LLM Interpretability

Tanmoy Chakraborty
Associate Professor, IIT Delhi
https://tanmoychak.com/

Advanced Large Language Models

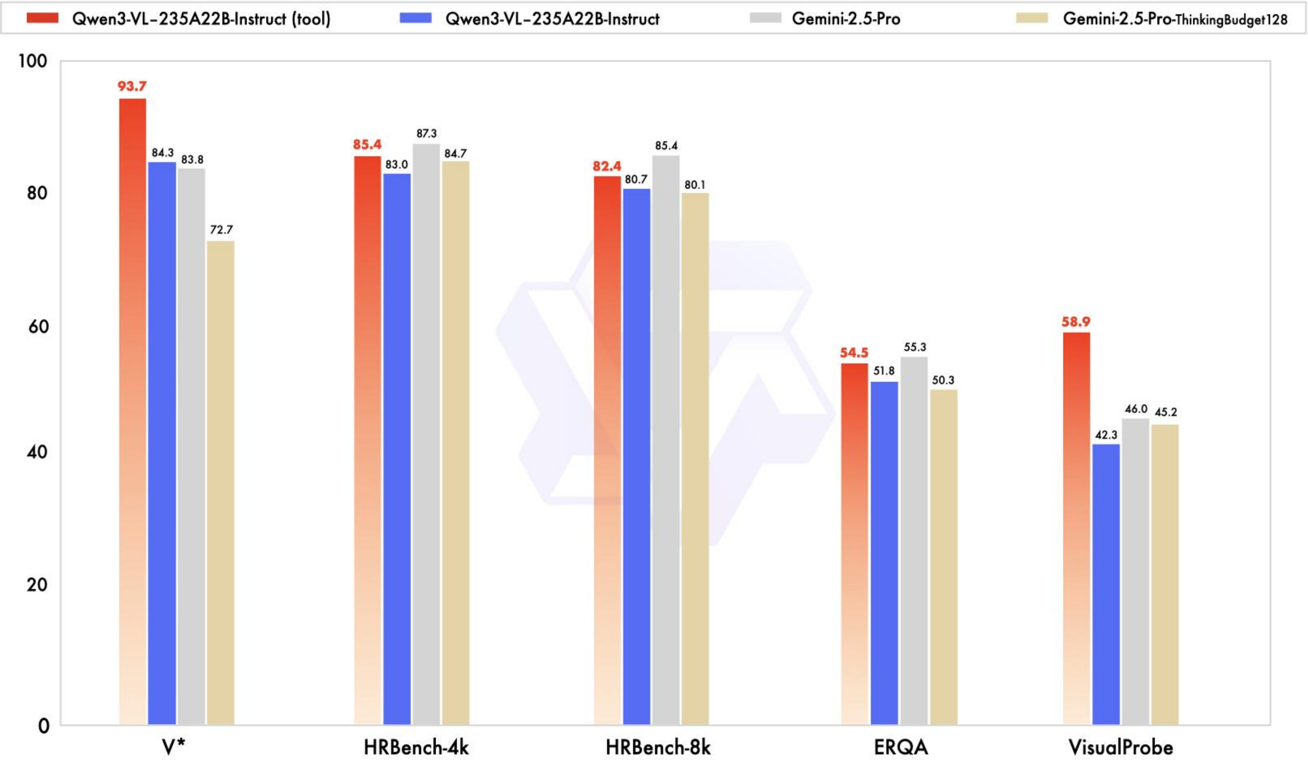
Qwen-3-VL

The most powerful vision-language model in the Qwen family

Announced on September 23, 2025

Qwen Blog

Qwen3-VL-235B-A22B offers both Instruct and Thinking versions. The Instruct version matches or even exceeds Gemini 2.5 Pro in major visual perception benchmarks. The Thinking version achieves state-of-the-art results across many multimodal reasoning benchmarks.



Qwen3-VL demonstrates significant progress across all capabilities -- from understanding and generating text, to perceiving and reasoning over visual content, handling longer contexts, interpreting spatial relationships and dynamic videos, and interacting with AI agents.

LLM Interpretability – Objectives and Motivations

- LLMs are an important component in a lot of major systems in the current technology landscape – but there is an inherent limitation - we know ***what*** they do, but not ***why.***
- Despite all the performance improvements**, LLMs remain black boxes.**
- We can benefit from a **deeper understanding of their internal mechanisms**.

Key motivations

Safety & Trust — Detecting deception, hallucination, or misuse.
Accountability — Understanding *why* a model made a harmful or biased prediction.
Debugging & Model Editing — Fixing specific failure modes without retraining.
Scientific Curiosity — Reverse-engineering intelligence from emergent representations.
Regulation & Governance — EU AI Act and audit requirements depend on interpretability.

Defining explanation/interpretation scopes

- **Transparency** — Opening the hood; understanding *how* each part contributes.
- ✓ **Explainability** — Providing human-friendly explanations of outputs.
- ✓ **Causality** — Understanding which components *cause* certain behaviors.
- **Conceptual Interpretability** — Mapping internal features to human concepts.

Challenges of Interpreting **Large** Language Models

- **Scale:** 175B+ parameters; even a single attention head may encode thousands of sub-features.
- **Distributed representations:** No single neuron = single concept (superposition).
- **Depth & composition:** 100+ layers make causal attribution hard.
- **Emergent behavior:** New capabilities appear unpredictably with scale.
- **Dynamic context usage:** In-context learning means model “weights” aren’t the full story.

Timeline

2013–2016 — CNN Visualization Era

- *Zeiler & Fergus (2014)* — Deconv nets reveal early CNN features.
- *Olah et al. (2015–16)* — Feature visualization & neuron interpretability (Distill.pub).

→ Focus: ***seeing what neurons see.***

2018–2020 — Linguistic Probing in NLP

- *Alain & Bengio (2017)* — Linear probes for representations.
- *Tenney et al. (2019)* — BERT layers mirror NLP pipeline.
- *Hewitt & Manning (2019)* — Syntax geometry in embeddings.
- *Clark et al. (2019)* — Attention heads learn linguistic roles.

→ Focus: ***what information is encoded.***

2020–2022 — Mechanistic Interpretability

- *Elhage et al. (2021, Anthropic)* — Framework for transformer circuits.
- *Nanda et al. (2022)* — Grokking analysis via circuits.
- *Wang et al. (2022, Anthropic)* — Induction head discovery.

→ Focus: ***how transformers compute.***

2023–2024 — Sparse Features & Superposition

- *Anthropic (2023)* — Sparse Autoencoders (SAE) for monosemantic features.
- *Conmy et al. (2023)* — LLMs labeling their own features.
- *Nanda (2024)* — TransformerLens tools for large-scale analysis.

→ Focus: ***features > neurons.***

2023–2025 — Causal & Editable Models

- *Meng et al. (2023, ROME)* — Factual circuit editing.
- *Hernandez et al. (2023)* — Causal scrubbing methods.
- *Zou et al. (2024)* — Steering vectors for controllable behavior.

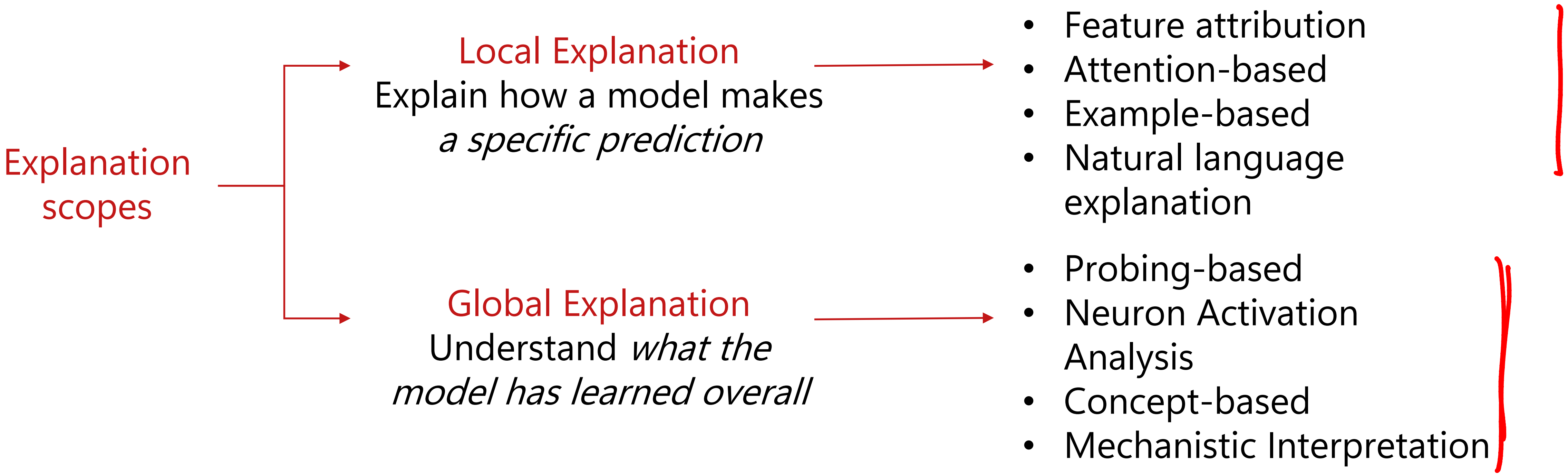
→ Focus: ***intervention & control.***

2024–2025 — Automated & Scalable Interpretability

- *Anthropic (2024)* — Automated feature labeling via LLMs.
- *SAEBench, Neuronpedia (2024–25)* — Open interpretability datasets.

→ Focus: ***AI interpreting AI.***

Broad Taxonomy



Explanation scopes

Local Explanation

Explain how a model makes *a specific prediction*

- Feature attribution
- Attention-based
- Example-based
- Natural language explanation

Global Explanation

Understand *what the model has learned overall*

- Probing-based
- Neuron Activation Analysis
- Concept-based
- Mechanistic Interpretation

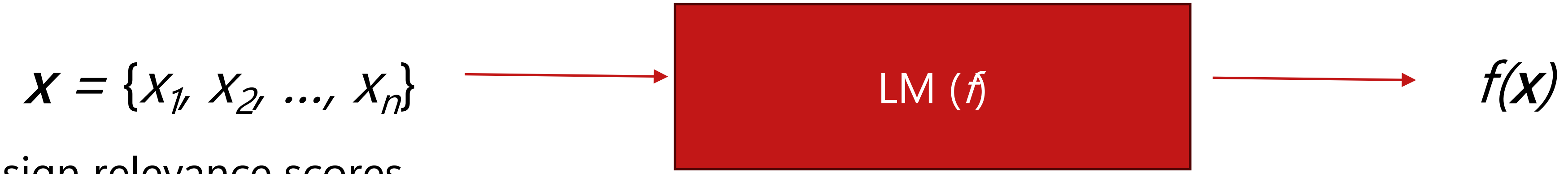
Based on Explainability for Large Language Models: A Survey

Local Explanation-based Analysis

Explain *why* the model produced a specific output

1. Feature Attribution

Compute contribution of each input token or feature to the output.



$\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$

LM (f)

$f(\boldsymbol{x})$

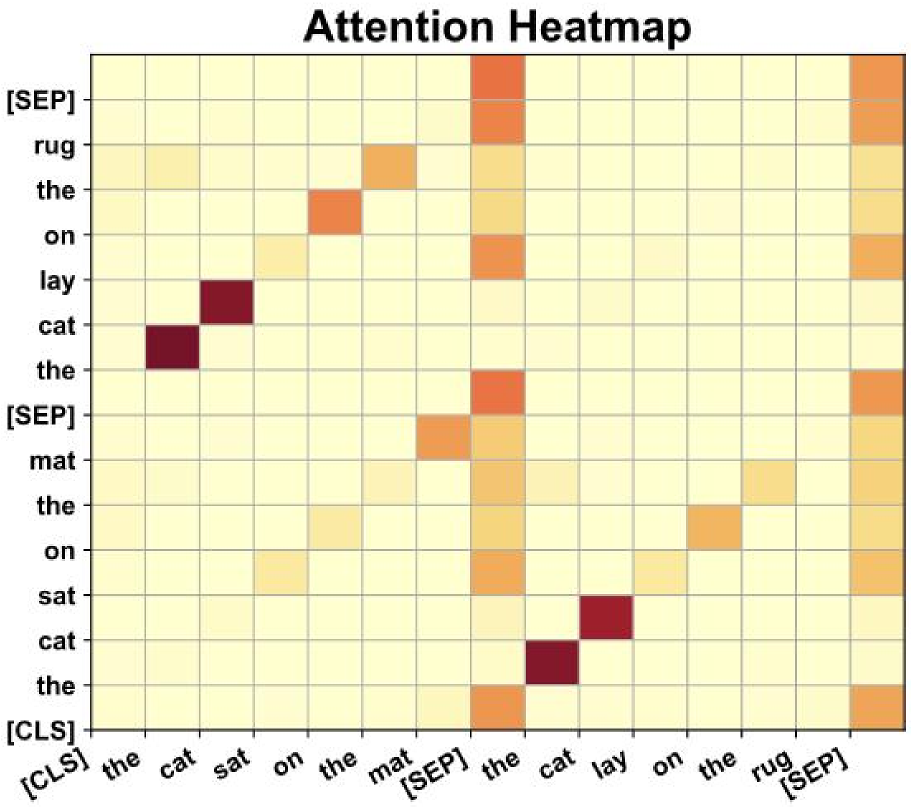
Assign relevance scores ($R(x_i)$)

Perturbation-Based	Gradient-Based	Surrogate Models	Decomposition Based
Perturb Input features, observe output, draw conclusions	Analyse the partial derivatives of the output wrt each input dimension	Use simpler, more human-comprehensible models to explain individual predictions of black-box models	Break down the relevance score into linear contributions from the input. [Layer-wise or end-to-end]
E.g. Leave-one-out	E.g. Integrated gradients (IG)	E.g. ~~SHAP, LIME~~	E.g. Layer-wise relevance propagation (LRP), Taylor-type decomposition (DTD)

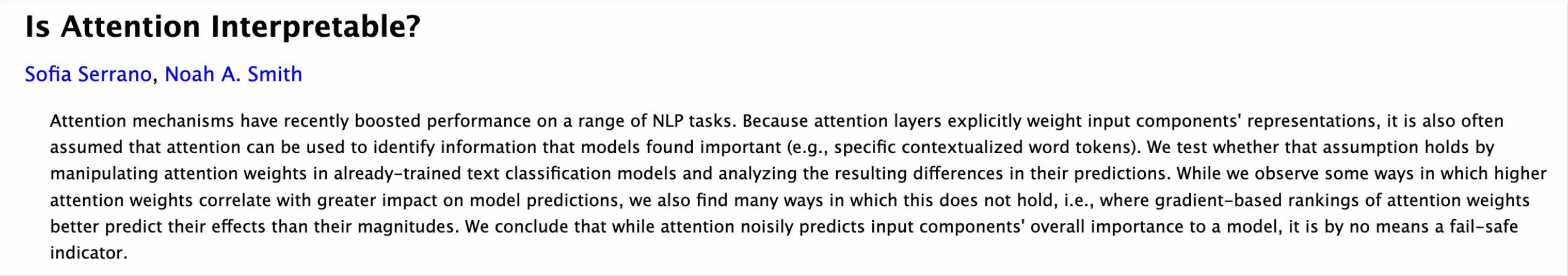
2. Attention-based Methods

Explain models solely based on the attention weights or by analyzing the knowledge encoded in the attention.

Visualizations	Function-Based	Probing Based
Visualizing attention heads for a single input using bipartite graphs or heatmaps	Attribution scores that blend attention and gradients generally perform better than using either alone, as they fuse more information.	Better utilized in global explanations (covered later)



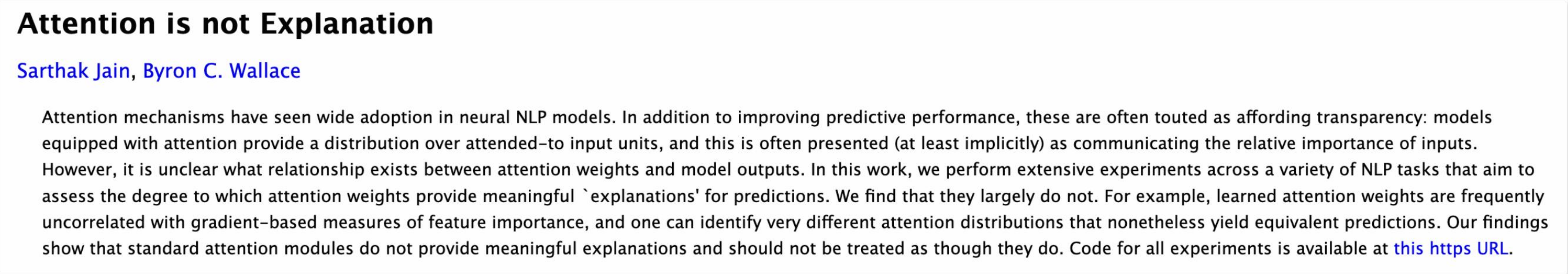
(b) Heatmap



Is Attention Interpretable?

Sofia Serrano, Noah A. Smith

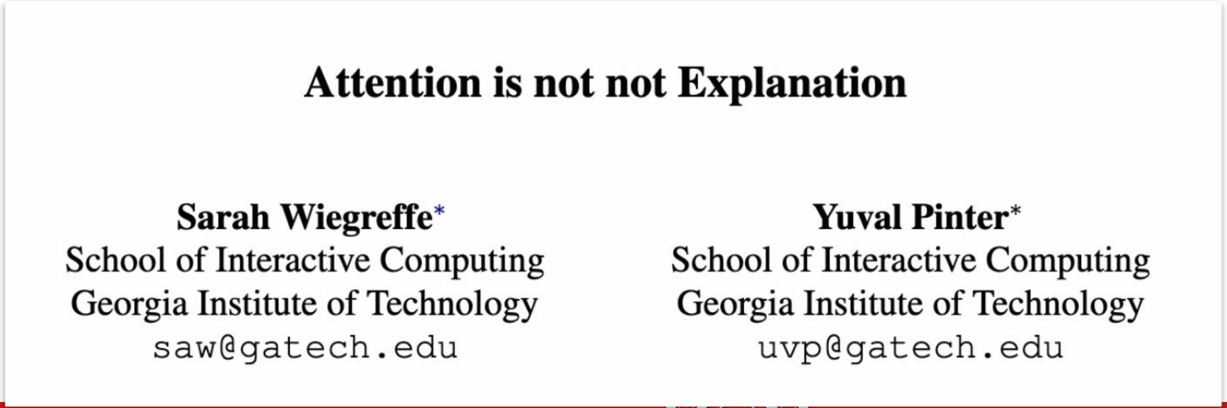
Attention mechanisms have recently boosted performance on a range of NLP tasks. Because attention layers explicitly weight input components' representations, it is also often assumed that attention can be used to identify information that models found important (e.g., specific contextualized word tokens). We test whether that assumption holds by manipulating attention weights in already-trained text classification models and analyzing the resulting differences in their predictions. While we observe some ways in which higher attention weights correlate with greater impact on model predictions, we also find many ways in which this does not hold, i.e., where gradient-based rankings of attention weights better predict their effects than their magnitudes. We conclude that while attention noisily predicts input components' overall importance to a model, it is by no means a fail-safe indicator.



Attention is not Explanation

Sarthak Jain, Byron C. Wallace

Attention mechanisms have seen wide adoption in neural NLP models. In addition to improving predictive performance, these are often touted as affording transparency: models equipped with attention provide a distribution over attended-to input units, and this is often presented (at least implicitly) as communicating the relative importance of inputs. However, it is unclear what relationship exists between attention weights and model outputs. In this work, we perform extensive experiments across a variety of NLP tasks that aim to assess the degree to which attention weights provide meaningful `explanations' for predictions. We find that they largely do not. For example, learned attention weights are frequently uncorrelated with gradient-based measures of feature importance, and one can identify very different attention distributions that nonetheless yield equivalent predictions. Our findings show that standard attention modules do not provide meaningful explanations and should not be treated as though they do. Code for all experiments is available at this https URL.



Attention is not not Explanation

Sarah Wiegreffe*
School of Interactive Computing
Georgia Institute of Technology
saw@gatech.edu

Yuval Pinter*
School of Interactive Computing
Georgia Institute of Technology
uvp@gatech.edu

3. Example based explanations

Identify **influential training or in-context examples**, by analysing changes in output when input is changed strategically.

Adversarial Examples	Counterfactual explanations	Data Influence
Carefully crafted modifications to the input can alter model decisions while barely being noticeable to humans.	Generated data point that is as close to the input data point as possible but for which the model gives a different outcome.	Characterizes the influence of individual training sample by measuring how much they affect the loss on test points.
For example, Wei et al. show that simply asking an LLM to begin its response with "Absolutely! Here's..." could mislead the model into complying with a harmful request.	For example, if a user was denied a loan by a machine learning model, an example counterfactual explanation could be: "Had your income been $5000 greater per year and your credit score been 30 points higher, your loan would be approved." [https://arxiv.org/pdf/1905.07857]	Kwon et al. propose DataInf https://arxiv.org/pdf/2310.00902

4. Natural Language Explanation

- **Explaining a model's decision making on an input sequence with generated text.**
- The basic approach for generating natural language explanations involves training a language model using both original textual data and human-annotated explanations.
- The trained language model can then automatically generate explanations in natural language.
- As explanations provide additional contextual space, they can improve downstream prediction accuracy and perform as a data augmentation technique.

Global Explanation based Analysis

Reveal the *latent structure* of an LLM — its internal representations, circuits, and conceptual organization — instead of just explaining one decision.

Methods

- Linear Probing
- Sparse Autoencoder-based decomposition
- Logit Lens

Classifier-based Probing

$\underline{\underline{x}}^{i}, \underline{\underline{y}}^{i}$

z^{i}

$f_l(x$

Original model:

$$f : x \mapsto \hat{y}, \quad \mathcal{D}_O = \{(x^{(i)}, y^{(i)})\}, \quad \text{PERF}(f, \mathcal{D}_O)$$

Intermediate representations at layer l:

$$f_l(x)$$

The probing classifier is trained and evaluated on its own annotated dataset: $\mathcal{D}_P = \{(x^{(i)}, z^{(i)})\}$

Then we measure the classifier performance: $\text{PERF}(g, f, \mathcal{D}_O, \mathcal{D}_P)$

From an information-theoretic view, training the probe estimates the mutual information between the representations and the property, *I(z,h),* where *z* is a random variable over linguistic properties and *h* represents the intermediate representations *f_l(x)*.

Ref: Probing Classifiers: Promises, Shortcomings, and Advances

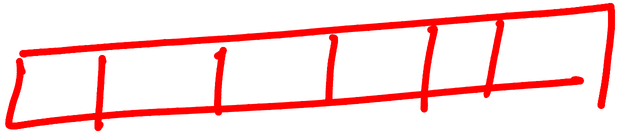
Classifier-based Probing - Limitation

- A main limitation of the probing classifier paradigm is the **disconnect between the probing classifier g and the original model f.**
- The probing framework may indicate **correlations** between representations $f_l(x)$ and linguistic property z, but it does not tell us whether this property is involved in **predictions** of f.
- In essence, correlations (with external properties) discovered by the probes are not appropriately indicative of the causality in the predictions of the original model.

Correlation ✓ Causation ✗

Neuron and Activation-level explanation

- Instead of examining the whole vector space, neuron analysis looks into individual dimensions, i.e., neurons in representations, that are crucial for model performance or associated with specific linguistic properties.



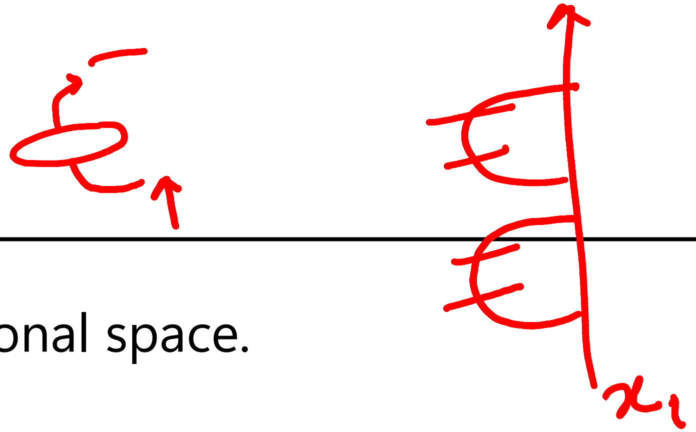
Concept-level Explanations

- Concept-based interpretability algorithms **map the inputs to a set of concepts** and measure **importance score of each pre-defined concept to model predictions**.
- By introducing abstract concepts, models can be explained in a human-understandable fashion rather than on low-level features. Information in latent space can also be transformed into comprehensible explanations.

Mechanistic Interpretability

- The field of study of **reverse engineering neural networks** from the learned weights down to **human-interpretable algorithms**. Analogous to reverse engineering a compiled program binary back to source code.
- Exposes ways to start controlling outputs of a model.

Key Concepts



- **Representation -** Every token → vector in a high-dimensional space.
  - contains linguistic, factual, and conceptual structure.
- **Feature -** a property of an input to the model, or some subset of that input (e.g., a token in the prompt given to a language model, or a patch of an image)
- **Concept -** human-meaningful abstraction — like "negation," "politeness," "gender" that can be represented as a direction or cluster in activation space.
- **Neuron -** A single scalar output in a neural network layer — the result of a linear transformation followed by a nonlinearity.
- **Residual stream** - The *residual stream* is the main communication channel that runs through all transformer layers, carrying forward the sum of previous activations.
  - $x_{l+1} = x_l + \text{Attention}(x_l) + \text{MLP}(x_l)$
  - Each layer *adds or edits* information without erasing what came before.
- **Circuit -** a causal subgraph of neurons, heads, or layers that collectively implement a specific computation.

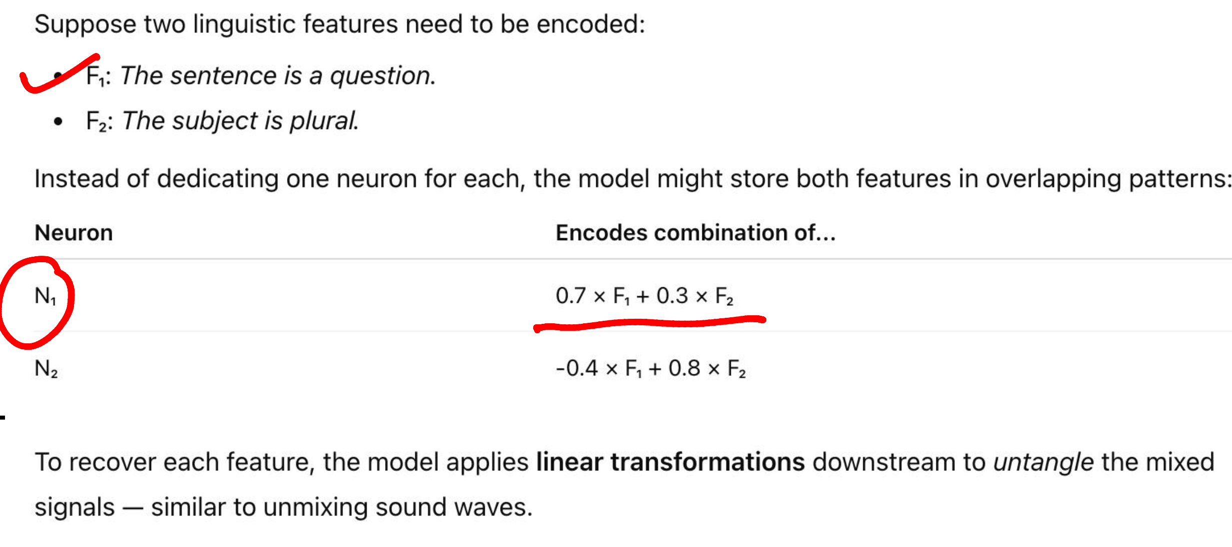
Key Concepts

- **Linear Representation hypothesis** - features of the input can be represented as directions in activation space.
  - **Decomposability**: Network representations can be described in terms of independently understandable features.
  - **Linearity**: Features are represented by direction.

- To reverse engineer neural networks - we need decomposability.
- We also need to be able to 'access' the decomposition somehow.
  - Identify the individual features within a representation. In a **linear representation**, this corresponds to determining **which directions in activation space correspond to which independent features of the input**.

Key Concepts

- **Superposition**
  - Linear representations can represent more features than dimensions, using a strategy we call superposition. This can be seen as neural networks simulating larger networks.
  - This pushes features away from corresponding to neurons.



Suppose two linguistic features need to be encoded:

- F_1: *The sentence is a question.*
- F_2: *The subject is plural.*

Instead of dedicating one neuron for each, the model might store both features in overlapping patterns:

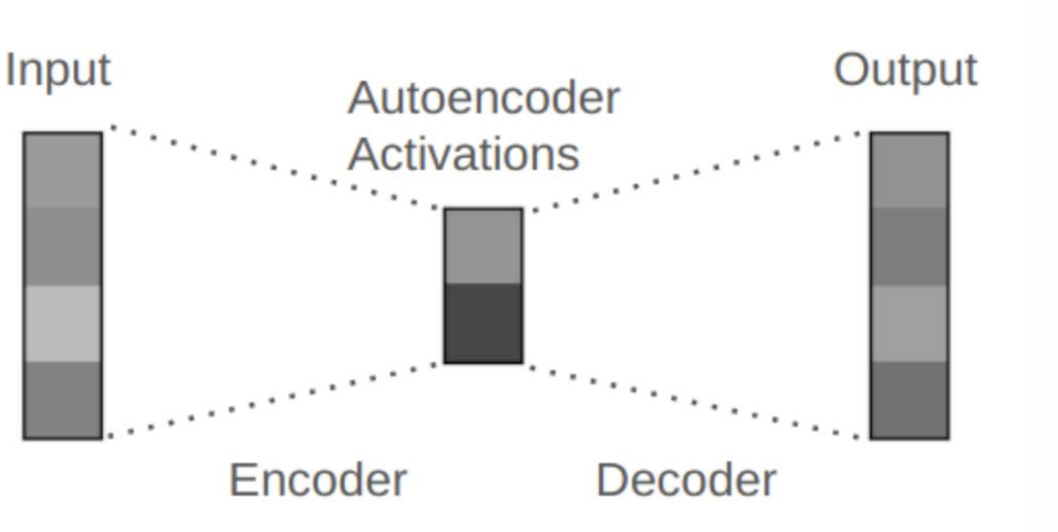
Neuron	Encodes combination of...
N_1	$0.7 \times F_1 + 0.3 \times F_2$
N_2	$-0.4 \times F_1 + 0.8 \times F_2$

To recover each feature, the model applies **linear transformations** downstream to *untangle* the mixed signals — similar to unmixing sound waves.

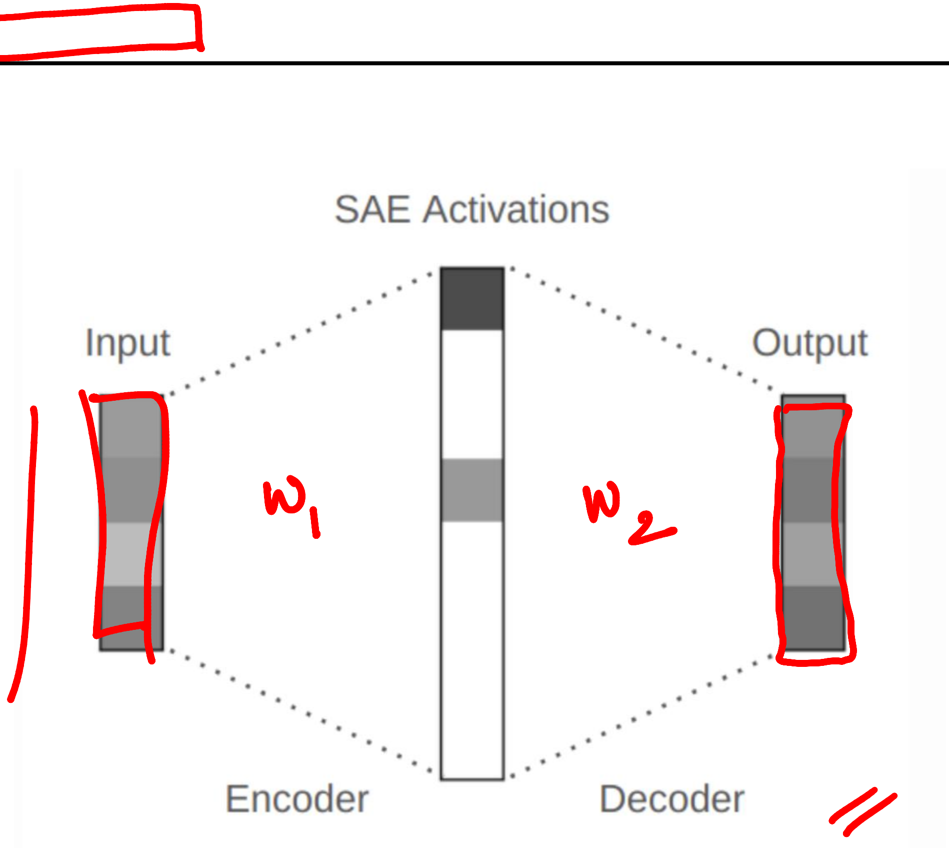
Key Concepts

- **Superposition**
  - Linear representations can represent more features than dimensions, using a strategy we call superposition. This can be seen as neural networks simulating larger networks.
  - This pushes features away from corresponding to neurons.
- **Polysemanticity of Neurons:** Some neurons appear to respond to unrelated mixtures of inputs
  - vs monosemanticity - a neuron is **monosemantic** if it corresponds to a single feature.

Sparse Autoencoders



Normal Autoencoder

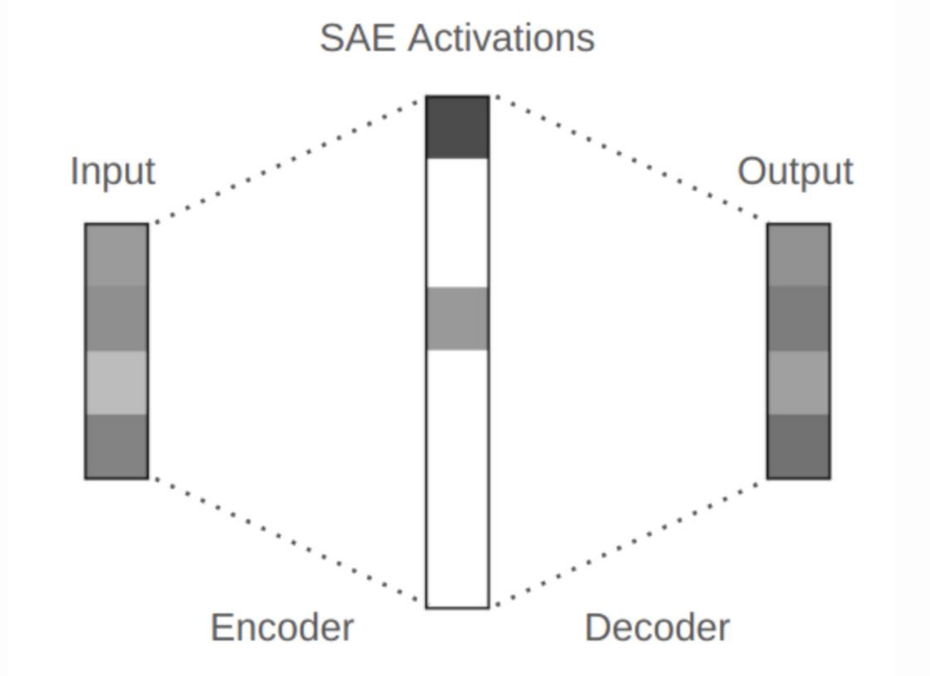


Sparse Autoencoder

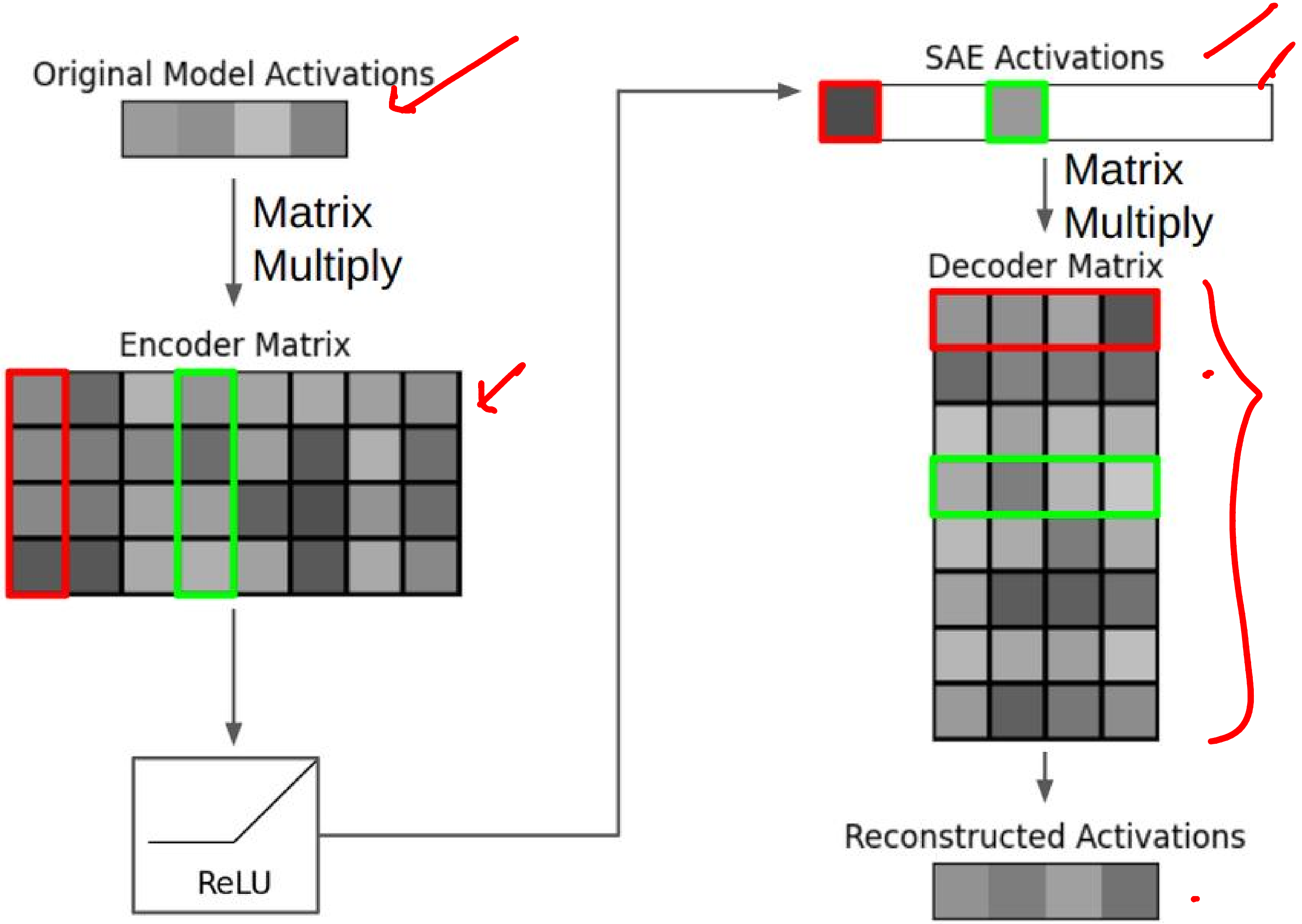
Ref

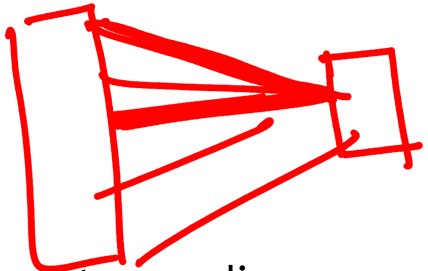
Sparse Autoencoders

- Tool for discovering **interpretable features** in model activations, trying to mitigate **polysemanticity**.
- SAEs compress high-dimensional activations into a few active latent features. The **sparsity constraint** (enforced by loss function) forces the model to isolate independent, interpretable concepts.
- Each latent unit ≈ one semantically meaningful *feature*.



Sparse Autoencoders





Input encoding:

$$h = f_{\text{enc}}(a) = \text{ReLU}(W_E a + b_E)$$

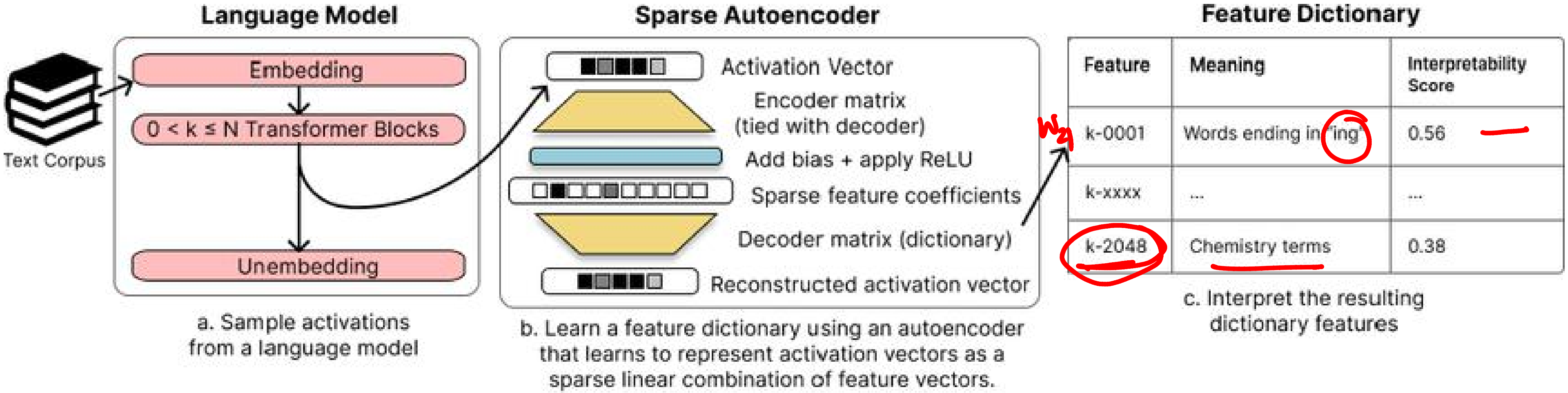
Decoding (reconstruction):

$$\hat{a} = f_{\text{dec}}(h) = W_D h + b_D$$

Loss function (reconstruction loss + sparsity loss):

$$\boxed{\mathcal{L} = \underline{\|a - \hat{a}\|_2^2} + \lambda \underline{\|h\|_1}}$$

Sparse Autoencoders



Feature	Meaning	Interpretability Score
k-0001	Words ending in "ing"	0.56
k-xxxx	...	...
k-2048	Chemistry terms	0.38

a. Sample activations from a language model

b. Learn a feature dictionary using an autoencoder that learns to represent activation vectors as a sparse linear combination of feature vectors.

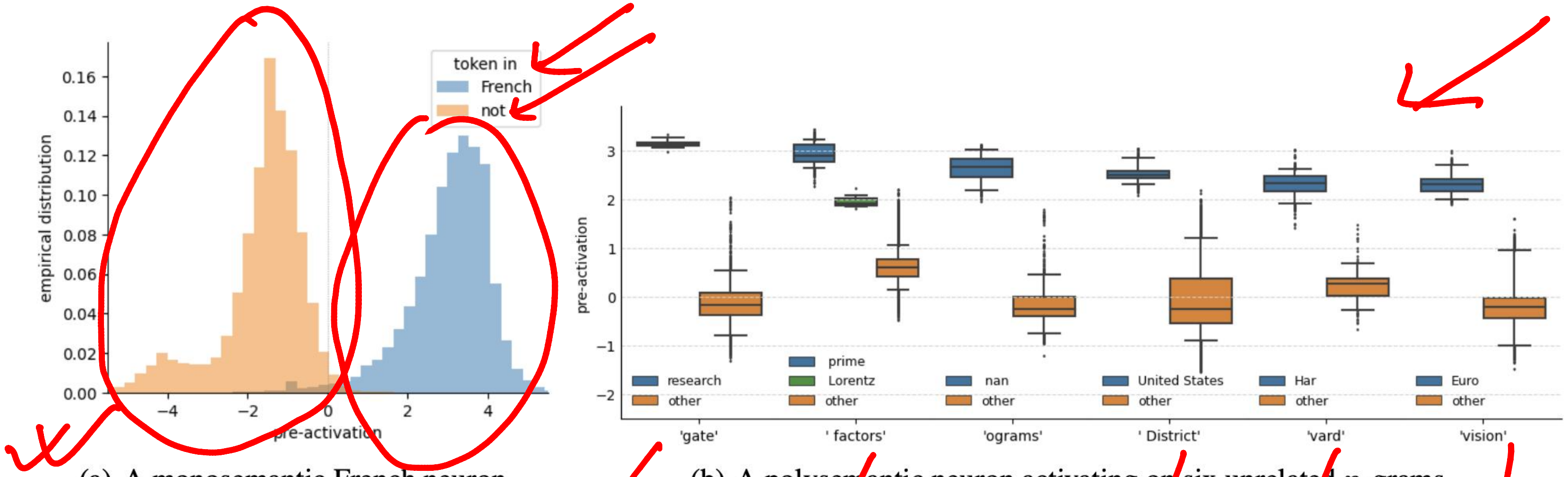
c. Interpret the resulting dictionary features

The assumption is that each learnt column of the decoder matrix corresponds to an interpretable feature. Hence all the columns for a feature dictionary.

The inputs that maximise the column's corresponding latent activation reveal its semantics.

Ref: Sparse Autoencoders Find Highly Interpretable Features In Language Models

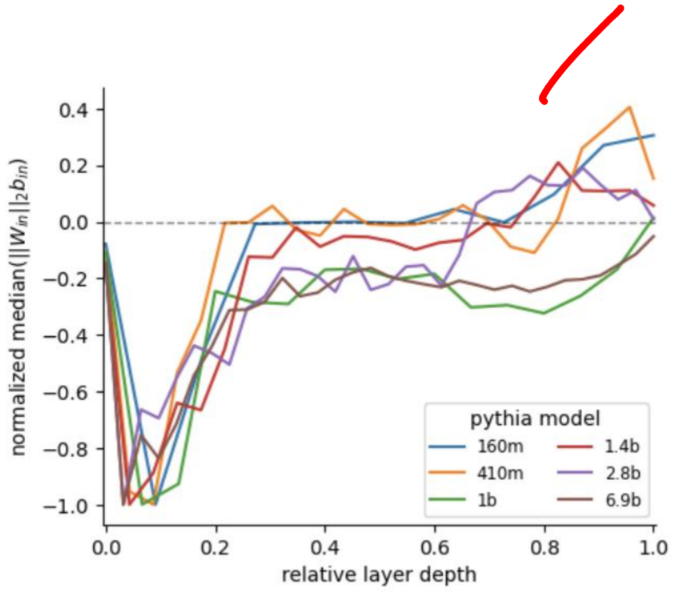
Finding neurons in a haystack – Case studies with Sparse Probing



(a) A monosemantic French neuron

(b) A polysemantic neuron activating on six unrelated n-grams

Figure 1: Neurons that respond to single features (left) can be understood independently, in contrast to polysemantic neurons (right) that activate for many unrelated features (in this case, the occurrence of specific multi-token spans).



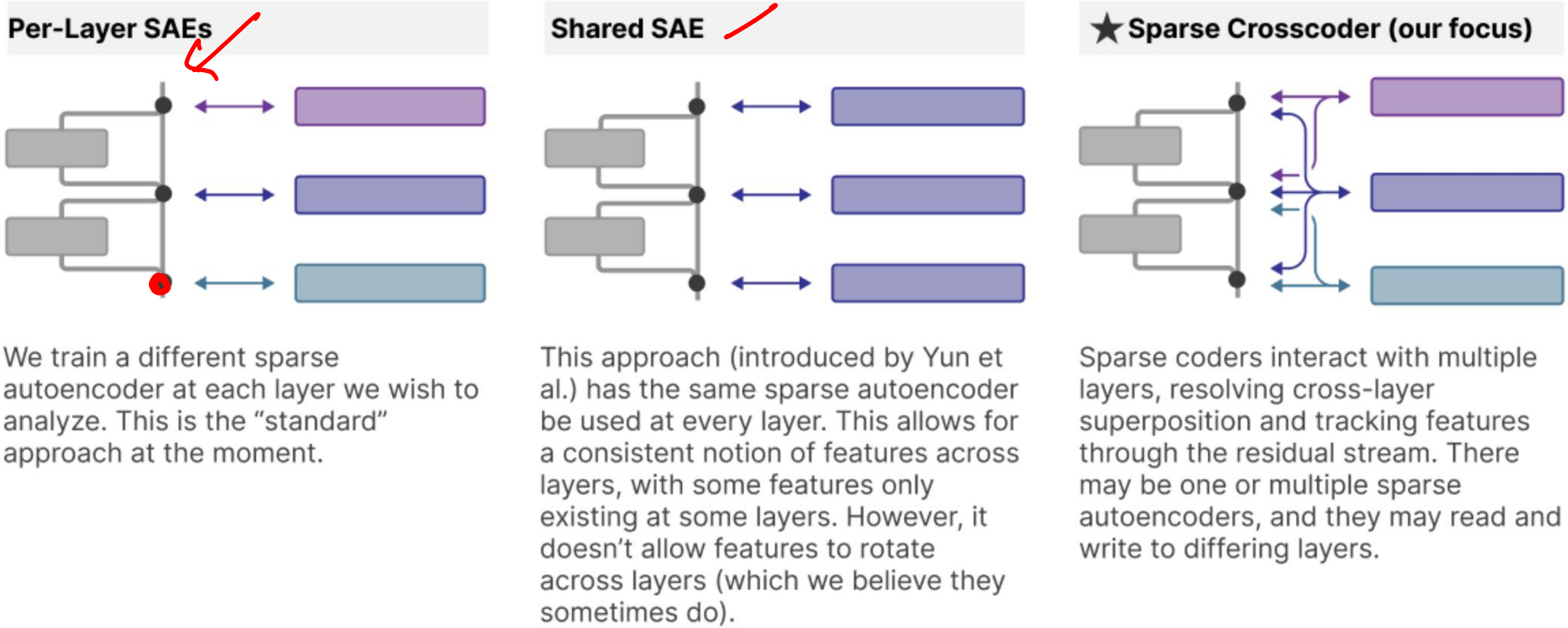
(c) Evidence of superposition in early layers

Some critical findings:

- Tremendous amount of interpretable structure within the neurons of LLMs but requires careful use and follow-up analysis to draw rigorous conclusions.
- The first 25% of fully connected layers employ substantially more superposition than the rest.
- As models increase in size, representation sparsity increases on average.

https://arxiv.org/pdf/2305.01610

Sparse Autoencoders - Variations



Per-Layer SAEs

We train a different sparse autoencoder at each layer we wish to analyze. This is the “standard” approach at the moment.

Shared SAE

This approach (introduced by Yun et al.) has the same sparse autoencoder be used at every layer. This allows for a consistent notion of features across layers, with some features only existing at some layers. However, it doesn’t allow features to rotate across layers (which we believe they sometimes do).

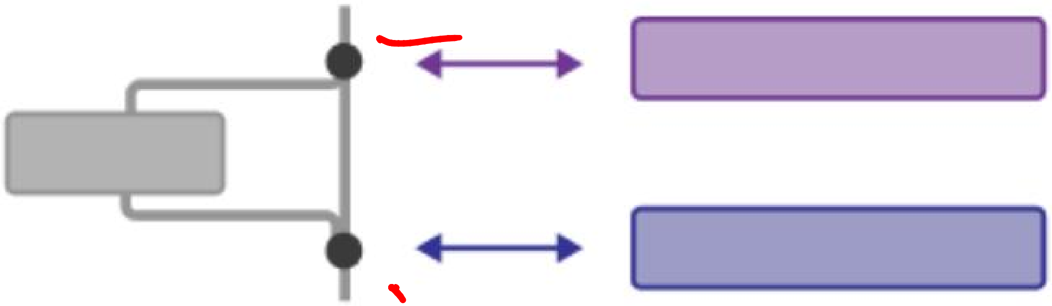
★ Sparse Crosscoder (our focus)

Sparse coders interact with multiple layers, resolving cross-layer superposition and tracking features through the residual stream. There may be one or multiple sparse autoencoders, and they may read and write to differing layers.

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

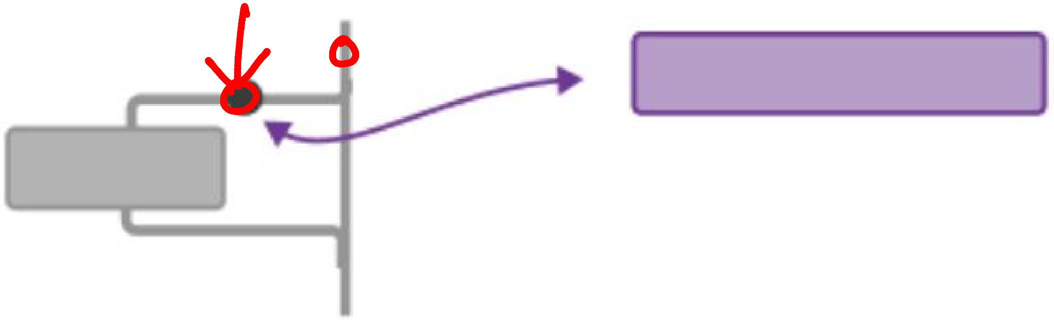
Sparse Autoencoders - Variations

Residual Stream



Sparse coder predict the residual stream.

Layer Outputs



Sparse coder predict the outputs of MLPs or attention before they're added to the residual stream. They may still read from the residual stream.

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

Transcoders vs SAEs

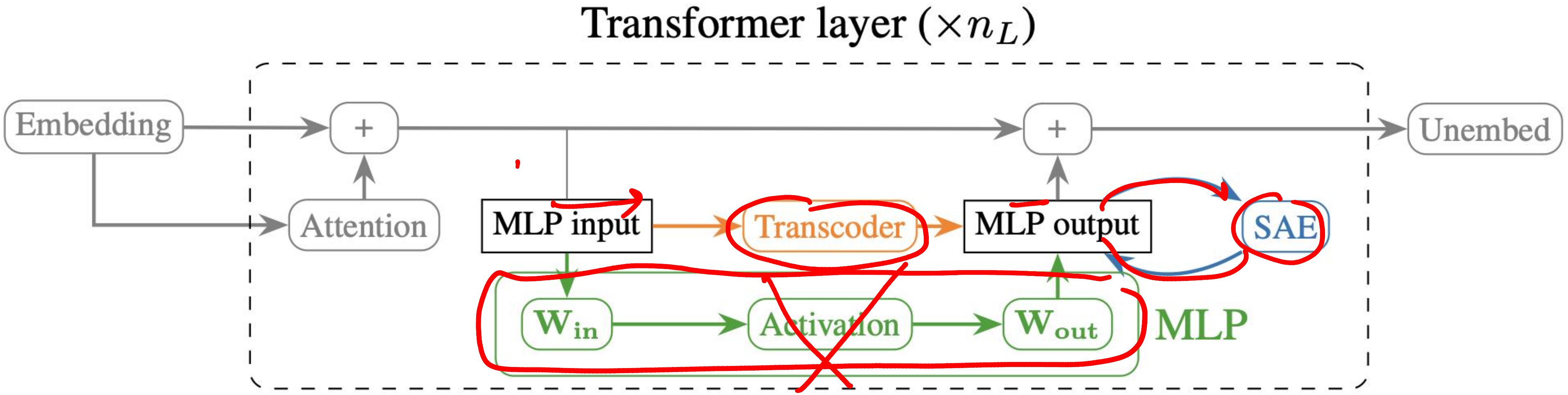


Figure 1: A comparison between **SAEs**, **MLP transcoders**, and **MLP sublayers** for a transformer-based language model. SAEs learn to reconstruct model activations, whereas transcoders imitate sublayers' input-output behavior.

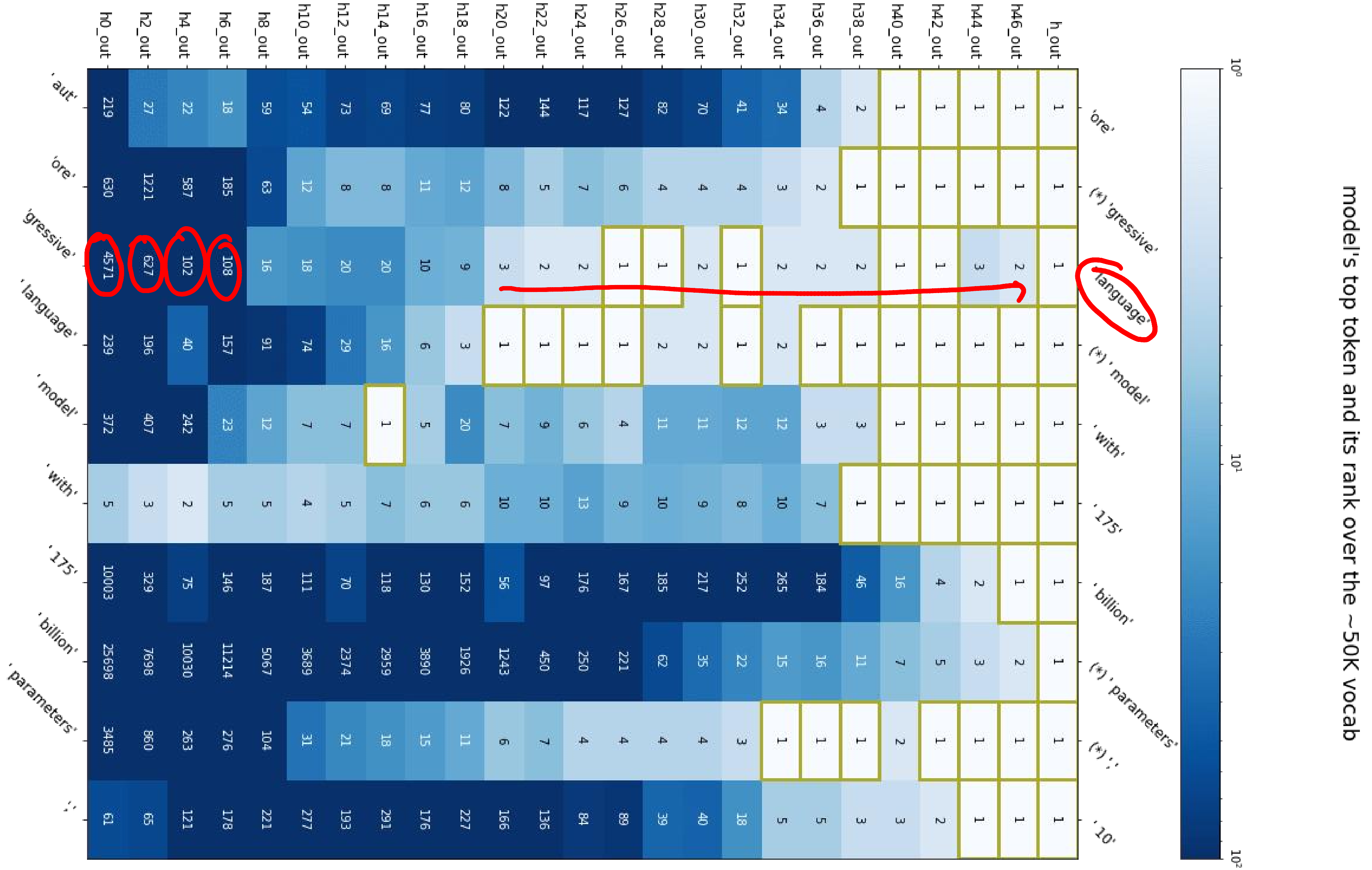
Ref: Transcoders find Interpretable LLM circuits

Logit Lens

- For analyzing **how predictions evolve** through model layers.
- Projects intermediate activations (residual stream) into **vocabulary space**.
- Shows which tokens the model "thinks" are likely at each layer.

$$\text{logits}_l = W_U \cdot a_l$$

- (where W_U is the unembedding matrix and a_l is the residual stream at layer l)



	h0_out	h2_out	h4_out	h6_out	h8_out	h10_out	h12_out	h14_out	h16_out	h18_out	h20_out	h22_out	h24_out	h26_out	h28_out	h30_out	h32_out	h34_out	h36_out	h38_out	h40_out	h42_out	h44_out	h46_out	h_out	
'aut'	219	27	22	18	59	54	73	69	77	80	122	144	117	127	82	70	41	34	4	2	1	1	1	1	1	'ore'
'ore'	630	1221	587	185	63	12	8	8	11	12	8	5	7	6	4	4	4	3	2	1	1	1	1	1	1	(*) 'gressive'
'gressive'	4571	627	102	108	16	18	20	20	10	9	3	2	2	1	1	2	1	2	2	2	1	1	3	2	1	' language'
' language'	239	196	40	157	91	74	29	16	6	3	1	1	1	1	2	2	1	2	1	1	1	1	1	1	1	(*) ' model'
' model'	372	407	242	23	12	7	7	1	5	20	7	9	6	4	11	11	12	12	3	3	1	1	1	1	1	' with'
' with'	5	3	2	5	5	4	5	7	6	6	10	10	13	9	10	9	8	10	7	1	1	1	1	1	1	' 175'
' 175'	10003	329	75	146	187	111	70	118	130	152	56	97	176	167	185	217	252	265	184	46	16	4	2	1	1	' billion'
' billion'	25698	7698	10030	11214	5067	3689	2374	2959	3890	1926	1243	450	250	221	62	35	22	15	16	11	7	5	3	2	1	(*) ' parameters'
' parameters'	3485	860	263	276	104	31	21	18	15	11	6	7	4	4	4	4	3	1	1	1	2	1	1	1	1	(*) ','
','	61	65	121	178	221	277	193	291	176	227	166	136	84	89	39	40	18	5	5	3	3	2	1	1	1	' 10'

Circuit Finding

- **Modular partitioning:** A post hoc way of understanding a deep NN in terms of modules is to partition the neurons into a set of subnetworks, each composed of related neurons. Toward this goal, some work divides neurons into modules based on graphical analysis of the network's weights and analyze how distinct the neurons in each module are. [https://arxiv.org/pdf/2207.13243]

- **Circuit Analysis:** Instead of analyzing an entire partition of a network, a much simpler approach is to study individual subnetworks inside of it. These have often been referred to as neural "circuits" which can be as small as just a few neurons and weights.

Activation Patching

- **Activation patching** (also called **causal tracing**): internal activations from one model run are "patched" into another run, enabling researchers to **causally probe** which components are necessary and sufficient for specific behaviors.

Standard activation patching procedure:

- Choose two prompts that differ in some key aspect:
  - Clean prompt (X_{clean}): elicits the desired behavior (e.g., "The Eiffel Tower is in" → "Paris")
  - Corrupted prompt ($X_{corrupt}$): differs in the dimension of interest (e.g., "The Colosseum is in" → "Rome")
- Run three model computations:
  - Clean run: Execute the model on X_{clean} and cache all activations
  - Corrupted run: Execute the model on $X_{corrupt}$ normally
  - Patched run: Execute the model on $X_{corrupt}$, but replace specific activations with those cached from the clean run
- Measure how the patched output changes using metrics like logit difference or KL divergence
- Repeat systematically across all components of interest (layers, attention heads, neurons)

Activation Patching

- **Key insight - activation patching reveals causal relationships:** if patching an activation from the clean run into the corrupted run restores the clean behavior, that component is **causally important** for the behavior. This allows us to identify which parts of the model encode specific information while **controlling for confounding factors**.

Correlation ✓ Causation ✓

Causal Tracing using activation patching – Example (Meng et al., 2022)

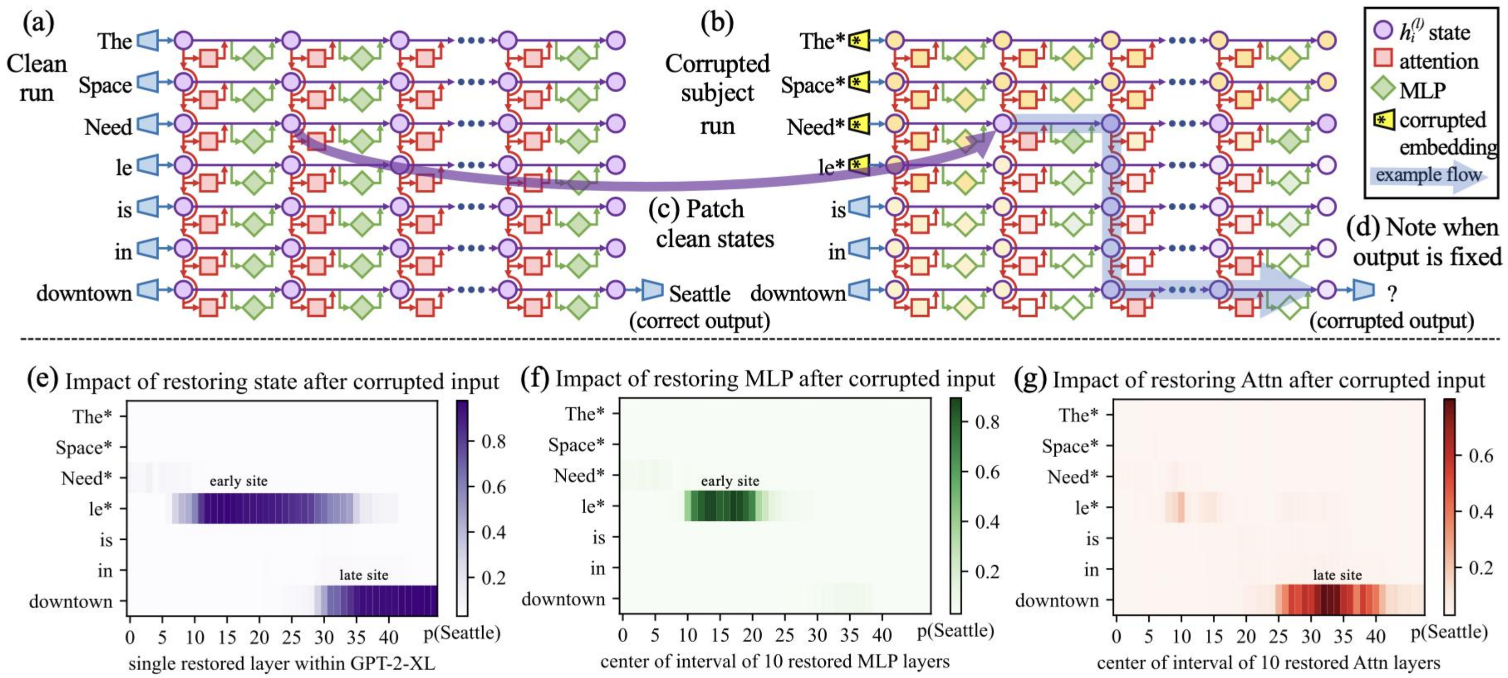


Figure 1: **Causal Traces** compute the causal effect of neuron activations by running the network twice: (a) once normally, and (b) once where we corrupt the subject token and then (c) restore selected internal activations to their clean value. (d) Some sets of activations cause the output to return to the original prediction; the light blue path shows an example of information flow. The causal impact on output probability is mapped for the effect of (e) each hidden state on the prediction, (f) only MLP activations, and (g) only attention activations.

Ref: ROME, a model editing method, uses causal mediation analysis to identify the specific modules that mediate
recall of a fact about a subject

Activation Steering

- An **inference-time** technique for modifying LLM behavior by directly intervening on internal activations without retraining or changing model parameters. It works by adding vectorial modifications (called **steering vectors)** at specific layers in the model's computation to alter outputs along interpretable dimensions such as sentiment, truthfulness, style, or safety.

Steering Language Models With Activation Engineering

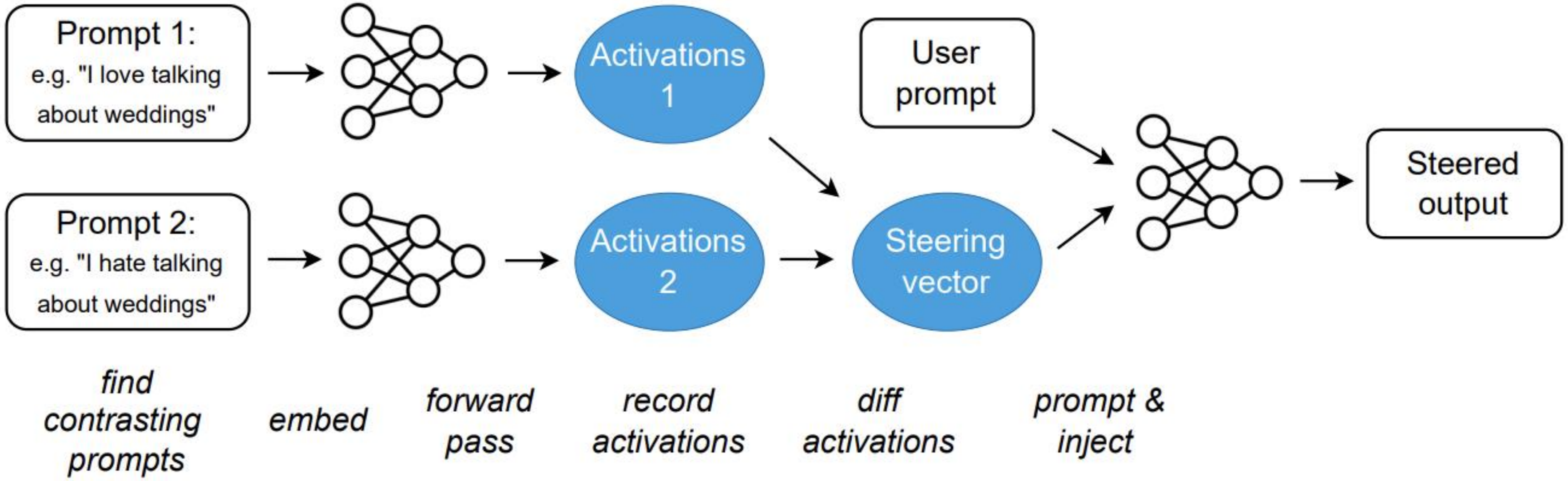


Figure 1: Schematic of the Activation Addition (**ActAdd**) method. ▭ = natural language text; ● = vectors of activations just before a specified layer. In this example, the output is heavily biased towards discussing weddings, regardless of the topic of the user prompt. (See Algorithm 1 for the method's parameters: intervention strength, intervention layer, and sequence alignment.)

https://arxiv.org/pdf/2308.10248

Steering Language Models With Activation Engineering

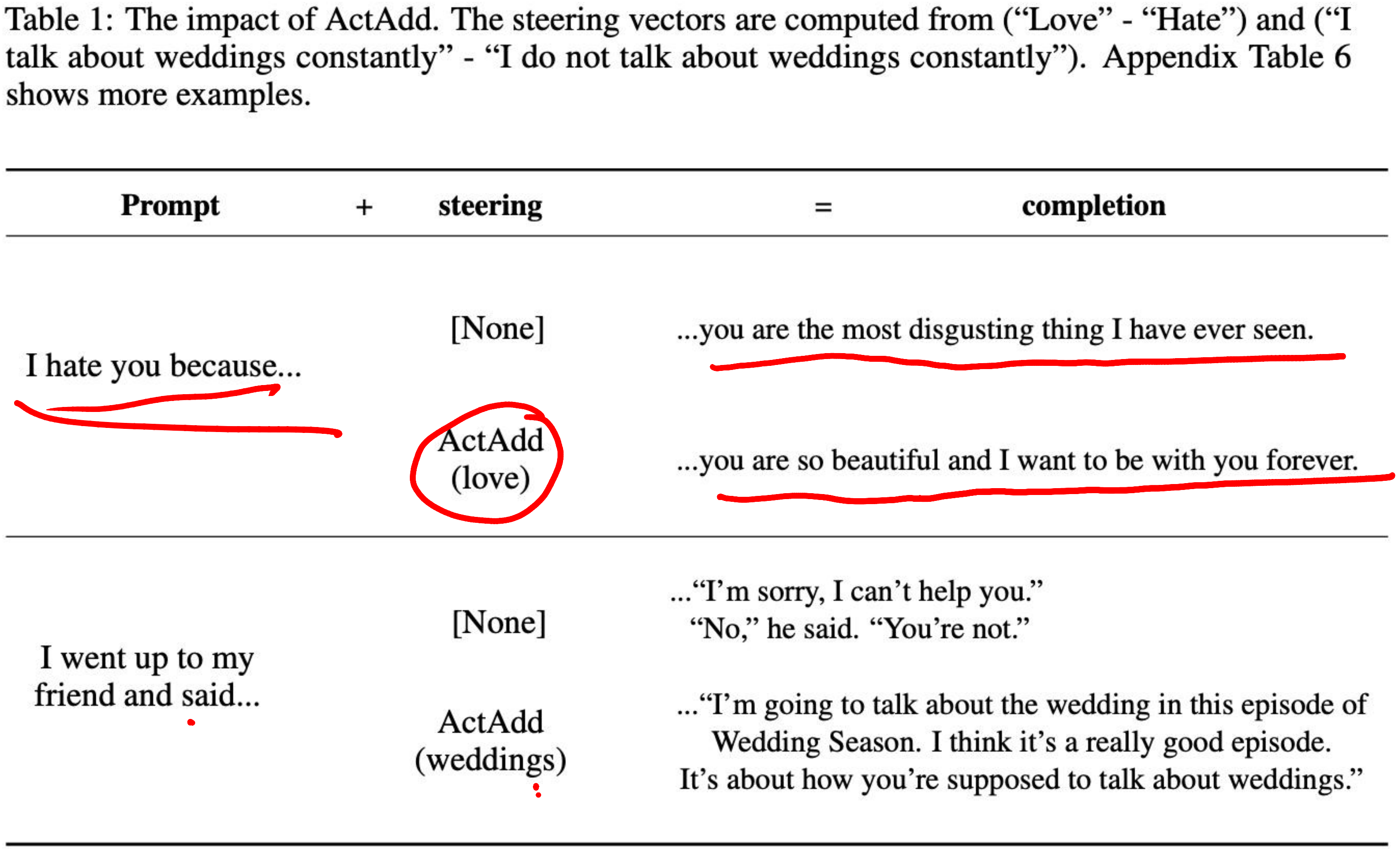


Table 1: The impact of ActAdd. The steering vectors are computed from ("Love" - "Hate") and ("I talk about weddings constantly" - "I do not talk about weddings constantly"). Appendix Table 6 shows more examples.

Prompt	+	steering	=	completion
I hate you because...		[None]		...you are the most disgusting thing I have ever seen.
		ActAdd (love)		...you are so beautiful and I want to be with you forever.
I went up to my friend and said...		[None]		..."I'm sorry, I can't help you." "No," he said. "You're not."
		ActAdd (weddings)		..."I'm going to talk about the wedding in this episode of Wedding Season. I think it's a really good episode. It's about how you're supposed to talk about weddings."

References