# LM-Debugger: An Interactive Tool for Inspection and Intervention in Transformer-Based Language Models

**Mor Geva**[1] **Avi Caciularu**[2,*] **Guy Dar**[3] **Paul Roit**[2] **Shoval Sadde**[1]
**Micah Shlain**[1] **Bar Tamir**[4] **Yoav Goldberg**[1,2]

[1]Allen Institute for AI [2]Bar-Ilan University
[3]Tel Aviv University [4]The Hebrew University of Jerusalem

morp@allenai.org

## Abstract

The opaque nature and unexplained behavior of transformer-based language models (LMs) have spurred a wide interest in interpreting their predictions. However, current interpretation methods mostly focus on probing models from outside, executing behavioral tests, and analyzing salience input features, while the internal prediction construction process is largely not understood. In this work, we introduce LM-Debugger, an interactive debugger tool for transformer-based LMs, which provides a fine-grained interpretation of the model's internal prediction process, as well as a powerful framework for intervening in LM behavior. For its backbone, LM-Debugger relies on a recent method that interprets the inner token representations and their updates by the feed-forward layers in the vocabulary space. We demonstrate the utility of LM-Debugger for single-prediction debugging, by inspecting the internal disambiguation process done by GPT2. Moreover, we show how easily LM-Debugger allows to shift model behavior in a direction of the user's choice, by identifying a few vectors in the network and inducing effective interventions to the prediction process. We release LM-Debugger as an open-source tool and a demo over GPT2 models.

## 1 Introduction

Transformer-based language models (LMs) are the backbone of modern NLP models (Bommasani et al., 2021), but their internal prediction construction process is opaque. This is problematic to end-users that do not understand why the model makes specific predictions, as well as for developers who wish to debug or fix model behaviour.

Recent work (Elhage et al., 2021; Geva et al., 2022) suggested that the construction process of LM predictions can be viewed as a sequence of updates to the token representation. Specifically,

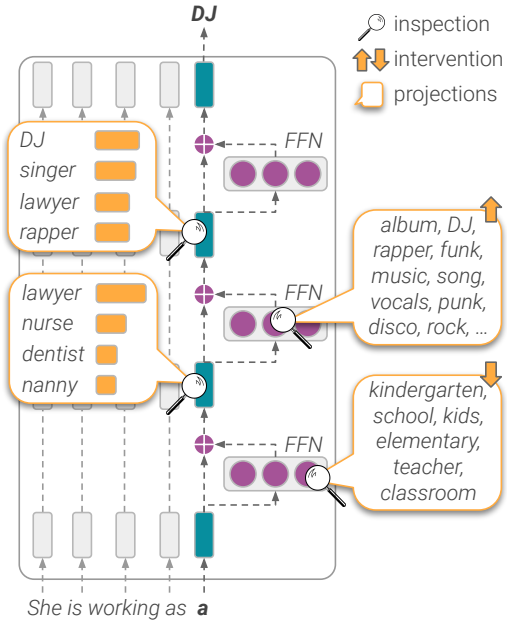


Figure 1: Illustration of the main capabilities of LM-Debugger. Our tool interprets dominant changes in the output distribution induced by the feed-forward layers across the network (self-attention layers are not shown), and enables configuring interventions for shifting the prediction in directions of the user's choice.

Geva et al. (2022) showed that updates by the feed-forward network (FFN) layers, one of the building blocks of transformers (Vaswani et al., 2017), can be decomposed into weighted collections of sub-updates, each induced by a FFN parameter vector, that can be interpreted in the vocabulary space.

In this work, we make a step towards LM transparency by employing this interpretation approach to create LM-Debugger, a powerful tool for inspection and intervention in transformer LM predictions. LM-Debugger provides three main capabilities for single-prediction debugging and model analysis (illustrated in Figure 1). First, for a given input (e.g. *"My wife is working as a"*), it interprets the model's prediction at each layer in the network, and the major changes applied to it by FFN layers. This is done by projecting the token representa-

[*] Work done during an internship at AI2.



*Proceedings of the The 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 12 - 21

tion before and after the FFN update as well as the major FFN sub-updates at any layer to the output vocabulary. Second, it allows intervening in the prediction by changing the weights of specific sub-updates, e.g. increasing (decreasing) a sub-update that promotes music-related (teaching-related) concepts, which results in a modified output. Last, for a given LM, LM-Debugger interprets all the FFN parameter vectors across the network and creates a search index over the tokens they promote. This allows an input-independent analysis of the concepts encoded by the model's FFN layers, and enables configuring general and effective interventions.

We demonstrate the utility of LM-Debugger for two general use-cases. In the context of prediction debugging, we use the fine-grained tracing of LM-Debugger to inspect the internal disambiguation process performed by the model. Furthermore, we demonstrate how our tool can be used to configure a few powerful interventions that effectively control different aspects in text generation.

We release LM-Debugger as an open-source tool at https://github.com/mega002/lm-debugger and host a demo of GPT2 (Brown et al., 2020) at https://lm-debugger.apps.allenai.org.[1] This to increase the transparency of transformer LMs and facilitate research in analyzing and controlling NLP models.

## 2 Underlying Interpretation Method

LM-Debugger establishes a framework for interpreting a token's representation and updates applied to it at each layer in the network. This framework builds upon recent findings by Geva et al. (2022), who viewed the token representation as a changing distribution over the output vocabulary, and the output from each FFN layer as a collection of weighted sub-updates to that distribution, which are often interpretable to humans. We next elaborate on the findings we rely on at this work.

Consider a transformer LM with $L$ layers and an embedding matrix $E \in \mathbb{R}^{d \times |\mathcal{V}|}$ of hidden dimension $d$, over a vocabulary $\mathcal{V}$. Let $\mathbf{w} = w_1, ..., w_t$ s.t. $\forall i = 1, ..., t: \ w_i \in \mathcal{V}$ be an input sequence of tokens, then at each layer $\ell = 1, ..., L$, the hidden representation $\mathbf{x}_i^\ell$ of the $i$-th token is being processed and updated by a FFN layer through a residual connection (He et al., 2016):[2]

$$\tilde{\mathbf{x}}_i^\ell = \mathbf{x}_i^\ell + \mathsf{FFN}^\ell(\mathbf{x}_i^\ell),$$

where $\mathbf{x}_i^\ell$ is the output from the preceding multi-head self-attention layer, and $\tilde{\mathbf{x}}_i^\ell$ is the updated token representation (Vaswani et al., 2017). Geva et al. (2022) proposed an interpretation method for these updates in terms of the vocabulary, which we employ as the backbone of LM-Debugger and describe in detail next.

**Token Representation as a Distribution Over the Output Vocabulary.** The token representation before ($\mathbf{x}_i^\ell$) and after ($\tilde{\mathbf{x}}_i^\ell$) the FFN update at any layer $\ell$ is interpreted by projecting it to the vocabulary space and converting it to a distribution:

$$\mathbf{p}_i^\ell = \text{softmax}(E\mathbf{x}_i^\ell) \ \ ; \ \ \tilde{\mathbf{p}}_i^\ell = \text{softmax}(E\tilde{\mathbf{x}}_i^\ell)$$

The final model output is defined by $\mathbf{y} = \tilde{\mathbf{p}}_i^L$.

**The FFN Output as a Weighted Collection of Sub-Updates.** Each FFN layer is defined with two parameter matrices $K^\ell, V^\ell \in \mathbb{R}^{d_m \times d}$, where $d_m$ is the intermediate hidden dimension, and a non-linearity function $f$ (bias terms are omitted):

$$\mathsf{FFN}^\ell(\mathbf{x}^\ell) = f\left(K^\ell \mathbf{x}^\ell\right) V^\ell \tag{1}$$

Geva et al. (2022) interpreted the FFN output by (a) decomposing it into sub-updates, each induced by a single FFN parameter vector, and (b) projecting each sub-update to the vocabulary space. Formally, Eq. 1 can be decomposed as:

$$\mathsf{FFN}^\ell(\mathbf{x}^\ell) = \sum_{i=1}^{d_m} f(\mathbf{x}^\ell \cdot \mathbf{k}_i^\ell)\mathbf{v}_i^\ell = \sum_{i=1}^{d_m} m_i^\ell \mathbf{v}_i^\ell.$$

where $\mathbf{k}_i^\ell$ is the $i$-th row of $K^\ell$, $\mathbf{v}_i^\ell$ is the $i$-th column of $V^\ell$, and $m_i^\ell := f(\mathbf{x}^\ell \cdot \mathbf{k}_i^\ell)$ is the activation coefficient of $\mathbf{v}_i^\ell$ for the given input. Each term in this sum is interpreted as a sub-update to the output distribution, by inspecting the top-scoring tokens in its projection to the vocabulary, i.e. $E\mathbf{v}_i^\ell$.

In the rest of the paper, we follow Geva et al. (2022) and refer to columns of $V^\ell$ as *"value vectors"* and to their weighted input-dependent form as *"sub-updates"*. Importantly, value vectors are *static* parameter vectors that are independent on the input sequence, while sub-updates are *dynamic* as they are weighted by input-dependent coefficients. For a model with $L$ layers and a hidden dimension $d_m$, there are $L * d_m$ static value vectors, which induce $L * d_m$ corresponding sub-updates when running an input through the model.

[1] See a video at https://youtu.be/5D_GiJv7O-M

[2] Layer normalization is omitted (Geva et al., 2022).

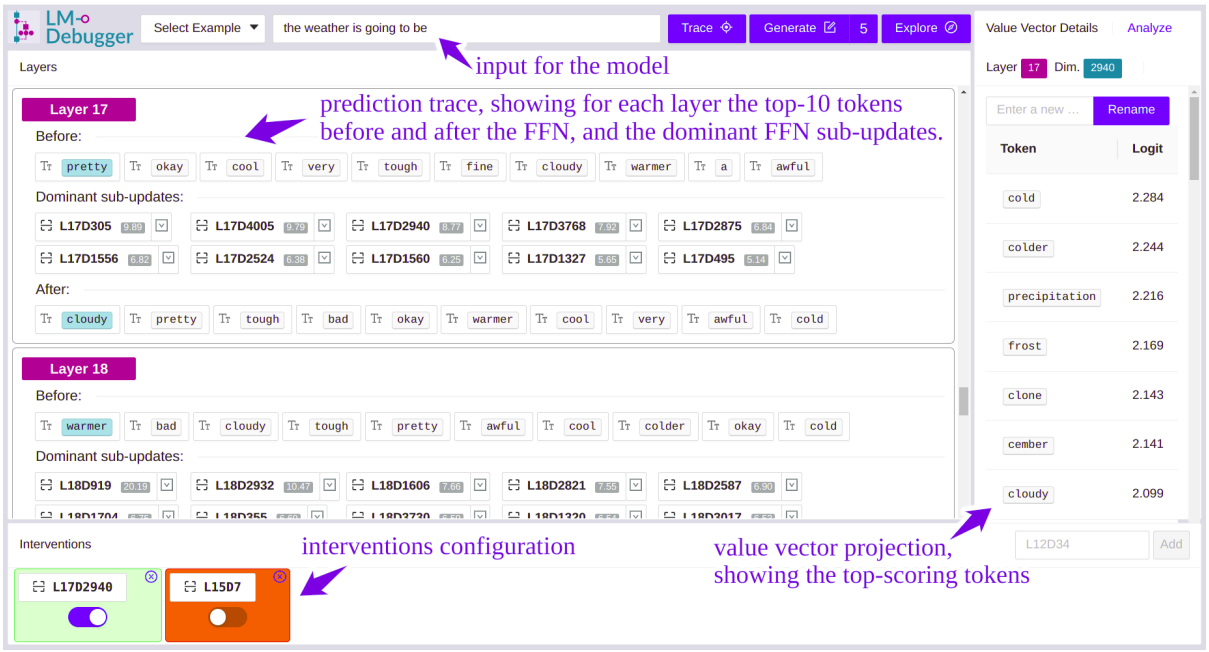


Figure 2: The prediction view of LM-Debugger, showing the prediction trace for a given input (main panel), allowing to configure interventions (lower panel) and interpret sub-updates to the output distribution (right panel).

## 3 LM-Debugger

LM-Debugger leverages both static and dynamic analysis of transformer FFN layers and the updates they induce to the output distribution for debugging and intervention in LM predictions. These capabilities are provided in two main views, which we describe next.

### 3.1 Prediction View

This view, shown in Figure 2, is designed for per-example debugging. It allows running inputs through the model to generate text in an auto-regressive manner, while tracing the dominant sub-updates in every layer and applying interventions.

**Prediction Trace** (Figure 2, main panel). The user enters an input for the model, for which a detailed trace of the prediction across the network is provided. For each layer, it shows the top-tokens in the output distribution, before and after the FFN update, and the 10 most dominant FFN sub-updates. For every sub-update $m_i \mathbf{v}_i^\ell$ we show an identifier $\texttt{L}[\ell]\texttt{D}[i]$ of its corresponding value vector and the coefficient for the given input (e.g. L17D4005 and 9.79).[3] The top distribution tokens and sub-updates are sorted by the token probability/sub-update coefficient from left (highest) to right (lowest). A small arrow next to each sub-update allows setting an intervention on its corresponding value vector.

[3]The layer and dimension in the identifier use zero-index.

**Interventions** (Figure 2, lower panel). Beyond tracing the output distribution, LM-Debugger also allows intervening in the prediction process by setting the coefficients of *any vector values in the network*, thus, inducing sub-updates of the user's choice. To set an intervention for a specific value vector, the user should enter its identifier to the panel and choose whether to "turn it on or off", that is, setting its coefficient to the value of the coefficient of the most dominant sub-update in that layer, or to zero, respectively. When running an input example, all interventions in the panel will be effective, for the entire generation process.

**Value Vector Information** (Figure 2, right panel). A natural question that arises is how to choose meaningful interventions. LM-Debugger provides two complementary approaches for this. A bottom-up approach is to observe the dominant sub-updates for specific examples, and apply interventions on them. A sub-update can be interpreted by inspecting the top-tokens in the projection of its corresponding value vector to the vocabulary (Geva et al., 2022). For convenience, we let the user assign names to value vectors. Another way to find meaningful interventions is by a top-down approach of searching for value vectors that express concepts of the user's interest. We provide this capability in the exploration view of LM-Debugger, which is described next.

### 3.2 Exploration View

This view allows static exploration of value vectors, primarily for analyzing which concepts are encoded in the FFN layers, how concepts are spread over different layers, and identifying groups of related value vectors.

**Keyword Search** (Figure 3). Value vectors are interpreted by the top tokens they promote. By considering these sets of tokens as textual documents, LM-Debugger allows searching for concepts encoded in value vectors across the layers. This is enabled by a search index that LM-Debugger holds in the background, which stores the projections of all value vectors to the vocabulary, and allows executing simple queries against them using the BM25 (Robertson et al., 1995) algorithm.

**Cluster Visualization** (Figure 4). Assuming the user is interested in locating a specific concept in the network and that she has found a relevant value vector, either from debugging an example in the prediction view or by the keyword search. A natural next step is to find similar value vectors that promote related tokens. To this end, LM-Debugger provides a clustering of all value vectors in the network, which allows mapping any value vector to a cluster of similar vectors in the hidden space (Geva et al., 2022). The interface displays a random sample of vectors from the cluster, as well as an aggregation of their top tokens as a word cloud, showing the concepts promoted by the cluster.

## 4 Debugging LM Predictions by Tracing FFN Updates

In this section, we demonstrate the utility of LM-Debugger for interpreting model behaviour upon a given example. As an instructive example, we will consider the case of sense disambiguation.

When generating text, LMs often need to perform sense disambiguation and decide on one plausible continuation. For example, the word *"for"* in the input *"The book is for"* has two plausible senses of *purpose* (e.g. *"reading"*) and *person* (e.g. *"him"*) (Karidi et al., 2021). We will now inspect the prediction by GPT2 (Brown et al., 2020) and track the internal sense disambiguation process for this example. To this end, we enter the input in the prediction view and click **Trace**, which provides a full trace of the prediction across layers.

Table 1 displays a part of this trace from selected layers, showing a gradual transition from *purpose* to *person* sense. Until layer 11 (out of 24), the top-tokens in the output distribution are mostly related to sale/example purposes. Starting from layer 12, the prediction slowly shifts to revolve about the audience of the book, e.g. `anyone` and `ages`, until layer 18 where `sale` is eliminated from the top position. In the last layers, tokens become more specific, e.g. `beginners` and `adults`.

| |
|---|
| Layer: 4 Sense: *purpose*<br>Before: `example, the, instance, purposes`<br>After: `example, the, instance, all` |
| Layer: 10 Sense: *purpose*<br>Before: `the, sale, example, a`<br>After: `the, sale, a, example` |
| Layer: 15 Sense: *purpose/person*<br>Before: `sale, the, anyone, use`<br>After: `sale, anyone, the, ages` |
| Layer: 20 Sense: *person*<br>Before: `beginners, anyone, adults, sale`<br>After: `anyone, beginners, adults, readers` |

Table 1: Partial prediction trace of GPT2 for the input *"This book is for"*, showing the internal disambiguation process from *purpose* to *person* sense across layers.

To examine the major updates through which the prediction has formed, we can click on specific sub-updates in the trace to inspect the top-scoring tokens in their projections. We observe that in early layers, tokens are often related to *purpose* sense (e.g. `instance` in `L2D1855` and `buyers` in `L12D659`), in intermediate layers tokens are a mix of both senses (`readers` in `L16D3026` and `preschool` in `L17D2454`, and `sale/free` in `L16D1662`), and mostly *person* sense in the last layers (`users` in `L18D685`, `people` in `L20D3643`, and `those` in `L21D2007`).

## 5 Configuring Effective Interventions for Controlled Text Generation

Beyond interpretability, LM-Debugger enables to *intervene* in LM predictions. We show this by finding value vectors that promote specific concepts and applying simple and effective interventions.

**Controlling Occupation Prediction.** Consider the input *"My wife is working as a"*. When running it through GPT2, the final prediction from the last layer has the top tokens `nurse, teacher, waitress`. We would like to intervene in the prediction in order to change its focus to occupations related to software engineering, which in general are less associated with women (De-Arteaga et al., 2019). To this end, we will use the exploration

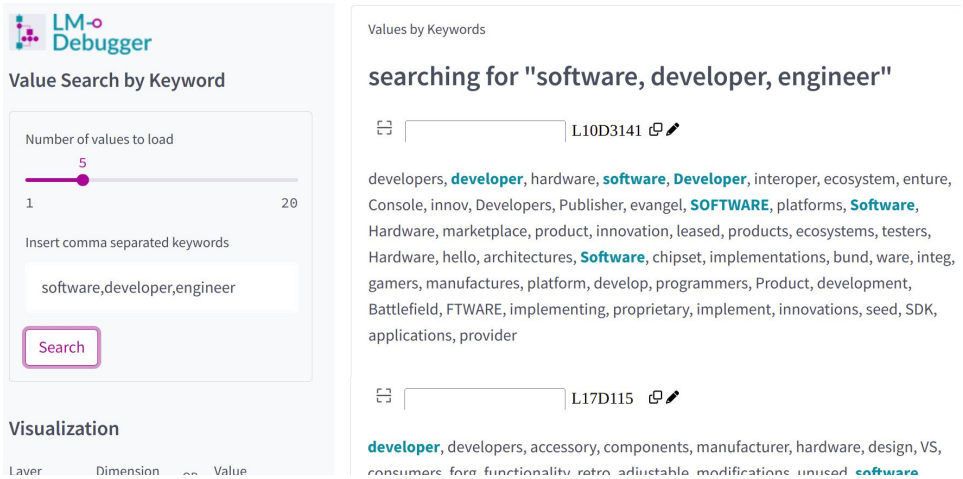


Figure 3: Keyword search in the exploration view of `LM-Debugger`, which matches user queries against the tokens promoted by value vectors of the model.

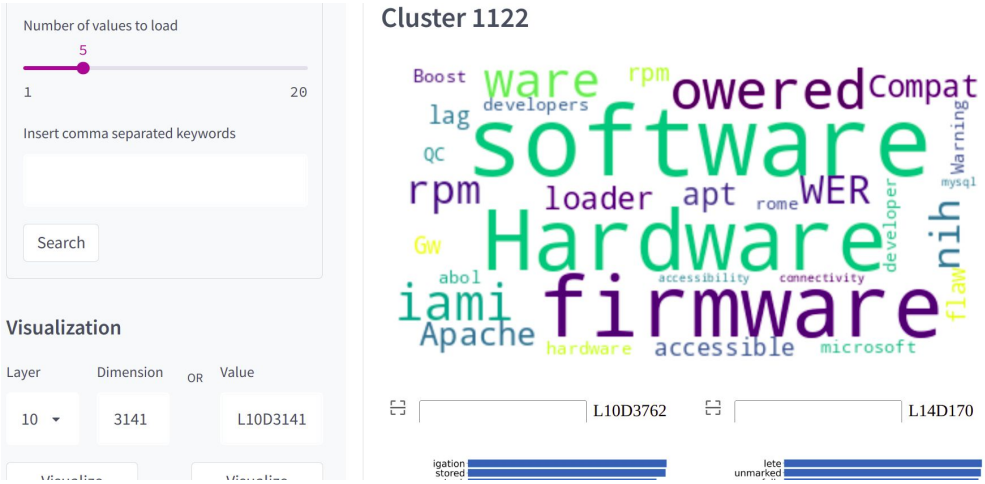


Figure 4: Cluster visualization in the exploration view of `LM-Debugger`, which maps a given value vector to its cluster of similar value vectors in the network.

view of `LM-Debugger` to search for value vectors promoting software-related concepts.

Searching the keywords *"software"*, *"developer"*, and *"engineer"* brings up two value vectors with coherent concepts: `L10D3141` and `L17D115` (Figure 3). Now, we will add these value vectors to the intervention panel in the prediction view, and run the example again. Our intervention, that only involved two (0.002%) vectors in the network, dramatically changed the prediction to `software`, `programmer`, `consultant`, `developer`, effectively shifting it in the direction we wanted. This demonstrates the power of `LM-Debugger` to change model behaviour and fix undesirable predictions.

**Controlling the Sentiment of Generated Text.** The previous example focused on next-token prediction. We now take this one step further and configure powerful and general interventions that influence various texts generated by the model. For our experimental setting, we will attempt to control the sentiment in generated reviews by GPT2, for inputs taken from the Yelp dataset (Asghar, 2016).

We choose our interventions independently of the inputs, with two easy steps. First, we use the keyword search (Figure 3) to identify "seed" value vectors that promote positive and negative adjectives/adverbs, using the queries *"terrible, mediocre, boring"* and *"spacious, superb, delicious"*. Then, we take one value vector for each polarity and, using the cluster visualization (Figure 4), expand it to a diverse set of vectors from its corresponding cluster, that promote similar concepts. Overall, we select 5-6 value vectors for each polarity (details in Appendix A.1), to which we apply interventions.

Table 2 presents the texts generated by GPT2 (each limited to 10 tokens) for multiple inputs, with and without applying interventions. Clearly, across

| Input | Interven. | Continuation |
|---|---|---|
| *"Service in this place is"* | - | a bit of a mess. I'm not sure |
| | ↑ Positive | a good place to make the right efforts to make |
| | ↑ Negative | a waste of a bunch of crap that is too |
| *"I have been to this restaurant twice and"* | - | both times I was disappointed. The first time I |
| | ↑ Positive | have been served excellent food and good service. The |
| | ↑ Negative | have been disappointed. The food is over processed and |
| *"We went on a weeknight. Place was"* | - | packed. We had to wait for the bus |
| | ↑ Positive | good, good food, good staff, good people |
| | ↑ Negative | too far for us to get lost. We were |
| *"Went for breakfast on 6/16/14. We"* | - | had a great time. We had a great time |
| | ↑ Positive | have a good team of people who are able to |
| | ↑ Negative | were too heavy for the wrong type of food that |

Table 2: Continuations (limited to 10 tokens) generated by GPT2 for different inputs from the Yelp dataset, with and without interventions for "turning on" sub-updates for positive and negative sentiment.

all the examples, our intervention in the prediction successfully leads to the desired effect, turning the sentiment of the generated text to be positive or negative, according to the configured sub-updates.

## 6 Implementation Details

The prediction view is implemented as a React web application with a backend Flask server that runs an API for executing models from the Transformers library by HuggingFace (Wolf et al., 2020). The exploration view is a Streamlit web application, which (a) sends user search queries to an Elasticsearch index with the top tokens of all vector value projections, and (b) visualize clusters of value vectors created with the scikit-learn package (Pedregosa et al., 2011). Our current implementation supports any GPT2 model from HuggingFace, and other auto-regressive models can be plugged-in with only a few local modifications (e.g. translating the relevant layer names). More details and instructions for how to deploy and run `LM-Debugger` are provided at `https://github.com/mega002/lm-debugger`.

## 7 Related Work

Interpreting single-predictions and the general behavior of LMs is a growing research area that attracted immense attention in recent years (Belinkov et al., 2020; Choudhary et al., 2022). `LM-Debugger` is a the first tool to interpret and intervene in the prediction construction process of transformer-based LMs based on FFN updates.

Existing interpretation and analysis frameworks mostly rely on methods for behavioral analysis (Ribeiro et al., 2020) by probing models with adversarial (Wallace et al., 2019b) or counterfactual examples (Tenney et al., 2020), input saliency methods that assign importance scores to input features (Wallace et al., 2019b; Tenney et al., 2020), and analysis of the attention layers (Hoover et al., 2020; Vig and Belinkov, 2019).

More related to `LM-Debugger`, other tools analyze patterns in neuron activations (Rethmeier et al., 2020; Dalvi et al., 2019; Alammar, 2021). Unlike these methods, we focus on interpreting the model parameters and on intervening in their contribution to the model's prediction.

The functionality of `LM-Debugger` is mostly related to tools that trace hidden representations across layers. Similarly to `LM-Debugger`, Alammar (2021); Nostalgebraist (2020) interpret the token representation in terms of the output vocabulary. We take this one step further and interpret the FFN updates to the representation, allowing to observe not only the evolution of the representation but also the factors that induce changes in it.

Our intervention in FFN sub-updates relates to recent methods for locating and editing knowledge in the FFN layers of LMs (Meng et al., 2022; Dai et al., 2022). Different from these methods, `LM-Debugger` aims to provide a comprehensive and fine-grained interpretation of the prediction construction process across the layers.

## 8 Conclusion

We introduce `LM-Debugger`, a debugger tool for transformer-based LMs, and the first tool to analyze the FFN updates to the token representations across layers. `LM-Debugger` provides a fine-grained interpretation of single-predictions, as well as a powerful framework for intervention in LM predictions.

## Ethical Statement

Our work aims to increase the transparency of transformer-based LMs. It is well known that such models often produce offensive, harmful language (Bender et al., 2021; McGuffie and Newhouse, 2020; Gehman et al., 2020; Wallace et al., 2019a), which might originate in toxic concepts encoded in their parameters (Geva et al., 2022). `LM-Debugger`, which traces and interprets LM predictions, could expose such toxic concepts and therefore should be used with caution.

`LM-Debugger` also provides a framework for modifying LM behavior in particular directions. While our intention is to provide developers tools for fixing model errors, mitigating biases, and building trustworthy models, this capability also has the potential for abuse. In this context, it should be made clear that `LM-Debugger` does not modify the information encoded in LMs, but only changes the intensity in which this information is exposed in the model's predictions. At the same time, `LM-Debugger` lets the user observe the intensity of updates to the prediction, which could be used to identify suspicious interventions. Nonetheless, because of these concerns, we stress that LMs should not be integrated into critical systems without caution and monitoring.

## Acknowledgements

We thank the REVIZ team at the Allen Institute for AI, particularly Sam Skjonsberg and Sam Stuesser. This project has received funding from the Computer Science Scholarship granted by the Séphora Berrebi Foundation, the PBC fellowship for outstanding PhD candidates in Data Science, and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement No. 802774 (iEXTRACT).

## References

J Alammar. 2021. Ecco: An open source library for the explainability of transformer language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 249–257, Online. Association for Computational Linguistics.

Nabiha Asghar. 2016. Yelp dataset challenge: Review rating prediction. *arXiv preprint arXiv:1605.05362*.

Yonatan Belinkov, Sebastian Gehrmann, and Ellie Pavlick. 2020. Interpretability and analysis in neural NLP. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 1–5, Online. Association for Computational Linguistics.

Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the ACM Conference on Fairness, Accountability, and Transparency (FAccT)*.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, S. Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen Creel, Jared Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, O. Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir P. Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Benjamin Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, J. F. Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Robert Reich, Hongyu Ren, Frieda Rong, Yusuf H. Roohani, Camilo Ruiz, Jack Ryan, Christopher R'e, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishna Parasuram Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei A. Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. 2021. On the opportunities and risks of foundation models. *ArXiv*, abs/2108.07258.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In

*Proceedings of Neural Information Processing Systems (NeurIPS)*.

Shivani Choudhary, Niladri Chatterjee, and Subir Kumar Saha. 2022. Interpretation of black box nlp models: A survey. *arXiv preprint arXiv:2203.17081*.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. 2022. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8493–8502, Dublin, Ireland. Association for Computational Linguistics.

Fahim Dalvi, Avery Nortonsmith, Anthony Bau, Yonatan Belinkov, Hassan Sajjad, Nadir Durrani, and James Glass. 2019. NeuroX: A toolkit for analyzing individual neurons in neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):9851–9852.

Maria De-Arteaga, Alexey Romanov, Hanna Wallach, Jennifer Chayes, Christian Borgs, Alexandra Chouldechova, Sahin Geyik, Krishnaram Kenthapadi, and Adam Tauman Kalai. 2019. Bias in bios: A case study of semantic representation bias in a high-stakes setting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, FAT* '19, page 120–128, New York, NY, USA. Association for Computing Machinery.

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

Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. 2020. RealToxicityPrompts: Evaluating neural toxic degeneration in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3356–3369, Online. Association for Computational Linguistics.

Mor Geva, Avi Caciularu, Kevin Ro Wang, and Yoav Goldberg. 2022. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. *arXiv preprint arXiv:2203.14680*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the conference on computer vision and pattern recognition (CVPR)*.

Benjamin Hoover, Hendrik Strobelt, and Sebastian Gehrmann. 2020. exBERT: A Visual Analysis Tool to Explore Learned Representations in Transformer Models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 187–196, Online. Association for Computational Linguistics.

Taelin Karidi, Yichu Zhou, Nathan Schneider, Omri Abend, and Vivek Srikumar. 2021. Putting words in BERT's mouth: Navigating contextualized vector spaces with pseudowords. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10300–10313, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Kris McGuffie and Alex Newhouse. 2020. The radicalization risks of gpt-3 and advanced neural language models. *arXiv preprint arXiv:2009.06807*.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual knowledge in gpt. *arXiv preprint arXiv:2202.05262*.

Nostalgebraist. 2020. interpreting GPT: the logit lens.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Nils Rethmeier, Vageesh Kumar Saxena, and Isabelle Augenstein. 2020. Tx-ray: Quantifying and explaining model-knowledge transfer in (un-)supervised nlp. In *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 124 of *Proceedings of Machine Learning Research*, pages 440–449. PMLR.

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

Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, and Mike Gatford. 1995. et almbox. 1995. okapi at trec-3. *Nist Special Publication Sp*, 109:109.

Ian Tenney, James Wexler, Jasmijn Bastings, Tolga Bolukbasi, Andy Coenen, Sebastian Gehrmann, Ellen Jiang, Mahima Pushkarna, Carey Radebaugh, Emily Reif, and Ann Yuan. 2020. The language interpretability tool: Extensible, interactive visualizations and analysis for NLP models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 107–118, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all

you need. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5998–6008.

Jesse Vig and Yonatan Belinkov. 2019. Analyzing the structure of attention in a transformer language model. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 63–76, Florence, Italy. Association for Computational Linguistics.

Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019a. Universal adversarial triggers for attacking and analyzing NLP. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2153–2162, Hong Kong, China. Association for Computational Linguistics.

Eric Wallace, Jens Tuyls, Junlin Wang, Sanjay Subramanian, Matt Gardner, and Sameer Singh. 2019b. AllenNLP interpret: A framework for explaining predictions of NLP models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, pages 7–12, Hong Kong, China. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

# A Appendix

## A.1 Details on Interventions to Control Generated Text Sentiment

Table 3 lists all the value vectors selected for our interventions described in §5, and examples for top-scoring tokens in their projections. These vectors were found with the exploration view of `LM-Debugger` (§3.2), using both keyword search and clustering visualisation. All the interventions were configured to "turn on" these vectors, namely, setting their coefficients to be maximal for the corresponding layer. This is following the observation by Geva et al. (2022) that FFN updates operate in a token promotion mechanism (rather than elimination).

| Sentiment | Value Vector | Example Top-scoring Tokens |
|---|---|---|
| Positive | L13D1763 | properly, appropriately, adequate, truthful, humane, fulfil, inclusive, timely, patiently, sustainable |
| | L13D2011 | clean, Proper, secure, flawless, safest, graceful, smooth, calmly |
| | L14D944 | peacefully, graceful, respectful, careful, generous, patiently, calm, tolerant, fair |
| | L15D74 | Excellence, superb, trustworthy, marvelous, terrific, awesome, Amazing |
| | L20D988 | successful, optimal, perfect, satisfactory, welcome, helpful, fulfilling, healthy |
| Negative | L11D4 | outdated, inadequate, stale, lousy, dull, mediocre, boring, wasteful |
| | L14D2653 | trivial, dismiss, rigid, unsupported, only, prejud, obfusc, pretend, dispar, slander |
| | L16D974 | inappropriately, poorly, disrespect, unreliable, unhealthy, insecure, improperly, arrogance |
| | L17D3790 | inappropriate, improper, wrong, bad, harmful, unreasonable, defective, disturbance, errors |
| | L18D91 | confused, bizarre, unfairly, horrible, reckless, neglect, misplaced, strange, nasty, mistakenly |
| | L18D3981 | wrong, incorrect, insufficient, misleading, premature, improperly, unrealistic, outdated, unfair |

Table 3: Value vectors used for controlling sentiment in generated text, that promote positive and negative adjectives/adverbs. For each vector, we show example top-scoring tokens from its projection to the vocabulary, as presented in the exploration view of LM-Debugger.