Attention Pattern Discovery at Scale

Jonathan Katzy
AI-enabled Software Engineering Lab
Delft University of Technology
`J.B.Katzy@TUDelft.nl`

Razvan Mihai Popescu
AI-enabled Software Engineering Lab
Delft University of Technology
`R.M.Popescu@tudelft.nl`

Erik Mekkes
AI-enabled Software Engineering Lab
Delft University of Technology
`E.J.Mekkes@Student.TUDelft.nl`

Arie van Deursen
Software Engineering Research Group
Delft University of Technology
`Arie.vanDeursen@TUDelft.nl`

Maliheh Izadi
AI-enabled Software Engineering Lab
Delft University of Technology
`M.Izadi@TUDelft.nl`

Abstract

Language models have scaled rapidly, yet methods for explaining their outputs are lagging behind. Most modern methods focus on a fine-grained explanation of individual components of language models. This is resource-intensive and does not scale well to describe the behavior of language models as a whole. To enable high-level explanations of model behavior, in this study, we analyze and track attention patterns across multiple predictions. We introduce Attention Pattern Masked AutoEncoder (AP-MAE), a vision-transformer–based approach that encodes and reconstructs large language model attention patterns at scale. By treating attention patterns as images, AP-MAE enables efficient mining of consistent structures across a large number of predictions.

Our experiments on StarCoder2 models (3B–15B) show that AP-MAE (i) reconstructs masked attention with high fidelity, (ii) generalizes across unseen model sizes with minimal degradation, and (iii) predicts whether a token will be correct, without access to ground truth, with up to 70% accuracy. We further discover recurring attention patterns demonstrating that attention patterns are structured rather than random noise. These results suggest that attention maps can serve as a scalable signal for interpretability, and that AP-MAE provides a transferable foundation for analyzing diverse large language models. We release code and models to support future work in large-scale interpretability.

1 Introduction

Understanding how large language models (LLMs) work internally is a central challenge in mechanistic interpretability. Prior progress has largely centered on finding exact explanations for generations. Using tools such as sparse autoencoders [8], transcoders [25, 10], ACDC [7], and path patching [14] to gain insights into what features are encoded in LLMs, and which circuits of an LLM are important for specific generations. However, this focus leaves a major component of transformer computation less understood: the attention patterns that dynamically route information between token representations. Unfortunately, attention patterns are inherently two-dimensional, unlike features in the

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Mechanistic Interpretability.

residual stream. This makes existing tooling ineffective for analyzing attention patterns, leaving them underexplored despite their central role in shaping information flow.

Although underexplored, attention patterns have been used in previous studies to validate the existence of computational circuits on a small scale [30], showing that there is potential to mine common patterns. We aim to mine attention patterns at a large scale and propose a novel method based on vision models. Vision models are designed for two-dimensional data, and can be used as encoder models to cluster similar patterns. More specifically, we train a Vision Transformer - Masked AutoEncoder (ViT-MAE) model [16] to reconstruct masked attention patterns. We name this novel model Attention Pattern - Masked AutoEncoder (AP-MAE). We then use the mined patterns to show that they are critical to the generation of correct outputs, and train a classifier to differentiate correct from incorrect predictions using only the discovered patterns.

We make the following contributions:

- We demonstrate that attention patterns can be effectively learned by a masked autoencoder, validating this as a tractable and informative object of study.
- We demonstrate that AP-MAE can be transferred between different models, resulting in only a minor increase in loss when evaluating attention patterns from a model not seen during training.
- We show that using only the attention patterns for classifying the correctness of a model prediction is a valid future path, in a noisy, real world setting.
- We release our code [1] and models [2] to be the basis of future work.

2 Related Works

Representation Learning The most basic element within the residual stream is the representation of features. To discover the features present at any location in the residual stream, models such as Sparse AutoEncoders (SAEs) can be utilized in a dictionary learning setting to map sparse features to human-relatable concepts [5]. Similarly, probing methods have also been shown to be effective, at mapping the residual stream between layers to the Language Model head, and thus understanding which tokens are represented at different parts of the model [4]. While SAEs and probing methods only look at one location, transcoders are trained to predict the outcome of a module given its input. This allows us to see a sparse representation of the calculation being done within a module [25, 10].

Circuit Level Analysis While the previous works focused on finding the representation of tokens at a location, how the model came to a given representation is also important. Transcoders can be used to trace attribution graphs throughout a model, and incorporate attention scores as an attribution strength within a graph [10]. Transcoders can be further abstracted to replacement networks, and crosscoders, and replacement networks which replace entire sections of a network with a sparse model [1]. This makes it easier to trace cross-layer circuits to explain an output. Other approaches find circuits by pruning graphs until the minimal elements remain to make a correct prediction [7] or by patching paths [14]. However, current approaches have not been able to find general global circuits, that generalize past given examples [19, 2]. These methods all have in common that they show certain behaviors are local to a subset of components of a model. Our approach does not focus on the exact circuits but on making general explanations of behaviors based on mined patterns at scale.

Attention Patterns As mentioned in previous sections, attention scores have been used to calculate attribution of different values to the output. However, the full attention patterns have been used to validate circuits and explain specific heads. For example, letter heads [18] show high attention to specific letters, and the rare word head showing the same patterns for rare words [27]. Furthermore, off-diagonal lines have been identified as induction heads, that look for previous copies of a token to be predicted [11]. For knowledge circuits an incorrect knowledge selection is found using the attention heads, explaining why a prediction was incorrect [30]. These works all show that small, certain heads produce a recognizable attention pattern. As a result, the field lacks methods for

[1]https://github.com/LaughingLogits/AP-MAE
[2]https://huggingface.co/collections/LaughingLogits/ap-mae-models-66b27a73536bb1306d55c4c4

Table 1: Training and architecture parameters for AP-MAE.

Model Inputs		Encoder		Decoder	
Pattern Size	256	Layers	24	Layers	8
Patch Size	32	Dim	512	Dim	512
Mask Ratio	0.5	Heads	16	Heads	8
Batch Size	480	MLP	2048	MLP	2048
Learning Rate	1.44×10^{-3}				

systematically mining and characterizing attention behaviors across models and tasks. We aim to automate the finding of common attention patterns at scale using AP-MAE.

3 Experimental Setting

One of the main criticisms we want to address in this work is the specificity of the discovered features/attribution graphs/circuits to crafted or selected inputs [2]. With this, we refer to the task templates and manually crafted prompts intended to elicit certain behaviors, rather than running the experiments on real-world data [19, 7].

In this work we focus on using inputs to the models that were not specifically crafted for a task. However, we also aim to leverage knowledge from mechanistic interpretability about the specificity and locality of certain features within a model. This means we need to have a corpus that we can query to automatically generate samples where the same task is being completed in different scenarios. To accomplish this, we focus on code rather than natural language due to its highly structured nature, which allows us to use parsing tools, such as tree-sitter, to mine similar completion tasks from a large corpus of data. More specifically, this investigation will focus on Java.

One challenge using real-world data has, is possible data contamination. Although the effect of this on the behavior of models (from a mechanistic interpretability point of view) is unexplored, we follow the general rule of machine learning not to use training data. As LLM training data is rarely made available, we focus our investigation on the Starcoder2 (SC2) [20] family due to their openness of sharing the exact training data. In order to have a decontaminated source of code files, we use The Heap [17] as it has already been deduplicated with regards to the training data of the SC2 models.

4 AP-MAE

The core of our proposed approach is the Attention Pattern – Masked Auto-Encoder (AP-MAE) model for identifying patterns in LLM attention outputs. To train AP-MAE, we model the patterns as 1 channel images and base our model on the original ViT-L architecture [9]. We apply a novel scaling method to the attention patterns to allow the training to converge.

Architecture AP-MAE is based on the ViT-L architecture with minor changes. We reduce the hidden size from 768 to 512 dimensions and scale the MLP down to 2048 parameters accordingly. We also remove the top right triangle of the data as this is masked in decoder-only attention patterns. For patches on the matrix's diagonal, we pad the masked values above the diagonal. Finally, we scale the attention patterns by taking the natural logarithm. This step is essential in allowing the model to fit to attention patterns, as demonstrated in the ablation study discussed below. Table 1 presents an overview of the architecture.

Data To train our models, we need to generate attention patterns. For this investigation, we use attention patterns generated by SC2 3B, 7B, and 15B. In order to generate attention patterns, we mimic the training procedure used for training the SC2 models. We mask spans between 3-10 tokens in a code file, at random locations. We then add the same sentinel tokens to the input data as were used during training. In order to ensure that the attention patterns used in this work are all the same size, we truncate the context to exactly 256 tokens in total.

As a final step in the training data selection, we sub-sample the attention patterns we get from the language models when generating. We also focus exclusively on attention patterns generated when

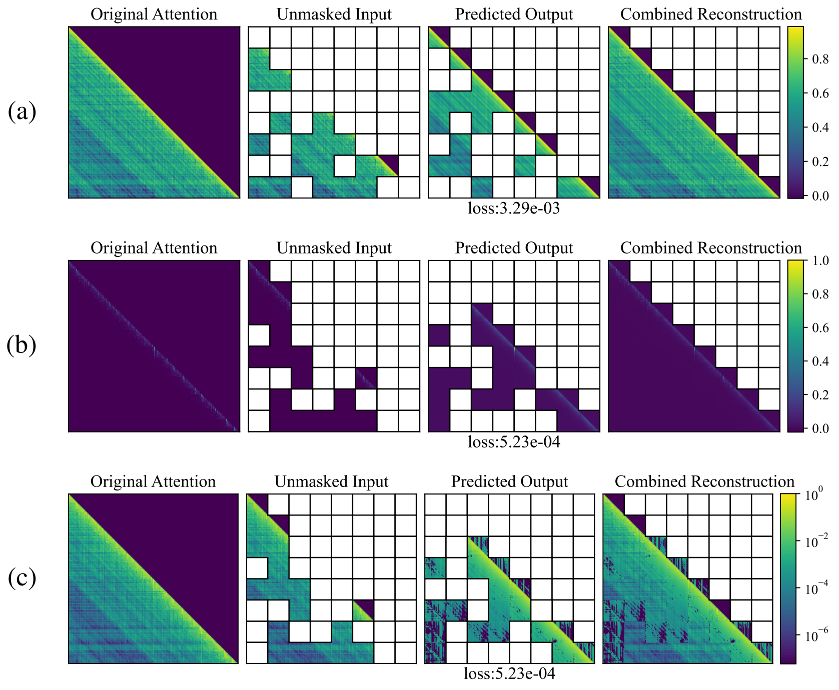

Figure 1: Comparison of attention pattern reconstruction methods: (a) log normalized attention pattern, (b) raw attention pattern, (c) raw attention pattern with pixel values scaled for visualization.

the prediction made by the model was correct. For each invocation of the target language model, we get 720 patterns for the 3B model, 1152 for the 7B model, and 1920 for the 15B model. As the generation procedure of these patterns is cheap from a time perspective, we sub-sample each generation to 25% of all attention heads. This allows us to get a larger variety of samples generated when the model is prompted in different locations in a code file. We ensure that when we subsample the patterns, we get 25% of the patterns from each layer. Other works have shown that there are signs of different behavior at different layers in a model, which we want to ensure we capture.

In Figure 1 (a), we show the initial attention pattern on the left. Then we display the masking and the reconstructed parts of the masked pattern. Finally, we present the combination of the original and reconstructed patterns.

Training Setup We train the models on eight A100 GPUs with a local batch size of 60 attention patterns. We train for $150,000$ batches using 72M attention patterns. The total training time is less than 100 GPU hours on our institution's cluster. Although we used eight A100 GPUs, all target models and AP-MAE models combined fit on one A100, making it possible to train on smaller servers. The limiting factor is the LLM size, as the AP-MAE encoder has 101M parameters and the decoder has 25M. We use the AdamW optimizer with a weight decay of 0.05 and a cosine annealing scheduler initialized with a global Learning Rate (LR) of 1.44×10^{-3}. We linearly upscale from the base LR of 1.5×10^{-4} for a batch size of 50 used by ViT-MAE.

4.1 Generalizability

One of the main advantages of analyzing attention patterns, compared to representations of features within a model, is that they have the same dimensions between models. To investigate if AP-MAE can take advantage of this we evaluate its ability to reconstruct attention patterns from models it was not trained on. We cross-evaluate the AP-MAE models on a test set containing all combinations of SC2 target models. Table 2 provides an overview of the results. We observe that evaluating the encoder models on attention patterns from other target models results in a loss often within one standard deviation of each other. This shows that there are opportunities to use an AP-MAE base model to make inferences about a target LLM's behaviors without training a new model every time.

Table 2: Cross-evaluation of AP-MAE

Trained \ Evaluated	SC2 3B Loss ($\times 10^{-3}$)	SC2 7B Loss ($\times 10^{-3}$)	SC2 15B Loss ($\times 10^{-3}$)
SC2 3B	7.07 ± 2.12	7.78 ± 2.05	9.53 ± 2.76
SC2 7B	7.55 ± 2.05	7.17 ± 2.12	9.29 ± 2.66
SC2 15B	9.57 ± 3.35	10.05 ± 3.79	7.59 ± 2.42

4.2 Ablation Study - Logarithmic Scaling

One of the preprocessing steps we used for the data is the log normalized scaling. To show that this step is necessary we conduct an ablation study by training an identical model, without scaling the attention patterns. In Figure 1 (b), we show an unscaled attention pattern together with its masking and reconstruction. Here it is difficult to see the reconstruction. In Figure 1 (c), we show the same pattern but scale the color gradient logarithmically to visualize the reconstruction. We see that the patches that were masked are corrupted. We compare this with the output of a model that has been trained on scaled attention patterns in Figure 1 (a). We see that when using logarithmic scaling, major patterns are reconstructed. While we cannot compare loss values directly, this shows the need for the logarithmic scaling of the attention patterns when training and evaluating the AP-MAE models.

5 Pattern mining

5.1 Setup

We begin by encoding the attention heads using AP-MAE and select specific tasks to use as inputs to the language model, based on findings from previous research. We then cluster the resulting representations, a step that poses significant challenges given the large scale of the problem.

Tasks Given the vast search space of possible circuits in Java, we leverage prior knowledge of circuits identified in LLMs to narrow our focus. Specifically, we select 11 tasks for pattern mining in attention heads, including a validation task in which the target model is probed with noise as an input.

1. Identifiers (1 task): One of the earliest benchmark circuits is the Indirect Object Identifier (IOI) circuit [28], where the model uses context to predict the correct token. Adapting this to source code, we mask a single identifier and task the target LLM with reconstructing it from the surrounding context.

2. Literals (3 tasks): Generating correct literals—spanning booleans, strings, and numbers—poses challenges distinct from identifier generation. Unlike identifiers, the correct literal values are not present in the input and must instead be inferred by the model (e.g., deducing $\pi = 3.14$). Prior work suggests that factual knowledge is encoded in the feed-forward layers of transformer models [29, 13], making this setting particularly suitable for probing systematic behaviors of attention heads. Moreover, evidence of arithmetic-related circuits has been reported, including a greater-than circuit [15] for numeric comparison and modules specialized for mathematical reasoning [19, 3].

3. Operators (3 tasks): Beyond selecting appropriate operand values, recent work has shown that LLMs exhibit operator-specific heuristics when performing arithmetic reasoning [23]. To complement literal selection, we include tasks focused on predicting the correct operator across three categories: boolean operators, arithmetic operators, and programming-specific assignment operators (e.g., +=).

4. Ending Statements (2 tasks): Finally, we evaluate the models' ability to complete complex syntactic structures. We consider two tasks. The first requires predicting the correct closing bracket for a statement, a capability that has been linked to specialized model circuitry [12]. The second task involves predicting line endings. Unlike brackets, line termination in Java is not syntactically required, making this task a blend of program correctness and modeling human coding conventions. Prior work has examined structural and stylistic features at line endings in natural language [19].

5. Baselines (2 tasks): To contextualize our results, we consider two baseline tasks. First, we include a random masking task, identical to the one used for generating attention patterns during AP-MAE training. Second, to verify that our method does not inadvertently cluster spurious or uninformative structures, we introduce a random token sampling task, where tokens are drawn uniformly from the

tokenizer vocabulary. This allows us to assess whether the discovered patterns reflect meaningful structure beyond random noise.

Encoding For encoding attention patterns, we employ the previously introduced AP-MAE model, using the representation of the [CLS] token as the embedding, following standard practice in encoding pipelines. For each target LLM, we select $10,000$ input samples per task. These samples are balanced such that half correspond to attention heads associated with correct predictions by the target LLM and the other half with incorrect predictions. In contrast to the AP-MAE training phase, we do not perform head subsampling in this setting.

Clustering Clustering all task samples is challenging due to both the high dimensionality of the representations (512 dimensions) and the sheer scale of the data (79.2M samples for SC2 3B, 126.7M for SC2 7B, and 211.2M for SC2 15B). The standard pipeline, dimensionality reduction with UMAP [22] followed by clustering with HDBSCAN [6], is computationally infeasible in this setting, as it requires pairwise distance computations across the full dataset. Instead of resorting to subsampling, we decompose the problem into smaller, tractable subproblems. Specifically, for each model head, we first reduce the representations to 8 dimensions with UMAP, and then cluster them with HDBSCAN. This yields up to 1920 independent clustering pipelines per model, each operating on approximately $110,000$ samples.

5.2 Identified Patterns

To assess whether repeated patterns exist and whether our clustering approach successfully captures them, we perform a qualitative evaluation of the resulting clusters. Figure 2 presents three groups, each containing five attention patterns. Panel (a) illustrates five representative clusters, providing an overview of the variation observed across patterns. In pattern (a)(I), we observe a prominent diagonal structure: the highest attention scores concentrate around the preceding few tokens, with alternating bands of higher and lower scores extending outward. This behavior resembles an induction head, a specialized mechanism that facilitates in-context learning [24]. From patterns (a)(III) and (a)(IV), we observe that these heads feature high attention on individual tokens, as indicated by the vertical attention lines. This behavior has previously been characterized as LLMs allocating disproportionate attention to rare words in the input sequence [27] or individual letters [18]. The high attention scores in (a)(III) around the diagonal in combination with the vertical lines, hint that some heads may exhibit multiple behaviors. In pattern (a)(III), we observe strong attention behavior reminiscent of the induction head, but distributed across multiple tokens. We also identify several recurring patterns that, to our knowledge, have not been previously documented. For instance, patterns (a)(II) and (a)(V) exhibit square-like structures with high attention diagonals that reappear at varying scales and frequencies across different heads; we highlight these two cases as representative extremes.

In addition to capturing distinct patterns, our method demonstrates robustness to variations in the ordering of input tokens. Figure 2(b) illustrates five patterns grouped within the same cluster (Figure 2(a)(I)). Although the intensity and position of the global diagonal lines vary, the general pattern is preserved. AP-MAE can capture these differences in pattern locations, allowing it to handle changes in the ordering of input tokens. Finally, we investigate whether heads generate patterns regardless of inputs. To this end, we visualize the heads obtained when feeding the model with random noise (Figure 2c). Unlike Figures 2(a) and 2(b), only a few discernible structures emerge. The most recognizable case is (c)(III), which closely resembles patterns observed in Figure 2(b), and stands as the most similar pattern we were able to identify. Notably, the characteristic square structures seen in (a)(II) and (a)(V) do not appear under noisy inputs. This absence suggests that such square patterns may serve as strong indicators of heads engaging in meaningful computation, an observation that highlights a promising direction for future research.

5.3 Pattern Distribution

We next examine how the discovered patterns are distributed across attention heads. Figure 3 reports the number of clusters identified in each head. A consistent trend emerges: as model size increases, a subset of heads produces a broader variety of patterns. In the 3B model, most heads yield only a few clusters, whereas the 7B model exhibits substantial diversity, particularly in later layers. The 15B model shows fewer heads with high diversity compared to the 7B model, though some still capture a

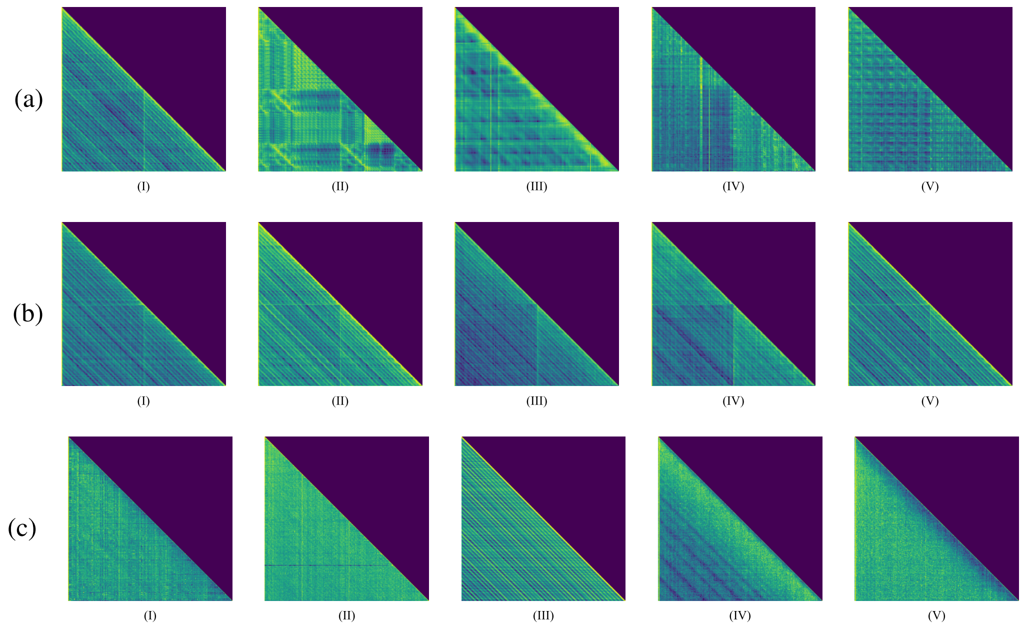

Figure 2: Comparison of different clustering results: (a) examples of different patterns found by clustering, (b) attention patterns within a single cluster, (c) attention patterns generated by noise.

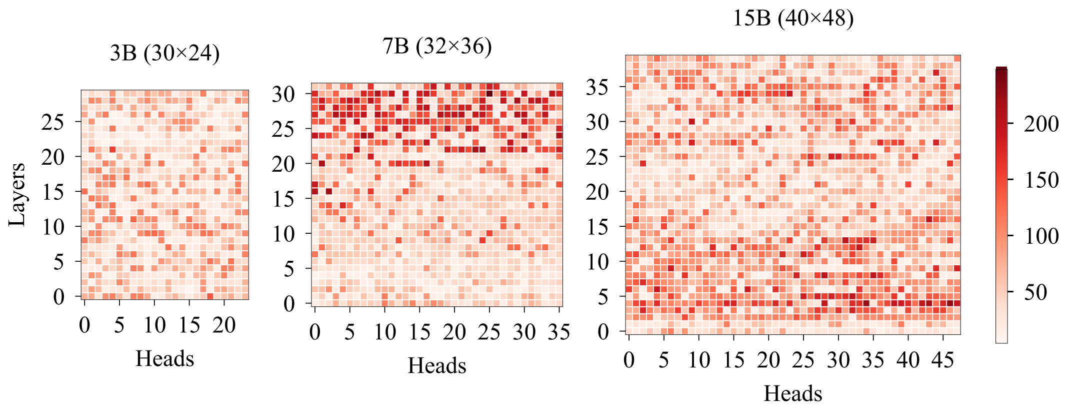

Figure 3: Distribution of the number of clusters in a head

wide range of patterns. These differences in distribution suggest increasing specialization of certain heads, potentially enabling them to handle a broader spectrum of noisy inputs.

6 Classification

6.1 Setup

To determine whether a target model's prediction is correct, we treat the output of each head as a categorical feature, using the cluster assignment of that head for the given prediction as its value. This formulation yields a tractable prediction problem with between 720 features (SC2 3B) and 1920 features (SC2 15B). We perform classification at the task level, training a dedicated predictor for each task. The Noise task is excluded, as it does not admit a notion of correctness.

For classification, we employ a gradient boosting decision tree model, CatBoost [26]. CatBoost offers two key advantages for our setting: (i) it enables the computation of SHAP values [21], which we use to quantify the contribution of each feature (here, individual transformer heads) to the distinction

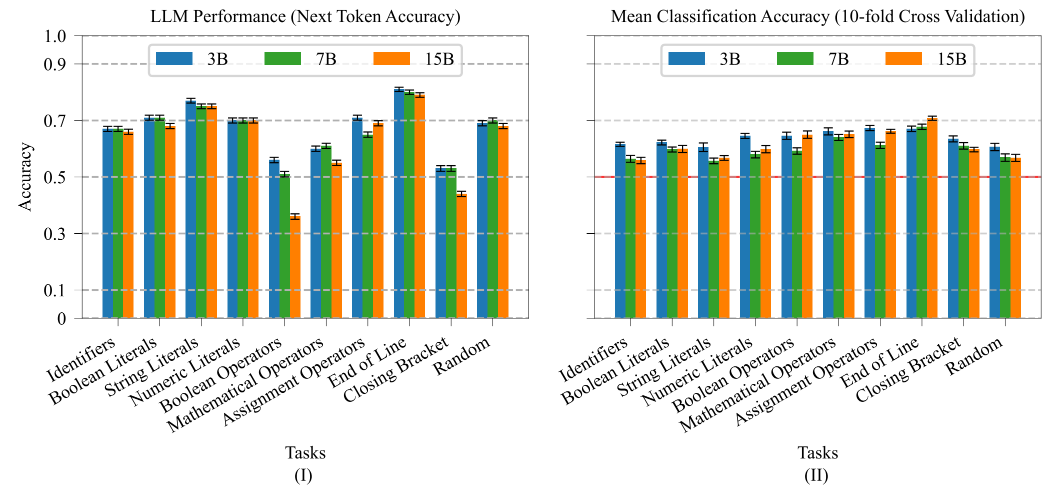

Figure 4: Performance of target LLMs on the studied tasks, and the accuracy of the CatBoost classifier

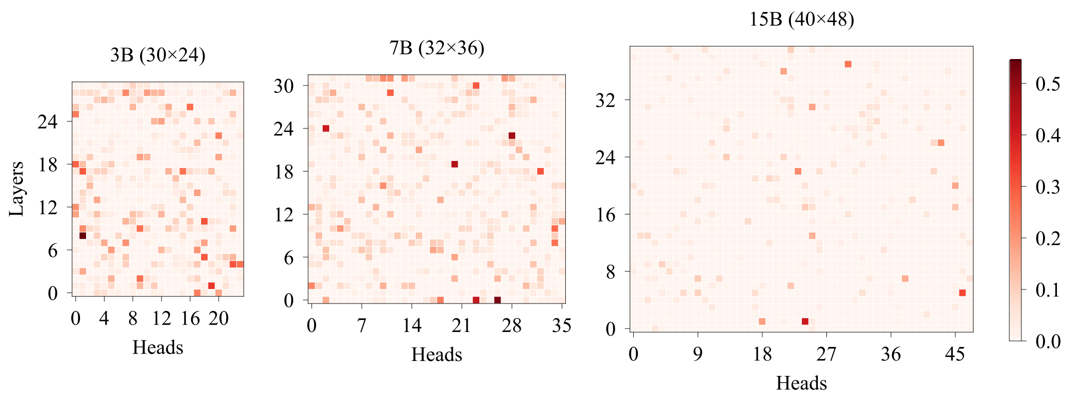

Figure 5: Difference in mean SHAP values per cluster for the CatBoost classifiers, classifying predictions for the End of Line task across all target sizes

between correct and incorrect predictions, and (ii) it natively handles categorical data, eliminating the need for additional preprocessing.

6.2 Performance

We give the results of the classification task in Figure 4 (II). We also include the performance of the target models in completing the next token prediction task in Figure 4 (I), to see if performance has an effect on our ability to correctly classify a prediction as correct or incorrect. For the classification task we plot the mean accuracy and the 95% confidence interval over a 10 fold cross validation using a 90, 10, 10 split. We see that the performance varies little between runs due to the small range of the 95% confidence interval. The first thing we see in Figure 4 is that there is no correlation between classification performance, and target model performance. Next, focusing only on Figure 4(II), we see that some tasks are indeed harder to classify as correct than others. Tasks such as Identifier, Boolean Literals, and String Literals perform similarly to the Random masking task, with accuracy scores between 55% and 60%. The best performing task is predicting the End of Line token, which has a mean accuracy of just over 70% for the 15B model.

To investigate which parts of the model are needed to differentiate between a correct and incorrect prediction, we use the maximum difference between mean SHAP values per category at each head. This will highlight heads that are both strong indicators of being a correct and incorrect prediction, depending on the pattern we detected in them. We plot these values in Figure 5. Here we plot the values mentioned above for the End of Line task. We see that the plots are sparse; a small number

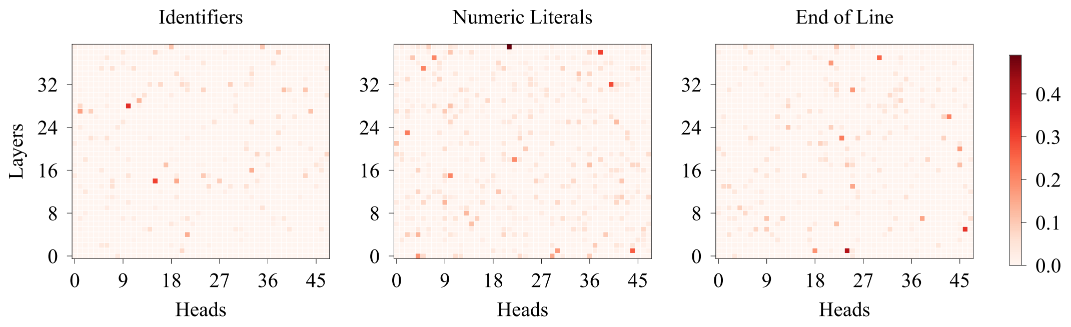

Figure 6: Difference in mean SHAP values per cluster explaining the global effect of each pattern in each head on the correctness classifier

of heads is enough to differentiate between correct and incorrect predictions. Furthermore, sparsity increases with model size, similar to the increase in classification accuracy. This allows us to highlight heads that are of interest when determining where LLMs make mistakes.

Finally, we want to know if there is a difference between tasks when it comes to predicting if an LLM generation is correct. To investigate this, we plot the same difference between mean SHAP values introduced earlier for different tasks targeting the 15B model in Figure 6. Here we see that for each task, the SHAP values are sparse, and different heads are highlighted. Showing that different tasks depend on different heads to predict if they are correct.

7 Limitations and Open Questions

We demonstrated that a large-scale analysis of attention patterns in LLMs offers a promising direction for understanding LLM behavior. Nevertheless, our study faces some limitations. Most notably, computational constraints required us to fix the input length to the models. This choice facilitated fairer comparisons across samples, ensuring that attention patterns were evaluated at a consistent scale, yet it does not fully capture the variability of sequence lengths. Our findings highlight that attention patterns can be systematically mined and are localized within LLMs. Future work should investigate how patterns evolve under varying input lengths, particularly for encoding and clustering.

We adopted a vision transformer architecture due to its scalability to large datasets, robustness to noisy inputs, and improved generalization under potential distribution shifts between pretraining samples and downstream tasks. However, in deployment scenarios where efficiency is critical, convolutional networks may remain preferable due to their lower computational overhead. In such cases, CNNs could also serve as feature extractors, enabling the analysis of whether attention heads emerge as compositions of fundamental local patterns.

To facilitate interpretation, we restricted our analysis to a subset of downstream tasks. Task selection was guided by prior knowledge of in-context learning, factual recall in model weights, and established mechanistic circuits, allowing us to capture a range of representative behaviors. Nonetheless, this choice inevitably risks overlooking behaviors that remain unknown. Future work should build on our findings by employing online learning techniques to mine behaviors at scale, enabling efficient exploration over larger data streams. Such approaches would help translate insights from mechanistic interpretability into more actionable benefits for real-world applications.

Our method is designed to identify patterns rather than provide explicit explanations. It can highlight interesting regions of a model with relatively low computational cost. An important direction for future work is to combine our approach with more fine-grained mechanistic interpretability methods.

8 Conclusion

In this work, we analyze the attention patterns generated by LLMs. We find that certain attention heads exhibit only a limited set of distinct patterns, and we provide an overview of the most commonly observed ones. Building on this characterization, we show that the type and position of an attention pattern can be predictive of whether a model produces a correct or incorrect output. Using SHAP values, we further demonstrate that, across a large number of samples, differences in prediction quality can often be attributed to a small subset of attention heads. Our experiments on data collected from publicly available repositories highlight both the robustness of this approach to noisy inputs and its practical applicability in real-world settings.

References

[1] Emmanuel Ameisen, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L. Turner, Brian Chen, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. Circuit tracing: Revealing computational graphs in language models. *Transformer Circuits Thread*, 2025. URL `https://transformer-circuits.pub/2025/attribution-graphs/methods.html`.

[2] Usman Anwar, Abulhair Saparov, Javier Rando, Daniel Paleka, Miles Turpin, Peter Hase, Ekdeep Singh Lubana, Erik Jenner, Stephen Casper, Oliver Sourbut, Benjamin L. Edelman, Zhaowei Zhang, Mario Günther, Anton Korinek, Jose Hernandez-Orallo, Lewis Hammond, Eric J Bigelow, Alexander Pan, Lauro Langosco, Tomasz Korbak, Heidi Chenyu Zhang, Ruiqi Zhong, Sean O hEigeartaigh, Gabriel Recchia, Giulio Corsi, Alan Chan, Markus Anderljung, Lilian Edwards, Aleksandar Petrov, Christian Schroeder de Witt, Sumeet Ramesh Motwani, Yoshua Bengio, Danqi Chen, Philip Torr, Samuel Albanie, Tegan Maharaj, Jakob Nicolaus Foerster, Florian Tramèr, He He, Atoosa Kasirzadeh, Yejin Choi, and David Krueger. Foundational challenges in assuring alignment and safety of large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL `https://openreview.net/forum?id=oVTkOs8Pka`. Survey Certification, Expert Certification.

[3] Tanja Baeumel, Daniil Gurgurov, Yusser al Ghussin, Josef van Genabith, and Simon Ostermann. Modular arithmetic: Language models solve math digit by digit. *arXiv preprint arXiv:2508.02513*, 2025.

[4] Nora Belrose, Zach Furman, Logan Smith, Danny Halawi, Igor Ostrovsky, Lev McKinney, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*, 2023.

[5] 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.

[6] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-based clustering based on hierarchical density estimates. In Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, editors, *Advances in Knowledge Discovery and Data Mining*, pages 160–172, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-37456-2.

[7] Arthur Conmy, Augustine Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 16318–16352. Curran Associates, Inc., 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/file/34e1dbe95d34d7ebaf99b9bcaeb5b2be-Paper-Conference.pdf`.

[8] Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=YicbFdNTTy`.

[10] Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. Transcoders find interpretable llm feature circuits. *Advances in Neural Information Processing Systems*, 37:24375–24410, 2024.

[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. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

[12] Xuyang Ge, Fukang Zhu, Wentao Shu, Junxuan Wang, Zhengfu He, and Xipeng Qiu. Automatically identifying local and global circuits with linear computation graphs. In *ICML 2024 Workshop on Mechanistic Interpretability*, 2024. URL `https://openreview.net/forum?id=b8sq8Y5VFo`.

[13] Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. *arXiv preprint arXiv:2012.14913*, 2020.

[14] Nicholas Goldowsky-Dill, Chris MacLeod, Lucas Sato, and Aryaman Arora. Localizing model behavior with path patching. *arXiv preprint arXiv:2304.05969*, 2023.

[15] Michael Hanna, Ollie Liu, and Alexandre Variengien. How does gpt-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model. *Advances in Neural Information Processing Systems*, 36:76033–76060, 2023.

[16] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022.

[17] Jonathan Katzy, Razvan Mihai Popescu, Arie van Deursen, and Maliheh Izadi. The heap: A contamination-free multilingual code dataset for evaluating large language models. In *Proceedings 2nd ACM international conference on AI Foundation Models and Software Engineering (FORGE 2025)*. ACM, 2025. URL `https://arxiv.org/abs/2501.09653`.

[18] Tom Lieberum, Matthew Rahtz, János Kramár, Neel Nanda, Geoffrey Irving, Rohin Shah, and Vladimir Mikulik. Does circuit analysis interpretability scale? evidence from multiple choice capabilities in chinchilla. *arXiv preprint arXiv:2307.09458*, 2023.

[19] Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. On the biology of a large language model. *Transformer Circuits Thread*, 2025. URL `https://transformer-circuits.pub/2025/attribution-graphs/biology.html`.

[20] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder 2 and the stack v2: The next generation, 2024. URL `https://arxiv.org/abs/2402.19173`.

[21] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[22] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *ArXiv e-prints*, February 2018.

[23] Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. Arithmetic without algorithms: Language models solve math with a bag of heuristics. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=O9YTt26r2P`.

[24] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *CoRR*, abs/2209.11895, 2022. URL `https://doi.org/10.48550/arXiv.2209.11895`.

[25] Gonçalo Paulo, Stepan Shabalin, and Nora Belrose. Transcoders beat sparse autoencoders for interpretability. *arXiv preprint arXiv:2501.18823*, 2025.

[26] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. *Advances in neural information processing systems*, 31, 2018.

[27] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1580. URL `https://aclanthology.org/P19-1580/`.

[28] Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=NpsVSN6o4ul`.

[29] Yunzhi Yao, Shaohan Huang, Li Dong, Furu Wei, Huajun Chen, and Ningyu Zhang. Kformer: Knowledge injection in transformer feed-forward layers. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 131–143. Springer, 2022.

[30] Yunzhi Yao, Ningyu Zhang, Zekun Xi, Mengru Wang, Ziwen Xu, Shumin Deng, and Huajun Chen. Knowledge circuits in pretrained transformers. *Advances in Neural Information Processing Systems*, 37:118571–118602, 2024.