On the Emergence of “Useless” Features in Next Token Predictors

Mark Rofin[1] **Jalal Naghiyev**[2] **Michael Hahn**[1]

Abstract

Why do models trained for next token prediction (NTP) learn to compute abstract features that appear to be useless for this task? We formalize three mechanisms of feature development in Transformers, differing in their role in NTP, and propose a method to estimate the influence of each mechanism on the emergence of specific features. We study this distinction experimentally by analyzing the representations of models trained on synthetic tasks, as well as those of an LLM. Our findings shed light on how Transformers develop and use hidden features, and how the NTP objective affects the training outcome.

1. Introduction

Large Language Models are usually pretrained with the objective of next token prediction (NTP). In this paradigm, a model learns to predict each token in a sequence given all previous tokens: in other words, it learns the distribution $p(x_{t+1} \mid x_1 \cdots x_t)$.

Thus, at each position, the model is incentivized to compute features that help predict the immediate next token. Hence, one could reasonably expect that the hidden representations at position t, computed by a model trained in this way, would contain only the information relevant for predicting x_{t+1}. However, a growing body of work on LLMs (and NTP-trained Transformers in general) shows that they know much more than that. For example, Transformers reconstruct abstract features of the input text (Templeton et al., 2024; Park et al., 2024b), infer the high-level structure of the processes generating their training data, forming ‘world models’ (Li et al., 2023; Karvonen, 2024; Shai et al., 2024), or implicitly predict the sequence multiple tokens ahead (Pal et al., 2023; Wu et al., 2024). Motivated by these intriguing findings, we ask:

Why do Transformers trained for NTP learn features that don’t help in the prediction of the immediate next token?

Towards answering this question, we develop a classification of features represented in Transformers. Based on the structure of information flow in causally masked Transformers, we show that features can be classified into three distinctive categories, which we refer to as *direct*, *pre-cached*, and *shared*. We then use our framework to understand the roles of those feature categories in Transformers trained on toy tasks (§3.2), Othello transcripts (§3.3), and language (§4.2).

2. Possible Causes of Feature Emergence

2.1. Definitions

We use $\mathcal{X}$ to denote a variable-length input space of sequences $x_1 \dots x_n$. We view a model T_θ as representing a function $x_1 \dots x_n \to \hat{x}_2 \dots \hat{x}_{n+1}$ such that

$$\hat{x}_{i+1}(x) = h_\theta^{L+1}(r_i^L) \qquad r_{\theta,i}^0(x) = h_\theta^0(x_i)$$
$$r_{\theta,i}^k(x) = h_\theta^k(r_1^{k-1} \dots r_i^{k-1}),\ k > 1$$

and $r_{\theta,i}^k(x) \in \mathbb{R}^d$. Here h_θ^0 and h_θ^{L+1} are embedding and unembedding layers, respectively, $r_{\theta,i}^k(x)$ are the values of the residual stream, and h_θ^k are Transformer blocks.

We call a *learned feature* any linear component of the residual stream at a specific layer and position $\langle w_i^k, r_{\theta,i}^k(x) \rangle$, where a vector $w_i^k \in \mathbb{R}^d$ defines the *feature direction*.

2.2. Information Flow Decomposition

To understand how features arise, we need to interpret the gradient-based training signal that produces them. For each training example in the task of NTP, the total loss $L(x, T_\theta(x))$ is obtained by summing $L(x_{j+1}, T_\theta(x)_j)$ over positions $j = 1 \dots N-1$. For conciseness, let L stand for $L(x, T_\theta(x))$, L_j stand for $L(x_{j+1}, T_\theta(x)_j)$ and $L_j^{\mathrm{sg}(k,i)}$ stand for L_j, but with a stop-gradient operator applied to $r_{\theta,i}^k(x)$ during the computation.

We fix position i and layer k and study all information paths in the computational graph of the model, classifying them by how they relate to $r_{\theta,i}^k(x)$. We argue that the gradient training signal can flow to θ through three types of paths, illustrated in Figure 1.

[1]Saarland University [2]Independent. Correspondence to: Mark Rofin <mrofin@lst.uni-saarland.de>.

ICML 2025 Workshop on Assessing World Models. Copyright 2025 by the author(s).

Firstly, a gradient signal can come from the immediate NTP (*direct learning*). This includes all paths passing through $r^k_{\theta,i}(x)$ and $\hat{x}_{i+1}$, and represents the effect of the information encoded in $r^k_{\theta,i}(x)$ on the prediction of the immediate next token. These paths are colored green in Figure 1. We formalize this by comparing the overall gradient with a gradient after a stop-gradient operator applied:

$$\nabla_\theta L^k_{i\,(\text{direct})} = \nabla_\theta L_i - \nabla_\theta L_i^{\text{sg}(k,i)} \tag{1}$$

Secondly, the information encoded in $r^k_{\theta,i}(x)$ affects the loss at positions $j > i$ because attention heads at those positions can attend to the position i. Thus, a gradient signal can come from prediction loss at future positions, after passing through one or more attention operations (*pre-caching*). This component includes the paths passing through $r^k_{\theta,i}$ and $\hat{x}_j,\ j > i+1$ (blue in Figure 1).

$$\nabla_\theta L^k_{i\,(\text{pre-cached})} = \nabla_\theta \sum_{j \neq i} \left[L_j - L_j^{\text{sg}(k,i)} \right] \tag{2}$$

Third, there are paths that do not pass through $r^k_{\theta,i}(x)$ at all. Hence, some gradient signal arises independently of $r^k_{\theta,i}(x)$. Since Transformer blocks use the same parameters to perform the computation at every position, this signal may influence the parameters computing it. We call this phenomenon *circuit sharing* and visualize the related paths in rose in Figure 1.

$$\nabla_\theta L^k_{i\,(\text{shared})} = \sum_j \nabla_\theta L_j^{\text{sg}(k,i)} \tag{3}$$

The term *pre-caching* is borrowed from Wu et al. (2024), who studied it as a possible explanation for the look-ahead in LLMs. We discuss the relation between our work and Wu et al. (2024) in Appendix A.

These three components provide an *exhaustive decomposition* of the gradient signal:

Theorem 2.1 (Loss gradients decomposition)**.** *For any layer k and position i,*

$$\nabla_\theta L = \nabla_\theta L^k_{i\,(\text{direct})} + \nabla_\theta L^k_{i\,(\text{pre-cached})} + \nabla_\theta L^k_{i\,(\text{shared})}$$

By Theorem 2.1, for each i and k, the total gradients that are backpropagated to the model parameters after computing the loss on one training batch can be split into three terms distinctive in their nature.

2.3. The Causes of Feature Emergence

So far, we have argued that there are three path types along which the loss signal, via its gradient, can pass to the model parameters. We now use this decomposition to study the extent to which a feature is produced, over the course of training, by each of the three components of the gradient. To this end, we aim to quantify how much each component of the gradient signal pushes the parameters towards developing a feature.

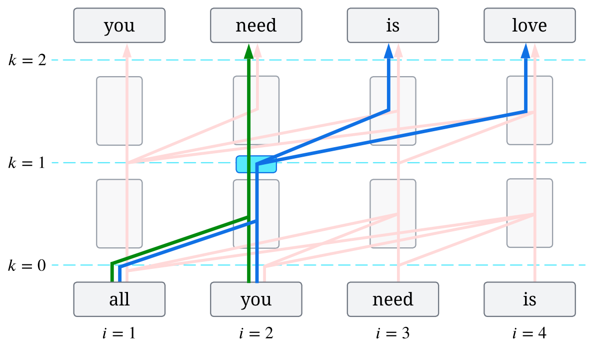

Figure 1. An illustration of the information flow decomposition for $i = 2$ and $k = 1$ into **direct**, **pre-cached**, and shared components. **Direct** and **pre-cached** paths must pass through the residual stream at position $i = 2$ and layer $k = 1$; any other path is considered shared. The turquoise rectangle indicates $r^{k=1}_{\theta,i=2}$.

Definition 2.2. We call *a feature mismatch* the value

$$R(x \mid \theta_1, \theta_2, w^k_i) = \frac{1}{2}\left(\langle w^k_i, r^k_{\theta_1,i}(x)\rangle - \langle w^k_i, r^k_{\theta_2,i}(x)\rangle\right)^2$$

The feature mismatch quantifies how much the projections of the residual streams onto the feature w^k_i differ between models parameterized by θ_1 and θ_2.

We now want to quantify the extent to which a single gradient update to an intermediate checkpoint θ_t narrows the feature mismatch when compared to the final checkpoint θ^*. By separately considering the three components of the gradient signal, we will be able to understand what role they each have on the development of the feature. We formalize this in terms of an *influence* $I(\theta, x, y \mid w^k_i, \theta^*, G)$:

Definition 2.3. For a vector $G \in \mathbb{R}^{|\theta|}$, we call *an influence of G* the value

$$I^k_i(\theta, x, y \mid w^k_i, \theta^*, G) = \frac{d}{d\varepsilon} R\left(x \mid \theta + \varepsilon G, \theta^*, w^k_i\right)\Big|_{\varepsilon=0}$$

Applying the decomposition from Theorem 2.1, for each feature, we can define *direct influence*:

$$I_{\text{direct}}(w^k_i, \theta) = I(\theta, x, y \mid w^k_i, \theta^*, \nabla_\theta L^k_{i\,(\text{direct})})$$

The definitions of *pre-cached* and *shared influences* are analogous.

Remark 2.4 (informal). Consider a model T_θ, trained for M steps of SGD with a small step size η. Then

$$R(x \mid \theta_0, \theta^*, w) \approx \eta \cdot \sum_{s \in S} \sum_{t=1}^{M} I(\theta_t, x_t, y_t \mid w^k_i, \theta^*, \nabla_\theta L_s)$$

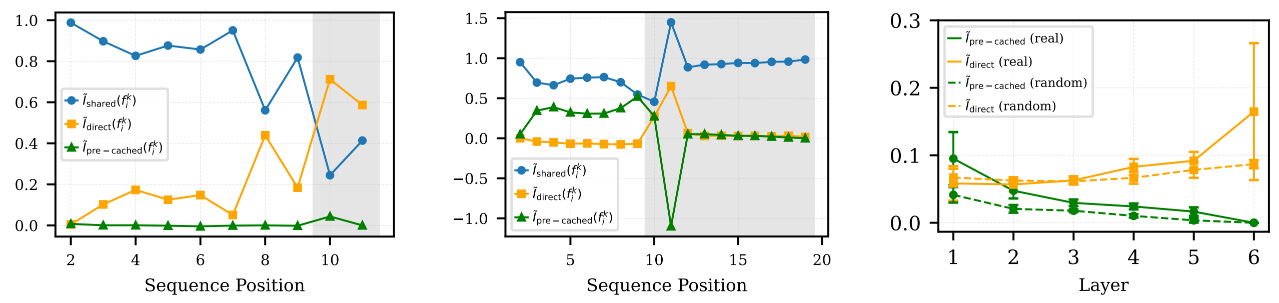

Figure 2. Left: integrated influence for the feature "the most frequent token so far" (Majority, $M = 10, K = 2$). *Center:* integrated influence for the feature "preceding token is A" (Conditioned Majority, $M = 10, K = 10$). *Right:* integrated influence for the board state representations as well as random features (Othello). The values are normalized by the sum of components. In the left and center plots, the white area represents the input phase (M tokens sampled uniformly) and the grey area represents the output phase (K tokens sampled according to the rules of the task).

Here $S = \{\text{direct}, \text{pre-cached}, \text{shared}\}$.

Remark 2.4 holds approximately due to the first-order approximation of feature mismatch $R(x \mid \theta_t, \theta^*, w)$. Indeed, expressing the change in feature mismatch at each step through its gradient and breaking it down into direct, pre-cached, and shared components leads to the equality above.

The remark shows that each loss component has its own influence on feature representation at every step of gradient descent. Integrated over the whole training process, this influence accounts for the discrepancy between the feature representation in the beginning and at the end of training[1].

Based on Remark 2.4, we can interpret the value $\widetilde{I}_{\text{direct}}(w_i^k) \equiv \sum_t I_{\text{direct}}(w_i^k, \theta_t)$ (*integrated direct influence*) as the overall impact of the direct loss component on the emergence of the feature w_i^k in the model, and similarly with the other two components.

We propose to use this decomposition in order to address the question stated in the introduction, answering which combinations of direct, shared, and pre-cached gradient signals produced a feature over the course of training, by evaluating the relative magnitudes of the three integrated components $\widetilde{I}_{\text{direct}}(w_i^k), \widetilde{I}_{\text{pre-cached}}(w_i^k), \widetilde{I}_{\text{shared}}(w_i^k)$.

This will allow us to understand how features develop even when they are not useful for immediate NTP.

3. Experiments

3.1. Method

For a given task, we train a Transformer twice. First, we do a standard training run, optimizing a Transformer to perform next token prediction. After we obtain the final checkpoint θ^*, for each feature of interest (for example, the color of a piece at a given position in the game of Othello), we train layer- and position-specific linear probes (Alain & Bengio, 2016; Belinkov, 2022), using the values of the residual stream of the model as input. Each of those probes represents one feature direction w_i^k.

Next, our target is to understand the reason behind the development of the direction w_i^k. We retrain the model from scratch with the same random seed and data order, repeating the training trajectory of the first run. For each batch in the training set, we compute $I_{\text{direct}}(w_i^k, \theta)$, $I_{\text{pre-cached}}(w_i^k, \theta)$, and $I_{\text{shared}}(w_i^k, \theta)$ (details in Appendix C.1). We sum those values across the batches, obtaining the integrated direct, shared, and pre-cached influences for each feature.

3.2. Toy Tasks

We first study two toy tasks where we have a clear understanding of the circuits required to solve them: *Majority* and *Conditioned Majority*. We train 2-layer Transformers to solve each task.

In **Majority**, each example x consists of M tokens sampled uniformly from the vocabulary of size V, and K tokens sampled from the set of the most frequent tokens so far: $\text{argmax}_t \ \text{count}(t, x_{\leqslant M})$. The task is solved by a simple uniform attention head computing the most frequent tokens. We track the influence components of the feature "the most frequent token so far". The first M tokens can be predicted trivially, without this feature, but the last K tokens require it. Accordingly, the gradient signal for this feature during training is dominated by shared paths for the first M tokens, and direct paths at the last K tokens (Figure 2a).

Conditioned Majority is designed to bring out the importance of pre-caching. The input consists again of two parts:

[1]Note that the equality 2.4 does not hold for more complex optimizers such as Adam due to the nonlinearities in momentum and adaptive step size.

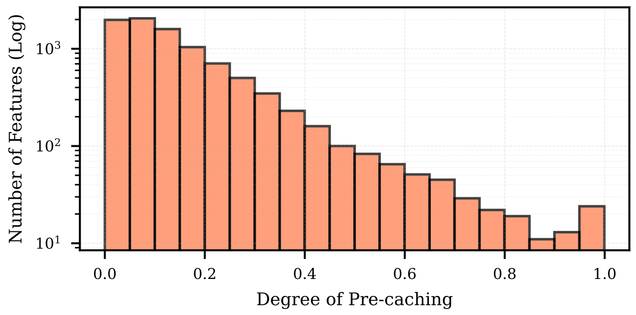

Figure 3. The distribution of pre-caching degree in SAE features of Gemma-2 2B (Layer 15).

M uniformly sampled tokens, followed by K uniform samples from the set of those tokens that followed the token “A” most often in the first part. The task requires a combination of two attention layers akin to induction heads (Olsson et al., 2022): the first layer attends to the preceding token and the second layer attends to the tokens succeeding A. We study the feature “the preceding token is A”. Indeed, the feature is important for future predictions when extracted from the first M tokens, whereas it is useless on the last K tokens. Consequently, the gradient signal over training has both shared and pre-cached components in the first part, but only shared components in the second part (Figure 2b).

3.3. Board State Representation in OthelloGPT

Next, we use our method to investigate the emergence of linear board state representations in Othello. Similar to prior work (Li et al., 2023; Nanda et al., 2023), we train a model for NTP on synthetically generated game transcripts. Due to resource constraints, we use a smaller board of 6×6 cells instead of the original 8×8.

We choose the colors of the four pieces in the center of the board as the features to study. We follow the methodology from Section 3.1. We also add four random directions in the residual space to the list of features to serve as baselines.

Results. Influence components across layers, averaged across random or real features, are shown in Figure 2c. Unsurprisingly, the features are mostly pre-cached in the shallow layers; direct influence affects the development of the features more than pre-caching in all but the first layer. However, in each layer, pre-caching influence for the board state features is higher than for the random features, indicating the development of some circuits transferring the board state information between positions in every layer.

4. Feature Roles in Trained Models

So far, we have studied the question *“Why has a certain feature emerged during training?”*. We now turn to a related, but distinct question: *“What is the role of that feature in a trained model?”*. We approach it by studying whether a given feature in a trained model serves as direct (useful for the immediate NTP), pre-cached (involved in information transfer between positions), or shared (useless for NTP at that position).

4.1. Measuring Pre-Caching in a Trained Model

The standard way of estimating the causal role of a feature in a Transformer is *an intervention* (Mueller et al., 2024): modifying the activations of a model during the forward pass to alter the representation of a feature and observing the changes in predictions. We employ this method to calculate the degree to which a given feature is involved in pre-caching.

Specifically, given an input, we run a forward pass without the intervention and record the output distribution at each step (p_j, where j is the position of the prediction). Next, we run one more forward pass, intervening on a feature at i-th position, and obtain the predictions under intervention p'_j. Then, we estimate $d_j = D_{\mathrm{KL}}(p'_j \mid\mid p_j)$ for each $j \geqslant i$, and use the value $(\sum_{j>i} d_j)/(\sum_{j\geqslant i} d_j)$ as the indicator of pre-caching degree: how much the feature influences future tokens compared to the immediate next one.

4.2. Pre-Caching of SAE features in Gemma 2

We employ the method described above to the features of a State-of-the-Art LLM: Gemma 2 (Gemma Team et al., 2024). We study the features from the Gemma-Scope (Lieberum et al., 2024) suite of Sparse Autoencoders for this model. During interventions, we simply ablate the SAE features one by one, zeroing out their activations in the hidden layer of the SAE (details in Appendix C.4).

The results are shown in Figure 3. Pre-caching degree seems to be distributed according to a power law, with most of the features being direct-only. The right side of the distribution, however, is unusually heavy, indicating the presence of a relatively high number of special pre-caching-only features. This finding supports the conclusions of Wu et al. (2024): pre-caching plays a relatively small role in LLMs, but that role is nevertheless nontrivial at large scale.

5. Discussion and Future Work

Our results indicate that all three forces, namely direct learning, pre-caching, and circuit sharing, are active in practice, and it is possible to trace the effect of each on feature learning. Moreover, the proposed approach can reveal new insights about the causes of emergence of specific features in particular tasks such as generating Othello games.

Application of our methods to tasks with more complex and nonlinear features, as well as establishing a precise link between the mechanisms behind feature emergence and the role of that feature in the final model checkpoint are exciting future research directions.

Acknowledgements

We thank Yash Sarrof, Aleksandra Bakalova, and Yana Veitsman for useful discussions and feedback on paper drafts.

References

Alain, G. and Bengio, Y. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

Bachmann, G. and Nagarajan, V. The pitfalls of next-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 2296–2318. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/bachmann24a.html.

Belinkov, Y. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219, 2022.

Bereska, L. F. and Gavves, E. Mechanistic interpretability for ai safety — a review. *TMLR*, April 2024.

Bricken, T., Templeton, A., Batson, J., Chen, B., Jermyn, A., Conerly, T., Turner, N., Anil, C., Denison, C., Askell, A., Lasenby, R., Wu, Y., Kravec, S., Schiefer, N., Maxwell, T., Joseph, N., Hatfield-Dodds, Z., Tamkin, A., Nguyen, K., McLean, B., Burke, J. E., Hume, T., Carter, S., Henighan, T., and Olah, C. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *International Conference on Machine Learning*, pp. 5209–5235. PMLR, 2024.

Dunefsky, J., Chlenski, P., and Nanda, N. Transcoders find interpretable llm feature circuits. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Engels, J., Liao, I., Michaud, E. J., Gurnee, W., and Tegmark, M. Not all language model features are linear. *arXiv preprint arXiv:2405.14860*, 2024.

Ferrando, J., Sarti, G., Bisazza, A., and Costa-Jussà, M. R. A primer on the inner workings of transformer-based language models. *arXiv preprint arXiv:2405.00208*, 2024.

Gao, L., la Tour, T. D., Tillman, H., Goh, G., Troll, R., Radford, A., Sutskever, I., Leike, J., and Wu, J. Scaling and evaluating sparse autoencoders. *arXiv preprint arXiv:2406.04093*, 2024.

Gemma Team, Riviere, M., Pathak, S., Sessa, P. G., Hardin, C., Bhupatiraju, S., Hussenot, L., Mesnard, T., Shahriari, B., Ramé, A., et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Gloeckle, F., Idrissi, B. Y., Roziere, B., Lopez-Paz, D., and Synnaeve, G. Better & faster large language models via multi-token prediction. In *International Conference on Machine Learning*, pp. 15706–15734. PMLR, 2024.

Hernandez, E., Sharma, A. S., Haklay, T., Meng, K., Wattenberg, M., Andreas, J., Belinkov, Y., and Bau, D. Linearity of relation decoding in transformer language models. In *The Twelfth International Conference on Learning Representations*, 2023.

Jenner, E., Kapur, S., Georgiev, V., Allen, C., Emmons, S., and Russell, S. J. Evidence of learned look-ahead in a chess-playing neural network. *Advances in Neural Information Processing Systems*, 37:31410–31437, 2024.

Joshi, A., Sharma, V., and Modi, A. CheckersGPT: Learning world models through language modeling. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pp. 576–588, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-srw.48.

Karvonen, A. Emergent world models and latent variable estimation in chess-playing language models. In *First Conference on Language Modeling*, 2024.

Li, K., Hopkins, A. K., Bau, D., Viégas, F., Pfister, H., and Wattenberg, M. Emergent world representations: Exploring a sequence model trained on a synthetic task. *ICLR*, 2023.

Lieberum, T., Rajamanoharan, S., Conmy, A., Smith, L., Sonnerat, N., Varma, V., Kramár, J., Dragan, A., Shah, R., and Nanda, N. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2. *arXiv preprint arXiv:2408.05147*, 2024.

Lin, J. Neuronpedia: Interactive reference and tooling for analyzing neural networks, 2023. URL https://www.neuronpedia.org. Software available from neuronpedia.org.

Lindsey, J., Templeton, A., Marcus, J., Conerly, T., Batson, J., and Olah, C. Sparse crosscoders for cross-layer features and model diffing. *Transformer Circuits Thread*,

2024. URL https://transformer-circuits.pub/2024/crosscoders/index.html.

Monea, G., Joulin, A., and Grave, E. Pass: Parallel speculative sampling. *arXiv preprint arXiv:2311.13581*, 2023.

Mueller, A., Brinkmann, J., Li, M. L., Marks, S., Pal, K., Prakash, N., Rager, C., Sankaranarayanan, A., Sharma, A. S., Sun, J., et al. The quest for the right mediator: A history, survey, and theoretical grounding of causal interpretability. *CoRR*, 2024.

Nanda, N., Lee, A., and Wattenberg, M. Emergent linear representations in world models of self-supervised sequence models. In Belinkov, Y., Hao, S., Jumelet, J., Kim, N., McCarthy, A., and Mohebbi, H. (eds.), *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 16–30, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.blackboxnlp-1.2. URL https://aclanthology.org/2023.blackboxnlp-1.2.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Pal, K., Sun, J., Yuan, A., Wallace, B., and Bau, D. Future lens: Anticipating subsequent tokens from a single hidden state. In *Proceedings of the 27th Conference on Computational Natural Language Learning (CoNLL)*, Singapore, December 2023.

Park, K., Choe, Y. J., Jiang, Y., and Veitch, V. The geometry of categorical and hierarchical concepts in large language models. In *ICML 2024 Workshop on Theoretical Foundations of Foundation Models*, 2024a.

Park, K., Choe, Y. J., and Veitch, V. The linear representation hypothesis and the geometry of large language models. In *International Conference on Machine Learning*, pp. 39643–39666. PMLR, 2024b.

Piotrowski, M., Riechers, P. M., Filan, D., and Shai, A. S. Constrained belief updates explain geometric structures in transformer representations. *arXiv preprint arXiv:2502.01954*, 2025.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rai, D., Zhou, Y., Feng, S., Saparov, A., and Yao, Z. A practical review of mechanistic interpretability for transformer-based language models. *arXiv preprint arXiv:2407.02646*, 2024.

Shai, A., Teixeira, L., Oldenziel, A., Marzen, S., and Riechers, P. Transformers represent belief state geometry in their residual stream. *Advances in Neural Information Processing Systems*, 37:75012–75034, 2024.

Templeton, A., Conerly, T., Marcus, J., Lindsey, J., Bricken, T., Chen, B., Pearce, A., Citro, C., Ameisen, E., Jones, A., Cunningham, H., Turner, N. L., McDougall, C., MacDiarmid, M., Freeman, C. D., Sumers, T. R., Rees, E., Batson, J., Jermyn, A., Carter, S., Olah, C., and Henighan, T. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*, 2024. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

Wu, W., Morris, J. X., and Levine, L. Do language models plan ahead for future tokens? In *First Conference on Language Modeling*, 2024.

Yuan, Y. and Søgaard, A. Revisiting the othello world model hypothesis. In *ICLR 2025 Workshop on World Models: Understanding, Modelling and Scaling*, 2025.

A. Related Work

Abstract features in Transformers. Our research is inspired by studies demonstrating that LLMs, and Transformers in general, learn to implicitly compute high-level features. Without claiming exhaustiveness, we list some of them here.

For various synthetic settings, it has been shown that Transformers implicitly reconstruct the latent variables of the data generation process, arriving to the so-called “world models”. In the domain of board games, this has been shown for Othello (Li et al., 2023; Nanda et al., 2023; Yuan & Søgaard, 2025), chess (Karvonen, 2024), and checkers (Joshi et al., 2024). In the controlled setting of Hidden Markov Models, Shai et al. (2024) and Piotrowski et al. (2025) have shown that Transformers calculate the belief state of the environment, closely aligned to the theoretically optimal prediction.

A broad field of research has also focused on finding abstract features implicit in general-purpose LLMs. LLMs have been shown to represent interpretable real-world relations linearly (Park et al., 2024b;a; Hernandez et al., 2023) and nonlinearly (Engels et al., 2024). Perhaps most well-known, interpretable linear features have been discovered in the internal representations of LLMs using such techniques as Sparse Autoencoders (Bricken et al., 2023; Templeton et al., 2024; Gao et al., 2024) or their variants (Lindsey et al., 2024; Dunefsky et al., 2024).

Future token prediction in Transformers. Another line of work studies look-ahead in Transformers: the emergent ability to predict multiple future tokens despite being trained to predict only one. Pal et al. (2023) show that future token predictions can be decoded from the internal representations of a pretrained LLM with nontrivial accuracy using probes and learned prompts. Jenner et al. (2024) find a similar effect in a neural network trained to play chess.

Some recent engineering efforts have been targeted to intentionally elicit look-ahead, using it to improve the performance and efficiency of Transformers. Cai et al. (2024) and Monea et al. (2023) predict multiple future tokens during decoding to speed up autoregressive text generation. Gloeckle et al. (2024) introduce an additional term encouraging look-ahead to the language modeling loss, enhancing both downstream capabilities and inference speed. Bachmann & Nagarajan (2024) use a similar method to bypass the limitations of the NTP objective.

Most relevant to our work, Wu et al. (2024) investigates the reasons behind the emergence of look-ahead and proposes two competing hypotheses: *breadcrumbs* and *pre-caching*. The pre-caching hypothesis posits that some tokens “prepare in advance” the information relevant for future tokens (and that information is picked by the look-ahead probes). The breadcrumbs hypothesis on the contrary, states that the features relevant for predicting the immediate next token are generally similar to the ones relevant for predicting future tokens, and hence the former can to some extent substitute the latter, enabling look-ahead. The results of Wu et al. (2024) suggest that the breadcrumbs hypothesis is likely closer to the truth for LLMs, though pre-caching plays some role in synthetic settings or large-scale models.

We build upon that work and borrow some of its terminology: most importantly, the term “pre-caching”. However, our contribution is distinct from that of Wu et al. (2024) in several aspects. We investigate breadcrumbs in more detail, breaking this hypothesis down into direct learning and circuit sharing. Crucially, we make a step from treating pre-caching and breadcrumbs as two general hypotheses about Transformers’ behavior to the analysis of their roles in development of specific features. Thus, our contribution can be seen as fine-graining the distinction between pre-caching and breadcrumbs introduced by Wu et al. (2024) and bringing it down to the level of individual features.

LLM Interpretability Our work broadly relates to the field of LLM interpretability and its subarea of mechanistic interpretability. We refer the reader to the surveys on this topic for the comprehensive review of its methods: Rai et al. (2024); Bereska & Gavves (2024); Ferrando et al. (2024).

B. Proof of Theorem 2.1

Theorem B.1 (Restated from Theorem 2.1). *For any layer k and position i,*

$$\nabla_\theta L(x, T_\theta(x)) = \nabla_\theta L^k_{i\,(\text{direct})}(x, T_\theta(x)) + \nabla_\theta L^k_{i\,(\text{pre-cached})}(x, T_\theta(x)) + \nabla_\theta L^k_{i\,(\text{shared})}(x, T_\theta(x)) \tag{4}$$

Where

$$\nabla_\theta L^k_{i\,(\text{direct})}\left(x, T_\theta(x)\right) = \nabla_\theta L(x_{i+1}, T_\theta(x)_i) - \nabla_\theta L\left(x_{i+1}, [T_\theta(x)]_i^{\text{sg}(i,k)}\right), \tag{5}$$

$$\nabla_\theta L^k_{i\,(\text{pre-cached})}\left(x, T_\theta(x)\right) = \sum_{j\neq i}\left[\nabla_\theta L(x_{j+1}, T_\theta(x)_j) - \nabla_\theta L\left(x_{j+1}, [T_\theta(x)]_j^{\text{sg}(i,k)}\right)\right], \tag{6}$$

$$\nabla_\theta L^k_{i\,(\text{shared})}\left(x, T_\theta(x)\right) = \sum_{j} \nabla_\theta L\left(x_{j+1}, [T_\theta(x)]_j^{\text{sg}(i,k)}\right) \tag{7}$$

Proof.

$$\nabla_\theta L(x, T_\theta(x)) = \nabla_\theta L\big(x, T_\theta(x)\big) + \nabla_\theta L\left(x, [T_\theta(x)]^{\text{sg}(i,k)}\right) - \nabla_\theta L\left(x, [T_\theta(x)]^{\text{sg}(i,k)}\right) =$$

$$= \sum_{j=1}^{N-1} \nabla_\theta L\left(x_{j+1}, [T_\theta(x)]_j^{\text{sg}(i,k)}\right) + \sum_{j=1}^{N-1} \nabla_\theta L\big(x_{j+1}, T_\theta(x)_j\big) - \sum_{j=1}^{N-1} \nabla_\theta L\left(x_{j+1}, [T_\theta(x)]_j^{\text{sg}(i,k)}\right) =$$

$$= \overbrace{\sum_{j=1}^{N-1} \nabla_\theta L\left(x_{j+1}, [T_\theta(x)]_j^{\text{sg}(i,k)}\right)}^{\nabla_\theta L^k_{i\,(\text{shared})}} + \overbrace{\left(\nabla_\theta L\big(x_{i+1}, T_\theta(x)_i\big) - \nabla_\theta L\left(x_{i+1}, [T_\theta(x)]_i^{\text{sg}(i,k)}\right)\right)}^{\nabla_\theta L^k_{i\,(\text{direct})}} +$$

$$+ \underbrace{\sum_{j\neq i}\left[\nabla_\theta L\big(x_{j+1}, T_\theta(x)_j\big) - \nabla_\theta L\left(x_{j+1}, [T_\theta(x)]_j^{\text{sg}(i,k)}\right)\right]}_{\nabla_\theta L^k_{i\,(\text{pre-cached})}}$$

□

C. Additional Experimental Details

C.1. Computing the Influence

In this section, we explain in more detail our methodology for computing the gradient influence terms during training.

By Definition 2.3,

$$I_i^k(\theta, x, y \mid w_i^k, \theta^*, G) = \frac{d}{d\varepsilon} R\left(x \mid \theta + \varepsilon G, \theta^*, w_i^k\right)\Big|_{\varepsilon=0}$$

A simple application of chain rule gives us an equivalent definition:

$$I_i^k(\theta, x, y \mid w_i^k, \theta^*, G) = \left\langle \nabla_\theta R\left(x \mid \theta, \theta^*, w_i^k\right), G\right\rangle$$

Thus, computing the influence of a given gradient term can be done by taking the inner product of that gradient term and the gradient of the feature mismatch.

Thus, to compute $I_{\text{direct}}(w_i^k, \theta)$, $I_{\text{pre-cached}}(w_i^k, \theta)$, and $I_{\text{shared}}(w_i^k, \theta)$ for a given i and k, we employ the following algorithm. First, we calculate $\nabla_\theta L_j$ for every j in a standard way. Next, we calculate $\nabla_\theta L_j^{\text{sg}(k,i)}$ in a similar manner, but during the forward pass of the model we detach the tensor corresponding to $r^k_{\theta,i}(x)$. After that, we compute the gradient decomposition terms according to the definitions in Section 2.2.

The only thing left is the gradient of the feature mismatch. We apply the linear probe defined by w_i^k to both $r^k_{\theta,i}(x)$ and $r^k_{\theta^*,i}(x)$ and compute the feature mismatch according to Definition 2.2. We run one more backpropagation to find $\nabla_\theta R$ and take its inner product with the gradient decomposition terms, obtaining the desired influence values.

C.2. Toy Tasks Experiments

For both Majority and Conditioned Majority, we generate training datasets of 100k objects using vocabulary size 3 (tokens “A”, “B”, “C”) and the input phase length $M = 10$. The output phase length K is 2 for Majority and 10 for Conditioned

Majority. We increased K for Conditioned Majority because we had observed that with low K, the strength of the training signal was insufficient to learn the task, leading to overfitting.

We train tiny Transformers with 2 layers and a hidden dimension of size 32. We use the architecture of GPT-2 (Radford et al., 2019). The models are trained for 2k steps with a batch size 128, a constant learning rate 10^{-3} and Adam optimizer.

The hidden representations for probing are extracted from the residual stream after the first Transformer block. All results are averaged across 5 runs with different random seeds.

C.3. Othello Experiment

Following the methodology of Li et al. (2023), we generate a synthetic dataset of one million legal Othello games. The size of the board in our experiment is 6×6. Each token represents a coordinate pair (x, y) of the cell where a piece is placed during the move.

Same as in the toy experiments, our model has the architecture similar to GPT-2. We use 6 Transformer layers and hidden dimension 256. The model is trained for 4k steps with a batch size 1024 and a constant learning rate $5 \cdot 10^{-4}$.

After training, the accuracy of legal move prediction is 0.98.

When training linear probes for the extraction of the piece color at a given position, we follow Nanda et al. (2023) and encode the pieces belonging to the player making the move as 1, and the pieces belonging to the other player as -1.

C.4. LLM Experiment with Gemma 2

Here, we provide additional details on our methodology for evaluating the pre-caching degree of features in Gemma 2 2B (Gemma Team et al., 2024).

We use the Gemma-Scope SAE release (Lieberum et al., 2024), specifically the SAE trained at the residual stream at layer 15 with 16k hidden features.

For each of those features, we run the following algorithm. We extract the top 5 sequences by the strength of the activation of that feature from Neuronpedia API (Lin, 2023). Then, for those sequences we run our algorithm of measuring pre-caching degree described in Section 4.1.

Let N be the length of the sequence. First, we run a forward pass and save the predicted logits at each position. Then, for every $i \in 1 \ldots N$, we run a forward pass again, but setting the activation of the SAE feature under study at position i (and only there) to 0. We record the predictions under ablation and, comparing them to the predictions without the ablation, calculate the pre-caching degree as described in Section 4.1. We track the change in prediction up to the distance of 10 tokens to the position under ablation.

We average the results across positions and sequences. This way, for each of the 16k hidden features of the SAE, we get the estimate of its pre-caching degree. The aggregated results are shown in Figure 3, and the examples of the discovered features with high pre-caching degree are listed below.

C.5. The Features with the Highest Pre-Caching Degree

This section demonstrates the examples of activating strings for the features with the highest pre-caching degree. For each feature, we print an activating sequence here and show the effect of this feature on predictions for that sequence in Figure 4. The sequences are obtained through Neuronpedia API (Lin, 2023).

The newlines are replaced with spaces and the ablated tokens are marked with **.

Note that all of those sequences turn out to be snippets of computer code. This may hint that the pre-cached features are most needed in the domain of code generation, and that is why the top 5 pre-cached features we found are related to code. We do not investigate this hypothesis in detail.

Feature 5106:

// _This _source _file _is _part **_of** _the _Swift . org _open _source _project // // _Copyright (c) _ 2 0 1 4 _– _ 2 0 1 8 _Apple _Inc . _and _the _Swift _project _authors // _Licensed _under _Apache _License _v 2 . 0 _with _Runtime _Library _Exception // // _See _https ://

Feature 14059:

<bos> package _org . base x . query . func . validate ; import _org . base x **.** query .*; import _org . base x . query . func .*; import _org . base x . query . value . item .*; import _org . base x . util .*; /** _* _Function _implementation . _* _* _@ author _Base X _Team _ 2 0 0 5 – 2

Feature 14626:

Object HideFlags : _ 0 __ m Corresponding Source Object : _{ fileID **:** _ 0 } __ m Prefab Instance : _{ fileID : _ 0 } __ m Prefab Asset : _{ fileID : _ 0 } __ m Name : _Floor __ m Shader : _{ fileID : _ 4 8 0 0 0 0 0 , _guid : _ 9 3 3 5 3 2 a

Feature 6928:

StringWriter ; import _org . apache . tika **.** T ika ; import _org . apache . tika . exception . T ika Exception ; public _class _App _ { ____ public _static _void _main (_String [] _args _) ____ { ________ try

Feature 2656:

. xml . ____ int _id _= _item . getItemId **();** ____ // noinspection _Simp lif iable If Statement ____ if _(id _== _R . id . action settings) _{ ________ return _true ; ____ }

Feature 7125:

. services . rek ogni tion . model . transform ; import _com . amazonaws **.** services . rek ogni tion . model .*; import _com . amazonaws . transform . Simple Type Json Un marshall ers .*; import _com . amazonaws . transform .*; import _com . amazonaws . util . json . Aws Json Reader ; /** _* _JSON _un marshaller _for _PO JO _Emotion _*/ class _Emotion

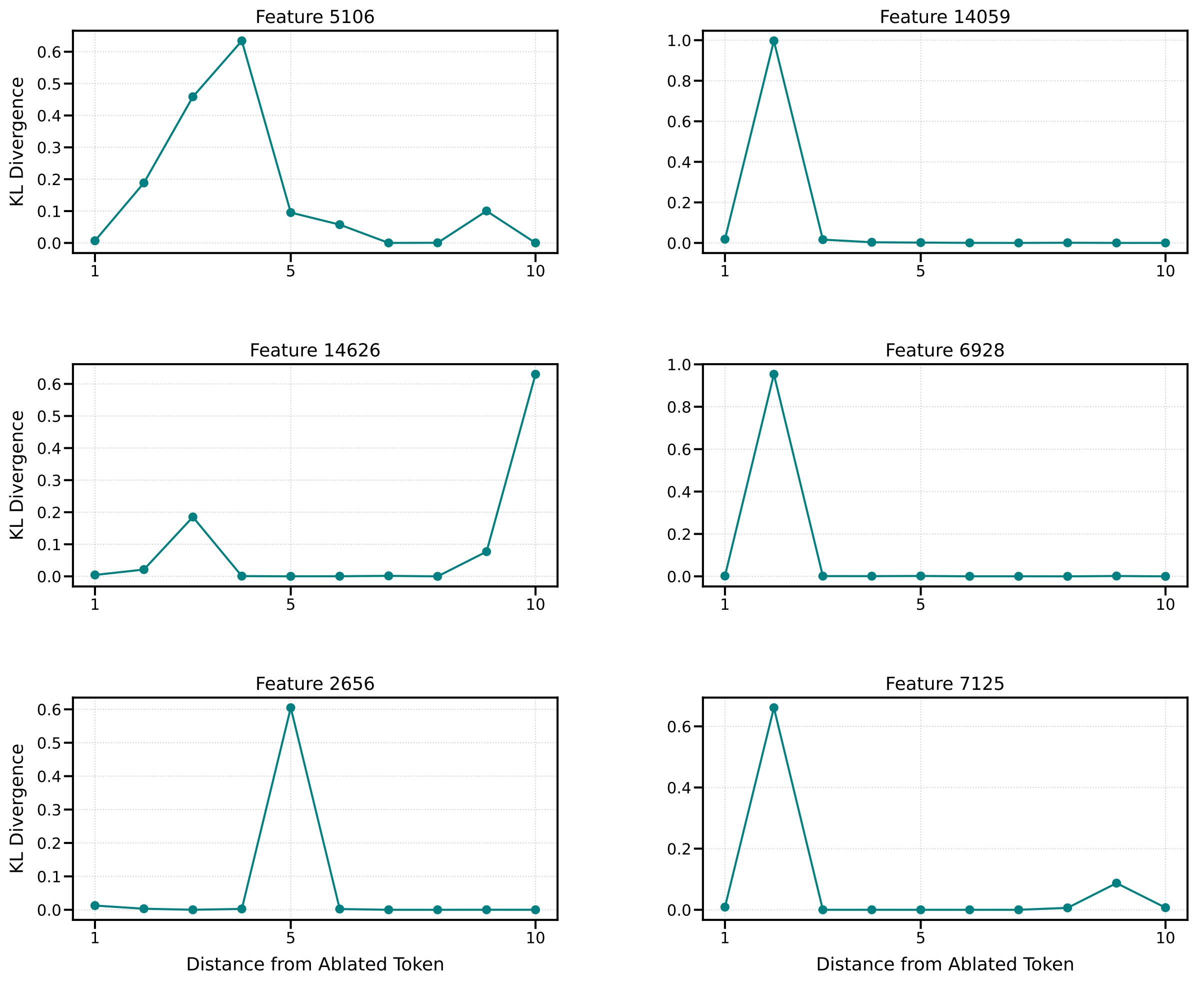

Figure 4. The difference between the predicted token distributions with and without ablation depending on the distance from the ablated token. The data from one sequence is plotted for each feature.