Tracking Universal Features Through Fine-Tuning and Model Merging

Anonymous ACL submission

Abstract

We study how features emerge, disappear, and persist across models fine-tuned on different domains of text. More specifically, we start from a base one-layer Transformer language model that is trained on a combination of the BabyLM corpus (Warstadt et al., 2023), and a collection of Python code from The Stack (Kocetkov et al., 2022). This base model is adapted to two new domains of text: TinyStories (Eldan and Li, 2023), and the Lua programming language, respectively; and then these two models are merged using these two models using spherical linear interpolation. Our exploration aims to provide deeper insights into the stability and transformation of features across typical transfer-learning scenarios using small-scale models and sparse auto-encoders.

1 Introduction

Language models are proving useful on an ever-widening range of tasks but there are still open questions about what is actually learned and represented in these models. Researchers have invested substantial energy into understanding language models, from both behavioural (Ribeiro et al., 2020) and mechanistic perspectives (Elhage et al., 2021). More recently, Bricken et al. (2023) proposed to study neural network features as a form of dictionary learning, in which a feature is a function that assigns values to data points. Their features are extracted from a sparse autoencoder using the activations of the MLP layer in a Transformer block. This approach to understanding what is learned by language models has proven successful, and can even control the output of language models pinning the features (Templeton et al., 2024).

In this paper, we study language model feature *evolution*, i.e. the emergence, disappearance, and persistence of features. We study feature evolution in two common transfer-learning settings: fine-tuning a language model to a new domain (Gururangan et al., 2020), and merging the weights of two models (Choshen et al., 2022). We conduct our experiments using a small language model trained from scratch on a combination of English text and the Python programming language.[1] This model is then separately fine-tuned on a different programming language (Lua), and more English text, after which the the fine-tuned models are merged back into a single model. Given this family of related models, we use sparse auto-encoders to extract and correlate feature activation patterns to study the evolution of features. We find that very few features persist between the studied models, but those that do persist are interpretable, e.g. they correspond to generic properties of the text, such as punctuation and formatting. We report case studies on a persistent feature that represents variable assignments in programming languages, and a disappearing feature that handles exceptions.

2 Related Work

Feature universality and convergence in Transformer language models is a growing area of interest. Prior work shows that models converge at universal feature representations (Li et al., 2015; Chughtai et al., 2023; Li et al., 2015; Huh et al., 2024), and more recent work shows these features can be extracted across different and divergent models (Bricken et al., 2023; Templeton et al., 2024; Cunningham et al., 2023). In Bricken et al. (2023), feature universality is the ability of a sparse auto-encoders to extract universal features across similar and divergent models.

Common approaches to adapting language mod-

[1] We believe that programming languages offer an interesting test-bed for feature evolution because on the one hand, they contain reserved keywords that overlap with common English words, e.g. “try”, “while”, “break”, etc; while on the other hand they have sequences not commonly seen in English text, such as code indentation blocks, e.g. “`\t`”, or “ ”. We study Python and Lua in this paper but we expect that our methods can be extended to other programming languages.

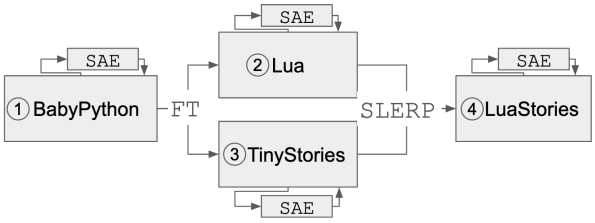

Figure 1: Overview of the experimental design. We start with a base model trained on BabyLM and Python code (1), which is fine-tuned (FT) on two new domains: the Lua programming language (2), and TinyStories (3). The fine-tuned models are merged into a single LuaStories model using spherical linear interpolation (SLERP) interpolation (4). For each of these models, we train a sparse auto-encoder on the MLP activations using the same data distribution as the original model.

els include fine-tuning on new data (Gururangan et al., 2020), and model merging (Choshen et al., 2022; Wan et al., 2024; Wortsman et al., 2022), especially for practical use cases (Goddard et al., 2024). However, little is known about the features in fine-tuned and merged models compared to their starting point. Our work extends Bricken et al. (2023) to study the feature dynamics of model merging, and fine-tuning, and further explore feature universality across models.

3 Methodology

The goal of our study is to investigate the evolution of Transformer language model features in common transfer-learning settings. In contrast to recent work on frontier models (Templeton et al., 2024; Gao et al., 2024), we use one-layer Transformers, which are quick to train and reproducible in lower-compute environments. Inspired by Bricken et al. (2023), our models are Mistral-like (Jiang et al., 2023) Transformers, with 1 self-attention block, followed by a 1024D MLP with a SiLU activation (Elfwing et al., 2018). The language model features are extracted using a sparse autoencoder trained on the MLP activations. Each language model and sparse auto-encoder can be trained in 10 hours, and 8 hours, respectively, on an NVIDIA A100 40GB GPU.[2]

3.1 Model Training and Fine-Tuning

The base model, `BabyPython`, is trained on a dataset containing the BabyLM 100M corpus (Warstadt et al., 2023) and a 10% sample of the Python subset of The Stack (Kocetkov et al., 2022), resulting in even amounts of Python and BabyLM tokens.[3] We fine-tune two further models: one on the Lua subset of The Stack, and the other on the TinyStories dataset (Eldan and Li, 2023). These fine-tuned models, named `Lua` and `TinyStories`, respectively, are then merged into a fourth model, `LuaStories`. All models are trained with an autoregressive log-likelihood loss. Figure 1 shows an overview of the relationship between the models.

3.2 Autoencoder Feature Extraction

The learned features used in our study are extracted from a sparse autoencoder, which is trained with an expansion factor of 16 on the output MLP activations of each Transformer language model.[4] We follow the general approach of Bricken et al. (2023), using 15 million tokens of 24-token blocks sampled uniformly from the combined trace of datasets used to train each respective model.[5] This process allows us to extract features from all of the underlying training distributions of each model. The sparse autoencoders are used to gather feature activation patterns, which are used to correlate individual extracted features across models, allowing us to observe the evolution of the features.

3.3 Model Merging

Given that all models start from the same base parameters, we can use generalised linear mode connectivity (Frankle et al., 2019) to enable model merging via spherical linear interpolation. Model merging techniques are used to combine multiple pre-trained models into a unified model that retains what is learned by the original models. We use spherical linear interpolation[6] (Shoemake, 1985), to interpolate all model parameters along a spherical path between the parameters of the Lua model to the parameters of the TinyStories model. At every fraction t along the path we measure the accuracy of the model corresponding to the interpolated parameters on both the Lua and TinyStories validation data, as measured by correct next-token

[2] Our code and weights will be available upon publication.

[3] 335.6M tokens in total, according to the model tokenizer.

[4] We follow best-practices, including tracking "dead" neurons in the autoencoder using the L0 norm, finding no significant problems. See Appendix B for details.

[5] For example, the `LuaStories` sparse auto-encoder is trained on data sampled from the combined BabyPython, Lua, and TinyStories datasets. Whereas the `Lua` sparse auto-encoder is trained on data sampled from the combined BabyPython and Lua datasets.

[6] We experimented with other merging methods, such as TIES-merging (Yadav et al., 2023), but none worked as well as Slerp. We leave further study of this for future work.

prediction. This allows us to select the merged model that retains the optimal balance between each model. Figure 5 in Appendix A shows the accuracy of the merged model at each fraction t along the interpolated path is shown. We pick the model corresponding to the parameters at equilibrium of the merged model accuracy on Lua and TinyStories at $t = 58\%$. This LuaStories model's accuracy is 20% lower than both original models in their respective domain, but approximately 20% better than the shared base model of the two original models. The LuaStories model is equally accurate in modelling Lua and TinyStories data.

3.4 Quantifying Feature Evolution

We quantify the evolution of features extracted from the sparse autoencoders of pairs of language models: a *parent* model and a *child* model. Following Bricken et al. (2023), features are *similar* if they take similar values over a diverse set of data, which is calculated by collecting feature activation patterns from the sparse autoencoder of each model.[7] We define a feature as *persisting* if there exists a feature pair with activation patterns that correlate more than 80% between the parent and child models.[8] A feature is *emerging* when we cannot find any feature in the parent model that correlates more than 80% with the features in the child model. And a feature is *disappearing* if there are no features in the child model that correlate sufficiently with a feature in the parent model.

4 Empirical Results

4.1 High-level Feature Flow

Figure 2 shows an overview of the emerging, disappearing, and persisting features across the models. First, we note that most of the features from the base model `BabyPython` disappear through fine-tuning, i.e. only 959 features persist into either of the fine-tuned models. In the final merged model, `LuaStories`, there are 1,210 features that can be traced back to either the `Lua` or `TinyStories` models, of which 729 emerged during fine-tuning, while the remaining 481 features persisted from the initial `BabyPython` model.

[7]We collect activations over 3 million tokens of data sampled from the common data distribution (see Footnote 5).

[8]The correlation threshold of 80% is based on several rounds of manual inspection. We find that sparse features with activation patterns that correlate more than 80% across are qualitatively similar enough to be considered the same.

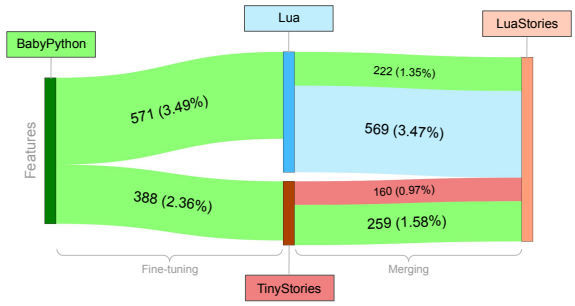

Figure 2: Overview of the features persisting through fine-tuning and model merging, showing volumes and trajectories of extracted features that emerge, persist and disappear. This overview omits the features that don't persist, and so the visual flows are scaled proportional to the persisting features. We note the share of features that persist from each model.

We use automated interpretability to explore the features that do persist through to the final LuaStories model (see Appendix C). By manually grouping verified explanations, we observe that 20% of features persisting from `BabyPython` → `Lua` → `LuaStories` track punctuation, and 5% can be explained as formatting features tracking indentation and different types of line breaks, which is a common property of the Python language. Although some features are seemingly monosemantically linguistic features, the remaining set of the features are less easily categorised. Overall, the majority of features flowing from the `BabyPython` model all the way to the `LuaStories` model are code related.

4.2 Feature Flow Case Studies

We now present a detailed examination of how two learned features evolve through our models. We study a feature that persists through both fine-tuning and merging, and we study a feature that disappears through fine-tuning. Feature behaviour is quantified using a proxy by checking the corresponding log-likelihood of a string under the under the hypothesised feature explanation, and under the full empirical activation distribution (see Appendix D for more details).

4.2.1 Persisting Variable Assignment Feature

A clear and persisting feature across all models is the variable assignment feature, originally found in the `BabyPython` feature #16336. This feature is activated by different types of variable assignment, as shown on a `Lua` example in Figure 3. We sweep for corresponding assignment

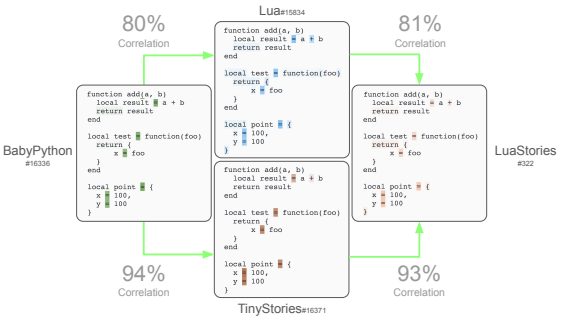

Figure 3: Visualisation of the feature activation patterns of the universally extracted variable assignment features found in each model. Each token is highlighted according to the feature's activation level, where darker background colour denotes higher level of activation. Additionally, we note the observed activation pattern correlations between each feature.

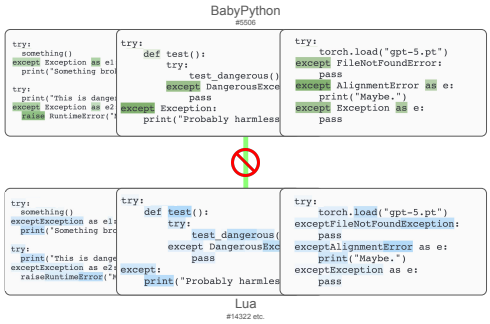

Figure 4: Examples of observed activation patterns of the `BabyPython` Python exception feature, and the closest matching feature in the `Lua` model, qualitatively showing insufficient correlation between the two.

features across each subsequent model, to find the strongest correlating feature activation patterns in the `Lua` and `TinyStories` model, and in turn in the `LuaStories` model. By analysing all similar features, we find a feature in each of the models that seems to be close to identical to the assignment feature found in the `BabyPython` model.

We find that our tracked set of features are qualitatively universal, by analysing top-activating tokens and contexts, and through manual inspection. To further confirm this, we look at the log-likelihood ratio under the "=" token feature hypothesis, and under the full empirical distribution computed as a 3 million token sample of the combined Lua and BabyPython dataset. Using this feature proxy, we find that the `BabyPython` feature has a log-likelihood ratio for assignments of 7.53, `Lua` has 6.58, `TinyStories` has 7.64, and `LuaStories` has 7.19. These features have a mean cross-correlation of 85.1%. Therefore, we conclude that the assignment feature persists, as per our 80% correlation threshold.

We observe the same universality across all models, confirming that our sparse auto-encoders are able to extract the same universal feature across all models. This further reinforces the claims of extraction universality of Bricken et al. (2023).

4.2.2 Disappearing Python Exception Feature

As we fine-tune the base model to instead specialise solely on Lua code, many syntactical Python constructs become redundant. Intuitively, we expect this to be reflected in the extracted features. In the previous section, we are able to trace and recover an assignment feature from the `TinyStories` model. In this case we show the feature that tracks Python code exceptions disappears during fine-tuning.

The Python exception feature in the BabyPython model tracks components of `except` clauses in Python code. Using the activation data for the Python exception feature in the BabyPython model, we sweep for similar activation patterns in the Lua model. Here we find a small handful of features.

Similar to §4.2.1 we look at the log-likelihood ratios. Here, BabyPython #5506 has a log-likelihood ratio of 11.38 under the simplified "`except`" token hypothesis, and under the full empirical distribution. Lua #14322 has a log-likelihood ratio of 8.59, and is 7 times more active for other tokens. These two features, while sharing some excitement for exceptions, correlate 40.81% across observed activation patterns. As such, we find that this Python exception feature disappears through Lua fine-tuning, because we don't see any matches for the BabyPython #5506 feature.

5 Conclusion

Using sparse auto-encoders, we have empirically mapped the evolution of learned features in small Transformer language models through realistic transfer-learning scenarios. We find that spherical linear interpolation of model parameters is able to maintain features of the parent models. We show that features are diluted through model merging and fine-tuning and, in our experiments, don't correlate as well as they would for two similar Transformers trained with the same hyper-parameters on identical data. Future work includes scaling this to deeper language models, and to further exploration of feature evolution in other domains.

Limitations

We conduct our experiments on just under half a billion tokens limited to the specific domains of English and programming languages, which may not capture feature evolution dynamics of larger and more diverse natural language corpora such as the Pile (Gao et al., 2020).

References

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

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

Leshem Choshen, Elad Venezian, Noam Slonim, and Yoav Katz. 2022. Fusing finetuned models for better pretraining.

Bilal Chughtai, Lawrence Chan, and Neel Nanda. 2023. A toy model of universality: Reverse engineering how networks learn group operations.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. 2023. Sparse autoencoders find highly interpretable features in language models.

Ronen Eldan and Yuanzhi Li. 2023. Tinystories: How small can language models be and still speak coherent english?

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. 2018. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2021. A mathematical framework for transformer circuits. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2021/framework/index.html.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. 2019. Linear mode connectivity and the lottery ticket hypothesis.

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

Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. 2024. Scaling and evaluating sparse autoencoders. *arXiv preprint arXiv:2406.04093*.

Charles Goddard, Shamane Siriwardhana, Malikeh Ehghaghi, Luke Meyers, Vlad Karpukhin, Brian Benedict, Mark McQuade, and Jacob Solawetz. 2024. Arcee's mergekit: A toolkit for merging large language models.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online. Association for Computational Linguistics.

Minyoung Huh, Brian Cheung, Tongzhou Wang, and Phillip Isola. 2024. The platonic representation hypothesis.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA.

Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, and Harm de Vries. 2022. The stack: 3 tb of permissively licensed source code.

Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. 2015. Convergent learning: Do different neural networks learn the same representations?

Neel Nanda and Joseph Bloom. 2022. Transformerlens. https://github.com/TransformerLensOrg/TransformerLens.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4902–4912, Online. Association for Computational Linguistics.

Ken Shoemake. 1985. Animating rotation with quaternion curves. *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*. Https://api.semanticscholar.org/CorpusID:11290566.

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

Fanqi Wan, Xinting Huang, Deng Cai, Xiaojun Quan, Wei Bi, and Shuming Shi. 2024. Knowledge fusion of large language models.

Alex Warstadt, Leshem Choshen, Aaron Mueller, Adina Williams, Ethan Wilcox, and Chengxu Zhuang. 2023. Call for papers – the babylm challenge: Sample-efficient pretraining on a developmentally plausible corpus.

Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. 2022. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. 2023. Ties-merging: Resolving interference when merging models.

A Slerp Interpolation curves

Figure 5 shows the accuracy of different stages of the interpolation between the Lua and TinyStories models.

B Sparse Auto-Encoders and Universality

We train our sparse-autoencoders according to directions in Bricken et al. (2023), using the Adam optimizer (Kingma and Ba, 2015) with $\beta_1 = 0.9$ and $\beta_2 = 0.9999$, a constant learning rate of $1e-4$, an L1-coefficient of $3e-4$, on activations corresponding shuffled blocks of 24 tokens in batches

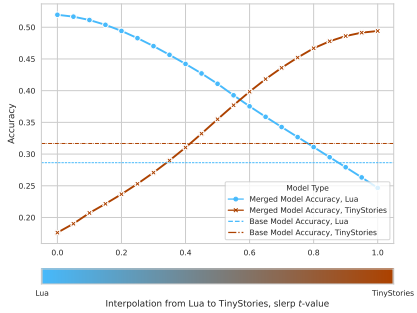

Figure 5: Observed accuracy trends of a merged model consisting of weights spherically linear interpolated between the Lua model and the TinyStories model, as measured on the validation datasets of the Lua and TinyStories, respectively, at each interpolation step (slerp t-value). The dashed baselines show the accuracy of the shared base model underlying the Lua and TinyStories model, on the same validation datasets.

of 128 blocks. To extract activations, we use TransformerLens (Nanda and Bloom, 2022), specifically targeting the `blocks.0.mlp.hook_post` activations of the one-layer Mistral-like models. Our final sparse auto-encoders all have a mean L0 norm between 25-35 and mean MSE loss lower than 0.0067, explained loss of more than 88% across all sparse auto-encoders, and were finally evaluated by manual and automated interpretability/explainability. For automated interpretability details see C.

Formally, similar to Bricken et al. (2023), let n be the input and output dimension of our sparse auto-encoder, and m be the auto-encoder hidden layer dimension. Given encoder weights $W_e \in \mathbb{R}^{m \times n}$, $W_d \in \mathbb{R}^{n \times m}$ with columns of unit norm, encoder biases $\mathbf{b}_e \in \mathbb{R}^m$, decoder biases $\mathbf{b}_d \in \mathbb{R}^n$, the operations and the loss function over dataset X are:

$$\bar{\mathbf{x}} = \mathbf{x} - \frac{1}{|\mathbf{x}|} \sum_{i=1}^{N} \mathbf{x}_i - \mathbf{b}_d \tag{1}$$

$$\mathbf{f} = \mathrm{ReLU}(W_e \bar{\mathbf{x}} + \mathbf{b}_e) \tag{2}$$

$$\hat{\mathbf{x}} = W_d \mathbf{f} + \mathbf{b}_d \tag{3}$$

$$\mathcal{L} = \frac{1}{|X|} \sum_{\mathbf{x} \in X} \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 + \lambda \|\mathbf{f}\|_1 \tag{4}$$

Our hidden layer is overcomplete, specifically $n = 1024$ and $m = 16n$, and the mean squared error is a mean over each vector element while the L1 penalty is a sum (λ is the L1 coefficient). The

hidden layer activations **f** are the learned features of the sparse auto-encoder.

C Automated Interpretability

To generate, evaluate and explore our extracted features at scale, we use automated interpretability using GPT-4 Turbo. We do this by sampling activation data similar to Bricken et al. (2023) tasking the large language model to provide a concise explanation of the observed feature activation pattern, without including verbatim token nor or activation samples, based on a prompt containing sampled pairs of token observations and the corresponding quantised feature activation. This prompt format follows that of Bills et al. (2023). To evaluate these explanations, we simulate the feature activation patterns using GPT-4 Turbo, on another similarly sampled set of feature activation pairs, and then measure the correlation between the simulated activations and the true observed activations. This is again similar to Bricken et al. (2023) and Bills et al. (2023). Using this approach to evaluate common features for each of our models, we arrive at Pearson correlation distributions that closely match those of Bricken et al. (2023).

To generate interpretations and evaluate a single feature costs approximately USD 0.11, using `gpt-4-turbo` via the OpenAI API. Therefore, it cost USD 185.68 to attempt to automatically explain all features in Figure 2.

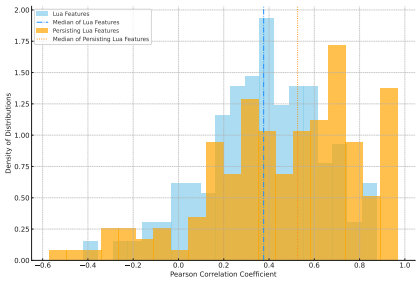

Figure 6: Histograms showing the distribution of observed correlations between automatically generated explanations and the true feature activation patterns of features in the Lua and LuaStories model.

In Figure 6 we show the correlation distributions used to evaluate the quality of our explanations of features in the Lua model. Here we distinguish between the features that persist through model merging to the LuaStories model and those that do not.

D Log-likelihood Ratio Feature Proxy

To analyse features, we adopt the log-likelihood ratio feature proxy used in Bricken et al. (2023). This is the log-likelihood ratio of a string under the feature hypothesis and under the full empirical distribution, where the feature hypothesis refers to a measure corresponding to the hypothesised explanation of a feature's activation pattern. For example, $\log\left(P\left(s|\text{variable assignment}\right)/P\left(s\right)\right)$. Following Bricken et al. (2023), we use log-likelihood proxies on the intuition that features will be incentivized to track log-likelihoods, since they linearly interact with the logits.

To compute $P(s)$, we use the unigram distribution over tokens and compute $P(s) = \prod_{t \in s} p(t)$. For the single-token features we deal with in the case studies of this work, we follow this same approach.