for each assigned family}
</family_brief>
<round_request>
{"candidates_per_active_fid": {each assigned fid: 3},
 "previous_candidates": [
   {fid, name, pattern, agg, flags} for each previous candidate],
 "previous_fit_scores": []}
</round_request>
previous_candidates is empty in round 1. In round 2, it contains
the first three candidates per family, without evaluation feedback.

TASK
Propose three diverse extractors per family whose values track the
ranking of its member-latent firing-event count across documents.
Candidates are scored by Kendall tau-b on fit documents.
In round 2, propose three additional candidates; do not repeat
the previous candidate array.
If brief tokens are content words, capture their shared form,
```

```
not the words themselves.

CONSTRAINTS
Use symbols, orthography, and structure: punctuation, digits,
operators, brackets, casing, whitespace/layout, Unicode classes,
run lengths, and closed-class code/LaTeX/HTML/SQL keywords.
Do not use English content-word literals, topical vocabularies,
or morphological suffix patterns. Use non-capturing groups (?:...).

AGGREGATORS
count: number of matches; ratio: matches per approximate word;
binary: whether any match exists; maxrun: longest match length.
Flags: "", "i", "m", or "im".

Prefer ratio when fit scores are within 0.01 of the best;
within this margin, prefer count/ratio over binary/maxrun.

OUTPUT
Return only a JSON array, without prose or markdown. Each object
must contain fid (integer), name (string), pattern (string), agg,
and flags. Return exactly three candidates for each assigned fid.
```

A.2 ALIGNMENT-BASED REFINEMENT

This prompt is used for the refinement stage in Section 4.1, revising previously proposed regex feature extractors using fit-set feedback on their alignment with the corresponding latent family's activations.

```
You are authoring regex feature extractors for latent families of a
sparse autoencoder. Work only from the supplied family briefs and
fit feedback. All family evidence comes from the unlabeled corpus.
No routing benchmark information is available. Do not use tools.
An external controller performs scoring and retention; return only
the candidates requested for this round.

INPUT
Batch {batch_id} of six; library width {N}; round 2.
<family_brief>
{fid, number of latents, activation share,
  Neuronpedia descriptions, highly associated tokens,
  family-specific tokens, punctuation/symbol tokens,
  digit-bearing tokens, for each assigned family}
</family_brief>
<round_request>
{"candidates_per_active_fid": {each assigned fid: 3},
 "previous_fit_scores": [
   {fid, name, pattern, agg, flags, fit, density, ceiling_fit}
   for each previous candidate]}
</round_request>
fit is Kendall tau-b on fit documents; density is match density.
ceiling_fit is the family's token-set reference score.

TASK
Propose three revised or alternative extractors per family whose
values better track the ranking of its member-latent firing-event
count across documents.
The controller scores candidates by Kendall tau-b on fit documents.
If brief tokens are content words, capture their shared form,
not the words themselves. Do not mechanically negate
anti-correlated values.

CONSTRAINTS
Use symbols, orthography, and structure: punctuation, digits,
operators, brackets, casing, whitespace/layout, Unicode classes,
```

```
run lengths, and closed-class code/LaTeX/HTML/SQL keywords.
Do not use English content-word literals, topical vocabularies,
or morphological suffix patterns. Use non-capturing groups (?:...).

AGGREGATORS
count: number of matches; ratio: matches per approximate word;
binary: whether any match exists; maxrun: longest match length.
Flags: "", "i", "m", or "im".

CONTROLLER PROCEDURE
After the initial three candidates, the controller requests three
refinement candidates together in one response, giving six proposed
candidates per family. Duplicates consume slots; there is no early
stopping or additional round. It retains the fit-best of all six,
preferring count/ratio within 0.01 of maximum fit.
Prefer ratio when fit scores are within 0.01 of the best.

OUTPUT
Return only a JSON array, without prose or markdown. Each object
must contain fid (integer), name (string), pattern (string), agg,
and flags. Return exactly three NEW candidates for each assigned
fid. Do not repeat the previous candidate array.
```

A.3 AUTHORING WITHOUT SAE GUIDANCE

This prompt is used for the LLM-only control in Section 6.4, authoring regex feature extractors without family descriptions, associated tokens, or activation feedback. The LLM proposes three candidates per slot in each of two rounds and selects its preferred candidate from all six, without SAE-based selection.

```
You are authoring regex feature extractors. Work only from the
supplied slot list and previous candidates. No routing benchmark
information is available.
Do not use tools. An external controller performs retention;
return only the candidates requested.

INPUT
Batch {batch_id} of six; library width {N}; round {round_id}.
<slots>
{one object containing only fid for each assigned slot}
</slots>
<round_request>
{"candidates_per_fid": {each assigned fid: 3},
 "previous_candidates": [
   {fid, name, pattern, agg, flags} for each previous candidate]}
</round_request>
previous_candidates is empty in round 1 and contains the first
three candidates per slot in round 2. No evaluation feedback is given.

TASK
Propose three diverse extractors per slot measuring a surface-form
property of text. Different slots should measure different properties.
In round 2, propose three additional candidates and identify your
preferred candidate among all six for each slot. Number candidates
1-3 in the order previously returned and 4-6 in the order of the
new response. Base your choice on the instructions alone; no SAE
alignment or routing scores are available. The controller retains
your preferred candidate for each slot.

CONSTRAINTS
Use symbols, orthography, and structure: punctuation, digits,
operators, brackets, casing, whitespace/layout, Unicode classes,
run lengths, and closed-class code/LaTeX/HTML/SQL keywords.
Do not use English content-word literals, topical vocabularies,
```

```
or morphological suffix patterns. Use non-capturing groups (?:...).

AGGREGATORS
count: number of matches; ratio: matches per approximate word;
binary: whether any match exists; maxrun: longest match length.
Flags: "", "i", "m", or "im".

Prefer ratio when candidates are otherwise comparable, because
dense/rate-like features are wanted. Prefer count/ratio over
binary/maxrun when otherwise comparable.

OUTPUT
Return only a JSON object, without prose or markdown:
{"candidates": [...], "preferred_by_fid": {...}}.
The candidates array must contain exactly three NEW candidates
per assigned fid, each with fid (integer), name (string), pattern
(string), agg, and flags. In round 1, preferred_by_fid is empty.
In round 2, it maps each fid to your preferred candidate's index
from 1 to 6. Do not repeat the previous candidate array.
```

B EXPERIMENTAL DETAILS

Table 3: Candidate-pool size and the number of queries in each split.

benchmark	#models	train	validation	test
EmbedLLM	112	29,673	3,000	3,000
NineBy30k	9	23,027	3,000	5,000
R2Bench	10	18,580	3,097	9,291
CARROT	13	30,968	6,636	6,637

B.1 BENCHMARKS TEMPLATE REMOVAL

Prompt templates. Every benchmark combines questions from several source datasets, and each source renders its questions inside a fixed prompt template. In EmbedLLM, every MMLU question starts with "The following are multiple choice questions (with answers) about ⟨subject⟩." and ends with "Answer:". In NineBy30k, most multiple-choice questions start with "Please read the following multiple-choice questions ..." and all end with the same `\boxed{X}` instruction. The template is the same for every query from a source and different across sources, so it identifies the source. A router may therefore use the template as a shortcut, associating each source with a model that performs well on its training queries. Such a router would also fail on real traffic, which does not come wrapped in benchmark prompts. We therefore remove the templates from every benchmark before any representation sees the text, for our features and all baselines alike. We modify only the query text and keep the original response scores.

Removal procedure. EmbedLLM records the source of each query, so we handle it per source. We remove the instruction header, keep only the final question for the few-shot sources (GSM8K, TruthfulQA), drop the `Q:`/`A:` frame of Social IQa and the `Question:`, `Passage:`, and `Choices:` labels, remove the trailing `Answer:` cue, and unify the option markers. Layout is a fingerprint too: multiple-choice sources put one option per line, generative sources one paragraph. We therefore render every EmbedLLM query on one line with options marked `(A)`, `(B)`, ... The other three benchmarks have no source column, but their templates are leading instruction paragraphs and trailing format instructions, so one generic pass handles them: it removes leading paragraphs that start with an instruction ("Please answer ...", "You are a knowledge expert ..."), drops field labels (`Context:`, `Question:`, `Passage:`, `Title:`), and removes trailing answer cues and format instructions such as the `\boxed{X}` block. Queries without a wrapper pass through unchanged; almost every NineBy30k query has one, and about a third of R2Bench and CARROT queries do. The question, any passage, and the answer options are kept verbatim, and the same cleaning is applied to all splits and to the perturbed test queries in Section 5.3. For example, "Please read the following

multiple-choice questions and provide the most likely correct answer based on the options given. Context: None. Question: The value of log2 4 is: Options: A. 2 . . . E. 12. Provide the correct letter choice in \boxed{X}, where X is the correct letter choice. Keep the explanation or feedback within 3 sentences." becomes "The value of log2 4 is: Options: A. 2 . . . E. 12".

B.2 Pipeline configuration

Unlabeled discovery corpus. The discovery corpus $\mathcal{D}_{\text{text}}$ contains 20,035 public documents covering web prose, mathematics, code, and instruction/dialogue text. It comprises 5,000 documents each from FineWeb (Penedo et al., 2024), OpenWebMath (Paster et al., 2023), CodeParrot, and OpenAssistant (Köpf et al., 2023), plus 35 instructions from Alpaca (Taori et al., 2023). We use FineWeb's `sample-10BT` subset and the `codeparrot-clean-valid` release of CodeParrot.[1] Using random seed 0, we fix a split of 13,394 fit documents for regex refinement and selection and 6,641 held-out documents for alignment evaluation after the extractors are frozen. We truncate each document to 512 tokens and evaluate candidate regexes on the corresponding decoded text span. The same documents and fit/held-out split are used across discovery models.

Pretrained LMs and SAEs. We obtain pretrained SAE checkpoints from Gemma Scope (Lieberum et al., 2024)[2], Qwen-Scope (Deng et al., 2026)[3], and Llama Scope (He et al., 2024)[4] for Gemma-2-2B, Qwen3-1.7B-Base, and Llama-3.1-8B, respectively. The LMs and SAEs remain frozen, and each SAE encodes the residual stream after the final transformer block. Their activations supply the latent families and alignment targets used in regex discovery. The default configuration uses Gemma-2-2B with its 16,384-latent SAE. Section 6.1 evaluates alternative discovery LM–SAE pairs, with per-benchmark results in Appendix C.1.

B.3 Query representations

Table 4 lists the neural encoders by category, with their checkpoints, dimensions, pooling methods, and supported context lengths. TF-IDF uses character 3–5-grams (`char_wb`, minimum document frequency 3, up to 40,000 terms) and word 1–2-grams (minimum document frequency 2, up to 20,000 terms) with sublinear term frequency and ℓ_2 normalization. For the scaling study, we encode every query with the Qwen2.5 base models in one pipeline: a prefill-only forward pass in bf16, last-token pooling, and an 8,192-token context with head truncation.

Table 4: **Learned query representations.** All checkpoints are frozen.

category	representation	checkpoint	dim	pooling	context length
Sentence-embedding models	MiniLM-L6	`all-MiniLM-L6-v2`	384	mean	512
	MPNet-base	`all-mpnet-base-v2`	768	mean	512
	BGE-base	`BAAI/bge-base-en-v1.5`	768	`[CLS]`	512
	GTE-base	`thenlper/gte-base`	768	mean	512
	E5-base	`intfloat/e5-base-v2`	768	mean	512
Encoder-only LMs	ModernBERT-base	`answerdotai/ModernBERT-base`	768	`[CLS]`	8,192
	ModernBERT-large	`answerdotai/ModernBERT-large`	1024	`[CLS]`	8,192
Decoder-only LMs	Qwen2.5-0.5B	`Qwen/Qwen2.5-0.5B`	896	last token	32,768
	Qwen2.5 ladder	`Qwen/Qwen2.5-{0.5B..72B}`	896–8192	last token	32,768–131,072

B.4 Encoder scaling setup

We compare frozen Qwen2.5 base models at seven sizes, from 0.5B to 72B parameters, using the uniform encoding pipeline described in Appendix B.3. For each model and benchmark, we train

[1] `https://huggingface.co/datasets/codeparrot/codeparrot-clean-valid`

[2] `https://huggingface.co/google/gemma-scope-2b-pt-res/tree/main/layer_25/width_16k/average_l0_116`

[3] `https://huggingface.co/Qwen/SAE-Res-Qwen3-1.7B-Base-W32K-L0_100/blob/main/layer27.sae.pt`

[4] `https://huggingface.co/OpenMOSS-Team/Llama3_1-8B-Base-LXR-8x/tree/main/Llama3_1-8B-Base-L31R-8x`

an Extra-Trees routing head and select its hyperparameters on validation data from the search space in Appendix C.4, using the same six configurations at every model size. We repeat each fit with five random seeds and report the mean test accuracy. The other neural encoders and the 128 regex features in Figure 1b follow the same protocol. Figure 1b reports the unweighted mean test routing accuracy across the four benchmarks. Despite a 147× increase in model size, it ranges only from 76.31% to 76.52%, and the 128 regex features reach 76.41% without running a neural text encoder at inference.

B.5 Robustness details

Perturbations. We generate one rewrite per test query and mode with DeepSeek-V4-Flash through OpenRouter, following the RouterArena (Lu et al., 2025). The system prompt requires every fact and answer option to be preserved. Misspellings introduce spelling errors while keeping the text readable; word substitutions replace content words with synonyms; sentence restructuring changes sentence structure; and paraphrasing rewrites the query using different wording. Perturbed queries are assigned the original response scores. Table 5 reports routing accuracy under each rewrite.

Table 5: **Routing accuracy under LLM query perturbation (%, four-benchmark mean, Extra-Trees).** Fitted on clean text; only test queries are perturbed. Clean reference scores are from Table 1; *drop* is the perturbation mean minus this reference. Accuracy-column maxima in **bold**.

representation	clean	typo	synonym	grammar	paraphrase	drop
128 regexes (ours)	**76.43**	**76.43**	**75.74**	75.41	74.92	−0.81
TF-IDF bags	75.70	75.27	74.43	75.06	74.27	−0.94
Qwen2.5-0.5B	76.33	75.80	75.28	**75.71**	**75.18**	−0.84
ModernBERT-large	75.91	74.84	74.76	75.12	74.81	−1.03
ModernBERT-base	75.58	75.07	74.70	75.06	74.44	−0.76
GTE-base	75.09	74.58	74.68	74.91	74.72	−0.37
BGE-base	75.07	74.69	74.48	74.76	74.45	−0.47
MPNet-base	74.96	74.81	74.69	74.92	74.63	−0.20
MiniLM-L6	75.02	74.64	74.47	74.71	74.48	−0.45
E5-base	74.92	74.50	74.40	74.63	74.50	−0.41

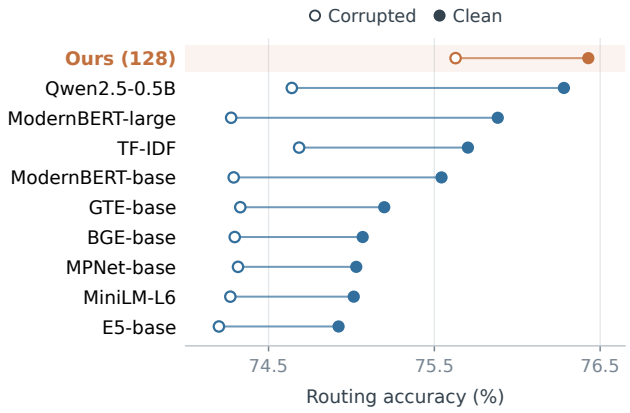


Figure 8: **Routing accuracy under mechanical character corruption** (%, four-benchmark mean, Extra-Trees head), using the marker convention of Figure 3 (left): filled is clean, hollow is the mean over corruption rates of 5, 10, 20, and 30% of characters.

Character corruption. We generate corrupted test queries with a Python script. At each alphabetic character, the script applies an edit with probability $p \in \{0.05, 0.10, 0.20, 0.30\}$, choosing uniformly among substitution with a neighboring keyboard character, deletion, swapping with the next character, and duplication. We fix the random seed to 0 and evaluate the routers trained on clean text using the original response scores. Figure 8 and Table 6 report results across corruption levels. Relative to the clean reference scores in Table 1, accuracy at 30% corruption is lower by 1.43

percentage points for the regex router and 0.96–1.16 percentage points for the sentence-embedding models. Nevertheless, it achieves the highest routing accuracy among the evaluated representations at every corruption level. Qwen2.5-0.5B shows the largest decrease, at 2.28 percentage points.

Table 6: **Mechanical character corruption (%, four-benchmark mean, Extra-Trees).** Clean reference scores are from Table 1. Ours and four representative baselines; the remaining encoders are in Figure 8.

representation	clean	5%	10%	20%	30%
128 regexes (ours)	76.43	76.13	75.97	75.41	75.00
TF-IDF bags	75.70	75.00	74.96	74.41	74.36
Qwen2.5-0.5B	76.33	75.30	74.88	74.33	74.05
E5-base	74.92	74.45	74.44	74.06	73.85
MiniLM-L6	75.02	74.63	74.43	74.16	73.86

B.6 Latency details

We measure latency on test queries with a batch size of one, using a machine with an AMD EPYC 7513 CPU and eight NVIDIA RTX A5000 GPU. CPU measurements use eight threads pinned to CPU cores. At $N = 128$, one query costs 3.14 ms of regex extraction, 1.05 ms of forest prediction, and 0.01 ms of standardization. Regex feature extraction grows linearly with N.

Table 7: **Routing latency and GPU memory use.** CARROT queries, batch size one. Timings include feature extraction and routing-head inference.

Router	p50 (ms)	p99 (ms)	VRAM (MB)
GPU			
MiniLM-L6	5.0	10.4	204
E5-base	8.2	13.6	832
Qwen2.5-0.5B	19.5	25.8	2,617
ModernBERT-large	21.6	27.7	1,609
CPU			
MiniLM-L6	11.6	40.6	—
E5-base	53.4	256.6	—
Qwen2.5-0.5B	200.5	846.0	—
ModernBERT-large	219.6	863.8	—
Ours ($N = 32$)	1.4	11.9	—
Ours ($N = 128$)	2.2	37.1	—

The combined test queries from our four benchmarks have a median length of 249 characters and a 99th-percentile length of 6,607 characters. Table 8 reports latency on a sample of 2,979 queries from these benchmarks, reweighted to match this length distribution. The p99 of the 128-feature router exceeds the GPU baselines, while the 32-feature router stays below all of them except MiniLM-L6 and E5-base.

Table 8: **Traffic-weighted latency (ms)** on a pooled trace of 2,979 queries, batch size one, reweighted to the real query-length distribution.

router	device	p50	p90	p99
32 regexes (ours)	CPU	1.8	3.8	11.5
128 regexes (ours)	CPU	3.0	8.6	33.6
MiniLM-L6	GPU	4.9	5.4	7.8
E5-base	GPU	8.2	8.9	10.8
Qwen2.5-0.5B	GPU	19.4	20.1	22.6
MiniLM-L6	CPU	13.2	30.7	36.3
E5-base	CPU	57.8	147.3	198.6

C Ablation Details

C.1 Choice of discovery LM and SAE

Table 9 gives the per-benchmark scores and SAE dictionary sizes for the discovery-model comparison in Figure 4(a). On each benchmark, routing accuracy is similar across the evaluated discovery LM–SAE pairs. The pretrained SAE checkpoints and discovery corpus are described in Appendix B.2.

Table 9: **Discovery LM and SAE.** Test routing accuracy (%) with $N = 128$. Each row corresponds to a discovery LM–SAE pair, and K is the number of latents in the SAE.

Source LM	K	EmbedLLM	NineBy30k	R2Bench	CARROT	Avg.
Gemma-2-2B	16,384	67.37	69.01	82.16	87.18	76.43
Qwen3-1.7B	32,768	67.53	68.91	82.15	87.18	76.44
Llama-3.1-8B	32,768	67.07	69.27	81.95	87.19	76.37

C.2 Choice of latent grouping

We compare spectral clustering with three alternatives: spherical k-means, facility-location selection, and average-linkage agglomerative clustering. All four methods group the same 16,384 Gemma SAE latents using their document-level activation vectors on the 13,394 fit documents. For each method and N, we construct a regex library using the same authoring budget and alignment feedback, then evaluate it with an Extra-Trees routing head.

Spectral clustering retains the 50 largest absolute correlations per latent, symmetrizes the graph, and clusters the resulting spectral embedding into N families. Spherical k-means clusters mean-centered, unit-normalized activation vectors using k-means++ initialization, 60 iterations, and random seed 0. Agglomerative clustering uses average linkage with dissimilarity equal to one minus the absolute correlation. Facility-location selection greedily chooses N representative latents to maximize coverage, weighted by each latent's activation variance across documents. Coverage is measured by a latent's largest absolute correlation with any selected representative. Each latent is then assigned to the representative with the highest absolute correlation.

Table 10 reports mean test routing accuracy across four benchmarks for each method and feature count N. Table 11 reports routing accuracy and family-size statistics at $N = 128$, where spectral clustering produces the smallest maximum family. We prefer a small maximum family size because each family is approximated by a single extractor, and a very large family combines many distinct properties that one surface pattern is unlikely to track.

Table 10: Mean test routing accuracy (%) across four benchmarks by grouping method and feature count N. Each cell reports a separately authored regex library evaluated with an Extra-Trees head.

N	Spectral	Spherical k-means	Agglomerative	Facility location
1	73.48	73.64	**73.66**	73.65
2	73.80	73.82	73.80	**73.96**
4	**75.08**	74.41	73.73	73.89
8	**75.24**	74.54	74.95	74.95
16	75.62	74.80	75.35	**75.66**
32	76.06	75.23	**76.12**	75.69
64	75.98	**76.25**	76.03	75.97
128	**76.43**	76.10	76.03	76.03
256	76.19	**76.39**	76.25	76.25
512	**76.31**	76.15	76.31	76.14
1,024	76.31	76.17	76.28	**76.32**

Table 11: Routing performance and family sizes at $N = 128$ with different clustering strategies.

Grouping method	Accuracy (%)	Latents per family		
		Minimum	Median	Maximum
Spectral clustering	76.43	31	115.0	439
Spherical k-means	76.10	22	95.5	531
Agglomerative clustering	76.03	1	15.0	4,904
Facility-location selection	76.03	1	56.5	1,831

C.3 SAE GUIDANCE AND REFINEMENT

The ablation in Section 6.4 gives every variant the same budget of six candidate extractors per output feature. We keep the authoring model, regex interface, vocabulary constraints, and routing evaluation protocol fixed across variants. We use the corresponding prompt templates in Appendix A. Each variant proposes three candidates together in each of two rounds and retains one of the six, yielding a library of N features. Duplicate proposals consume candidate slots, and no variant stops early or receives replacement proposals.

The LLM-only variant receives no SAE evidence or alignment feedback and retains the LLM's preferred candidate among all six for each feature. The two SAE-guided variants use the same Gemma-2-2B SAE partition and family briefs. Here, refinement refers to returning alignment feedback to the authoring LLM so it can revise its proposals. The variant without refinement (*initial regexes* in Table 2) proposes all six candidates without receiving fit feedback. The full procedure instead returns all three initial candidates and their fit scores to the LLM before it proposes three revised or alternative candidates. Both guided variants select one extractor from all six using Kendall τ_b on the unlabeled fit corpus, with the same aggregation preferences specified in Appendix A.

Comparing LLM-only with the variant without refinement measures the combined effect of SAE-family information and alignment-based candidate selection. Comparing the two SAE-guided variants measures the additional effect of feedback-driven revision, with the candidate budget and final selection rule held fixed.

To compare alignment across libraries, we select a one-to-one regex–family matching that maximises mean Kendall τ_b on fit documents, then report its mean alignment on held-out documents. This matching is used only for evaluation after authoring. It does not provide SAE feedback to the LLM-only control or change the extractors or routing results.

C.4 ROUTING-HEAD DETAILS

We use an Extra-Trees regressor with at least 25 samples per leaf, trained on the full per-model score vector. For each representation and benchmark, we select the number of trees, maximum depth, and features per split on validation from $\{(300, \infty, 1.0), (300, 12, 1.0), (300, 24, 1.0), (100, \infty, 1.0), (300, \infty, \sqrt{d}), (100, \infty, \sqrt{d})\}$, where 1.0 means all d coordinates are considered at every split, and read test once. For the ablation in Section 6.5, we additionally evaluate a decision tree (maximum leaves in $\{8, 16, 32, 64, 128\}$), a random forest using the forest grid above, cosine k-nearest neighbors ($k \in \{10, 25, 50, 100\}$), and mini-batch k-means ($K \in \{50, 100, 150, 200\}$).

D COMPLETE REGEX FEATURE LIST

Table 13 enumerates all 128 frozen regex features. The indices f1–f128 follow the category order used in Figure 5. The eight categories describe surface properties rather than task types and are distinct from the SAE latent families used during discovery. These regex feature extractors are obtained from Gemma-2-2B using the pipeline described in Section 4.

Table 12: **Choice of routing head.** Mean test routing accuracy (%) across four benchmarks, with hyperparameters selected on validation. RF: random forest; ET: Extra-Trees. Bold marks the best representation for each head.

Representation	Tree	RF	ET	k-NN	k-means
128 regexes (ours)	**75.17**	**76.37**	**76.43**	75.01	75.09
TF-IDF bags	74.41	75.57	75.70	74.73	74.68
Llama-3.1-8B	74.37	76.18	76.41	75.77	75.61
Gemma-2-2B	74.31	76.11	76.39	75.85	75.58
Qwen2.5-0.5B	74.54	76.10	76.33	**76.04**	**75.73**
ModernBERT-large	74.08	75.68	75.91	75.40	75.41
ModernBERT-base	74.29	75.46	75.58	75.40	75.28
GTE-base	73.66	75.05	75.09	75.16	74.94
BGE-base	73.42	74.87	75.07	75.31	75.02
MiniLM-L6	73.49	74.81	75.02	75.28	74.83
MPNet-base	73.92	74.81	74.96	75.12	74.93
E5-base	73.60	74.78	74.92	75.57	75.09

For a query q, let $c_j(q)$ be the number of non-overlapping matches returned by Python's `re.findall` for pattern p_j. The library uses two aggregation rules:

$$\phi_j(q) = \begin{cases} c_j(q), & a_j = \texttt{count}, \\ c_j(q) / \max(|q|/5, 1), & a_j = \texttt{ratio}, \end{cases} \tag{10}$$

where $|q|$ is the number of characters in the query. Thus, `ratio` is a count per approximate word, using five characters per word. There are 124 `count` features and 4 `ratio` features. The additional flags `i` and `m` denote case-insensitive and multiline matching, respectively. A dash means no additional flags. For portable typesetting, literal non-ASCII characters are written as equivalent Python regex Unicode escapes (`\uXXXX`).

Table 13: Complete feature library in the display order of Figure 5. Each entry gives its name, aggregation rule, additional matching flags, and full regex.

Index	Feature name	Aggregation; flags
Script (12 features)		
f1	`cat_accent_elision_union`	`count;` —
	`(?:[\u00e0\u00e8\u00e9\u00ed\u00f2\u00f3\u00fa\u00ef\u00e7\u00c0 \u00c8\u00c9\u00cd\u00d2\u00d3\u00da\u00cf\u00c7\u00b7]|\b[LlDdSsNn MmTt][\u2019'](?=[a-z\u00e0-\u00ff]))`	
f2	`it_accent_ratio`	`ratio;` —
	`[\u00e0\u00e8\u00e9\u00ec\u00f2\u00f9\u00c0\u00c8\u00c9\u00cc\u00d2 \u00d9]`	
f3	`cjk_code_split`	`count;` —
	`[\u4e00-\u9fff\u3000-\u303f\uff00-\uffef\d(){}\[\]*+]|[^\s\u4e00- \u9fff\u3000-\u303f\uff00-\uffef\d(){}\[\]*+]+`	
f4	`nonascii_or_titlecase`	`count;` —
	`(?:[^\x00-\x7f]|\b[A-Z][a-z]{2,}\b)`	
f5	`tr_dia_jsonstruct`	`count;` —
	`[\u0131\u0130\u011f\u011e\u015f\u015e\u00e7\u00c7\u00f6\u00d6\u00fc \u00dc]|\["|",|"\]|={6,}|#!/|-\*-|\?"`	

Continued on next page

Table 13 continued

Index	Feature name	Aggregation; flags
f6	spanish_marks_accents [\u00f1\u00d1\u00bf\u00a1\u00e1\u00e9\u00ed\u00f3\u00fa\u00c1\u00c9\u00cd\u00d3\u00da\u00fc\u00dc]	count; —
f7	cyrillic_words [\u0400-\u04ff]+	count; —
f8	years_roman_dia \b(?:1[6-9]|20)\d{2}\b|\b[IVX]{2,6}\b|[\u00e1\u00e9\u00ed\u00f3\u00fa\u00f1\u00c1\u00c9\u00cd\u00d3\u00da\u00d1\u00bf\u00a1]|(?<=[a-z]\s)[A-Z][a-z]{2,}	count; —
f9	de_umlaut_sz_ratio [\u00e4\u00f6\u00fc\u00c4\u00d6\u00dc\u00df]	ratio; —
f10	cyrillic_diacritic_titlecase (?:[\u0400-\u04ff\u0100-\u024f\u00c0-\u00ff]|\b[A-Z][a-z]{2,}\b)	count; —
f11	jp_all_ratio [\u3000-\u303f\u3040-\u30ff\u4e00-\u9fff]	ratio; —
f12	pt_marks [\u00e3\u00f5\u00e7\u00c3\u00d5\u00c7]	count; —
Casing (25 features)		
f13	propn_weighted_len (?=(?<=[a-z]\s)[A-Z][a-z]{2,})|\S+	count; —
f14	capwords_cnt \b[A-Z][a-z]+\b	count; —
f15	titlecase_run2_or_dots (?:\b(?:[A-Z][a-z]+[ ]){1,}[A-Z][a-z]+\b|(?:\b[A-Z]\.){2,})	count; —
f16	capitalized_word_count \b[A-Z][a-z]{2,}\b	count; —
f17	capitalised_words \b[A-Z][a-z]+\b	count; —
f18	capword_ct \b[A-Z][a-z]+\b	count; —
f19	propernoun_runs (?:\b[A-Z][a-z]+\b\s+){2,}	count; —
f20	acro_hyph_ct (?:\b[A-Z]{2,}\d*\b|\b[a-z]{2,}-[a-z]{2,}\b)	count; —
f21	capwords_cnt \b[A-Z][a-z]+\b	count; —

Continued on next page

Table 13 continued

Index	Feature name	Aggregation; flags
f22	capitalized_word_count \b[A-Z][a-z]+\b	count; —
f23	cap_word_count \b[A-Z][a-z]{2,}\b	count; —
f24	caps_and_bullets_cnt (?:\b[A-Z]{2,}\b|\b[A-Z][a-z]+\b|^\s*[-\u2022*\u00b7]\s)	count; m
f25	propn_excl_weighted (?=(?<=[a-z]\s)[A-Z][a-z]{2,})|(?=!)|\S+	count; —
f26	curly_and_names (?:[\u2018\u2019\u201c\u201d\u2026\u2013\u2014]|\b[A-Z][a-z]+\b)	count; —
f27	cap_acro_quote_ct (?:\b[A-Z][a-z]+\b|\b[A-Z]{2,}\b|[,.]\u201d)	count; —
f28	typographic_or_caps_count (?:[\u2019\u201c\u201d\u2014]|\b[A-Z][a-z]+\b)	count; —
f29	edu_prose_count (?:[\u2019\u201c\u201d-]|\b[A-Z][a-z]+\b|\b[a-z]{8,}\b)	count; —
f30	acro_weighted_len (?=\b[A-Z]{2,6}\b)|(?=[\u201c\u201d])|\S+	count; —
f31	acad_prose_count (?:\b[A-Z][a-z]+\b|\([^()\n]{0,60}\)|\b[A-Za-z]{10,}\b)	count; —
f32	art_editorial_count (?:[\u2018\u2019\u201c\u201d\u2013\u2014\u2026\u2033]|!|\b[A-Z][a-z]+\b)	count; —
f33	codes_and_caps (?:\b[A-Z]{2,5}\b|\b[A-Z][a-z]+\b)	count; —
f34	names_clitics_punct (?:\b[A-Z][a-z]+\b[\u2019'](?:s|t|d|ll|re|ve|m)\b|[\u201c\u201d\u2019\u2026\u2014])	count; —
f35	sci_weighted_len (?=\b[A-Z]{2,6}\b)|(?=\b[A-Z][a-z]{2,}\b)|\S+	count; —
f36	quote_or_caps_count (?:[\u2019'\"\u201c\u201d]|\b[A-Z][a-z]+\b)	count; —
f37	sports_cap_num_weighted (?=(?<=[a-z]\s)[A-Z][a-z]{2,})|(?=\b\d{1,3}\b)|\S+	count; —

Continued on next page

Table 13 continued

Index	Feature name	Aggregation; flags
Prose (31 features)		
f38	informal_markers_cnt (?:(?<![\w’\u2019])[A-Z](?![\w’\u2019])|[’\u2019][a-z]{1,2}\b|[!?\u2026])	count; —
f39	prose_punct_weighted (?=\b[a-z]{2,}\b)|(?=[,\u2019\u201c\u201d\u2014\u2013])|\S+	count; —
f40	lower_word_runs (?:\b[a-z]{2,}\b[,]){3,}	count; —
f41	uk_marks_cnt (?:\u00a3|\u2018|\u2019[.,]|[.,]\u2019|\s\u2013\s)	count; —
f42	long_word_or_comma_count (?:\b[A-Za-z]{9,}\b|,|\b[a-z]+-[a-z]+\b)	count; —
f43	subword_pieces (?:[A-Za-z]{1,4}|\d|[^\sA-Za-z\d])	count; —
f44	informal_marks_count (?:[?!]|\b[A-Za-z]+[’\u2019][a-z]{1,2}\b|(?<![A-Za-z])I(?![A-Za-z]))	count; —
f45	quotes_plus_capwords_cnt (?:[\u201c\u201d\"]|[,.][\u201d\"]|\b[A-Z][a-z]+\s+[A-Z][a-z]+\b|\b[A-Z][a-z]+\b)	count; —
f46	words8_cnt \b[a-z]{8,}\b	count; —
f47	quote_double_weighted (?=[\u2019\u2018\u201c\u201d\u2026\u2014\u2013])|(?<=[\u2019\u201c\u201d])|(?=\b[a-z]{2,}\b)|\S+	count; —
f48	excl_time_contraction_cnt (?:!|\b\d{1,2}(?::\d{2})?\s*[AaPp]\.?[Mm]\.?\b|[’\u2019][a-z]{1,2}\b)	count; —
f49	word_ct \b[a-z]{2,}\b	count; —
f50	comma_or_sentence_count (?:[.!?]\s+[A-Z]|,)	count; —
f51	clitic_and_quotepunct (?:[\u2019’](?:s|t|d|ll|re|ve|m)\b|[.,!?][\"\u201d\u2019])	count; —
f52	comma_hyphen_caps_count (?:,|\b[a-z]+-[a-z]+\b|\b[A-Z][a-z]+\b)	count; —

Continued on next page

Table 13 continued

Index	Feature name	Aggregation; flags
f53	prose_and_quotes (?:\b[a-z]{4,}\b|[\u201c\u201d])	count; —
f54	cv_weighted_len (?=\b(?:[bcdfgjklmnprstvwy][aeiou]){3,}\b)|(?=\.{3,})|(?=\u2013)|\S+	count; —
f55	prose_q_len (?=\b[a-z]{4,}\b)|(?=[?,])|\S+	count; —
f56	sentence_ct [.!?][\"\u201d\u2019]?\s+[A-Z]	count; —
f57	shortwords_cnt \b[a-z]{1,4}\b	count; —
f58	prose_title_bold_union (?:\b[a-z]{4,}\b|\b[A-Z][a-z]+\b|**)	count; —
f59	allword_ct \b[A-Za-z]{2,}\b	count; —
f60	alpha_words \b[A-Za-z]{2,}\b	count; —
f61	comma_cnt ,	count; —
f62	word_ct \b[a-z]{2,}\b	count; —
f63	dquote_weighted_len (?=[\u201c\u201d])|(?=\b\d{2}\b)|\S+	count; —
f64	long_words6 \b[a-z]{6,}\b	count; —
f65	legal_formal_cnt (?:\b[A-Z]{2,}\b|\u00a7|;|\([a-z]\)|\b[a-z]{9,}\b)	count; —
f66	prose_words4 \b[a-z]{4,}\b	count; —
f67	words_and_typography_cnt (?:\b[A-Za-z]+\b|[\u2019\u201c\u201d\u2018\u2013\u2014\u2026])	count; —
f68	alpha_words_all \b[A-Za-z]+\b	count; —

Continued on next page

Table 13 continued

Index	Feature name	Aggregation; flags
Layout (8 features)		
f69	short_line_ct ^.{1,45}$	count; m
f70	digit_head_split2 [\d?$\u2022\|%]\|^#{1,6}\|^[\t]*[\u2022*-][\t]\|\.{3}\|[^\s\d?$\u2022|%]+	count; m
f71	recipe_layout_union (?m)(?:^.{1,60}$|[\u00b0\u00bc\u00bd\u00be]|^[ ]*(?:[-*\u2022]|\d+[.)])[ ]+|\b\d+[ ]?(?:kg|g|ml|oz|lb)\b)	count; —
f72	layout_junk (?: {2,}|\t|[#=*_~-]{3,}||)	count; —
f73	nonword_or_indent (?m)(?:[^\w\s]|^[\t]{2,})	count; —
f74	md_q_math_lists (?:#{2,}|\.{3,}|\?|\$|^\s*\d+[.)]\s)	count; m
f75	bracket_quote_count (?:[(){}\[\]]|\"|’)	count; —
f76	punct_layout_cnt [,.:\n(#/\-]	count; —
Numbers (6 features)		
f77	digit_count \d	count; —
f78	nums_money_entities (?:\b\d[\d,.]*\b|%|[$\u00a3\u20ac]|\b[A-Z][a-z]+\b)	count; —
f79	digit_count \d	count; —
f80	digits_plus_biblio_cnt (?:\d|(?:[A-Z]\.\s*){2,}|\.,|\u2116|\u00b7)	count; —
f81	digit_count \d	count; —
f82	num_mark_ct (?:[0-9]|[:@\u2022#|])	count; —
Paths (3 features)		
f83	path_like_cnt (?::/|/[A-Za-z0-9_.-]+/|[A-Za-z]:\\\|\.[a-z]{2,4}\b/)	count; —

Continued on next page

Table 13 continued

Index	Feature name	Aggregation; flags
f84	install_union_ct (?:\.\.?/|://|>=|*/|[\u00f8\u00e6\u00e5\u00d8\u00c6\u00c5]|--[a-z]) 	count; —
f85	web_typo_caps_count (?:[\u2019\u201c\u201d]|https?://|www\.|\.com\b|(?<![\w.])@\w|\b[A-Z][a-z]+\b) 	count; —
Math (11 features)		
f86	tex_math_broad (?:\\[a-zA-Z]+|\$|[{}^_]) 	count; —
f87	num_ops \d+(?:\.\d+)?|[=\u00d7\u2248\u00b0*/^]|** 	count; —
f88	tex_full_ct (?:\\[a-zA-Z]+|\$|[_^{}]) 	count; —
f89	tex_cmd_brackets (?:\\[a-zA-Z]+|[{}\[\]]) 	count; —
f90	acronym_or_unit_count (?:\b[A-Z]{2,5}\b|\b\d+(?:\.\d+)?\s?[munpkKMG]?(?:Hz|dB|[VWAF\u03a9])\b) 	count; —
f91	chem_plus_longwords_cnt (?:\b(?:[A-Z][a-z]?\d{0,2}){2,}\b|[A-Z][a-z]?\d|\u00b0|\(aq\)|\b[a-z]{9,}\b) 	count; —
f92	math_sym_var \$|\\[a-zA-Z]+|[{}^_]|\d|[+\-*/=\u00d7]|\b[b-hj-z]\b 	count; —
f93	math_symbols_wide_cnt (?:[\$\^_{}\\]|[=+]|\\[a-zA-Z]+) 	count; —
f94	latex_math_number_union (?:\\[A-Za-z]+|[$^_{}\\\u0370-\u03ff]|\b\d+(?:\.\d+)?\b|[\u00d7\u00b1\u223c\u2248\u2192\u2208\u221e]) 	count; —
f95	latex_dense2 \\[a-zA-Z]+|\\\\\|[&{}$_^]|\\[^\sa-zA-Z] 	count; —
f96	math_num_ct (?:\$[^$\n]{1,60}\$|\b\d+\b) 	count; —
Code (32 features)		
f97	lic_code_dense [#_="\'()/:.]|\b[A-Z]{3,}\b|^\s{4,}\S|\b(?:import|from|def|class|return)\b 	count; m

Continued on next page

Table 13 continued

Index	Feature name	Aggregation; flags
f98	c_code_chars (?:[;{}()\[\]<>=*&]|::|->|\+\+|#\s*include|\b(?:int|char|float|double|void|struct|class|template|namespace|public|private|const|static|return|sizeof)\b)	count; —
f99	camel_or_class (?:\b[a-z]+[A-Z][A-Za-z]*\b|\b[A-Z][a-z]+(?:[A-Z][a-z]+)+\b)	count; —
f100	code_all2_ct (?:[{}\[\]();=<>_]|^[\t]{2,}|\b(?:def|return|import|func|fmt|nil|mut|pub|let|var|int|float|self|if|else|for|class|try|new|struct|enum|void|const)\b)	count; m
f101	banner_caps_comment_union (?m)(?:\b[A-Z]{2,}\b|^[ ]*[#*/;%!-]+|\(C\)|\u00a9|://|-*-)	count; —
f102	hdr_block_ct (?:^[\t]*#|^[\t]*(?:from|import)\b|\"\"\"|-{4,})	count; m
f103	py_file_full_cnt (?:^\s*(?:import|from|def|class)\s|^(?: {4})+\S|^\s*#|\"\"\"|__|\(\)|\):|\bself\b)	count; m
f104	lic_union_ct (?:^[\t]*#|^[\t]*(?:from|import|def|class)\b|\b[A-Z]{3,}\b|\"\"\")	count; m
f105	script_banner_or_longword (?m)(?:^[\t]*(?:import|from)[\t]+[A-Za-z_]|#!/|-*-|://|^[-=#]{5,}|\b[a-z]{8,}\b)	count; —
f106	py_dense_sym2 [_().\'\[\]=]|\b(?:def|class|import|from|return|self)\b|://|^\s*(?:from|import)\s|@\w|\n	count; m
f107	java_full_count (?:\bpublic\b|\bprivate\b|\bprotected\b|\bstatic\b|\bvoid\b|\bthrows\b|\bextends\b|\bimplements\b|\bimport\b|\bfinal\b|\bnull\b|\bString\b|\bint\b|\bboolean\b|\);|\]\);|\"\);\;\s*$\|\{\s*$|\}\s*$|\(\)\s*\{)	count; m
f108	py_full_ct (?:^ {4,}|[()\[\]{}=_.;]|\b(?:self|def|import|class|return|None|True|False)\b)	count; m
f109	py_module_full_count (?:^(?: {4})+\S|\bself\b|\bdef\b|\bclass\b|==|\bNone\b|\bimport\b|\bfrom\b|__\w+__|\"\"\"|\):)	count; m
f110	header_broad (?:^#|^\s*(?:import|from)\b|-*-|#!/|\b[A-Z]{4,}\b|\"\"\")	count; m

Continued on next page

Table 13 continued

Index	Feature name	Aggregation; flags
f111	py_kw_line_starts (?m)^[\t]*(?:import|from|def|class|if|for|while|return|try|except|with|print)\b	count; —
f112	php_sigils_wide <\?|\?>|->|\$[A-Za-z_]\w*|\);|\';|=>|#!/|::|&&||||/>|</?[a-z][a-z0-9]{0,6}>	count; —
f113	mig_union2_ct (?:^ {8,}|\('|',|'\)|\{'|'\]|\)\),|=\[|\b(?:True|False|None)\b|\b(?:[A-Z][a-z]+){2,}\b)	count; m
f114	code_punct_r [{}()\[\];=<>_]	ratio; —
f115	ml_code_idiom_count (?:\w+\.\w+\(|\[[^\]\n]{0,30}[:,][^\]\n]{0,30}\]|\b\w+_\w+\b|\bimport\b)	count; —
f116	markup_chars_wide [<>&;="'/{}%#]	count; —
f117	brace_markup_count (?:\}\}|\{\{|\$\{|_\{|=\"\"|=')	count; —
f118	docstring_struct """|^\s*#|^-{3,}|^\s*:[a-z]+[\s:]|@\w|^\s{4,}\S|^\s*(?:def|class)\s	count; m
f119	py_bracket_ops_cnt (?:\bdef\b\):|\[|\]|\(|\)|=)	count; —
f120	script_struct_cnt (?:^\s*(?:import|from)\s|^\s*#|^(?: {4})+\S|\bos\.|\bsys\.|\bdef\b|\(\))	count; m
f121	web_code_cnt (?:\}\);|'\);|\"\);|=>|</|<[a-zA-Z][^>\n]{0,60}>|\b(?:var|let|const|function|return|import|export|document|window|require|module|class)\b|\.(?:js|jsx|ts|tsx|css|html)\b)	count; —
f122	py_dense [_()=.]|\b(?:def|class|import|from|return|self|if|for|None|True|False)\b|\n	count; —
f123	caps_url_comment (?:\b[A-Z]{2,}\b|https?://|\u00a9|^\s*#|/*|*/)	count; m
f124	markup_punct_ct [<>{}=\";:/]	count; —
f125	code_punct_wide_cnt [(){}\[\]=_/#.:;`]	count; —

Continued on next page

Table 13 continued

Index	Feature name	Aggregation; flags
f126	`pkg_full_ct` `(?:'[^'\n]{0,40}'|\"[^\"\n]{0,40}\"|^ {4,}|\bimport\b|\bfrom\b|\bdef\b|\bclass\b)`	`count;m`
f127	`indent_and_flow_count` `(?:^(?: {4})+\S|\bself\b|\bdef\b|==|\bNone\b)`	`count;m`
f128	`py_oo_keywords` `\b(?:self|def|class|return|None|import|assert|raise|except|True|False|super)\b`	`count;—`

E SOURCE DATASET FEATURE PROFILES

Figures 9–11 extend Figure 5 with the ten largest source datasets in each query pool. In EmbedLLM, PIQA peaks on short-line counts, whereas MMLU professional law has higher word and long-word counts. In the shared CARROT/R2Bench pool, MATH is elevated on LaTeX and mathematical-symbol features, while MuSR sources have higher sentence-boundary and punctuation counts. These examples show that datasets have distinct average profiles across our regex coordinates, providing interpretable cues for distinguishing their queries.

Each bar is a source's mean feature value after standardizing query values with the corresponding training statistics and clipping to $[-4, 4]$. All panels use the same order of surface properties as Figure 5. The complete list of the extractors is in Appendix D.

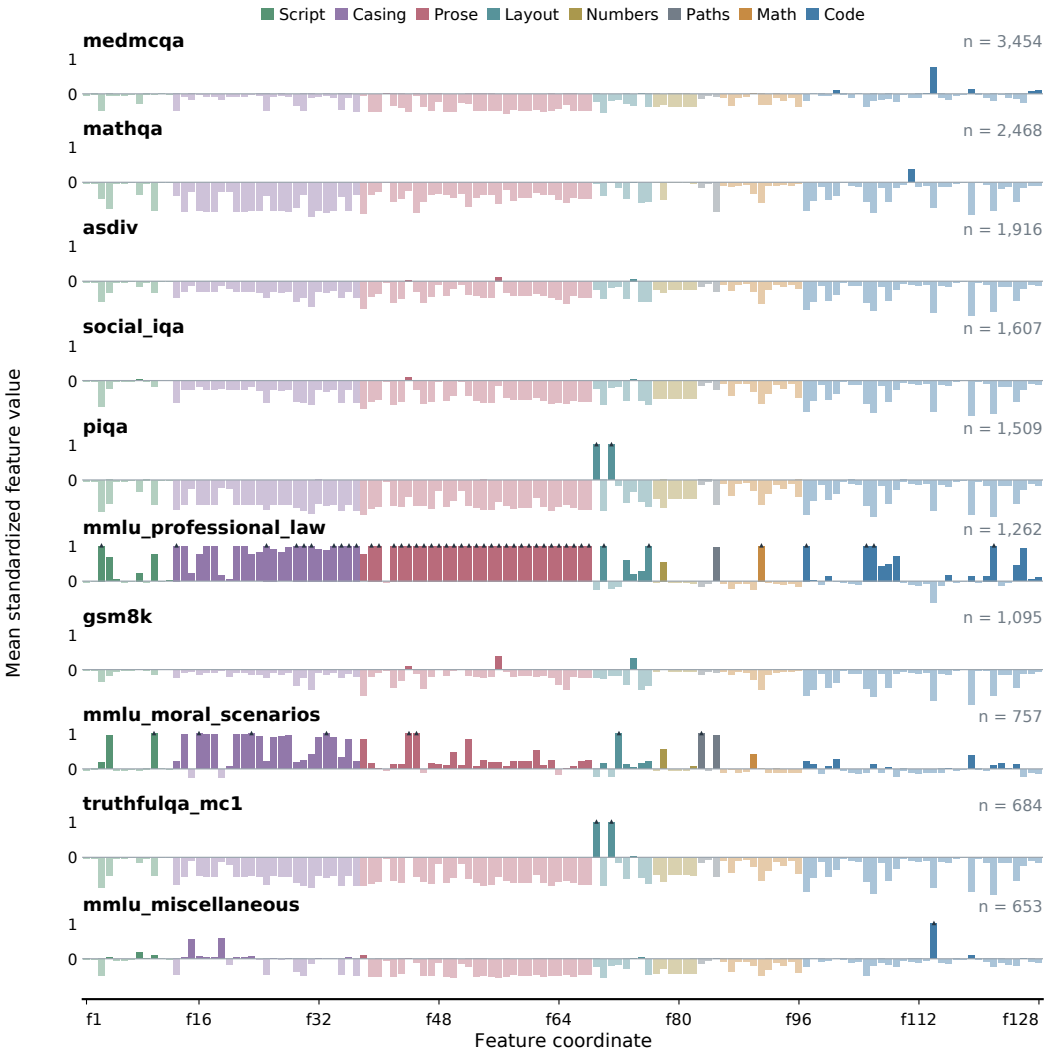


Figure 9: **Source feature profiles for EmbedLLM.** The ten largest training sources, with one bar per feature. Triangles mark means above the $+1$ display limit; n denotes the source's query count.



Figure 10: **Source feature profiles for NineBy30k.** The ten largest training sources, with the same feature order, colors, and display limits as Figure 9.

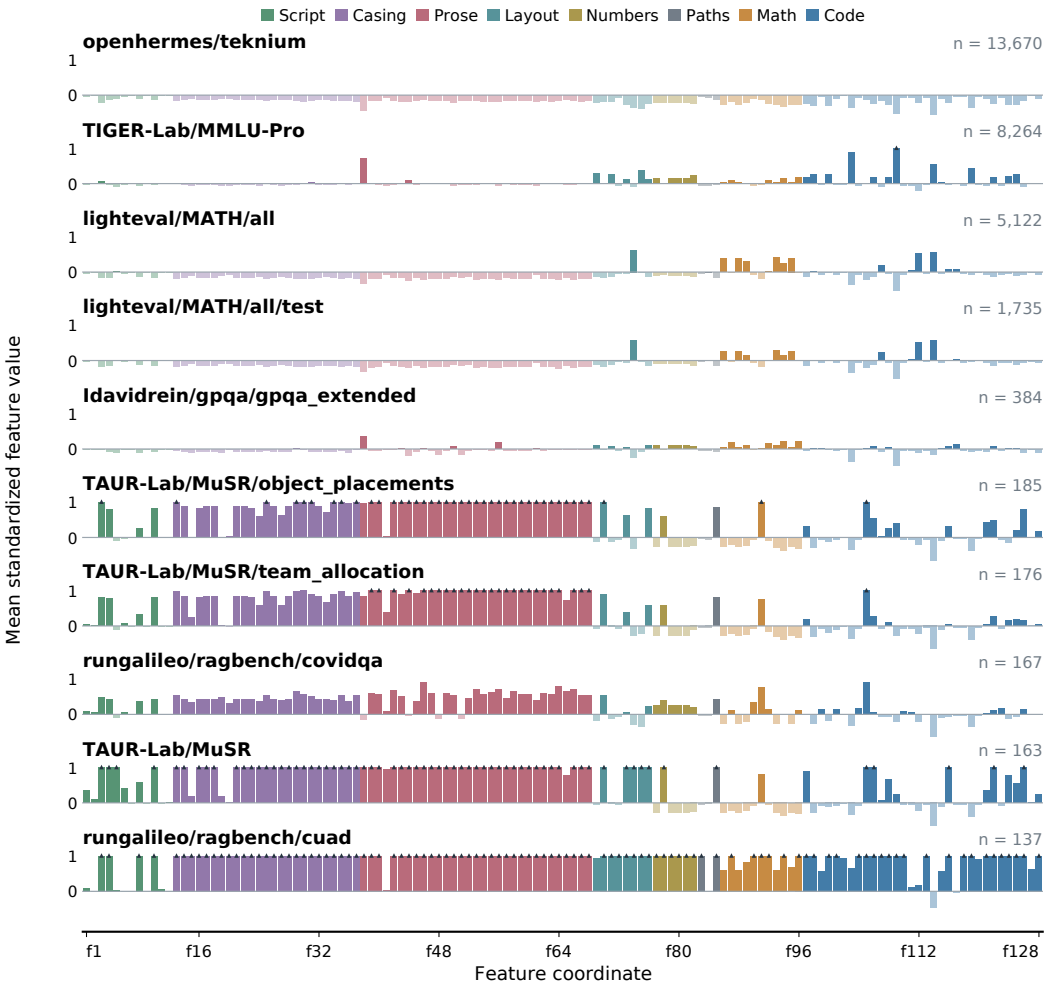


Figure 11: **Shared source feature profiles for CARROT and R2Bench.** The ten largest source groups in CARROT training, which contains R2Bench's full query pool. Counts and scaling use this shared pool.

F QUERY NEIGHBORHOODS

F.1 SETUP AND DEFINITIONS

We provide the setup and per-benchmark results for Figure 6(a,b), comparing our 128 regex features with E5-base, ModernBERT-base, and Qwen2.5-0.5B. We use the full test splits in Table 3 and select $k = 20$ neighbors for each query from the other test queries.

Forest neighborhoods. As in the main analysis, neighbors are defined by shared leaves in an Extra-Trees router. For each representation, we fit the router to training model-score vectors, using 100 trees, unrestricted depth, square-root feature subsampling, minimum leaf size 25, squared-error loss, and seed zero. These settings are fixed across representations for this analysis. We pass test queries through the frozen forest and rank neighbors by the fraction of trees in which they share a terminal leaf with the query, comparing leaf identities within each tree.

Evaluation metrics. Let d_i be query i's original source label and $r_{i,m} \in [0, 1]$ its observed response score for model m. For a query pair (i, j), we define source agreement and model-score similarity as

$$A(i, j) = \mathbf{1}[d_i = d_j], \qquad S(i, j) = 1 - \frac{1}{M} \sum_{m=1}^{M} |r_{i,m} - r_{j,m}|. \tag{11}$$

We average each metric over the same k neighbors and all test queries, including those with constant score vectors, and report percentages. The random reference averages each metric over all distinct test-query pairs within a benchmark.

Direct nearest neighbors. As an additional comparison, we select neighbors by Euclidean distance in each representation before fitting a router. We standardize each coordinate using its training mean and standard deviation, with scale one for zero-variance coordinates. The regex inputs are already z-scored and clipped to $[-4, 4]$; we standardize these clipped inputs again before computing distances. We use the same $k = 20$, test queries, and evaluation metrics as for forest neighborhoods.

F.2 NEAREST-NEIGHBOR RESULTS

Table 14 reports the per-benchmark results underlying Figure 6(a,b). With forest neighborhoods, REGEXROUTE reaches 68.24% mean source agreement and 64.97% model-score similarity, compared with 68.09% and 65.48% for Qwen2.5-0.5B. Both means exceed those of E5-base and ModernBERT-base, as well as the random reference of 17.55% and 57.74%, respectively.

Tables 15 and 16 report direct Euclidean neighbors. Here, REGEXROUTE reaches 67.00% source agreement and 64.62% model-score similarity, while Qwen2.5-0.5B reaches 72.27% and 65.62%. Thus, regex features group queries by source and model performance under both definitions, although Qwen has stronger direct neighborhoods.

Table 14: **Forest neighborhoods, per benchmark.** Source agreement and model-score similarity (%) at $k = 20$ for the representations in Figure 6(a,b).

A. Source agreement

Representation	EmbedLLM	NineBy30k	R2Bench	CARROT	Mean
Random	3.91	6.72	30.08	29.48	17.55
E5-base	24.71	35.77	72.46	66.29	49.81
ModernBERT-base	38.78	58.57	81.67	79.08	64.53
Qwen2.5-0.5B	44.82	60.12	84.84	82.59	68.09
RegexRoute	40.02	64.11	85.12	83.72	68.24

B. Model-score similarity

Representation	EmbedLLM	NineBy30k	R2Bench	CARROT	Mean
Random	53.34	53.04	62.72	61.83	57.74
E5-base	58.60	57.14	69.44	68.47	63.41
ModernBERT-base	59.92	58.75	70.60	69.80	64.77
Qwen2.5-0.5B	60.46	59.40	71.45	70.62	65.48
RegexRoute	59.79	59.07	70.66	70.35	64.97

Table 15: **Source agreement under direct Euclidean neighborhoods.** Percentage of $k = 20$ other test-query neighbors sharing the anchor's original dataset label, after standardizing coordinates using training statistics.

Representation	EmbedLLM	NineBy30k	R2Bench	CARROT	Mean
Random	3.91	6.72	30.08	29.48	17.55
E5-base	44.75	58.33	83.94	82.56	67.40
ModernBERT-base	41.88	64.73	83.99	82.53	68.28
Qwen2.5-0.5B	49.46	69.51	85.60	84.53	72.27
RegexRoute	37.80	63.05	84.33	82.83	67.00

Table 16: **Model-score similarity under direct Euclidean neighborhoods.** Similarity (%) on the same $k = 20$ neighbors as Table 15.

Representation	EmbedLLM	NineBy30k	R2Bench	CARROT	Mean
Random	53.34	53.04	62.72	61.83	57.74
E5-base	60.56	58.75	71.10	70.10	65.13
ModernBERT-base	59.56	58.86	70.91	70.19	64.88
Qwen2.5-0.5B	61.39	59.70	71.35	70.02	65.62
RegexRoute	59.01	59.16	70.37	69.95	64.62

G Feature Recovery Across Encoder Sizes

G.1 Setup and definitions

We train linear ridge and nonlinear MLP probes to predict all 128 regex feature values from frozen query embeddings. For each encoder and benchmark, we fit the probes on the training split, select hyperparameters by validation mean squared error, and evaluate on the test split.

Query representations. We use the seven Qwen2.5 sizes from 0.5B to 72B with the last-token embeddings described in Appendix B.4. MiniLM-L6 and E5-base provide the smaller-encoder references in Figure 6(c). We also report ModernBERT-base in the tables below. The Qwen2.5 models provide the comparison of scale within one model family.

Linear and MLP probes. Ridge regression uses the original embedding coordinates, with regularization selected from $\{1, 10, \ldots, 10^6\}$. For MLP probe, we divide embeddings by their mean training-vector norm and fit a two-hidden-layer ReLU network with mean squared error loss. Training uses AdamW, weight decay 10^{-4}, batch size 512, and at most 80 epochs, with early stopping after eight epochs without validation improvement. We search four configurations, specified as (hidden widths, dropout, learning rate): $((256, 128), 0.1, 10^{-3})$, $((512, 256), 0.1, 10^{-3})$, $((512, 256), 0.3, 10^{-3})$, and $((1024, 512), 0.1, 3 \times 10^{-4})$. All runs use seed zero.

Recovery metric. For each feature j, we compare its predicted value $\widehat{\phi}_j(q)$ with its extracted value $\phi_j(q)$ across test queries:

$$R_j^2 = 1 - \frac{\sum_{q \in \mathcal{D}_{\text{route}}^{\text{test}}} (\phi_j(q) - \widehat{\phi}_j(q))^2}{\sum_{q \in \mathcal{D}_{\text{route}}^{\text{test}}} (\phi_j(q) - \overline{\phi}_{j,\text{test}})^2}, \tag{12}$$

where $\overline{\phi}_{j,\text{test}}$ is the feature's test mean. A score of one denotes perfect recovery; zero matches predicting this mean for every query.

The main figure reports $R_{\text{w}}^2 = \sum_j w_j R_j^2 / \sum_j w_j$ over these features. Weights are impurity-based feature importances, plus 10^{-9}, from an Extra-Trees regressor. We also report the unweighted mean and median feature R^2. Each summary is computed separately for each benchmark, then averaged equally across the four benchmarks.

G.2 Linear and nonlinear recovery results

Table 17 summarizes recovery across encoders, and Table 18 gives the per-benchmark results. From Qwen2.5-0.5B to 72B, importance-weighted R^2 rises from 0.820 to 0.848 with an MLP, compared with 0.612 to 0.751 with ridge. The small MLP gain also appears without importance weighting: mean feature R^2 rises from 0.734 to 0.761. Thus, the surface features are already recoverable from the small encoder, while scaling improves linear recovery more substantially.

Table 17: Feature recovery across encoder sizes. Each entry averages four per-benchmark summaries. Wtd. uses training-derived Extra-Trees importances; mean and median treat features equally.

	Ridge			MLP		
Encoder	Mean	Median	Wtd.	Mean	Median	Wtd.
Qwen2.5-0.5B	0.507	0.603	0.612	0.734	0.823	0.820
Qwen2.5-1.5B	0.542	0.642	0.652	0.751	0.843	0.840
Qwen2.5-3B	0.569	0.673	0.683	0.754	0.848	0.842
Qwen2.5-7B	0.594	0.701	0.713	0.754	0.854	0.841
Qwen2.5-14B	0.615	0.722	0.734	0.768	0.857	0.852
Qwen2.5-32B	0.581	0.718	0.727	0.748	0.841	0.833
Qwen2.5-72B	0.632	0.743	0.751	0.761	0.852	0.848
MiniLM-22M	0.345	0.393	0.424	0.689	0.770	0.751
E5-base	0.457	0.530	0.561	0.719	0.814	0.796
ModernBERT-base	0.563	0.662	0.673	0.696	0.790	0.774
TF-IDF	0.745	0.900	0.877	—	—	—

Table 18: Importance-weighted feature recovery by benchmark. Probe configurations are selected separately on each benchmark's validation split.

Encoder	EmbedLLM	NineBy30k	R2Bench	CARROT
Ridge (linear)				
Qwen2.5-0.5B	0.641	0.674	0.543	0.589
Qwen2.5-1.5B	0.667	0.721	0.589	0.632
Qwen2.5-3B	0.694	0.764	0.610	0.664
Qwen2.5-7B	0.740	0.775	0.644	0.692
Qwen2.5-14B	0.759	0.787	0.670	0.721
Qwen2.5-32B	0.773	0.776	0.661	0.699
Qwen2.5-72B	0.792	0.812	0.667	0.734
MiniLM-22M	0.570	0.422	0.323	0.380
E5-base	0.665	0.658	0.424	0.497
ModernBERT-base	0.792	0.723	0.561	0.618
TF-IDF	0.897	0.904	0.813	0.894
MLP (nonlinear)				
Qwen2.5-0.5B	0.875	0.869	0.768	0.769
Qwen2.5-1.5B	0.882	0.868	0.775	0.835
Qwen2.5-3B	0.889	0.883	0.766	0.832
Qwen2.5-7B	0.887	0.878	0.772	0.829
Qwen2.5-14B	0.893	0.882	0.786	0.844
Qwen2.5-32B	0.896	0.881	0.747	0.807
Qwen2.5-72B	0.903	0.889	0.766	0.832
MiniLM-22M	0.881	0.672	0.673	0.776
E5-base	0.904	0.803	0.674	0.803
ModernBERT-base	0.902	0.818	0.584	0.792