Steering Large Language Models using Conceptors: Improving Addition-Based Activation Engineering

Joris Postmus
University of Groningen
Groningen, Netherlands
j.postmus@student.rug.nl

Steven Abreu
University of Groningen
Groningen, Netherlands
s.abreu@rug.nl

Abstract

Large language models have transformed AI, yet reliably controlling their outputs remains a challenge. This paper explores activation engineering, where outputs of pre-trained LLMs are controlled by manipulating their activations at inference time. Unlike traditional methods using a single steering vector, we introduce conceptors–mathematical constructs that represent sets of activation vectors as ellipsoidal regions. Conceptors act as soft projection matrices and offer more precise control over complex activation patterns. Our experiments demonstrate that conceptors outperform traditional methods across multiple steering tasks. We further use Boolean operations on conceptors for combined steering goals that empirically outperform additively combining steering vectors on a set of tasks. These results highlight conceptors as a promising tool for more effective steering of LLMs. Our code is available on github.com/jorispos/conceptorsteering.

1 Introduction

Large language models (LLMs) have rapidly advanced AI capabilities [1], but their potential to spread misinformation [2], reinforce biases [3], and develop harmful behaviors [4] highlights the urgent need for methods to understand and control their outputs. Various methods, including reinforcement learning from human feedback (RLHF) [5], supervised fine-tuning [6], and prompt engineering [7], have been proposed to steer LLM outputs toward desired patterns. However, RLHF and fine-tuning are computationally expensive and struggle with generalization [8, 9], while prompt engineering often produces inconsistent results [10].

Activation engineering [11, 12] has recently been proposed as a new steering method which works by directly modifying the model's activations at inference time without changing the model's parameters and without expensive optimization. A steering vector that represents desired behavior can be computed directly or (more commonly) contrastively from positive and negative examples [13]. However, finding contrastive prompts to identify complex patterns is not always possible and, more importantly, the performance of activation addition for steering is not reliable [12].

This paper introduces an alternative to the predominant approach for steering LLMs using activation engineering. Instead of averaging or subtracting a set of activation vectors to form a steering vector, we use the cached activations to compute a *conceptor* [14], which we refer to as a "steering matrix". Instead of manipulating the LLM's activations using vector addition, the activations are (softly) projected using a matrix-vector multiplication with the steering matrix. We contribute the following: (1) we introduce a novel application of conceptors [14] as steering mechanisms for LLMs, (2) we apply this mechanism to function vectors [15] on GPT-NeoX and GPT-J, and (3) we show how a Boolean algebra on conceptors [16] can be used for combining steering targets on GPT-J.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

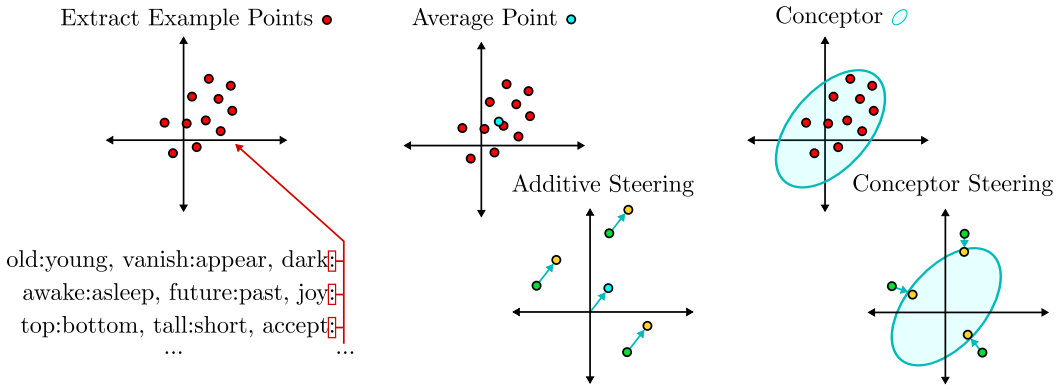


Figure 1: Illustration showing the basic geometric difference between additive and conceptor steering using a set of activations for the antonym task. Additive steering acts as a translation of the activation vectors by a fixed steering vector. Conceptor steering acts as a (soft) projection onto a target ellipsoid.

2 Background

Adding steering vectors to the residual stream has been used to control the output of LLMs across various domains [12, 13, 17]. The use case that will mainly be focused on here are the findings from the paper by Todd *et al.* [15]. Their work showed that a steering vector can be extracted from the residual stream that captures the activation space of an input-output function (e.g. a function that takes a word and returns its antonym). This steering vector can then be added to the residual stream at inference time to steer the model toward performing the captured function. See Figure 6 in Appendix A.1.1 for an illustration of function vector tasks.

Their baseline method works as follows. First, a set of in-context learning (ICL) prompts P_f that demonstrate a particular task f (the execution of an input-output function) are compiled. Then, for each prompt $p_i^f \in P_f$ (*e.g.*, $p_1^{antonym} =$ `hot:cold,old:`), the final token's activations $h_\ell(p_i^f)$ are cached at a specific layer ℓ from the residual stream h. The cached activation vectors are then averaged into the steering vector $\bar{h}_\ell^f$ for task f at layer ℓ:

$$\bar{h}_\ell^f = \frac{1}{|P_f|} \sum_{p_i^f \in P_f} h_\ell(p_i^f) \tag{1}$$

To steer the model towards performing this function, the function (steering) vector $\bar{h}_\ell^f$ can be added (without additional re-normalization) to the residual stream at layer ℓ when the model would be completing a prompt containing a previously unseen input:

$$h'_\ell = \beta_{\text{add}}\, \bar{h}_\ell^f + h_\ell \tag{2}$$

where h'_ℓ is the steered activation and $\beta_{add} > 0$ is a hyperparameter. The performance of additive steering can further be improved by a technique called *mean-centering* [18], see Appendix A.2.1.

3 Conceptors as Steering Matrices

Conceptors can broadly be defined as a neuro-computational mechanism designed to encapsulate and manipulate the state space of neural activations [14]. A conceptor matrix C is a positive semi-definite matrix that captures the principal directions and variances of a set of neural activation vectors. This structure can be visualized as a high-dimensional ellipsoid that describes the overall shape and spread of the activations' "underlying pattern", or state space region. See Figure 2 for a visual illustration. Because conceptors are computed from the cloud of activation vectors and encode the correlations between activations, conceptors can better capture the activation space of complex patterns compared to simple point representations, which discard information about correlations. This difference is illustrated geometrically in Figure 1.

Conceptors have been used to control pattern-generating RNNs effectively across various behaviors [16], prevent catastrophic forgetting and enhance continual learning in feedforward networks [19],

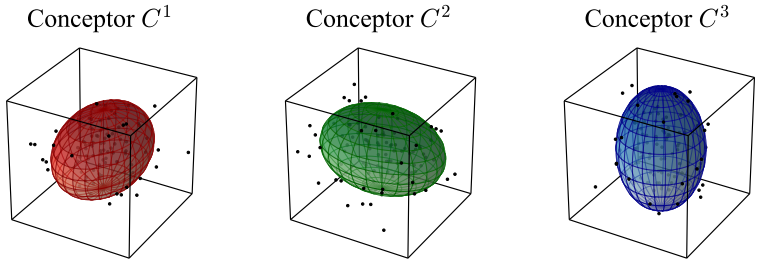


Figure 2: Illustration of three conceptors as ellipsoids that capture the state space region of different sets of neural activations in 3D space (black points). Reproduced from Jaeger [14].

remove bias subspaces in LLMs like BERT and GPT [20], and distill linguistic abstractions into knowledge graphs from contextual embeddings [21, 22].

One way to formalize the conceptor matrix C, is through an optimization that minimizes the reconstruction error while using a regularization term. The objective function to be minimized is:

$$\min_C \|X - XC\|_F^2 + \alpha^{-2}\|C\|_F^2$$

where X is a matrix of neural activation vectors (stacked as rows), $\|\cdot\|_F$ is the Frobenius norm, and α is the regularization parameter also referred to as the conceptor's *aperture*. This aperture parameter α balances the trade-off between accurately representing the activation pattern and maintaining a generalized representation. The closed-form solution to this problem is given by:

$$C(R, \alpha) = R\left(R + \alpha^{-2} I\right)^{-1} \quad \text{with} \quad R = \frac{X^T X}{n} \tag{3}$$

where n is the number of samples, and I is the identity matrix of the same dimensionality as R.

The eigenvalues μ_i of the conceptor matrix C are defined as:

$$\mu_i = \begin{cases} \frac{\lambda_i}{\lambda_i + \alpha^{-2}} & \text{for } 0 < \lambda_i < 1 \text{ and } 0 < \alpha < \infty \\ 0 & \text{for } 0 < \lambda_i < 1 \text{ and } \alpha = 0 \\ 1 & \text{for } 0 < \lambda_i < 1 \text{ and } \alpha = \infty \\ 0 & \text{for } \lambda_i = 0 \text{ and } 0 \leq \alpha \leq \infty \\ 1 & \text{for } \lambda_i = 1 \text{ and } 0 \leq \alpha \leq \infty \end{cases}$$

where λ_i represents the eigenvalues of the correlation matrix R. These eigenvalues μ_i fall within the interval $[0, 1]$ and are influenced by the aperture parameter α. When α is large, the eigenvalues μ_i approach 1 and C approaches the identity matrix, causing the conceptor to allow for more signal components to pass through the projection of the states with the conceptor matrix Cx. Conversely, when α is small, the eigenvalues μ_i approach 0, causing the conceptor to allow for less variability. In the extreme case of $\alpha \to 0$, the conceptor tends to the zero mapping.

We can use the conceptor for steering by collecting activations $h_\ell(p_i^f)$ into X and then compute the associated conceptor C_ℓ^f using Equation 3, and finally steer new hidden activations h_ℓ with:

$$h'_\ell = \beta_c C_\ell^f h_\ell \tag{4}$$

where h'_ℓ is the steered activation and $\beta_c > 0$ is a hyperparameter. We can think of this as a "soft projection". A projection matrix has eigenvalues that are either zero or unity, but the conceptor matrix has "soft" eigenvalues between zero and unity. Thus, the conceptor "softly projects" the activation vector h_ℓ toward the pattern represented by C_ℓ^f by scaling its components according to the patterns' principal directions.

3.1 Boolean Operations on Conceptors

We can combine multiple steering matrices using the conceptor Boolean operations as defined by Jaeger [16]. We begin with the OR operation on conceptors, which can be interpreted as merging the

data from which each conceptor is computed by adding the covariance matrices on which C_1 and C_2 were computed. Given that C_1 was computed with the covariance matrix R_1 and C_2 was computed with the covariance matrix R_2, the conceptor that is computed on the sum of the two covariance matrices $R_1 + R_2$ is defined as $C_1 \vee C_2$:

$$C_1 \vee C_2 = (R_1 + R_2)(R_1 + R_2 + \alpha^{-2} I)^{-1} \tag{5}$$

$$C_1 \vee C_2 = \left(I + \left(C(I - C)^{-1}\right) + B(I - B)^{-1}\right)^{-1}\Big)^{-1} \tag{6}$$

The NOT operation on a conceptor C is defined as the conceptor $\neg C$ that is computed on a covariance matrix R^{-1} that is the inverse of the original covariance matrix R for conceptor C. Intuitively, $\neg C$ can be interpreted as the conceptor that arises from data that which co-vary inversely to the data giving rise to C:

$$\neg C = R^{-1}(R^{-1} + \alpha^{-2} I)^{-1} \tag{7}$$

$$\neg C = I - C \tag{8}$$

For our experiments, we use the AND operation which can now be obtained from the NOT and OR operations using de Morgan's law $a \wedge b = \neg(a \vee b)$ such that the conceptor $C_1 \wedge C_2$ is computed using the correlation matrix $(R_1^{-1} + R_2^{-1})^{-1}$. This leads to:

$$C_1 \wedge C_2 = (R_1^{-1} + R_2^{-1})^{-1} \left((R_1^{-1} + R_2^{-1})^{-1} + \alpha^{-2} I\right)^{-1} \tag{9}$$

$$C_1 \wedge C_2 = (C_1^{-1} + C_2^{-1} + I)^{-1} \tag{10}$$

3.2 Computational Complexity of Conceptor Steering

The cost of computing a conceptor steering matrix is dominated by the matrix inversion and matrix-matrix multiplication of the activation correlation matrix $R = XX^T/n$ (see Equation 3). This correlation matrix is a $n \times n$-dimensional matrix where n is the dimension of the activation vectors (typically <4096 for the model sizes we presented, or up to 8192 for larger models such as Llama-2-70B), so the complexity of the conceptor computation is $\mathcal{O}(n^3)$. This computation is done entirely offline and the cost is amortized over all future applications of the steering method. The final conceptor $C \in \mathbb{R}^{n \times n}$ takes $O(n^2)$ memory – the same amount as a weight matrix acting on the activation vectors. For 32-bit floating point numbers, this amounts to 17MB for $n = 2048$, 67MB for $n = 4096$, or 268MB for $n = 8192$.

During inference, conceptor steering adds an extra matrix-vector multiplication Cx with the activation vector x. However, the additional memory and inference cost for applying the conceptor can be eliminated by fusing the conceptor with the succeeding weight matrices for the query, key and value weight matrices. This is equivalent to replacing the existing weight matrix W_x with the conceptor-fused weight matrix $W_x^C = W_x C$. This fusing of operations is standard practice when optimizing networks for inference. We note that there may be an overhead cost for switching the conceptor steering on and off which amounts to the cost of changing the network's computational graph during inference. We believe this overhead to be negligible during auto-regressive generation on a single data sample, but it must be considered when using batch sizes larger than one.

4 Experiments

For our experiments, we will use EleutherAI's GPT-J 6B and GPT-NeoX 20B models, as done in previous works on activation steering [15, 18]. For all experiments, we find optimal hyperparameters for each steering method at every layer. The details of our grid search for α and β_c for conceptor-based steering and β_{add} for additive steering can be found in Appendix A.1.2.

4.1 Function Steering

We compare conceptor-based and additive steering mechanisms on their ability to steer a given model towards correctly executing a set of functions. We test both methods on GPT-J with 6B parameters and GPT-NeoX with 20B parameters. For each function, the described experiment will be repeated five times with different random seeds, and all reported results are averaged across across these five

runs. The examples of the input-output functions come from the dataset by Todd *et al.* [15]. We use the following subset of five functions [18]: antonyms (e.g. good→bad), present-past (e.g. go→went), English-French (e.g. hello→bonjour), singular-plural (e.g. mouse→mice), country-capital (e.g. Netherlands→Amsterdam), and capitalize (e.g. word→Word). To ensure comparability of our results, we follow [15] as closely as possible. For more details, see Appendix A.1.1.

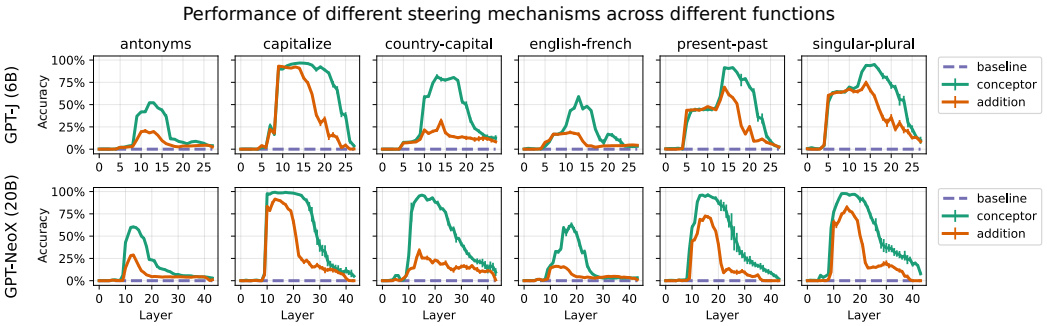


Figure 3: Comparison of the accuracy on all six function tasks for conceptor-based steering against additive steering across all layers for GPT-J and GPT-NeoX. For explanation, see main text.

The results in Figure 3 show that conceptor-based steering outperforms additive steering (the baseline method reported in Ref. [15]) for every task on both tested models. In line with previous findings [15, 18], steering is most effective across layers 9-16 for GPT-J and layers 10-30 for GPT-NeoX.

Table 1 and Figure 4 show that mean-centering (as outlined in Appendix A.2.1) provides a small improvement for both addition-based and conceptor-based steering. Mean-centering improves the performance of additive steering by as much as 2x (on the country-capital task). For conceptor-based steering the improvements of mean-centering are relatively smaller – at most 5% on the country-capital task. Conceptor-based steering outperforms additive steering on all tasks, even comparing additive steering with mean-centering against conceptor-based steering without mean-centering.

Table 1: The effect of mean centering on conceptor-based and addition-based steering on the GPT-J (6B) model, across simple function vector tasks. Results show the best performance across all hyperparameters and across all layers.

	antonyms	capitalize	country-capital	english-french	present-past
Addition	20.54%	93.16%	32.04%	18.88%	69.66%
Addition (MC)	31.20%	95.00%	63.90%	34.32%	83.32%
Conceptor	52.14%	**96.68%**	81.62%	59.02%	91.56%
Conceptor (MC)	**52.82%**	96.26%	**85.32%**	**61.32%**	**91.88%**

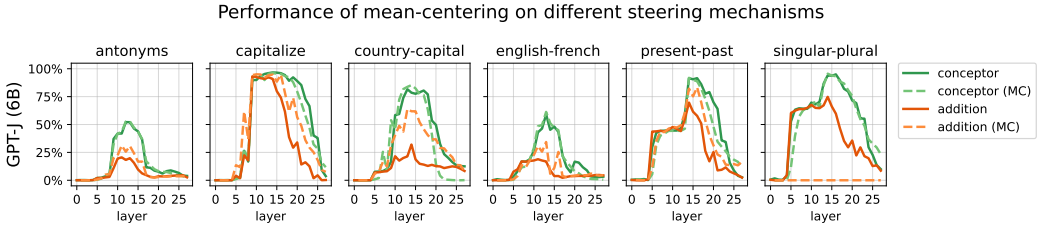


Figure 4: The effect of mean centering on conceptor-based and addition-based steering on the GPT-J (6B) model across all layers, computed on five different function vector tasks (% accuracy). The line shows the best average performance across five runs for the best hyperparameters for the given layer.

4.2 Steering Composite Functions

We further conducted experiments where two conceptors, each representing one of three different functions, were combined using the AND operator. The input-output example dataset for this function was generated using GPT-4o. To present the baseline for how well non-combined steering mechanisms perform, we show results for the conceptor $C^{1,2}$ and the steering vector $h_\ell^{1,2}$ that were each computed on the compound function directly. We then combine the conceptors computed on the individual functions C^1 and C^2 using the AND operation as $C^1 \wedge C^2$, and we combine the steering vectors h_ℓ^1 and h_ℓ^2 using their arithmetic mean $\frac{1}{2}(h_\ell^1 + h_\ell^2)$.

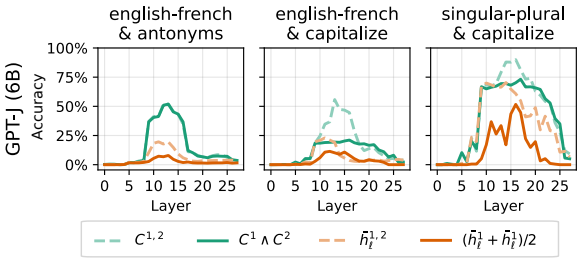


Figure 5: Performance of additive steering and conceptor steering on composite functions. For explanation of the figure caption, see text. Dashed lines represent the "baseline" where the steering mechanism is computed on the composite task. Solid lines show task arithmetic.

Figure 5 shows the performance of all compared methods across all layers of the GPT-J model. In line with the results from Section 4.1, the conceptor baseline outperformed the additive baseline on all three tasks. The AND-combined conceptor outperformed the mean-combined steering vectors. On one of the three tasks, english-french & antonyms, the AND-combined conceptor even outperforms the additive baseline.

5 Conclusion

In our experiments, conceptor-based steering generally outperformed addition-based methods. Further research should be conducted to assess the mechanisms' impact on the model's overall capabilities, the performance on more complex behaviors/tasks, and the scalability to larger models.

A limitation of conceptors is their reliance on more data points to build accurate representations. Additionally, the inherent mathematical structure and additional required computations makes it more computationally expensive compared to simple addition-based methods. However, while more expensive than addition-based approaches, they are still much cheaper than alternatives like RLHF and fine-tuning. Conceptors also introduce a new hyperparameter, the aperture α, that may require tuning for optimal performance. In our experiments, we found a single aperture value, $\alpha = 0.1$, yields the best performance across all experiments[1], but this finding must be verified for new models and steering tasks.

Despite these challenges, conceptor-based steering methods could offer a more precise and effective way to steer LLMs compared to traditional addition-based methods, proposing a fundamental shift in what is possible with activation engineering. Our experiments on conceptor-based steering further suggest that region-based representations may allow for more flexible and nuanced steering compared to point-based representations. The proposed method could have significant positive implications for debiasing models, aligning models with human values, and overall AI safety.

[1]More precisely, the aperture value $\alpha = 0.1$ is within 10% of the best-performing aperture value across all experiments and models. In most experiments, it is the single best value. See Appendix A.3 for more details.

References

[1] Bo Xu and M. Poo. Large language models and brain-inspired general intelligence. *National Science Review*, 10, 2023. doi: 10.1093/nsr/nwad267.

[2] Yikang Pan, Liangming Pan, Wenhu Chen, Preslav Nakov, Min-Yen Kan, and William Yang Wang. On the risk of misinformation pollution with large language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=voBhcwDyPt.

[3] Isabel O. Gallegos, Ryan A. Rossi, Joe Barrow, Md Mehrab Tanjim, Sungchul Kim, Franck Dernoncourt, Tong Yu, Ruiyi Zhang, and Nesreen K. Ahmed. Bias and fairness in large language models: A survey. *arXiv*, abs/2309.00770, 2024. URL https://arxiv.org/abs/2309.00770.

[4] Toby Shevlane, Sebastian Farquhar, Ben Garfinkel, Mary Phuong, Jess Whittlestone, Jade Leung, Daniel Kokotajlo, Nahema Marchal, Markus Anderljung, Noam Kolt, Lewis Ho, Divya Siddarth, Shahar Avin, Will Hawkins, Been Kim, Iason Gabriel, Vijay Bolina, Jack Clark, Yoshua Bengio, Paul Christiano, and Allan Dafoe. Model evaluation for extreme risks, 2023. URL https://arxiv.org/abs/2305.15324.

[5] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9781713871088.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

[7] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 55(9), jan 2023. ISSN 0360-0300. doi: 10.1145/3560815. URL https://doi.org/10.1145/3560815.

[8] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *arXiv*, abs/1606.04838, 2018. URL https://arxiv.org/abs/1606.04838.

[9] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv*, abs/1606.06565, 2016. URL https://arxiv.org/abs/1606.06565.

[10] Banghao Chen, Zhaofeng Zhang, Nicolas Langren'e, and Shengxin Zhu. Unleashing the potential of prompt engineering in large language models: a comprehensive review. *ArXiv*, abs/2310.14735, 2023. doi: 10.48550/arXiv.2310.14735.

[11] Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=aLLuYpn83y.

[12] Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. Activation addition: Steering language models without optimization, 2024. URL https://arxiv.org/abs/2308.10248.

[13] Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Turner. Steering llama 2 via contrastive activation addition. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15504–15522, Bangkok, Thailand, August 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.828.

[14] Herbert Jaeger. Conceptors: an easy introduction. *arXiv*, abs/1406.2671, 2014. URL https://arxiv.org/abs/1406.2671.

[15] Eric Todd, Millicent Li, Arnab Sen Sharma, Aaron Mueller, Byron C Wallace, and David Bau. Function vectors in large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=AwyxtyMwaG.

[16] Herbert Jaeger. Controlling recurrent neural networks by conceptors. *arXiv*, abs/1403.3369, 2017. URL https://arxiv.org/abs/1403.3369.

[17] Teun van der Weij, Massimo Poesio, and Nandi Schoots. Extending activation steering to broad skills and multiple behaviours, 2024. URL https://arxiv.org/abs/2403.05767.

[18] Ole Jorgensen, Dylan Cope, Nandi Schoots, and Murray Shanahan. Improving activation steering in language models with mean-centring. *arXiv*, abs/2312.03813, 2023. URL https://arxiv.org/abs/2312.03813.

[19] Owen He. *Continual lifelong learning in neural systems: overcoming catastrophic forgetting and transferring knowledge for future learning*. PhD thesis, University of Groningen, 2023.

[20] Li S. Yifei, Lyle Ungar, and João Sedoc. Conceptor-aided debiasing of large language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=M6BJfQ9oup.

[21] Jesper Kuiper. Using conceptors to extract abstraction hierarchies from corpora of natural text: Combatting word polysemy using word sense disambiguation techniques. Master's thesis / essay, University of Groningen, Groningen, Netherlands, January 2024.

[22] Paul Bricman. Nested state clouds: Distilling knowledge graphs from contextual embeddings. Bachelor's Project Thesis, University of Groningen, Supervisors: Prof. Dr. Herbert Jaeger, Dr. Jacolien van Rij-Tange, July 2022. URL https://fse.studenttheses.ub.rug.nl/27840/.

[23] Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, and Stefanie Tellex. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

A Appendix

A.1 Experimental Details

All experiments were run on NVIDIA GPUs. The GPT-NeoX model was run on one NVIDIA RTX A6000 with 48GB of VRAM, and the GPT-J model was run on one NVIDIA GeForce RTX 4090 with 24GB of VRAM. Each hyperparameter sweep took less than 18 hours of compute time per model and per task.

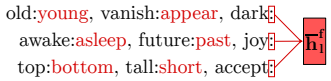
A.1.1 Function Steering

All the experimental configurations (number of experiments, number of ICL prompts and examples per prompt, accuracy metric, etc.) were, unless mentioned otherwise, adopted from Ref. [15] to ensure comparability of results.

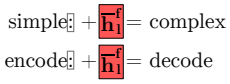
For each experiment, to generate the 4 steering mechanisms, we first compile $N_p = 100$ (ICL) prompts that demonstrate the respective input-output function. The prompts are formed by randomly sampling $N = 10$ input-output pairs from the function pairs dataset. If for a specific function, the dataset contains less than $N_p \times N = 1000$ input-output examples, this sampling is done with replacement. For each prompt p_i^f, the last input-output pair has the output stripped, resulting in the format:

$$p_i^f = "x_1 : y_1, x_2 : y_2, ..., x_{N-1} : y_{N-1}, x_N : "$$

where x represents the input tokens of a randomly sampled (input, output) pair, y represents the corresponding output tokens, N represents the number of sampled input-output pairs, and $i \in \{1, \ldots, N_p\}$. A very simple example where $N_p = 3$ and $N = 3$ can be seen in Figure 6a.



(a) Extraction of the antonym function (steering) vector $\bar{h}_\ell^f$ at layer l using 3 ICL prompts.



(b) Antonym steering vector in 2 zero-shot contexts.

Figure 6: Visualization of how an antonym function (steering) vector can be extracted and applied. Example from [15]

.

Formally, for each function $f \in F$ in our set of in-context learning (ICL) tasks, we have compiled a set P_f of ICL prompts $p_i^f \in P_f$. Each prompt p_i^f is a sequence of tokens with N input-output exemplar pairs (x, y) that demonstrate the function f mapping between x and y. For each experiment, we generate N_p such prompts.

Now that the ICL prompts have been generated, we need to extract the relevant activations. Todd *et al.* [15] showed that the neural representations of the functions are encoded in the activation vector of the last token (":") of the prompt, right before the transformer would auto-regressively start generating the output token(s). Moreover, the point in the residual stream h at which the functions were most strongly encoded was shown to be at the beginning of layers $L = \{9, \ldots, 16\}$, right before MHA and FFN [15].

Formally, for each function $f \in F$ and each prompt $p_i^f \in P_f$, the activation vectors $h_\ell^f(p_i^f)$ are extracted from the residual stream h at each relevant layer $l \in L$ from the last token's (":") activation vector.

For each function $f \in F$ and each layer $l \in L$, we now have N_p cached activation vectors $h_\ell^f(p_i^f)$ aimed to encode the neural representation of f at layer l. Using this, we can generate the layer-specific steering mechanisms for each function as follows:

- The standard additive steering mechanism $\bar{h}_\ell^f$ is generated by averaging over all the cached activation vectors $h_\ell^f(p_i^f)$ respectively as described in Equation 1.

- The additive steering mechanism with mean-centering $\bar{h}_\ell^{f,\text{mc}}$ is computed by taking the previously generated steering mechanism $\bar{h}_\ell^f$ and subtracting μ_{train} as described in Equation 11.
- The regular conceptor steering mechanism C is computed as described in Equation 3 using the aperture value α_{reg}. The correlation matrix R is computed as $R = \frac{X^T X}{N_p}$, where X is the matrix of all $h_\ell^f(p_i^f)$ stacked activation vectors.
- The mean-centered conceptor steering mechanism C^{mc} is computed with some minor adjustments. The matrix X is formed by subtracting μ_{train} from the activation vectors $h_\ell^f(p_i^f)$ before stacking them. This results in an adjusted correlation matrix R:

  $$R = \frac{(X - \mu_{\text{train}})^T (X - \mu_{\text{train}})}{N_p}$$

  The mean-centered conceptor matrix C^{mc} can then be calculated as described in Equation 3 using the aperture value α_{mc} and the adjusted correlation matrix R.

To test the performance of the generated steering mechanisms, new sets of $N_t = 1000$ input-output pairs are randomly sampled from the function pairs dataset for each experiment. This is done with replacement for functions where the dataset contains less than N_t pairs. An input prompt p_t is formatted as $p_t = "x :"$, where x is a tokenized input from an input-output pair. The tokenized output y from the pair is left out from p_t as it will be used to test the accuracy of the steering mechanisms. For each experiment, we now have N_t test input prompts p_t.

To test the accuracy of the steering mechanisms, we apply the layer-specific steering mechanisms on independent forward passes and record their subsequent output. This means that for our experimental configuration, across the functions $f \in F$, the 5 experiments, the 4 steering mechanisms (excluding the baseline), the N_t number of test prompts, and the number of layers $l \in L$, there will be $6 \times 5 \times 4 \times 1000 \times 8 = 960,000$ forward passes, each with a steering intervention.

Each steering intervention will consist of a layer-specific steering mechanism modifying the residual stream h at the mechanisms' respective layer l. This modification can be defined as transforming the unmodified residual stream activation vector h_ℓ into the steered activation vector h'_ℓ. The steering mechanisms' modification can be described as follows:

- For the standard additive steering mechanism, the averaged activation vector $\bar{h}_\ell^f$ is multiplied by the injection coefficient β_{add} and added to the residual stream activation vector h_ℓ:

  $$h'_\ell = \beta_{\text{add}}\, \bar{h}_\ell^f + h_\ell$$

- For the additive steering mechanism with mean-centering, the mean-centered average activation vector $\bar{h}_\ell^{f,\text{mc}}$ is multiplied by the injection coefficient β_{add} and added to the residual stream activation vector h_ℓ:

  $$h'_\ell = \beta_{\text{add}}\, \bar{h}_\ell^{f,\text{mc}} + h_\ell$$

- For the regular conceptor steering mechanism, the residual stream activation vector h_ℓ is multiplied using the conceptor matrix C and further multiplied with the rescaling coefficient β_{c}:

  $$h'_\ell = \beta_{\text{c}}\, C\, h_\ell$$

- For the mean-centered conceptor steering mechanism, the residual stream activation vector h_ℓ is first adjusted by subtracting μ_{train}. This adjusted vector is then multiplied with the mean-centered conceptor matrix C^{mc} and further multiplied with the rescaling coefficient β_{c}. Finally, μ_{train} is added back to the result:

  $$h'_\ell = \beta_{\text{c}}\, C^{\text{mc}}\, (h_\ell - \mu_{\text{train}}) + \mu_{\text{train}}$$

- For the baseline condition, no modifications are made to the residual stream.

  $$h'_\ell = h_\ell$$

After the respective modifications have been made to the residual stream, the forward passes will continue as usual. At the end of each forward pass, the final logits are converted into probabilities using a softmax, and the token with the highest probability is selected. This means that at the end of one experiment, we have N_t single-token outputs for each layer-specific steering mechanism. These tokens can now be compared with the first token of output y that corresponds with the input x of the initial prompt p_t. Based on how many of the N_t outputs were correctly identified, a top-1 accuracy is calculated for each layer-specific steering mechanism. This experiment is repeated 5 times for each function $f \in F$ to account for variability caused by the random sampling for the generation of the steering mechanisms and test sets.

A.1.2 Hyperparameter optimization

The performance of the steering mechanisms in the function vector experiments was optimized through a grid search over all hyperparameters. Firstly, we try steering at each layer of the model. For conceptor-based steering, we do a grid search for the aperture value α with possible values from $\{0.001, 0.0125, 0.05, 0.1\}$ and the scaling coefficient β_c with possible values from $\{0.5, 1.0, 2.0, 3.0, 4.0, 5.0\}$. For additive steering, we run a grid search over the scaling coefficient β_{add} with possible values from $\{0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0\}$. The results from these hyperparameter sweeps are shown in Appendix A.3

A.2 Additional Experimental Results

A.2.1 Mean centering

An important improvement for additive steering is a technique called *mean-centering*, put forward by Jorgensen *et al.* [18]. This method enhances the effectiveness of steering vectors by reducing the inherent bias present in the activation space of LLMs. Activation vectors in LLMs tend to be anisotropic, meaning that they are not evenly distributed around the origin, but are instead offset in a consistent direction. This can negatively impact the steering vector's performance as the bias vector b representing this offset, does not encode any specific task-related information, diluting the steering vector's effectiveness.

First, the steering vector $\bar{h}_\ell^f$ for a specific function f is computed by averaging the activations at layer ℓ on a set of ICL prompts demonstrating the input-output function P_f (as defined in Equation 1).

$\bar{h}_\ell^f$ now encodes the task-specific behavior but may still be affected by biases in the model's overall activation space. Mean-centering attempts to mitigate this by subtracting the mean activation of a broader dataset that represents the general activation space of the model. This is done by computing the mean activation vector μ_{train} over a large, representative set of prompts D_{train} from the model's training data.

The mean activation vector μ_{train} was calculated using the same procedure described by Jorgensen *et al.* [18]: A subset from the dataset used to train GPT-2 was compiled [23]. The subset was constructed by storing all entries from the folders `urlsf_subset01-1/data` and `urlsf_subset01-182/data`. After this, only entries that contained less than 500 tokens (using the GPT-2 Tokenizer) were retained. This resulted in 210 entries from which the final 10 were removed, leaving a dataset of 200 entries. The mean activation vector μ_{train} was then computed by averaging the activations over this dataset.

Implementing the mean-centering performance enhancement for steering toward the execution of functions can be done as follows:

$$\bar{h}_\ell^{f,\text{mc}} = \bar{h}_\ell^f - \mu_{\text{train}} \quad \text{with} \quad \mu_{\text{train}} = \frac{1}{|D_{\text{train}}|} \sum_{d \in D_{\text{train}}} h_\ell(d) \tag{11}$$

where $\bar{h}_\ell^f$ is as described in Equation 1, and D_{train} is the dataset for which the mean-centered vector μ_{train} is computed. This refinement leads to a steering vector that can more effectively guide the model toward the specific task and has been shown to have a positive impact on the overall steering effectiveness [18].

A.3 Hyperparameter Sweep Results

In the following section, we present results from the hyperparameter optimization described in Appendix A.1.2, in order to assess the sensitivity of both steering mechanisms (additive and conceptor-based) to the hyperparameters.

A.3.1 Conceptor Steering

Figure 7 shows that the optimal choice of aperture and beta parameters for the conceptor steering mechanism is constant at $\alpha = 0.05$ and $\beta_C = 2.0$ across all tasks for the GPT-J model (for the layer with the maximum performance). Figure 8 shows similar behavior for the GPT-NeoX model, although the optimal beta parameter is $\beta = 1$ and the optimal aperture parameter changes to $\alpha = 0.0125$ for the country-capital task, and $\alpha = 0.1$ for the english-french task, and $\alpha = 0.05$ for all other tasks. This shows that hyperparameter choices are robust for conceptor steering, but still benefit from task-specific and model-specific optimization.

We further show the performance of conceptor-based steering across all layers and different beta values (taking the best-performing aperture value) for the GPT-J model in Figure 9 and for the GPT-NeoX model in Figure 10. For the GPT-J model, the best-performing layers are typically layers 12-14 with some variability (present-past being a few layers later at 14-17, and capitalize working well across layers 9-19). For the GPT-NeoX model, conceptor steering reaches (near-)maximum performance at layer 15 across all tasks, with layer 15 being at around one third of the depth of the model. Figures 11 and 12 show the performance of conceptor-based steering across all layers and different aperture values (taking the best-performing beta value) for the GPT-J model and the GPT-NeoX model, respectively, and show a similar pattern as described above.

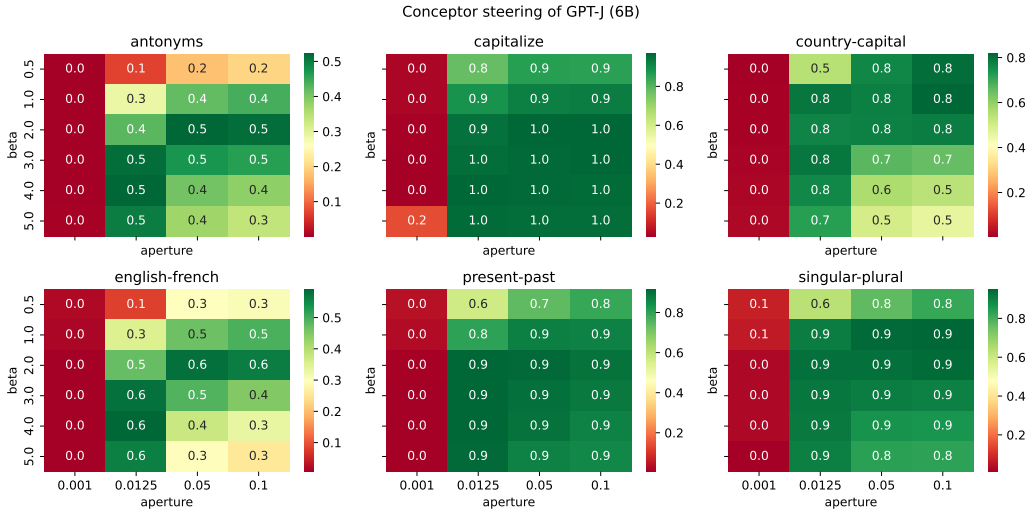


Figure 7: Performance results of the grid search across aperture and beta values (for the optimal layer) for the GPT-J (6B) model, using conceptor-based steering.

A.3.2 Additive Steering

Additive steering only has two hyperparameters that were being optimized: the layer on which steering was done, and the beta value that determines the "steering strength". Figure 13 shows the performance of additive steering on the GPT-J model across all layers and beta values. Similarly to the results of conceptor-based steering, additive steering works best across layers 9-14 with peak performance always between layers 12-14. The best-performing beta values are 2.0, 3.0, and 4.0, although 2.0 is sufficient to reach peak performance for all tasks. Figure 14 shows the performance of additive steering on the GPT-NeoX model across all layers and beta values. Similar to the best-performing conceptor-based steering hyperparameters, additive steering works best on layers 12-16. The optimal beta values are 1.5 and 2.0.

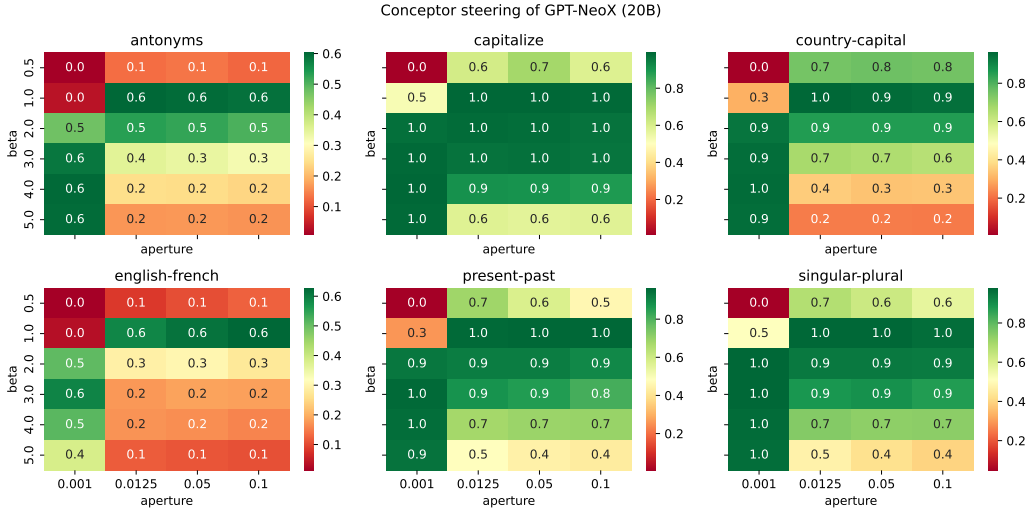


Figure 8: Performance results of the grid search across aperture and beta values (for the optimal layer) for the GPT-NeoX (20B) model, using conceptor-based steering.

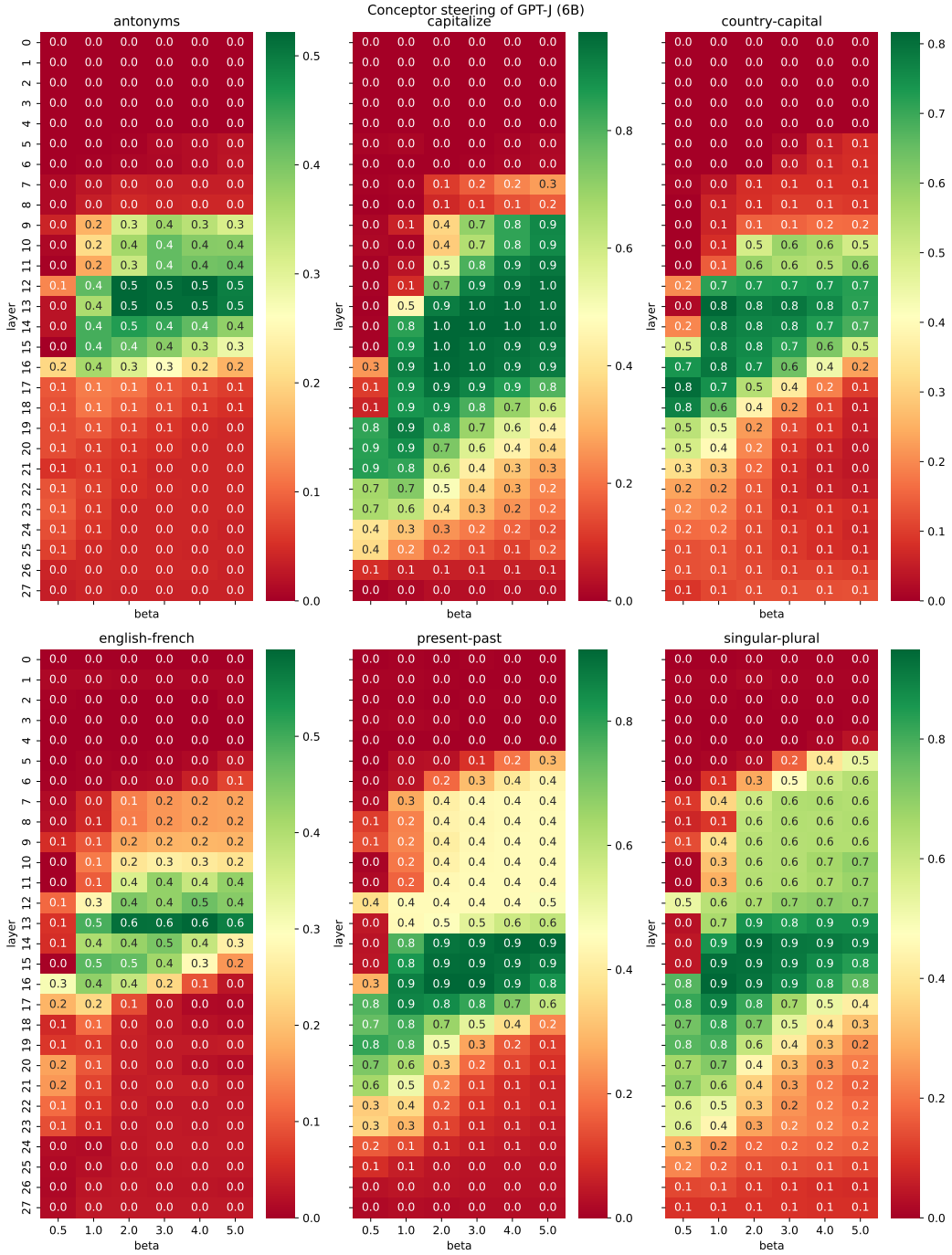


Figure 9: Performance results of the grid search across layers and beta values (for the optimal aperture value) for the GPT-J (6B) model, using conceptor-based steering.

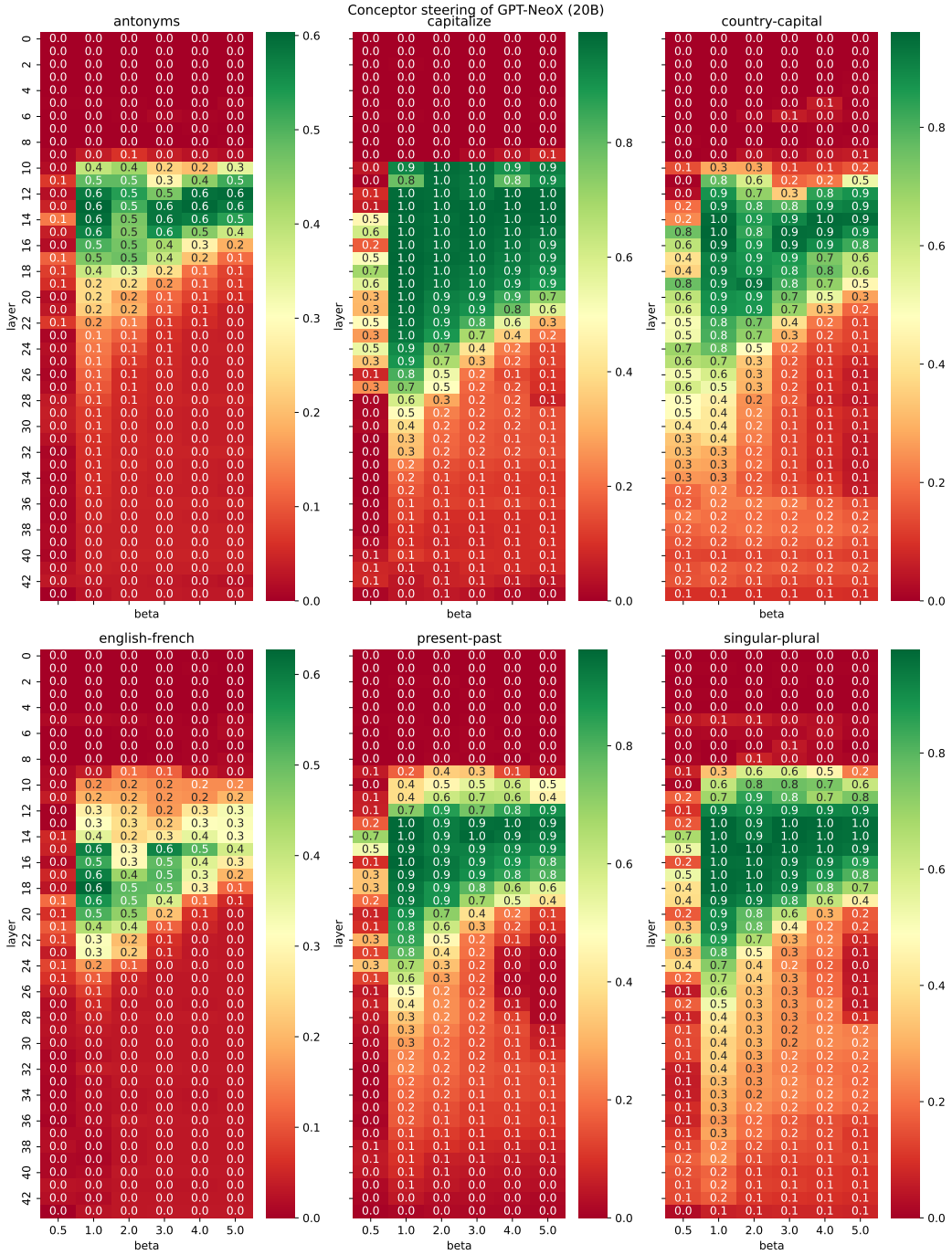


Figure 10: Performance results of the grid search across layers and beta values (for the optimal aperture value) for the GPT-NeoX (20B) model, using conceptor-based steering.

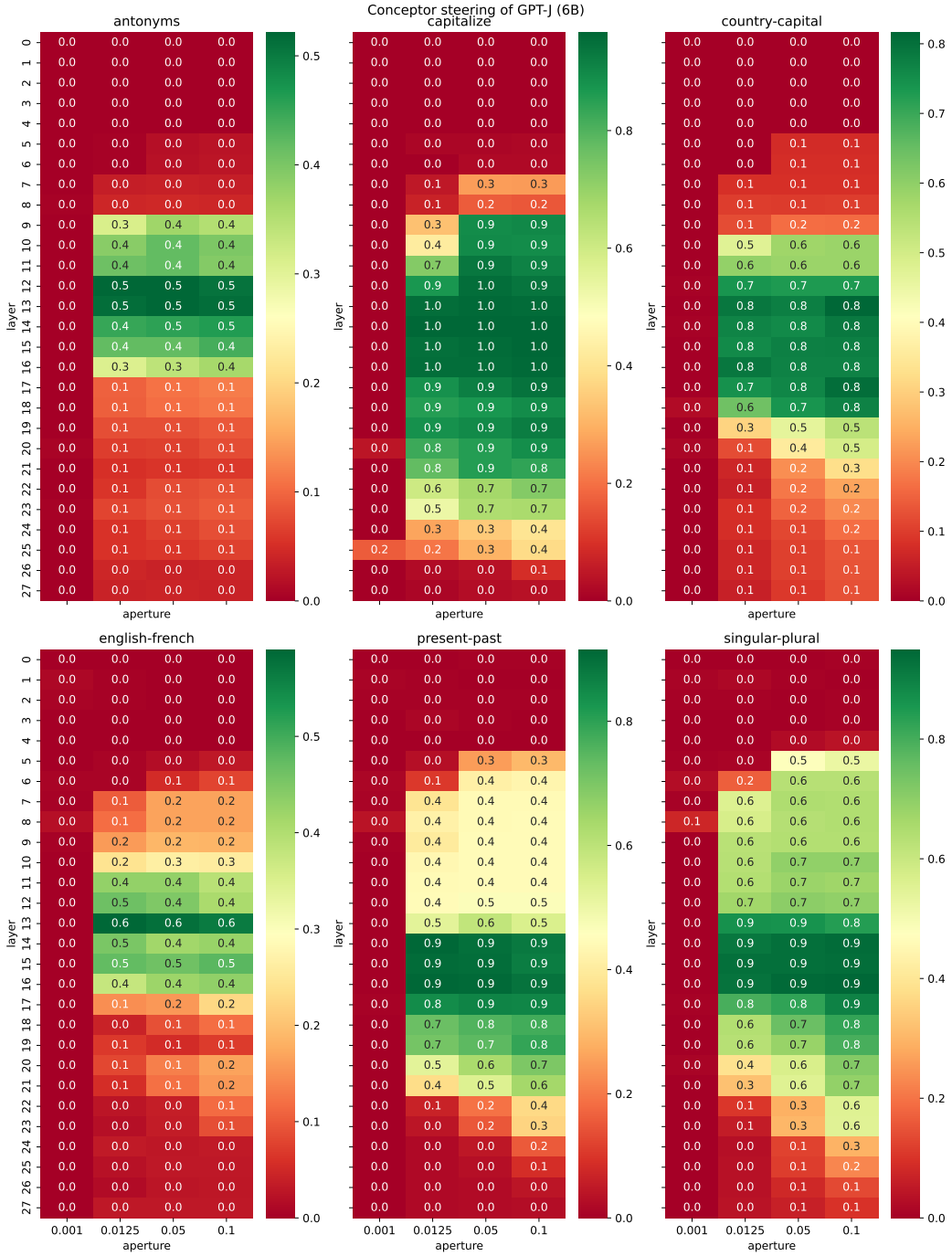


Figure 11: Performance results of the grid search across layers and aperture values (for the optimal beta value) for the GPT-J (6B) model, using conceptor-based steering.

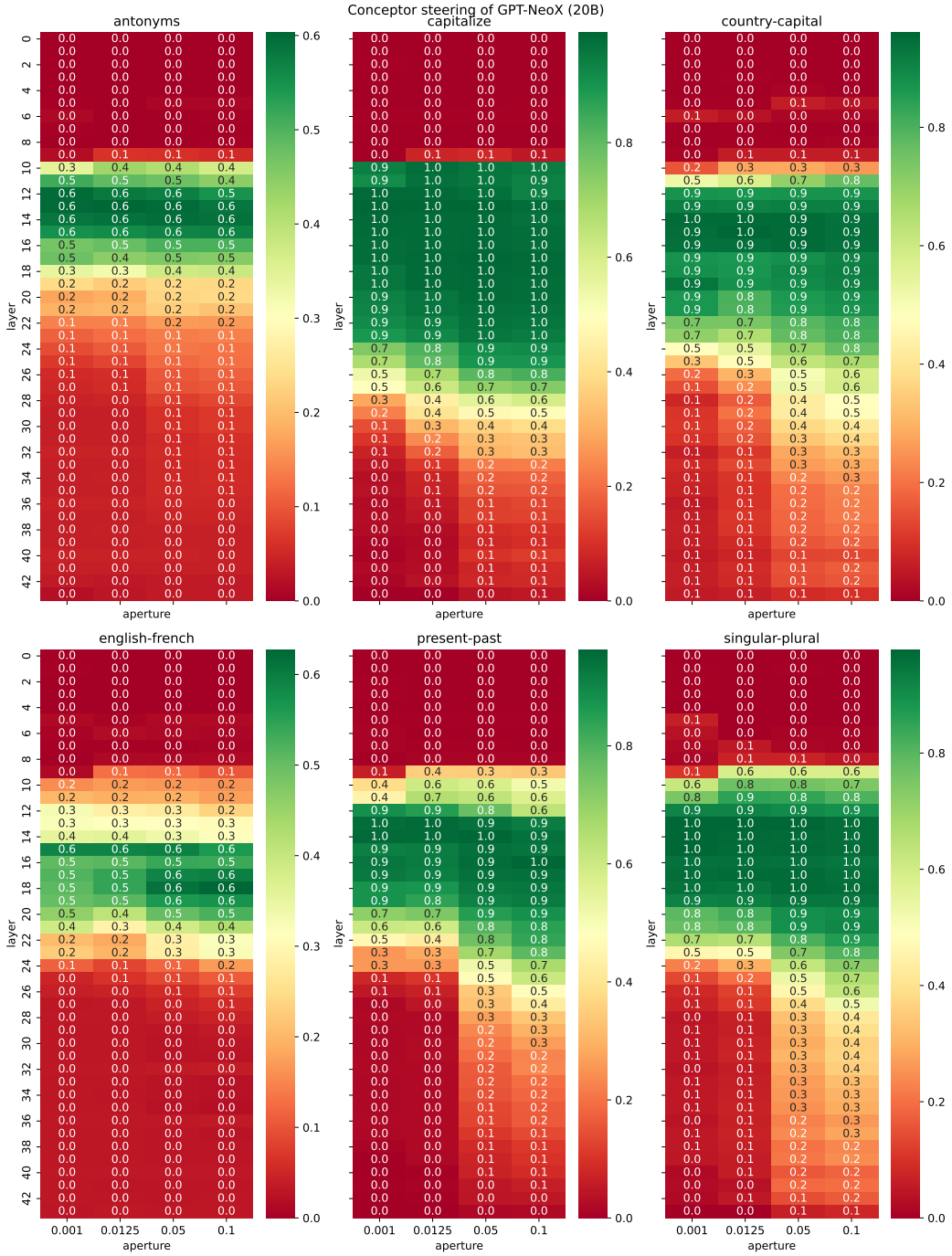


Figure 12: Performance results of the grid search across layers and aperture values (for the optimal beta value) for the GPT-NeoX (20B) model, using conceptor-based steering.

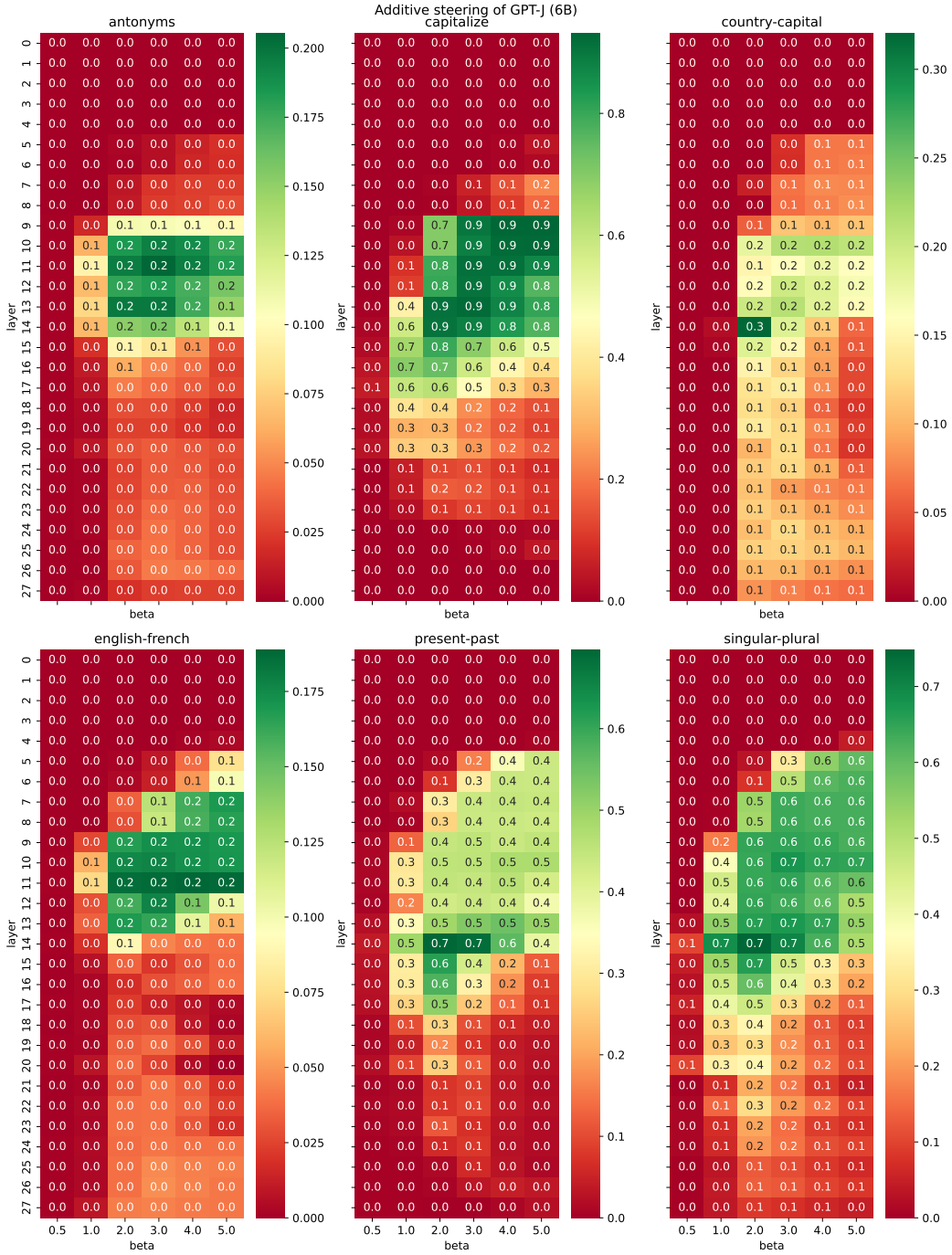


Figure 13: Performance results of the grid search across layers and beta values for the GPT-J (6B) model, using additive steering.

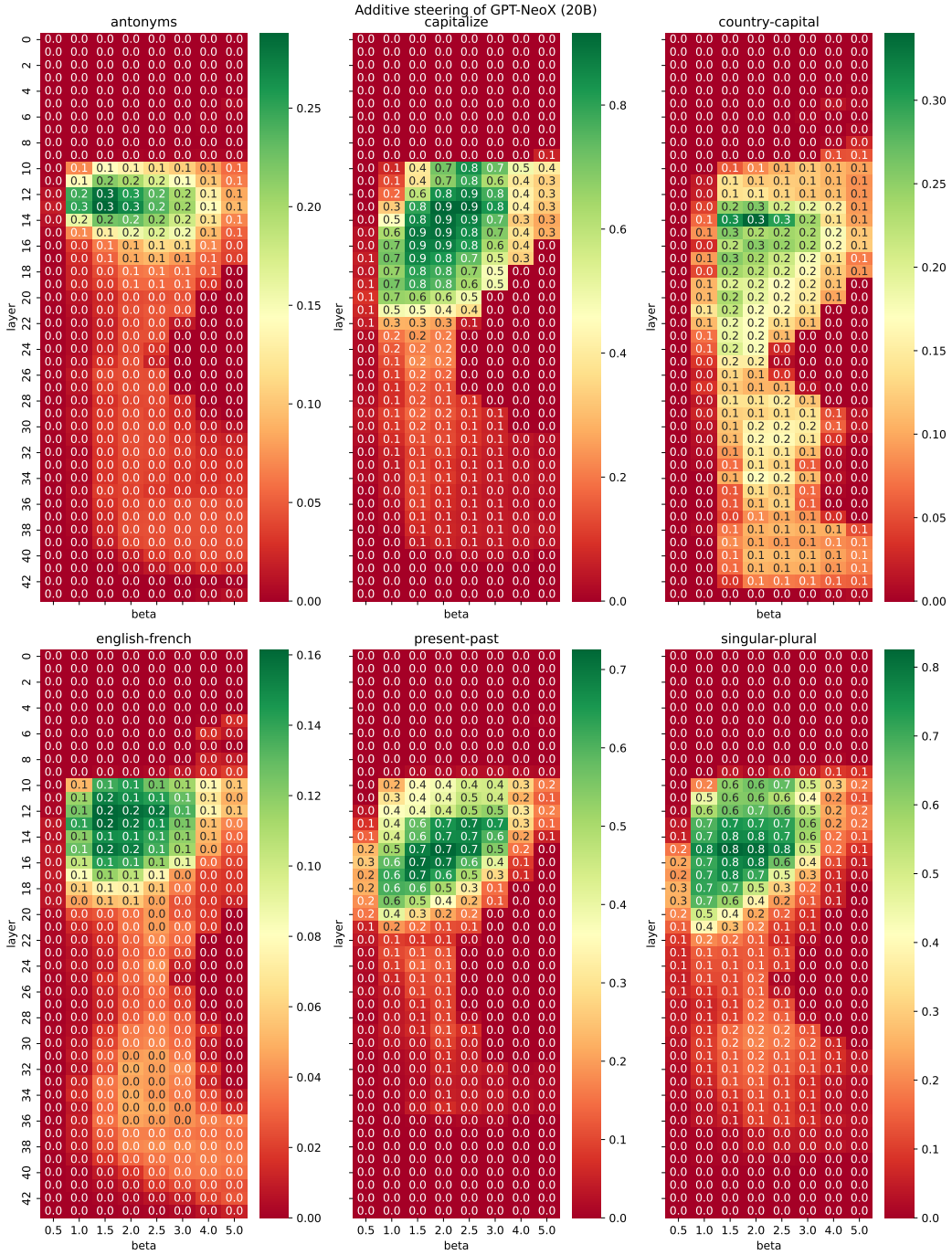


Figure 14: Performance results of the grid search across layers and beta values for the GPT-NeoX (20B) model, using additive steering.