Model Interpretibility for FM

- Open Problems in Mechanistic Interpretability
- Position-aware Automatic Circuit Discovery

Open Problems in Mechanistic Interpretability

Matthew Nguyen (ttn5cv)

Contents

Introduction

Introduction

- **Emergent AI Capabilities:** Modern AI systems achieve impressive capabilities through deep neural networks that learn autonomously from training data, meaning their internal mechanisms are not explicitly designed or fully understood by developers.
- **Enhanced Control and Trust:** Gaining insight into these neural mechanisms could improve human control, monitoring, and trust in AI systems, which is critical for deploying them safely in high-stakes and ethically-sensitive settings.
- **Scientific and Engineering Opportunities:** The field of mechanistic interpretability seeks to understand how neural networks make decisions, offering the potential to uncover new scientific principles and enhance practical applications in various domains.

Motivation

- Although several recent reviews of mechanistic interpretability research and related topics exist this paper takes a more forward-looking stance.

- They discuss not only where the frontier is today, but also which directions we might benefit most from prioritizing in the future.

Types of Interpretability

- **Interpretability by design:** This thread focuses on constructing AI models to be transparent from the outset, often using inherently interpretable architectures such as decision trees, linear models, or additive models alongside classical attribution techniques to explain sensitivity to inputs and training data.
- **Local attribution methods:** This thread targets understanding individual decisions made by large-scale neural networks by leveraging scalable attribution techniques that highlight the influence of specific inputs on model outputs.
- **Mechanistic interpretability:** This thread investigates the internal computational structures and shared mechanisms that enable neural networks to generalize across diverse tasks, aiming to reveal how circuits, representation subspaces, and information flow patterns contribute to broad model performance.

Goals

- Monitor AI systems for signs of dangerous or emergent cognition
- Enable targeted modifications to internal mechanisms and parameters to adjust model behavior.
- Predict how models will act in unforeseen situations or acquire new capabilities.
- Enhance inference, training, and overall model performance to meet specific preferences.
- Extract latent knowledge from models to improve our understanding of the world.

Methods and Foundations of Open Problems

Mech Interp Approaches

- Neural networks learn parameters that enable them to implement algorithms, which process input data through a series of transformational steps to produce an output.
- Mechanistic interpretability is the study of understanding how different network components contribute to these step-by-step processes.
- There are two primary strategies for mechanistic interpretability: reverse engineering and concept-based interpretability

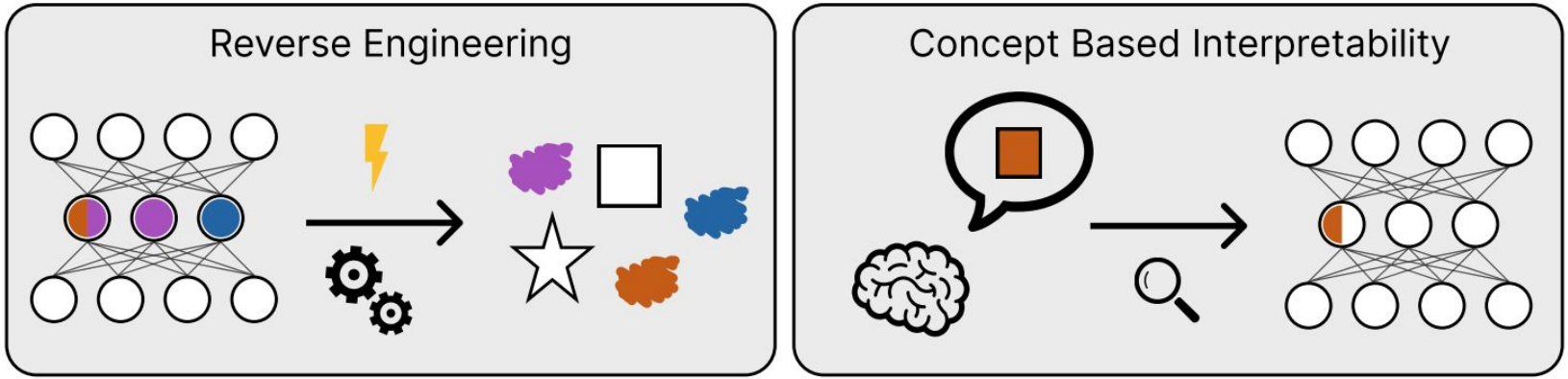


Figure 1: Two approaches to neural network interpretability. (Left) Reverse Engineering is characterized by decomposing networks into functional components and describing how those components interact to produce the network's behavior. It thus aims to 'identify the roles of network components' (Section 2.1). (Right) Concept-based interpretability on the other hand attempts to discover human concepts within neural network internals. It thus aims to 'identify the network components for given roles' (Section 2.2).

Reverse Engineering

- Although large language models produce human-like text, they use fundamentally different cognitive processes and problem-solving methods. For instance, a small model can outperform humans in next-token prediction yet struggle with tasks that even a young child can manage.

- To understand these "alien" mechanisms, developing reverse-engineering methods is essential for uncovering and interpreting the internal strategies these models employ.

- Reverse engineering generally involves three steps: decomposition, description of components, and validation of descriptions

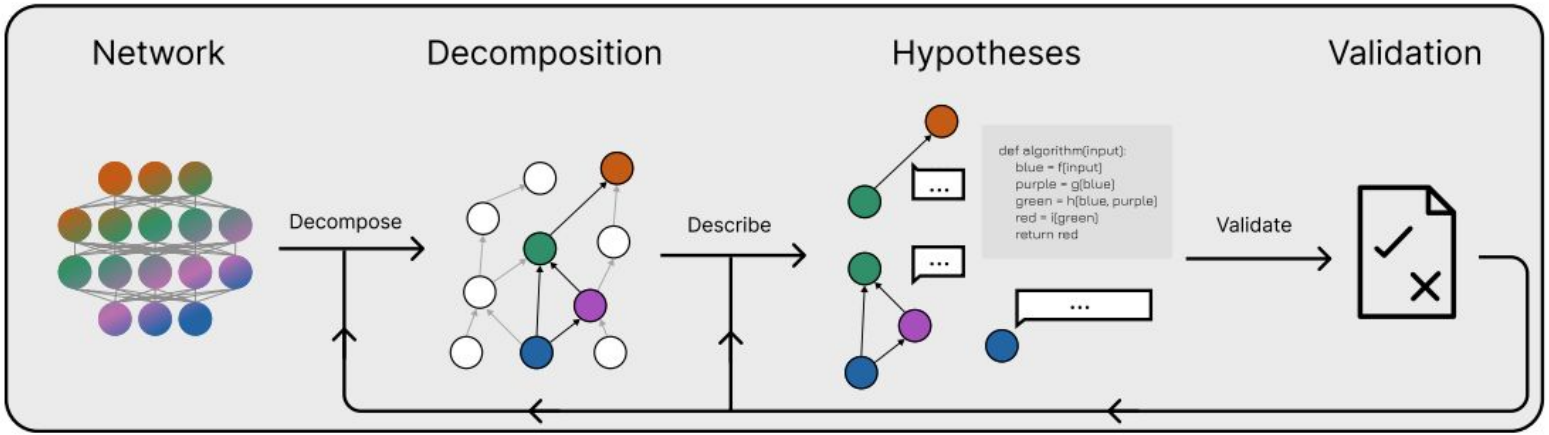


Figure 2: The steps of reverse engineering neural networks. (1) Decomposing a network into simpler components. This decomposition might not necessarily use architecturally-defined bases, such as individual neurons or layers (Section 2.1.2). (2) Hypothesizing about the functional roles of some or all components (Section 2.1.3). (3) Validating whether our hypotheses are correct, creating a cycle in which we iteratively refine our decompositions and hypotheses to improve our understanding of the network (Section 2.1.4).

Step 1: Neural Network Decomposition

- Early approaches sought to interpret neural networks by breaking them down into individual neurons, attention heads, or layers—mirroring the neuron doctrine in neuroscience
- Research revealed that both neurons and attention heads are polysemantic, meaning they respond to multiple features, which complicates their interpretation as distinct units
- Finally, observations indicate that network representations often span multiple layers, suggesting that relying solely on natural architectural components can be misleading for understanding neural network behavior

Decomposition by dimensionality reduction

- Since single neurons fall short in capturing neural network computations, researchers are exploring groups or patterns of neurons as the fundamental computational units.
- Methods involve feeding models with diverse unlabeled inputs, gathering hidden activations, and searching for structured activation vectors that might reflect the underlying computation.
- Techniques such as PCA, SVD, and non-negative matrix factorization have been employed to reveal these patterns, although they are no longer the dominant approaches in mechanistic language model analysis.

Decomposition by sparse dictionary learning (SDL)

- **Exploits Sparse Activation:** SDL is built on the superposition hypothesis, leveraging the idea that neural networks can encode more features than they have activation dimensions by ensuring each feature activates only sparsely.
- **Two-Layer Neural Decomposition:** It typically employs a simple two-layer network—an encoder that produces sparse latent activations and a decoder that serves as a dictionary aligning with the feature directions in the hidden activations (with variants like sparse autoencoders, transcoders, and crosscoders).
- **Key Approach in Mechanistic Interpretability with Limitations:** While SDL is currently the leading unsupervised method for uncovering hidden features in neural networks, it comes with substantial practical and conceptual challenges despite its innovative approach.

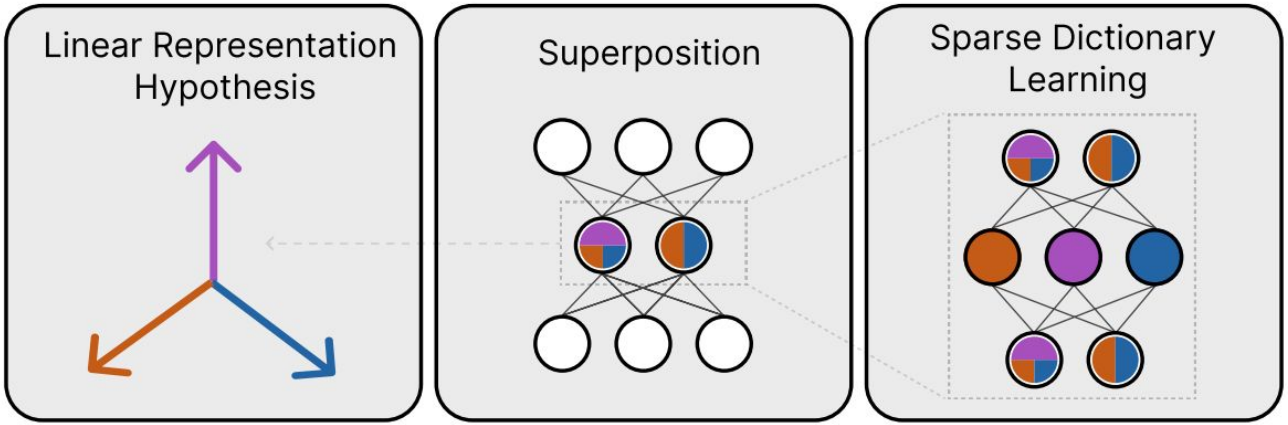
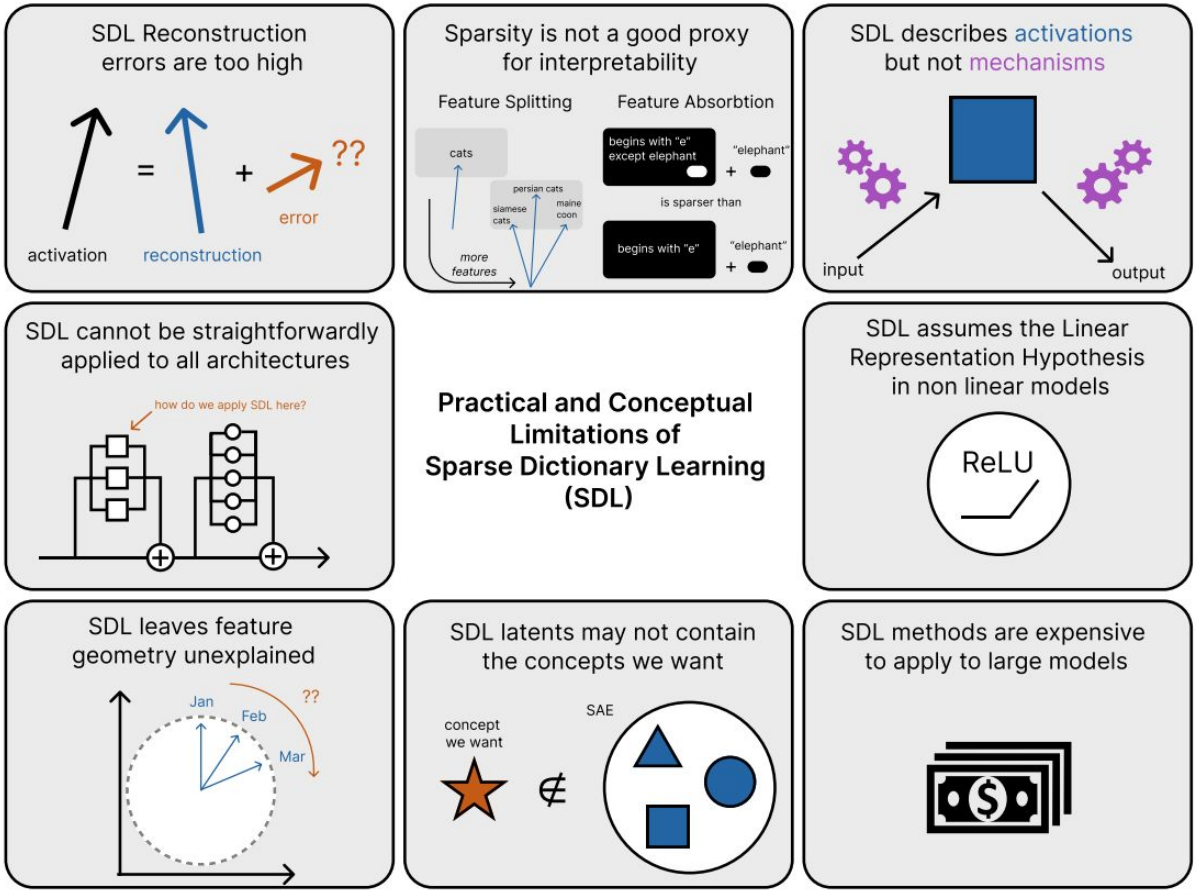


Figure 3: Three ideas underlying the sparse dictionary learning (SDL) paradigm in mechanistic interpretability. (Left) The linear representation hypothesis states that the map from 'concepts' to neural activations is linear. (Middle) Superposition is the hypothesis that models represent many more concepts than they have dimensions by representing them both sparsely and linearly in activation spaces. (Right) SDL attempts to recover an overcomplete basis of concepts represented in superposition in activation space.


SDL Reconstruction errors are too high
activation
reconstruction
error
??
Sparsity is not a good proxy for interpretability
Feature Splitting
Feature Absorbtion
cats
persian cats
siamese cats
maine coon
more features
begins with "e" except elephant
"elephant"
is sparser than
begins with "e"
"elephant"
SDL describes activations but not mechanisms
input
output
SDL cannot be straightforwardly applied to all architectures
how do we apply SDL here?
Practical and Conceptual Limitations of Sparse Dictionary Learning (SDL)
SDL assumes the Linear Representation Hypothesis in non linear models
ReLU
SDL leaves feature geometry unexplained
Jan
Feb
Mar
??
SDL latents may not contain the concepts we want
concept we want
SAE
SDL methods are expensive to apply to large models

Intrinsic Interpretability

- Instead of training solely for performance and interpreting models post hoc, intrinsic interpretability focuses on embedding interpretability directly into the training process to yield more decomposable models

- Proposed methods include forcing network activations to use discrete codes, employing sparser activation functions (like TopK or SoLU), limiting attention superposition, and utilizing approaches such as mixture of experts to encourage sparsely activating, individually interpretable components.

Step 2: Describing the functional role of components

- After decomposing the network, the next reverse engineering step is to hypothesize and describe the functional role of each component

- Descriptions of the functional role of neural network components can either indicate (1) the cause of a component's activation or (2) what occurs after that component has been activated

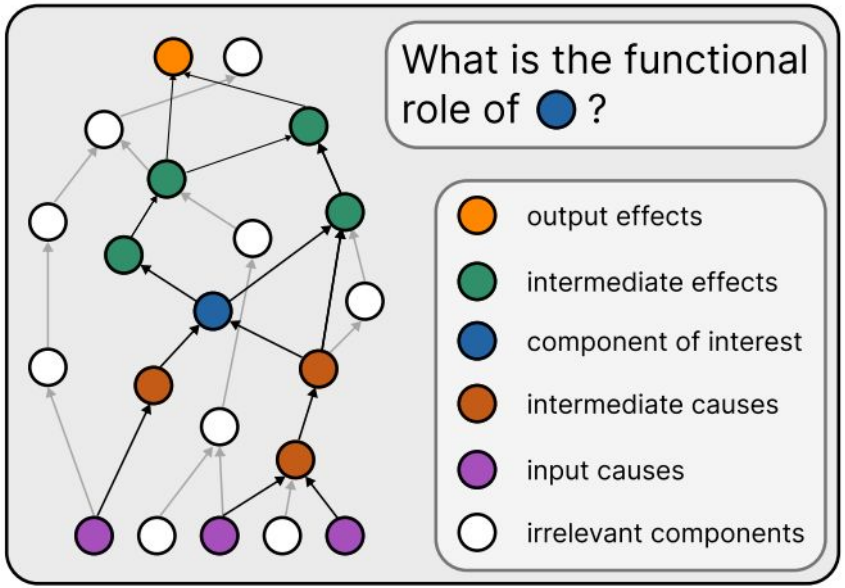


Figure 5: To study the functional role of the blue component numerous approaches are possible. We could study its causes: the purple input components or the red intermediate components via e.g. feature synthesis (Olah et al., 2017a; 2020b), maximum activating examples (Olah et al., 2017a; Bricken, 2023), or attributions (Sundararajan et al., 2017). Or we could study its effects: the orange output components or the green intermediate components via e.g. the logit lens (Nostalgebraist, 2020), activation steering (Turner et al., 2024), or attributions (Marks et al., 2024).

Explanations for what causes components to activate

- **Highly activating dataset examples:** This method identifies inputs that strongly trigger a neural component to hypothesize its underlying function.

- **Attribution methods:** These approaches quantify the causal impact of input features on a component’s activation via gradient, sampling, or perturbation techniques.

- **Feature synthesis:** This strategy combines exemplar-based insights with gradient-based attributions to generate synthetic inputs that maximize a component's activation under regularization constraints.

Explanations for the downstream effects of components

- **Studying the direct effect:** This approach employs techniques like the logit lens and tuned lens to convert intermediate activations into output predictions.

- **Causal interventions:** These methods perturb or replace specific neural activations or connections during the model's forward pass (e.g., via patching, ablation, or path patching) to isolate and measure their causal contributions.

- **Observing the effects on sequential behavior:** This strategy involves injecting activated components into new contexts—through techniques such as steering, patchscopes, or analyzing chains-of-thought—to interpret the role of these components in generating sequential outputs.

Zeqiang Ning(avr7qy)

Step 3: Validation of description

- Initial descriptions of network components' functional roles should be treated as hypotheses
- However, mixing up hypotheses with conclusions has been commonplace in mechanistic interpretability research, such as some model interpretations fail sanity checks
- Two ways that can simplify the validation process:
  - Model organism
    - In certain natural sciences, such as neuroscience, researchers study a few extensively researched species known as model organisms.
    - Knowledge gained from these organisms can be generalized and applied to other species. Similarly, researchers in interpretability are exploring which neural networks could serve as "model organisms" in the field of mechanistic interpretability.
    - Open-source, easy to use, low-cost, representative of a wide range of systems and behaviors, and have a reproducible training process with publicly available datasets
  - Benchmark
    - Benchmark has proven to be a highly effective evaluation method in other areas of machine learning. Therefore, designing systematic benchmarking frameworks to validate interpretability in the field of mechanistic interpretability is also a promising direction.

Step 3: Validation of description

- Criteria: Does the hypothesis make good predictions about the neural network's behavior?

- Forms of validating:
  - Predicting activations and counterfactuals
  - Predicting and explaining unusual failures or adversarial examples
  - Handcrafting a network that reconstructs a network behavior
  - Testing on ground truth
  - Using the hypothesis to achieve particular engineering goals
  - Using the hypothesis to achieve specific engineering goals competitively

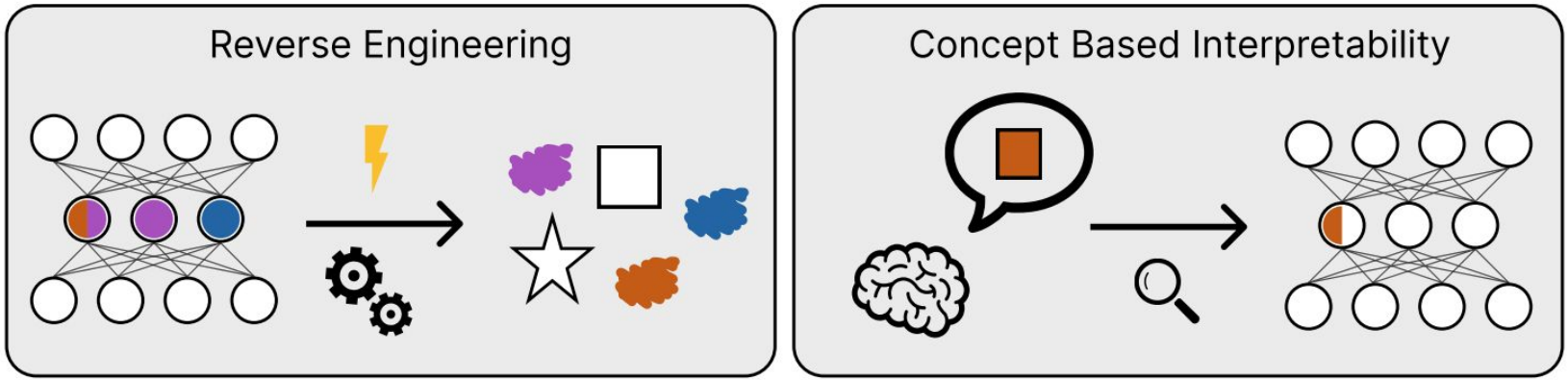


Figure 1: Two approaches to neural network interpretability. (Left) Reverse Engineering is characterized by decomposing networks into functional components and describing how those components interact to produce the network's behavior. It thus aims to 'identify the roles of network components' (Section 2.1). (Right) Concept-based interpretability on the other hand attempts to discover human concepts within neural network internals. It thus aims to 'identify the network components for given roles' (Section 2.2).

Concept-based interpretability

Concept-based probes

- When we try to locate a human-understandable concept within a neural network, an intuitive approach is to "probe" it.

- A probe is a classifier that is trained to predict a concept from the hidden representations of another mode

- For example, if we have an image classification model and want to know whether the model uses the concept of "ears" when recognizing a cat, we can use a trained probe to detect the concept of "ears."You extract image patches from a dataset that either contain or do not contain "ears", label them as 1 or 0 respectively, and train a simple linear classifier.

- If I erase the ear region in the input image and observe the model's prediction for "cat", and it still classifies the image as a cat, then it suggests that the model does not rely on the concept of "ears" to recognize a cat.

Concept-based interpretability

Challenge 1: Probes need carefully chosen data for well-defined concepts

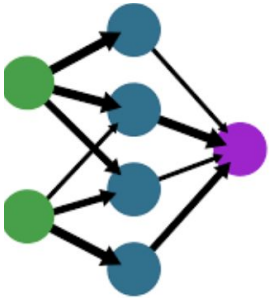
- We should know in advance what concept we are looking for
- We should be able to construct a high-quality dataset for that concept
- So you can only probe concepts that you have already clearly defined and labeled, but you cannot discover concepts in the hidden layers that you have not yet thought of
- It is time-costing and impractical
- Solution: Contrastive Consistency Search (CCS)
    - It does not probe a specific individual concept but instead explores a general "direction" in the activation space, similar to unsupervised learning.

Concept-based interpretability

Challenge 2: Probes detect correlations, rather than causal variables

- Even if a probe can accurately predict the presence of a concept, it does not necessarily mean that "the model is actually using this concept to make decisions."
- A major risk faced by probing methods is that, in addition to finding truly relevant features, they may also uncover spurious correlations caused by the high dimensionality of hidden activations.
- To improve the causal relevance of probe directions, we can use counterfactual data — that is, we intervene on the concept of interest (for example, observe what happens to the model's output if a dog in the image is replaced with a cat).

Proceduralizing mechanistic interpretability into circuit discovery pipelines



- Task Definition
  - Choose a task that the model performs and the corresponding dataset
- Decomposition
  - Represent the model as a DAG(directed acyclic graph)
  - Nodes: Represent activations in the network, such as the output of a specific layer or an individual neuron (architectural components,like attention heads）
  - Edges: Represent the connections between different activations, i.e., "abstract weights." These may correspond to actual neural network weights or more conceptual forms of dependency.
- An initial description step: Identify task-relevant vs. -irrelevant subgraphs
  - Identify task-relevant nodes/edges by casual interventions
  - For example, in the task of recognizing a cat in an image, we can intervene on certain nodes (e.g., by masking out the region corresponding to the cat's ears) and observe the model's response to see whether it can still successfully recognize the cat.

Proceduralizing mechanistic interpretability into circuit discovery pipelines

- An iterative description-validation loop
  - After recognize the related subgraph, we need to know the functions of nodes/edges
  - Design experiments to test hypotheses
  - Loop until confident understanding of the functions of all the edges and nodes

- Final Validation
  - Evaluate based on three attributes
  - Faithfulness (the extent to which the circuit approximates the behavior of the entire neural network)
  - Minimality (are all parts necessary?)
  - Completeness (did we miss anything important?)

Limitations of Circuit Discovery Pipelines

- Concept-based tasks → only capture average behavior, not individual cases.
- Imperfect decompositions → architectural & latent bases are flawed.
- Low faithfulness → causal metrics may mislead, contributing to circuits often unfaithful.
- Scalability issues → interventions are costly.
- Backup/negative behavior → important but will inhibit task performance.
- Streetlight effect → tasks chosen are too simple; hard tasks remain unsolved.

Axes of mechanistic interpretability progress

Decomposition vs. Description

- Decomposition = how we split the network.
- Description = how deeply we understand each part.
- For description, deep = causal relationship, shallow = surface-level

Extent of Understanding Needed

- Some goals need single neuron but some needs whole model

Task Distribution Scope

- For some model, we can based on narrow tasks (e.g. behavior monitoring).
- Understand full task coverage (e.g. verification).

When Understanding Happens

- For some goals, understanding the mechanisms of a trained model is sufficient; but for more ambitious goals, it is necessary to understand both the model's mechanisms and how these mechanisms evolve during the learning process.

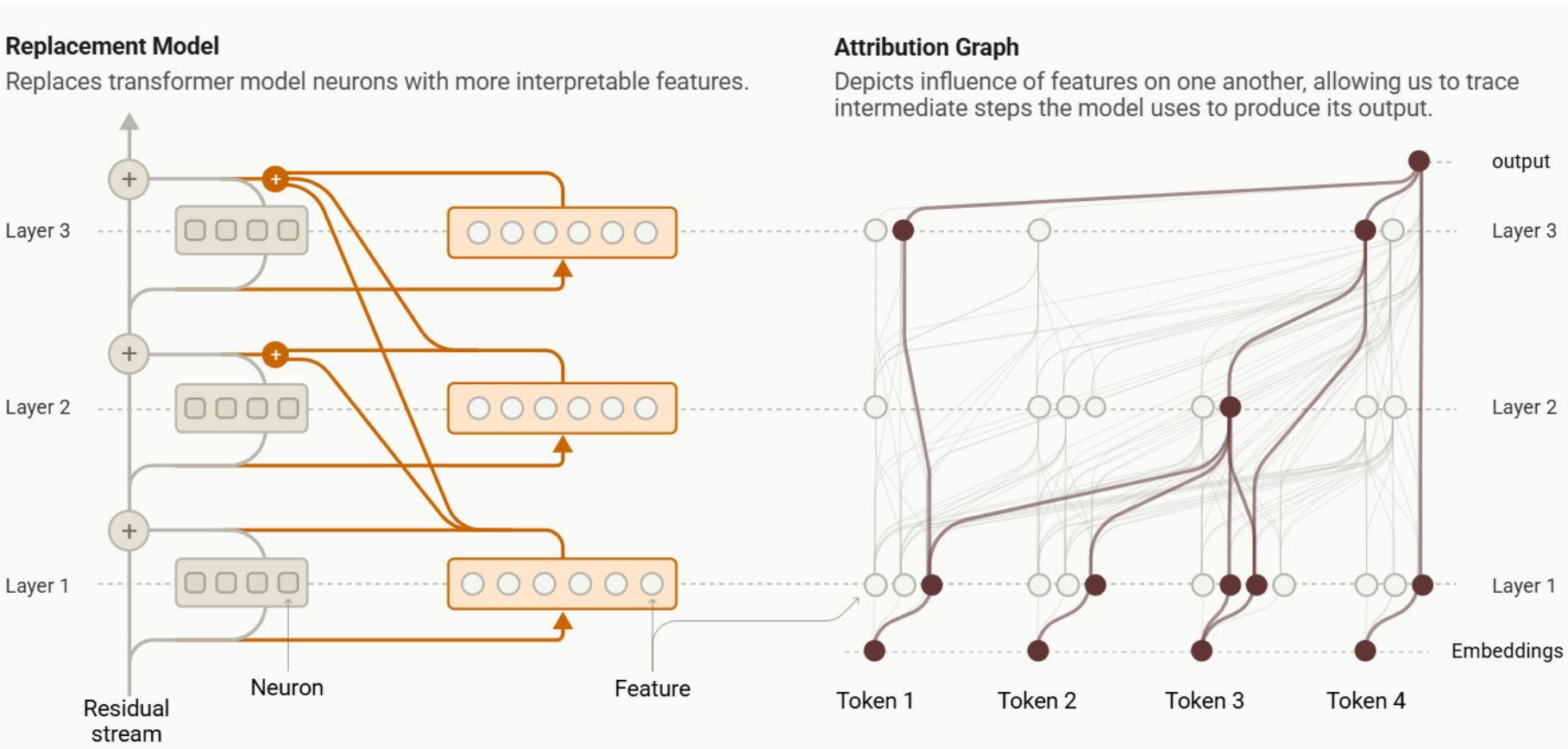
Position-aware Automatic Circuit Discovery

Wenhao Xu (wx8mcm)

What is Circuit

- A circuit is a minimal subgraph of the model's computation graph that performs a specific function.
- Nodes: components like attention heads, neurons, or layers.
- Edges: connections that pass activations (e.g., attention patterns).
- Used to explain how models compute.

Introduction



Why Automate Circuit Discovery?

- Manual methods (e.g., IOI circuit): precise but not scalable, prone to bias, and hard to generalize.
- Automatic methods: use patching, gradients, or heuristics to identify relevant components.
- However, they mostly assume position-invariance.

The Limitation of Position-Invariant Circuits

- Many automatic methods treat components as equally important across all token positions.
- This position-agnostic assumption misses cross-positional mechanisms.
- Leads to imprecise and bloated circuits.

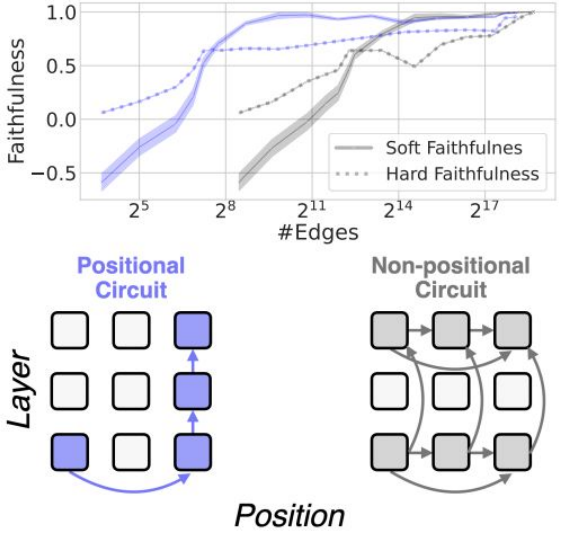
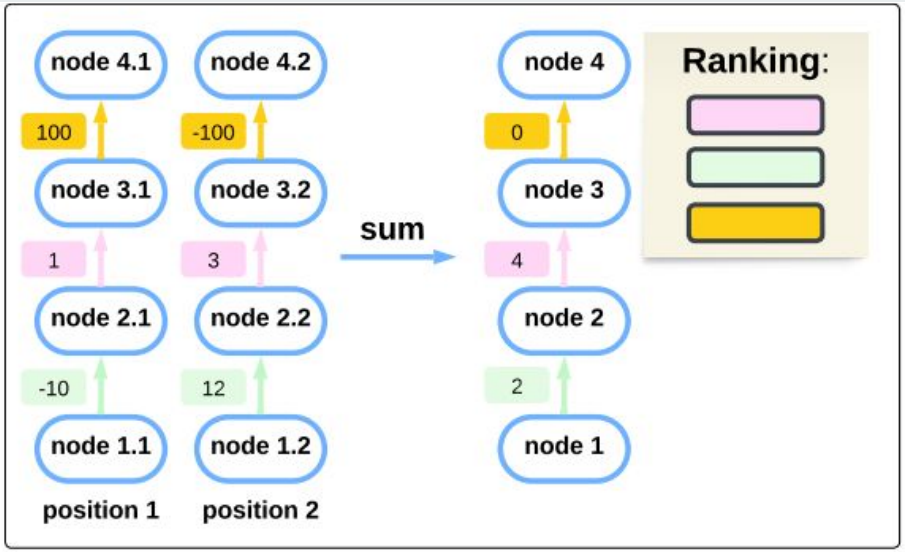
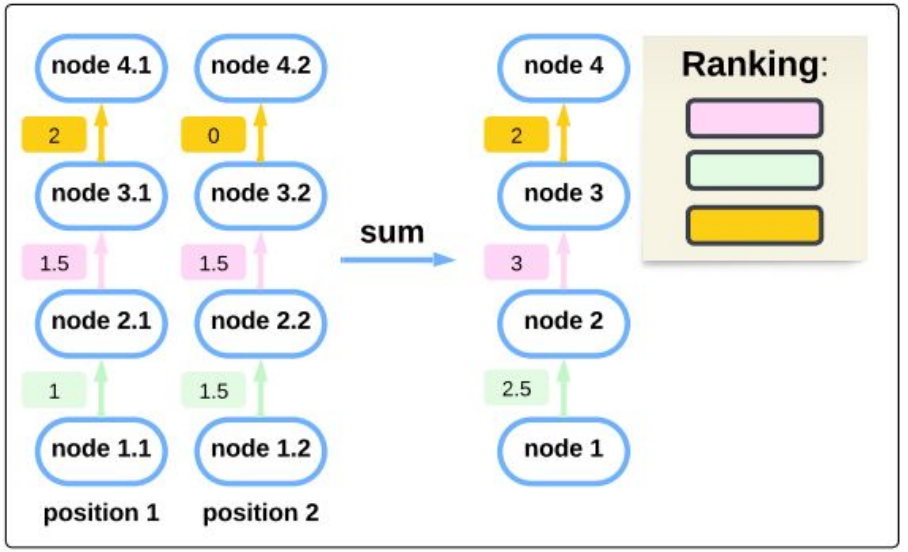


Figure 1: Positional vs. non-positional circuits. In a non-positional circuit, the same edges must be included at all positions. A positional circuit can distinguish between the same edge at different positions. This specificity yields better trade-offs between circuit size and faithfulness. It can also increase both precision and recall.

Failure Modes in Non-Positional Circuit Discovery





- Cancellation Across Positions:
  - Positive and negative effects from the same component cancel each other out.
  - Leads to low recall.
- Overestimation:
  - Small effects across many positions are summed up.
  - Leads to low precision.

Quantifying the Problems

- Evaluation done using the IOI task on GPT2-small.
- Edge importance rankings differ significantly depending on aggregation method.
- Cancellation: positive/negative effects cancel across positions.
- Overestimation: frequent but weak contributions dominate rankings.

Cancellation				
$K\%$	Diff	$\text{Diff}_{\text{Control}}$	ρ	ρ_{Control}
1	17.1%	3.9%	0.760	0.985
5	13.4%	2.4%	0.831	0.991
10	12.1%	2.3%	0.877	0.992
Overestimation				
$K\%$	Diff	$\text{Diff}_{\text{Control}}$	ρ	ρ_{Control}
1	17.5%	3.6%	0.772	0.984
5	14.6%	2.1%	0.811	0.993
10	12.4%	2.2%	0.864	0.993

EAP Formula Recap

- Estimates indirect effect (IE) of edge e=(u,v) on metric M
- Based on counterfactual activation zu∗z_u^*zu∗ and gradient at v.

$$g(e) = M(x|e = e_{x'}) - M(x) \approx (z_u^* - z_u)^\top \nabla_v M(x) \tag{1}$$

- Approximation avoids expensive full-patching.
- Still position-agnostic in Syed et al. (2023).

Modeling Attention Head Outputs in PEAP

- PEAP focuses on attention heads, where each head at each token position is a separate node.
- Output of head h_t^i is:

$$z_{h_t^i} = W_O^i(\text{softmax}(\frac{q_t^i {K_t^i}^\top}{\sqrt{d_k}})V_t^i) \in \mathbb{R}^{d_{\text{model}}} \quad (2)$$

- W_O^i : output projection for head i
- q,k,v: standard attention vectors at position t

Approximating Edge Effects Across Positions

$$M(x|e = e_{x'}) - M(x) \approx (z^*_{h^i_t} - z_{h^i_t})^\top \nabla_{z_{h^i_t}} M(x) \quad (3)$$

$$z^*_{h^i_t} = W^i_O(\text{softmax}\left(\frac{q^i_t K^{i\top}_t}{\sqrt{d_k}}\right)[v^i_1, ..., v_{t'}{}^*, ..., v^i_t]) \quad (4)$$

$$z^*_{h^i_t} = W^i_O(\text{softmax}\left(\frac{q^i_t[k^i_1, ..., k_{t'}{}^*, ..., k^i_t]^\top}{\sqrt{d_k}}\right)V^i_t) \quad (5)$$

$$z^*_{h^i_t} = W^i_O(\text{softmax}\left(\frac{[q^i_t k^{i\top}_1, ..., q^{i*}_t k_{t'}{}^\top, ..., q^i_t k^{i\top}_t]}{\sqrt{d_k}}\right)V^i_t) \quad (6)$$

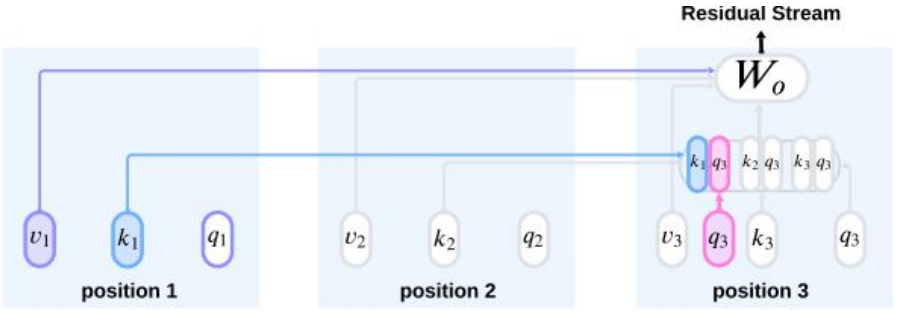


Figure 3: Illustration of the attention mechanism from the perspective of position 3. We approximate how patching v_1, k_1 or q_3 impacts the downstream metric via the output of the attention head at position 3.

Constructing the Final Circuit

- Once all position-aware edge scores are computed:
  - Use a greedy algorithm (adapted from Hanna et al., 2024b).
  - Select top edges to form a minimal faithful circuit.
- Can measure:
  - Soft Faithfulness
  - Hard Faithfulness

Why Position Matters in Circuits

- Transformers are position-sensitive by design (via positional embeddings & attention).
- Position-agnostic circuit discovery:
  - Misses causal interactions between positions.
  - Produces noisy and bloated circuits.
- Calls for a position-aware method of attribution and circuit discovery.

The Challenge of Variable-length Inputs

- PEAP works best on templated inputs (fixed length, consistent token positions).
- In real datasets, examples vary:
- “The war lasted from 1741 to...” vs. “The Black Plague ended in...”
- Same semantic role (e.g., subject) might appear at different positions.
- Positional analysis breaks when token roles shift.
- Goal: Align semantic meaning, not absolute position.

Introducing Dataset Schemas

- A schema defines semantic spans (like “Subject”, “Time”, “Verb”) for each example.
- These spans:
  - Have consistent meanings across examples.
  - Can contain multiple tokens.
  - Enables semantic alignment across variable-length inputs.

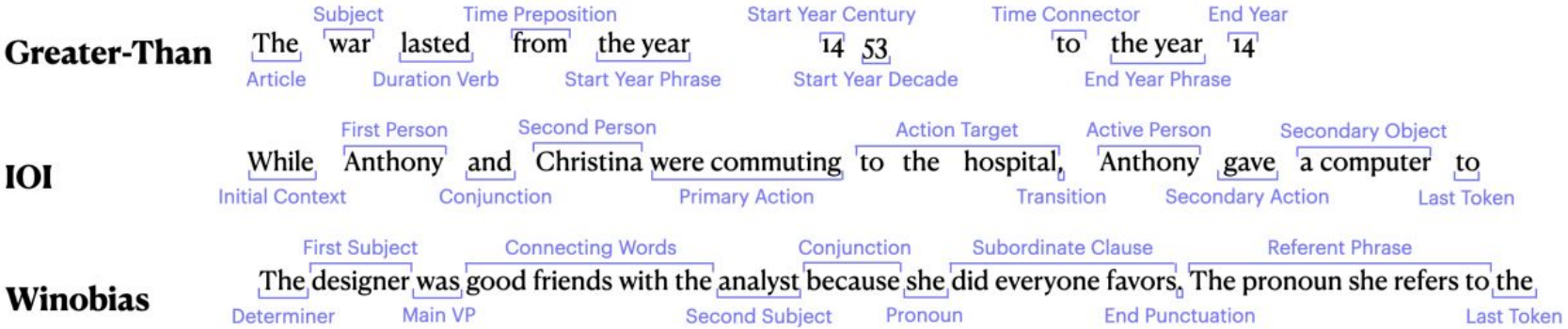


Figure 4: Example schema for each task. We show examples from the LLM+Mask method. See §A for examples of human-designed schemas.

Abstracting Computation with Schemas

- Each example's computation graph is transformed into an abstract graph, where nodes represent schema spans.
- Edges between schema spans correspond to groups of edges between individual tokens assigned to those spans.
- This transformation enables computation-level abstraction, allowing model behavior to be generalized across examples with differing lengths and token orderings.

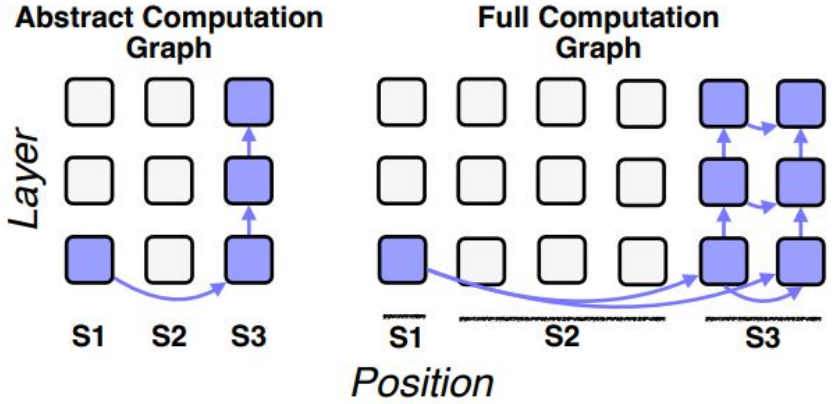


Figure 5: Circuits defined over schemas. Every node/edge at position s in the abstract computation graph is mapped to a set of nodes/edges in the full computation graph within the span s.

Aggregating Edge Importance Across Inputs

- The contribution of a schema-level edge is estimated by aggregating PEAP scores of all corresponding token-level edges across the dataset.
- This provides a single importance score for each span-to-span connection that captures its typical influence across examples.
- The result is a generalized attribution graph that represents consistent computational patterns tied to semantic structure.

Projecting Schema Circuits to Instance-specific Circuits

- Once a schema-level circuit is discovered, it can be applied to new examples by mapping schema edges back to token-level edges.
- This produces instance-specific circuits that are consistent with the generalized abstraction.
- These circuits are then used for faithfulness evaluation, measuring how well the circuit captures the model's behavior for that input.

Scalable Schema Generation via Large Language Models

- Manual schema creation is impractical for large datasets or diverse tasks.
  - The authors design a pipeline using LLMs to:
  - Generate schema definitions from representative examples.
  - Assign span labels to all inputs.
- The pipeline ensures that token spans are:
  - Complete (all tokens labeled),
  - Consistently ordered, and
  - Aligned across the dataset.

Incorporating Saliency into Schema Discovery

- Token saliency scores are computed to identify which tokens most strongly influence the model's output.
- These scores are provided to the LLM as highlighted tokens during schema generation.
- The LLM is instructed to assign highly salient tokens to separate, distinct schema spans.
- This results in schema definitions that are more aligned with model behavior and produce circuits with higher faithfulness.

Limitations of the Current Approach

- Schema assumes consistent span ordering across examples:
  - Cannot capture tasks with variable logical structure or non-linear inputs.
- No formal theory on what makes a schema optimal:
  - Current approaches are heuristic and data-driven.
- Saliency scores, while useful, are gradient-based and can be noisy:
  - Could be replaced with more principled attribution methods in future work.

Evaluating Position-aware Circuits in Practice

- Objective: Evaluate the faithfulness and efficiency of PEAP-discovered circuits.
- Faithfulness measures:
  - Soft faithfulness: How closely does the circuit replicate the model's output scores?
  - Hard faithfulness: Does the circuit lead to the same predicted output as the full model?
- Two model architectures tested:
  - GPT2-small (124M parameters)
  - LLaMA-3–8B (8 billion parameters)
- Three tasks selected to evaluate structure-sensitive reasoning:
  - IOI (Indirect Object Identification): Identify which noun a pronoun refers to.
  - Greater-Than: Determine whether a year is greater than a previous year in context.
  - Winobias: Gender coreference resolution to test for social bias.

Evaluation Pipeline

- For each task:
  - PEAP is used to discover circuits (with and without schema abstraction).
  - Circuits are evaluated by measuring output fidelity compared to the full model.
- Circuits are compared based on:
  - Number of edges (smaller = more interpretable)
  - Faithfulness at fixed circuit sizes

Circuits Compared in Each Experiment

- The following circuit types are compared:
  - Non-positional circuits (baseline)
  - Position-aware token-level circuits (PEAP)
- Schema-level circuits:
  - Human-designed schema
  - LLM-only generated schema
  - LLM + Saliency-enhanced schema

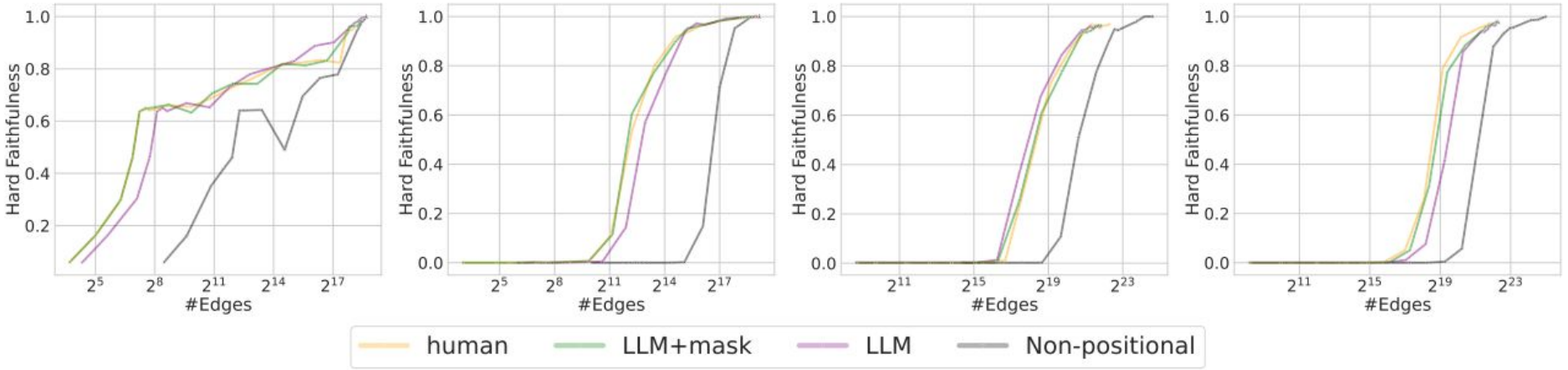


Figure 6: Hard faithfulness curves for GPT-2-small on Greater-Than (left) and IOI (mid-left), and for Llama-3-8b on IOI (mid-right) and Winobias (right).

Final Takeaways

- This paper advances the field of interpretability by showing how:
  - Positionality and semantic abstraction work hand in hand.
- With PEAP and schemas:
  - Faithful and interpretable circuits can be discovered automatically,
  - Even in large models and real-world settings.
- This sets a foundation for future interpretability pipelines that are:
  - Scalable
  - Semantically meaningful
  - Aligned with internal model computations

Using mechanistic interpretability for better control of AI system behavior

Activation Steering

- Adds a fixed activation vector to the model's intermediate layer based on the Linear Representation Hypothesis
- Allows steering of the model's behavior with fewer side effects
- Future Direction: Steering entire mechanisms, not just individual features

Machine Unlearning

- Removing undesirable knowledge or sensitive data from models
- Interpretability helps precisely target mechanisms responsible for certain knowledge and evaluate unlearning through white-box methods
- Current Limitation: Intermediate activation methods are not yet competitive

Model Editing

- Aims to precisely modify specific knowledge while preserving other capabilities
- Progress depends on better network decomposition and high-quality knowledge representation and description methods

Interpretability and Finetuning

- Interpretability methods can improve finetuning efficiency (by modifying local parameters) and better debug the side effects of finetuning

Chenxu Li (jnr2jp)

Contents

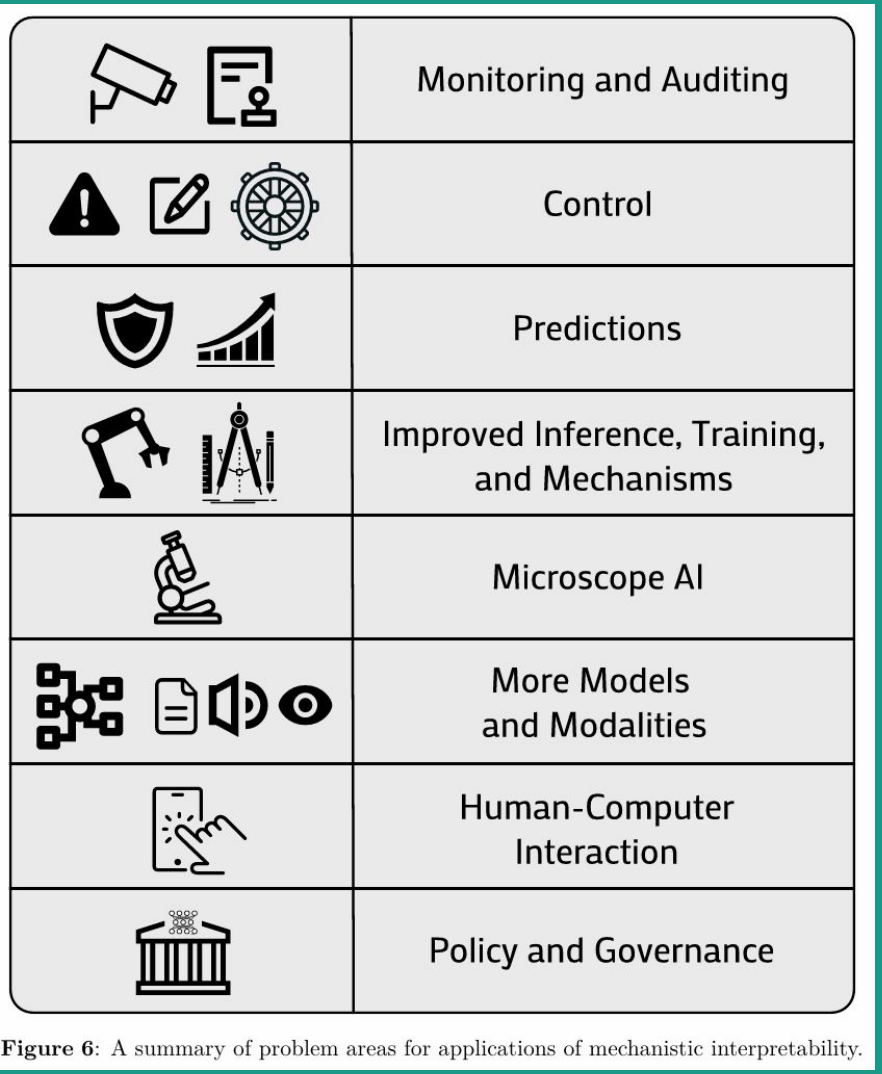


Figure 6: A summary of problem areas for applications of mechanistic interpretability.

- Using mechanistic interpretability for better
  - Predictions
  - Improvement
  - Extensions
  - Human computer interaction

Predictions about AI systems

Interpretability helps predict behavior in novel situations

- **Jailbreaking**
  By understanding the “jailbreaking” mechanisms within the model, it is possible to foresee how users might circumvent security protections. (Lee et al., 2025; Arditi et al., 2024)

- **Trojans/Backdoors**
  A backdoor inside the model will trigger unexpected behavior when encountering certain activations. (Casper et al., 2023)

- **Other behavioral indicators**

Predictions about AI systems

Interpretability helps predict capabilities that arise during training or finetuning

- **Black box** (OpenAI et al., 2024; Anthropic, 2024)
- **Model scale** (Wei et al., 2022; Schaeffer et al., 2023)
- **Phased capability transition** (Michaud et al., 2023; Wang et al., 2024b; Park et al., 2024a)
- **Internal structure** (Olsson et al., 2022)
- **Training Data** (Berglund et al., 2024; Reddy, 2024)
- **Finetuning's two-way effect** (Prakash et al., 2024; Jain et al., 2024; Lee et al., 2025)

Using interpretability to improve inference and training

- **Accelerated Reasoning**
  - Optimize computational workflows by understanding internal generation processes
- **Improving model distillation**
  - Develop more efficient distillation methods to identify and fill in neglected functions
- **Optimizing the training process**
  - Improving the selection of training data through mechanism analysis
- **Modular design and reconfiguration**
  - Decompose the network into functional modules to facilitate customization of specific computing structures
- **Fault Intervention and Optimization**
  - Removing faulty reasoning mechanisms through internal intervention

Using mechanistic interpretability for“microscope AI”

Cases

- In the AlphaZero study, new chess game concepts were extracted through microscopic methods and taught to top international chess players (Schut et al., 2023).
- A CNN model was used to analyze defendant photos and judges’ verdicts to reveal the correlation between facial features and verdicts (Ludwig & Mullainathan, 2023).
- Analyze the CNN model and learn previously unknown cell morphological features to predict immune cell protein expression (Cooper et al., 2022).

Expanding the applicable model family

Current interpretability foucus on three major families of models:

- CNN-based image model
- BERT-based text model
- GPT-based text model

New model families:

- Image Field
  - Diffusion Model
  - Vision Transformers
- Language Field
  - RWKV
  - State Space Models（SSMs）

Can the current interpretability results be extended to other models and application scenarios?

Human computer interaction with model internals

- Control is closely related to understanding neural networks → Promote new human-computer interaction modes
- Helps users intuitively control and debug AI systems according to their needs

Practical application examples

- Real-time monitoring dashboard
  - Display internal features that affect the chatbot's responses and provide timely error warnings (Chen et al., 2024; Zou et al., 2023a; Viégas & Wattenberg, 2023)
- Hybrid interface
  - Combining direct manipulation with text interaction to provide users with richer control options (Carter & Nielsen, 2017)

Social and philosophical challenges

- Potential advantages and limitations of interpretive techniques
  - Deep insights into AI models are expected to advance scientific understanding and control
  - However, most current tools have not yet truly used this capability to improve actual system security.

- Lack of unified paradigm and goals

- Consider the development and deployment context

Conclusion

Mechanistic interpretation has made significant progress in methods and applications, but major challenges remain to achieve many ambitious goals ！

Questions?

Thank you!