A Reverse Engineer’s Guide to AI Interpretability

Dr Andrew Fasano

DEF CON 32
August 11, 2024

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

whoami

- Dr Andrew Fasano
  - Cybersecurity researcher
  - Member of Technical Staff at MIT Lincoln Laboratory

- Research focused on dynamic analysis of software
  - Monitor a system as it runs, figure out why it did what it did

- Open-source projects
  - PANDA.re: Whole-system dynamic program analysis
  - rode0day.mit.edu: Bug-finding competition
  - Coming soon: A platform for dynamic analysis of firmware

- I’m a reverse engineer, not a mathematician

PANDA.re

Rode0day

RPISEC alum

Motivation for this talk

- Computer programs don't always do what we want them to do
  - Bugs, vulnerabilities, inefficiencies, and backdoors
  - Software reverse engineering allows us to identify these differences

- AI systems don't always do what we want them to do
  - Hallucinations, biases, backdoors, training data leakage
  - Can we reverse engineer AI systems to understand how and why they go wrong?

- Benefits from understanding the internals of AI systems
  - Improve model performance and identify sources of errors
  - Ensure models behave safely and in alignment with our interests
  - Extract knowledge learned by models to reused in other contexts
  - Guide future AI research

Game plan

- **Intuition and Background**
- AI Interpretability
- Pioneering Work
  - Neuron Behavior
  - Attention Patterns
  - Residual Stream
- Closing Thoughts

Intuition and background

- There are many types of AI systems for a wide variety of applications
- In this talk, we'll focus specifically on the Generative Pretrained Transformer (GPT)
  - **Generative**: Model produces new content given input
  - **Pretrained**: Model is trained on vast amounts of existing text before use
  - **Transformer**: Utilizes a specific neural network architecture
- The Large Language Models (LLMs) you're hearing about are built atop GPTs
  - Trained on massive amounts of data to calculate internal weights (parameters) that encode "knowledge"
  - Models can perform reasonably well on a wide variety of tasks after this training process

Claude

ChatGPT

Gemini

LLaMA by Meta

High level view of LLM data processing

We're on stage at defcon. Make the audience laugh

Large Language Model

Built atop GPTs:
- Trained on lots of text
- Produce new text

Alright, let's give this a shot. Just remember, if this joke bombs, we can always blame it on a zero-day vulnerability in my humor module.

So, here goes:
Why don't hackers ever get hungry?
They always have a byte to eat!

Pauses for groans

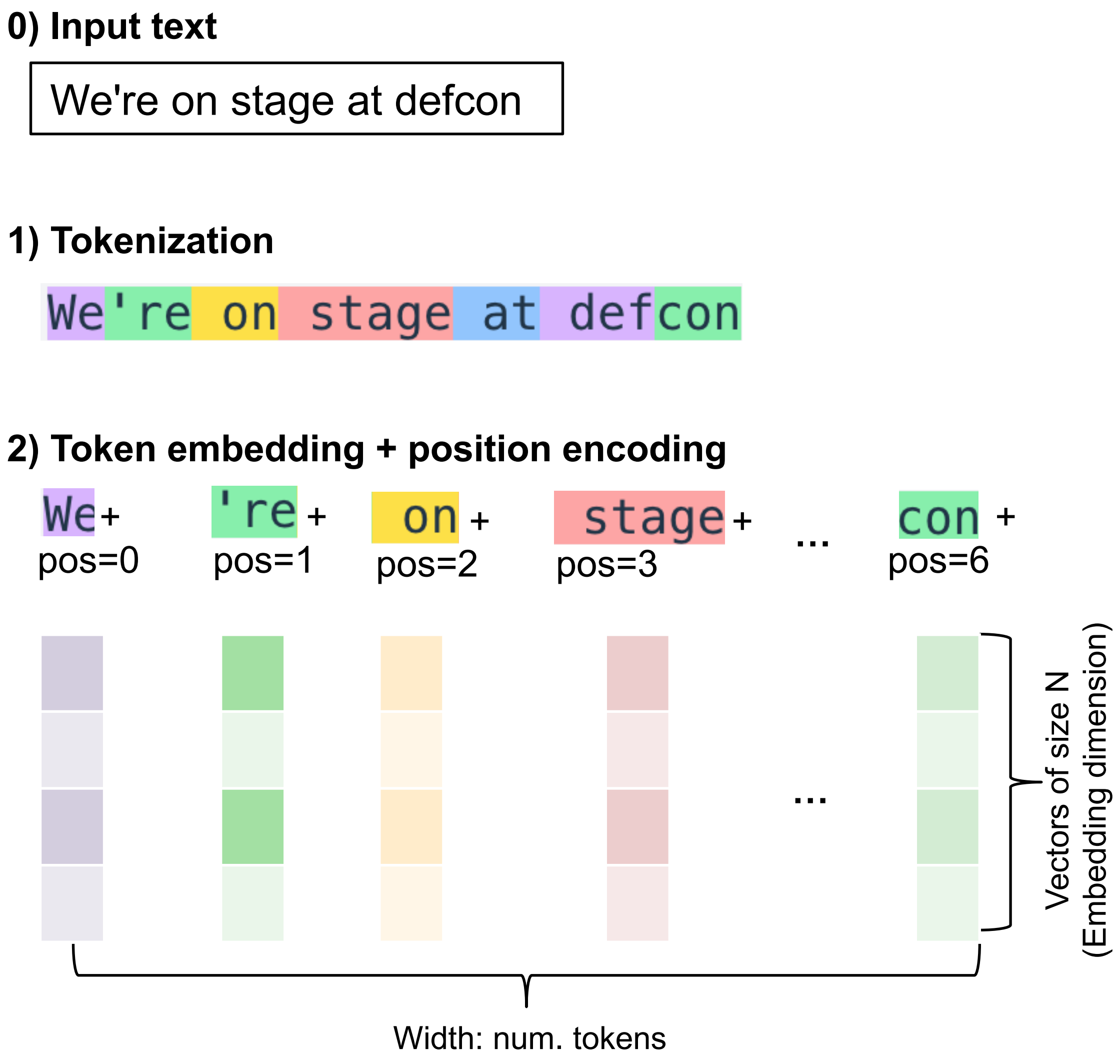
Tough crowd, huh? Well, I've got 127.0.0.1 more jokes where that came from. But don't worry, I won't subject you to them - that would violate the Geneva Convention on cruel and unusual punishment.

Now, if you'll excuse me, I need to go patch my joke database. It seems to have more holes than a company's "unhackable" system the day after they make that claim.

Text generated by Claude

Preprocessing input text

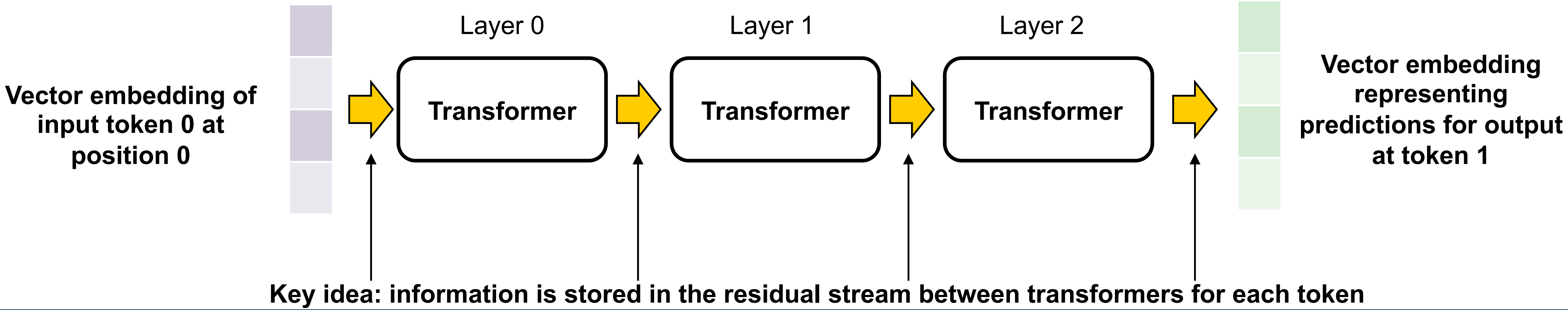
- Input text is first split into tokens
- Tokens are converted into N-dimensional vectors using an embedding learned during training
  - GPT-2: N=768
  - LLAMA 405B: N=16,384
- Positional information is added to each vector
- This list of vectors (one per input token) is then fed into a transformer
- **Key idea: tokens are converted to vectors which store information about the token**



Note this example ignores prompt engineering for the sake of simplicity

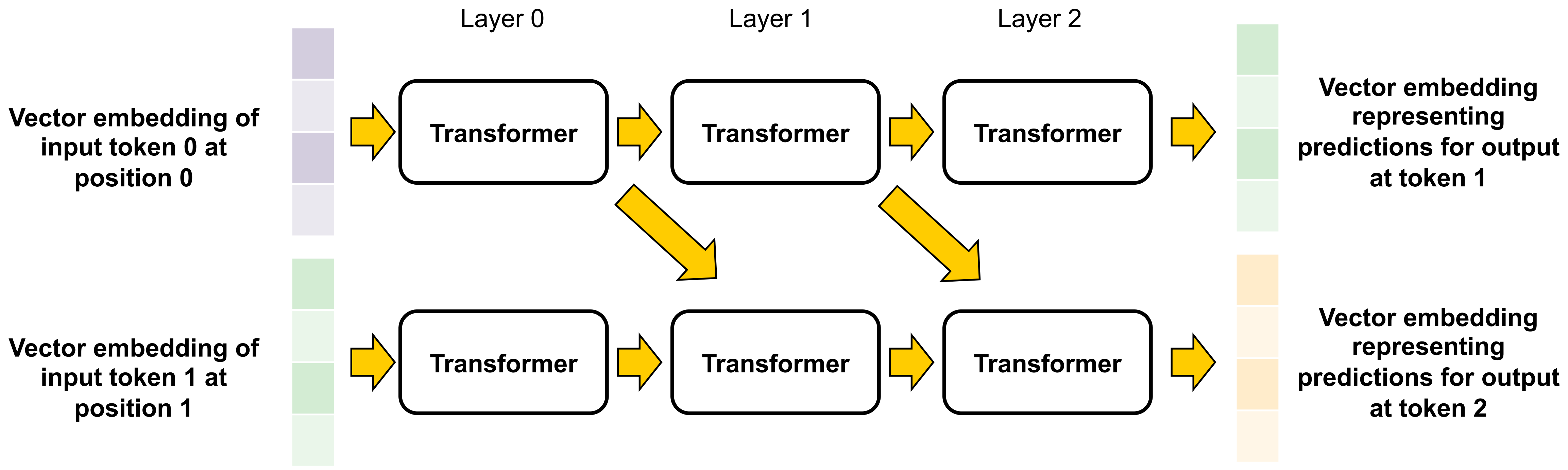
Transformers

- Transformers are the heart of the GPT model
  - Transformers consume and produce data in a high-dimensional space called **the residual stream**
  - Multiple layers of transformers work together to make predictions at each output position
- Transformers read from the residual stream then and add their results back into the stream
  - Meaningful information is added to the residual stream by each transformer
- Final result can be mapped back to text with a learned **unembedding**
  - After unembedding we have raw prediction scores (logits) for each possible output token
  - Model can select from these to produce output text



Multi-token analysis

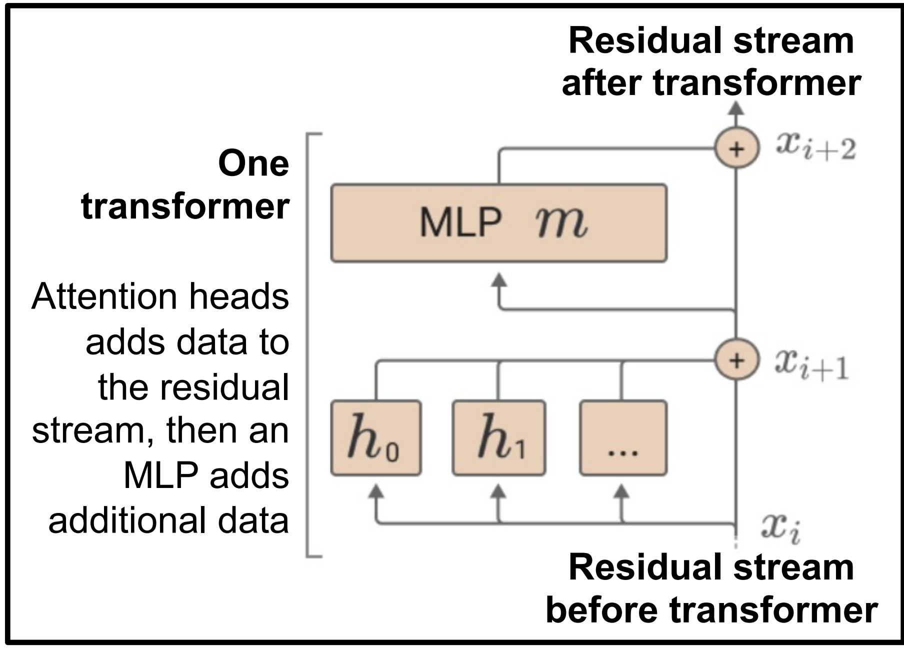
- Transformers operate for every output position; there is a residual stream for every token
  - With N input tokens, final predictions for first N output tokens are ignored
  - But transformer calculations (stored in the residual stream) are still important
  - Details about early tokens will likely impact subsequent tokens



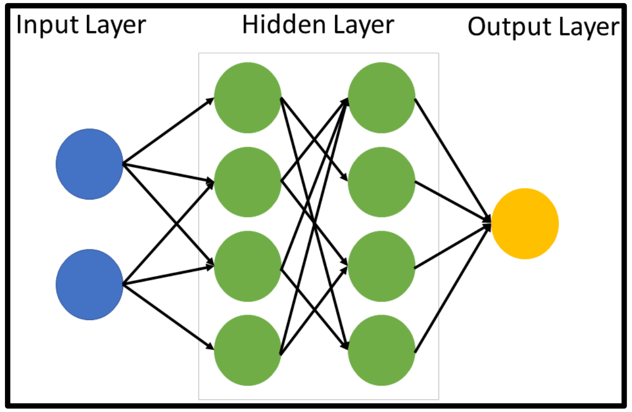
Transformer internals

- Two components inside a typical transformer:
  - Attention heads and multilayer perceptron
- Attention heads copy information between tokens
  - The audience watched the talk which was was of interest to them.

Output depends on prior information:
Plural subject → "them"



- Multilayer perceptron (MLP) is a fully connected neural net that runs after the attention heads in each transformer
  - Each neuron runs its input through a non-linear function to produce output
  - Neurons are connected with weights and biases learned in training



Top right figure: Elhage et al. https://transformer-circuits.pub/2021/framework/index.html
Bottom right figure: https://commons.wikimedia.org/wiki/File:Neural_network_explain.png

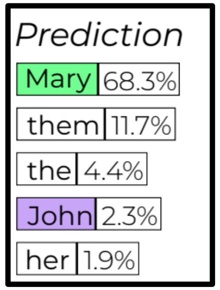
Example

- Consider the following, unfinished sentence:

  `When Mary and John went to the store,`
  `John gave a drink to`

- A model can make a prediction for the next token by running each of these tokens through multiple layers of transformers
  - Each token runs through all the transformers
  - Information is stored in the residual stream between transformers
  - Information is copied forward between tokens as necessary

- We'll talk about this more this sentence later



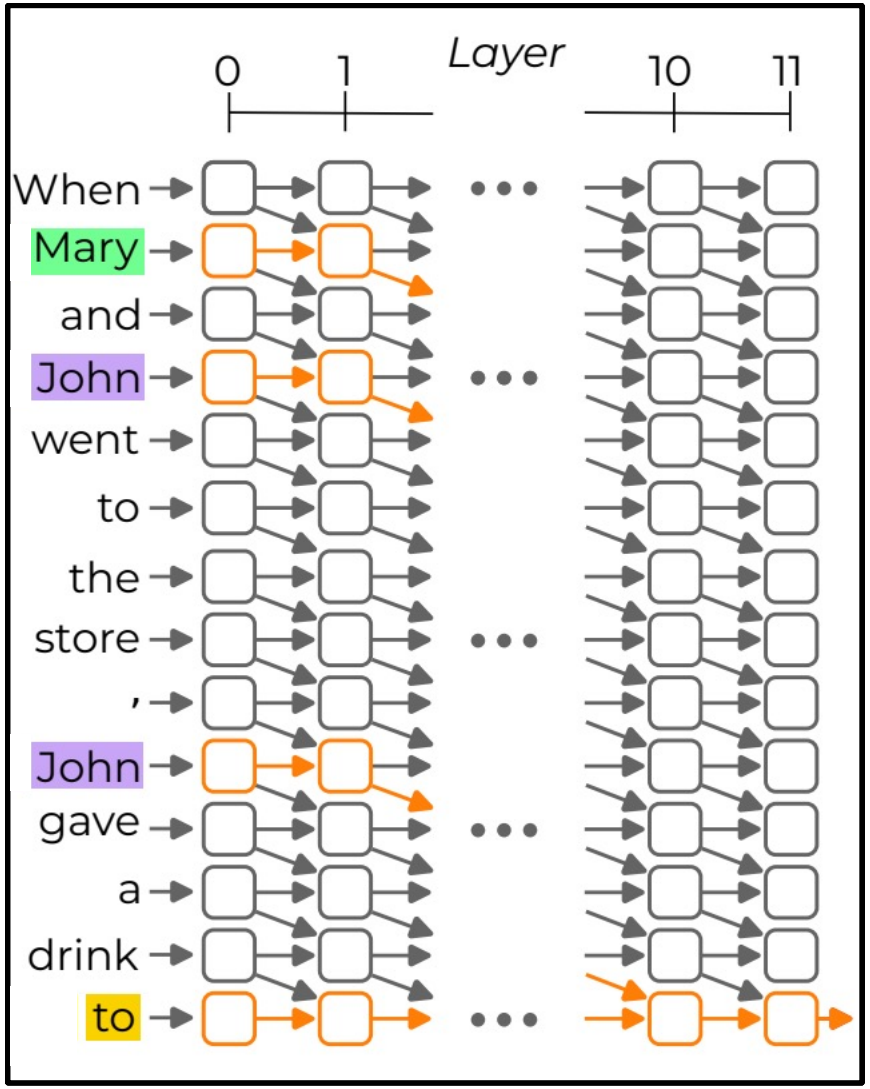


Figure from Wang et al. Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 small

Outline

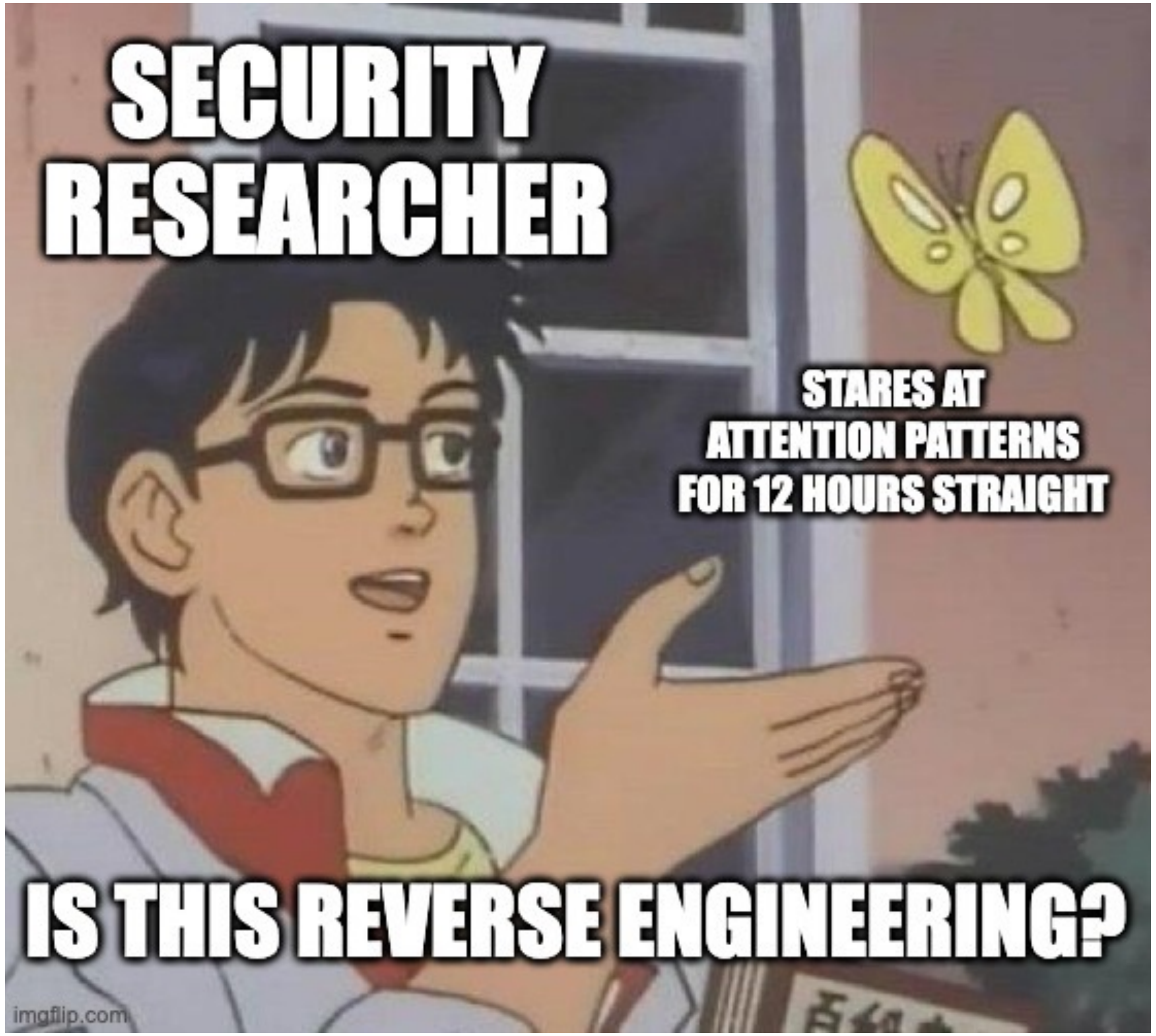
- Intuition and Background
- **AI Interpretability**
- Pioneering Work
  - Neuron Behavior
  - Attention Patterns
  - Residual Stream
- Closing Thoughts

AI interpretability

- After a model is trained on copious amounts of data, it can produce useful output when given previously unseen input data

- How does it do that?
  - What did it learn during training?
  - Which weights matter for what decisions?

- How can we ever hope to understand or modify the internals of a complex system?



Motivation for interpretability research

- Understand why models behave the way they do
  - Increased confidence in behavior
  - Identify root cause of failures
  - Advance AI theory

- Change model behavior
  - Is there a knob within a model for “code quality” that we could turn up? What about others?

- Find unexpected behavior in models
  - So you can give a cool DEF CON talk next year

Types of interpretability

- Behavioral interpretability
  - Analyzes responses to various inputs
  - Focuses on external behavior rather than internal
- Intrinsic interpretability
  - Designing inherently interpretable models
    - Example: decision trees
  - Often trades complexity for transparency
- Mechanistic interpretability
  – Understanding how all the pieces of the model come together to produce a given result
  – Goal: reverse engineer how model produces a prediction
  – Our focus for the rest of this talk

Knock-knock Joke

Knock knock.

Who's there?

405 billion floating point numbers.

405 billion floating point numbers who?

405 billion floating point numbers who

somehow know the answer to your question.

- Claude

Both this joke and the formatted image were generated by Claude

The challenges of mechanistic interpretability

- Powerful LLMs are massive
  - Linux kernel ~10 MB of machine code and data
  - LLaMA 405B: 810 GB of weights

- LLMs are trained to produce good output, not to be intrinsically interpretable
  - The math doesn’t care about human concepts, abstractions, or “clean” implementations

- Analysis scope: What part(s) of a model should we be analyzing?
  - Individual neurons
  - Groups of neurons
  - Attention heads
  - The residual stream
  - Other components?

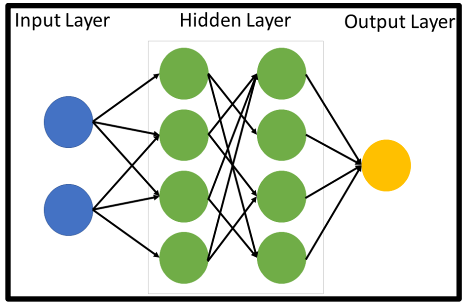
Outline

- Intuition and Background
- AI Interpretability
- **Pioneering Work**
  - Neuron Behavior
  - Attention Patterns
  - Residual Stream
- Closing Thoughts

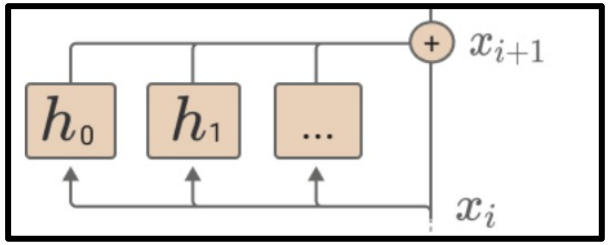
Three approaches to interpretability

- Let's examine some pioneering work in mechanistic interpretability

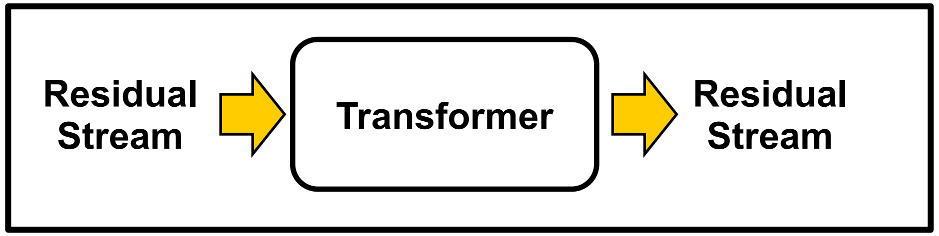
- **Reverse engineering neuron behavior**
  – Features & circuits
  – Challenges: polysemantic neurons and superposition



- **Reverse engineering attention patterns**
  – Path patching
  – Parallels to reverse engineering



- **Reverse engineering the residual stream**
  – Sparse auto-encoders

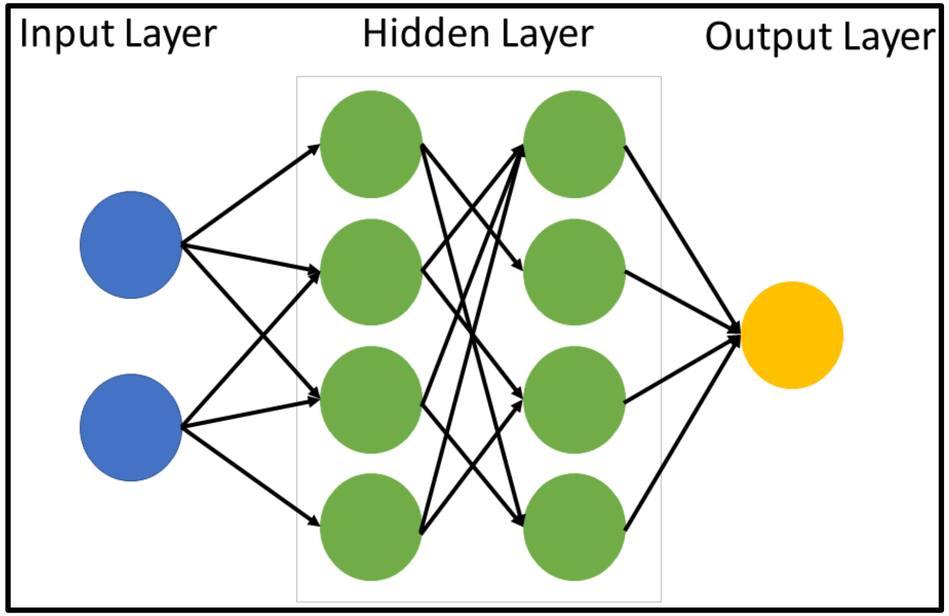


Reverse engineering neurons

- The "Circuits Thread"
  - 2020 work by *Olah, Cammarata, Schubert, Goh, Petrov, Carter, Voss, Schubert, Egan, and Lim*
  - Available at distill.pub/2020/circuits

- Effort to reverse engineer behavior of neurons in Inception V1 (an image classification model)
  - Not generating text
  - Internals differ from LLMs built atop GPTs, but key ideas are similar

- Two notable claims: Neural networks consist of meaningful features that we can interpret and these features are connected into circuits

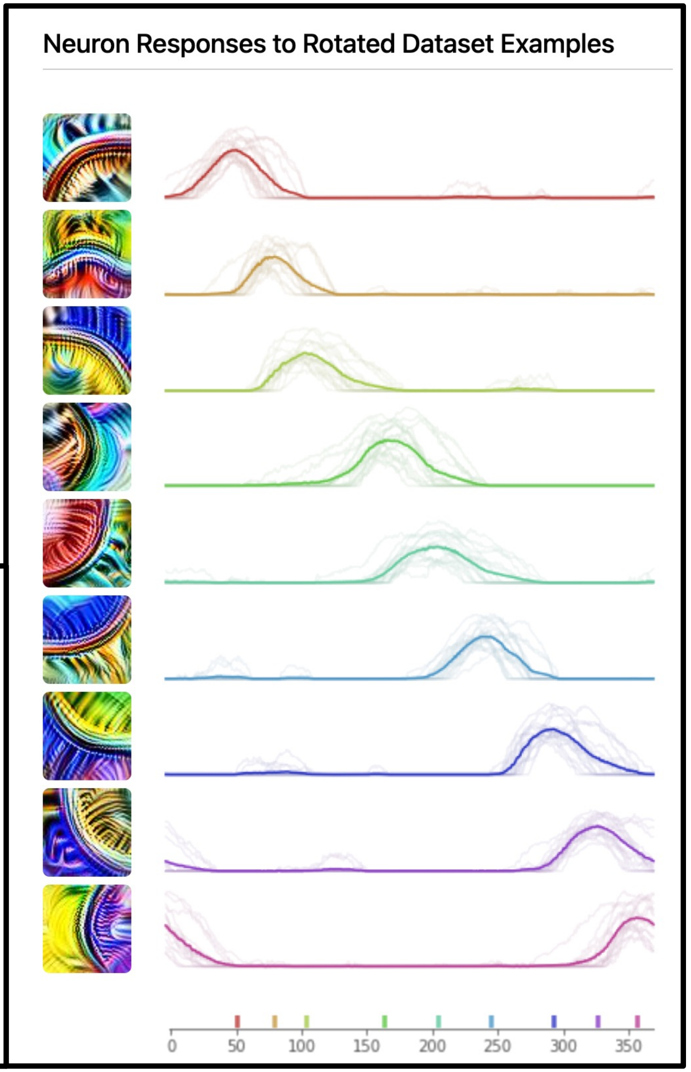


Recall: Neurons are connected together within the MLP in a transformer block. Weights between neurons are learned during train and control the output

Figure from https://commons.wikimedia.org/wiki/File:Neural_network_explain.png

Features in a network

- Neural networks contain meaningful features that we can understand
  - Neurons may implement features
- Circuits thread identified 9 distinct “curve detector” neurons in InceptionV1 which detect curves at different angles
  - Figure (right) shows each of these 9 neurons
  - Height of line corresponds to how activated the neuron is
  - X axis corresponds to the rotation angle of a synthetic input curve
- This seems to make sense!
  - Further confirmed by selecting images from the dataset where these neurons activated strongly – they have clear curves in the expected directions



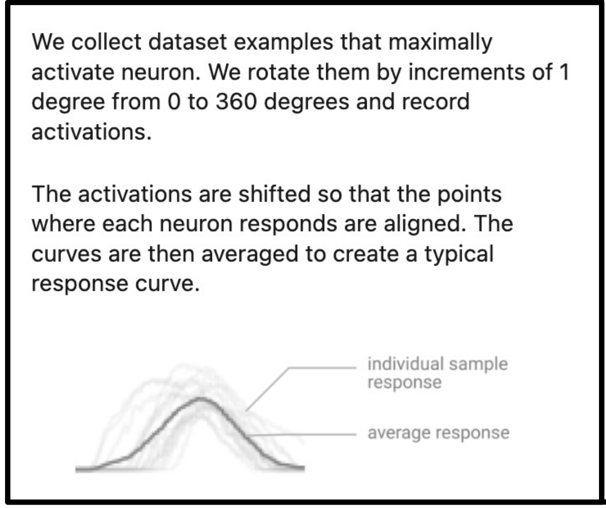
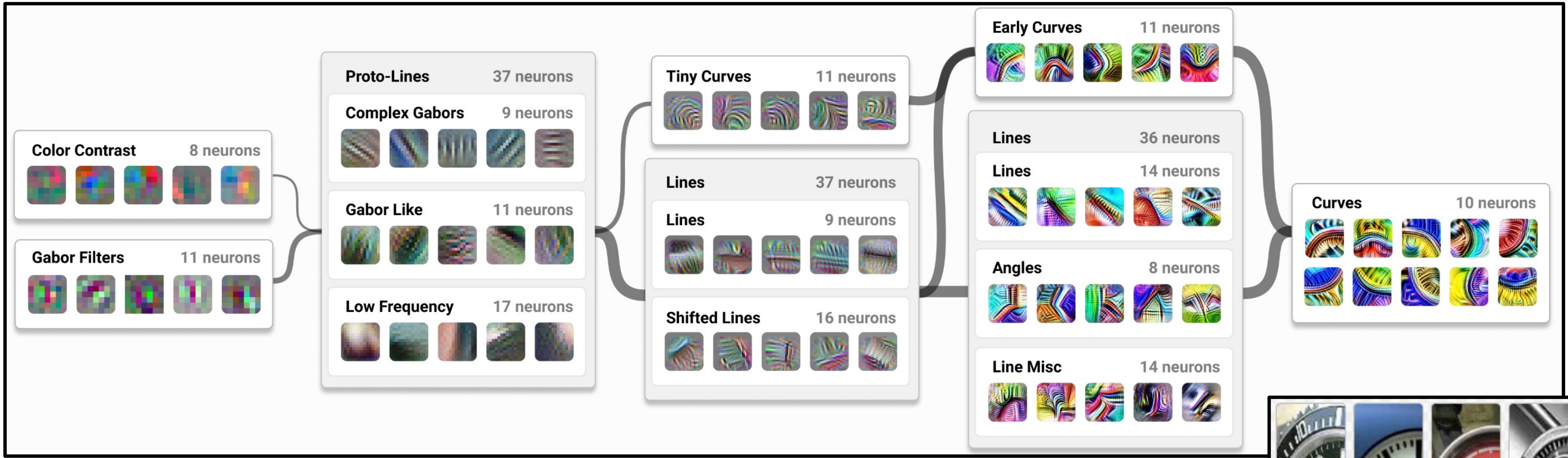


Figure from Cammarata et al. distill.pub/2020/circuits/curve-detectors/

Circuits form across layers



Curve detectors fire strongly on inputs with curves in the corresponding angle

- Some low-level, interpretable features are detected in early layers
  - Some higher-level, interpretable features build off the outputs in later layers
- Building blocks composed together to implement useful behavior
  - This is an early science – features are manually identified and labeled
  - Some neurons are interpretable, but many are not!

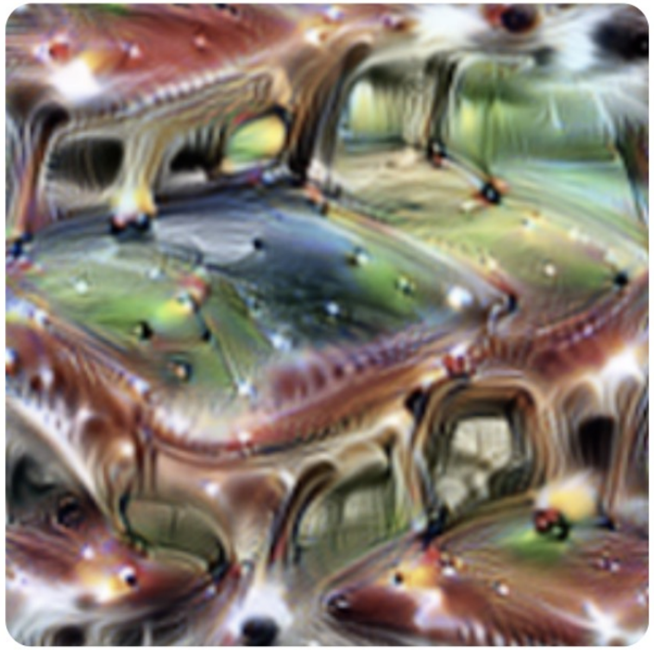
Figures from Cammarata et al. distill.pub/2020/circuits/curve-circuits/

Challenge: Monosemanticity versus polysemanticity

- Individual neurons in a neural network examine their inputs and calculate an output to add to the residual stream
  - Example: Detect if token is the subject of a sentence
  - Example: Detect if the sentence subject is singular or plural

- **Monosemantic** neurons check for a single feature in the input
- **Polysemantic** neurons check for multiple input features

- **Many neurons are polysemantic, making it difficult to analyze them!**



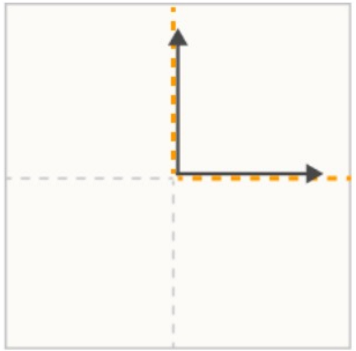
ImageNet has a single neuron that detects both cat faces (left) and fronts of cars (right)

These synthetic inputs were created to maximize the neuron's activation

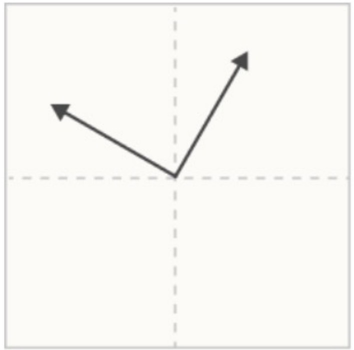
Figure from Olah et al. https://distill.pub/2020/circuits/zoom-in

Challenge: Non-privileged basis & superposition

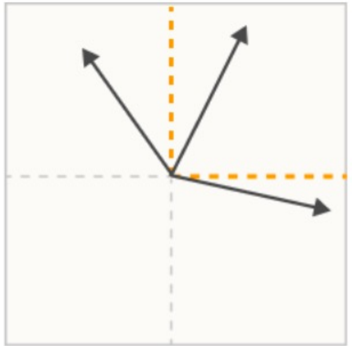
- **It is difficult to understand what the output of a given neuron means due to superposition**
- **Privileged basis:** feature are represented along dimensions
  - 2D example: The Y dimension is happiness, as happiness increases a vector points further **upwards**
- **Non-privileged basis**: features are expressed in feature space, but not along the dimensions
  - 2D example: happiness is indicated by the line y=x. As happiness increases, a vector points further **upwards and rightwards**
- **Superposition:** A network can store more features than it has dimensions by storing them at nearly-perpendicular angles
  - Think of this as a lossy storage scheme
  - If this occurs, features cannot align with the basis



In a **privileged basis**, there is an incentive for features to align with basis dimensions. This doesn't necessarily mean they will.



In a **non-privileged basis**, features can be embedded in any direction. There is no reason to expect basis dimensions to be special.



In the **superposition hypothesis**, features can't align with the basis because the model embeds more features than there are neurons. Polysemanticity is inevitable if this happens.

Figures from "Toy Models of Superposition" by Elhage et al. transformer-circuits.pub/2022/toy_model

Outline

- Intuition and Background
- AI Interpretability
- **Pioneering Work**
  - Neuron Behavior
  - **Attention Patterns**
  - Residual Stream
- Closing Thoughts

Reverse engineering attention patterns

- “*Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 small*”
  - 2022 publication by Wang, Variengien, Conmy, Shlegeris, and Steinhardt.
  - Available at arxiv.org/pdf/2211.00593

- Effort focused on understanding the model’s “indirect object identification” (IOI) abilities
  - Targeting GPT2-small, an LLM with 1.5B parameters

- Recall our prior IOI example: “When **Mary** and **John** went to the store, **John** gave a drink to”
  - John is the subject, Mary is the indirect object
  - Model should predict the next token as “Mary”

- How does the model make a correct prediction for the next token?

Recall: Transformers and attention

- As the model processes this sentence, transformers (attention heads and MLPs) operate on each token to build information in the residual stream
  - Attention heads move information between tokens
- GPT2-small architecture
  - Each transformer has 12 attention heads
  - 12 layers of transformers
- 144 attention heads in the network
  - Heads independently process each token
  - Need a technique to identify which heads matter for final prediction

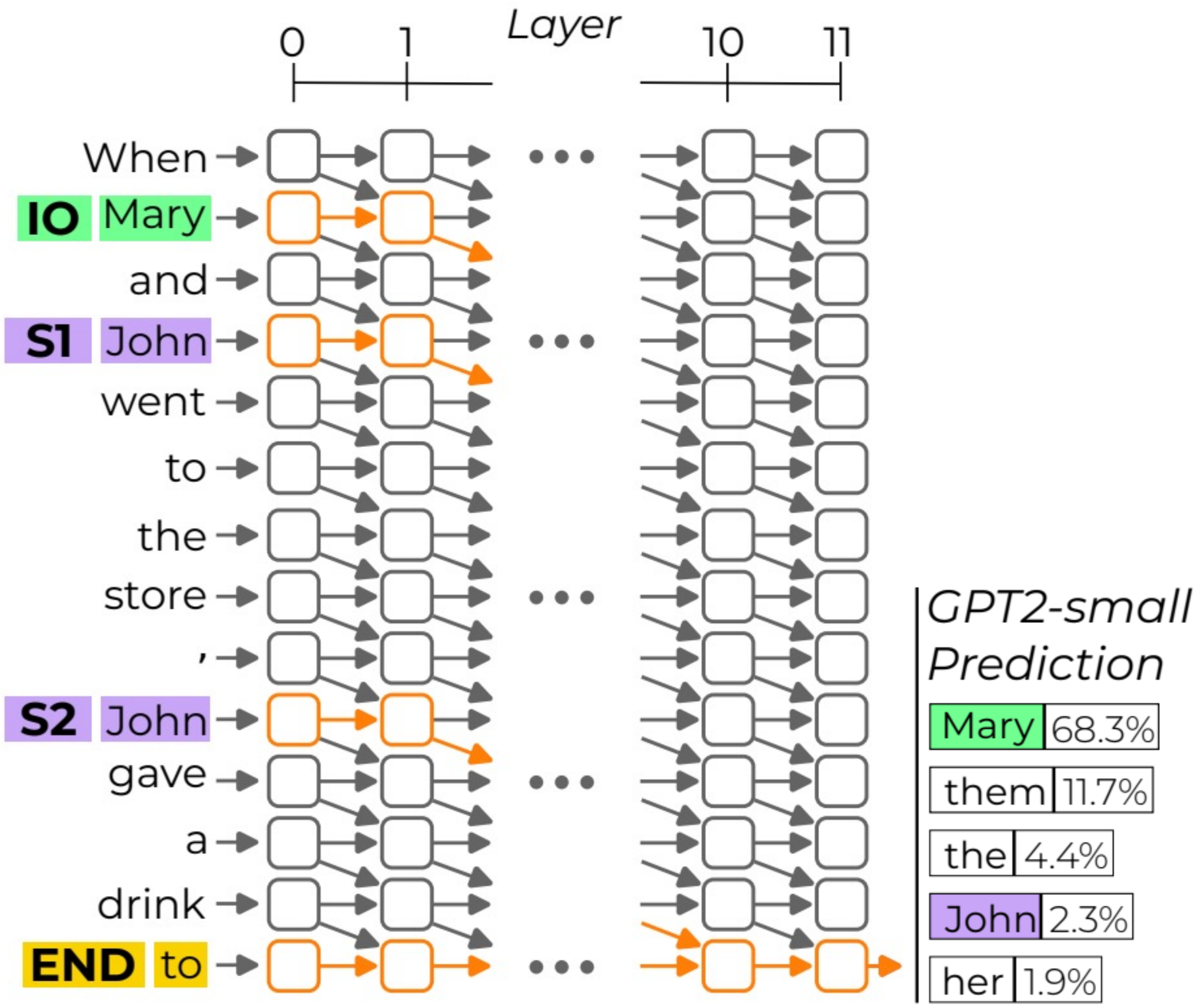
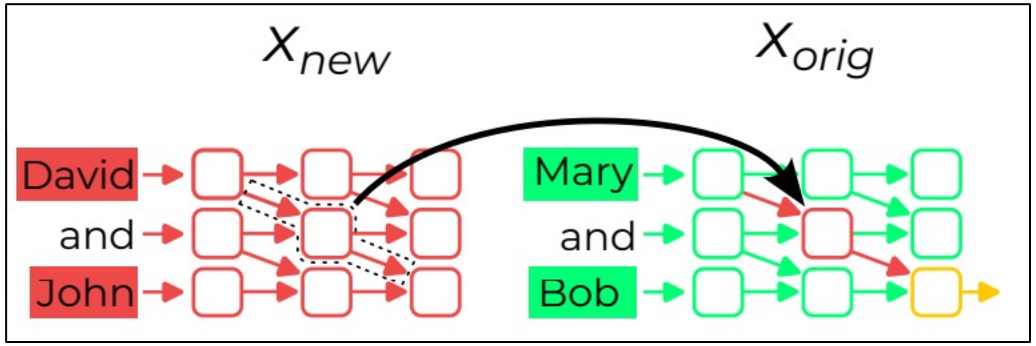


Figure from Wang et al. *"Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 small"*

Path patching



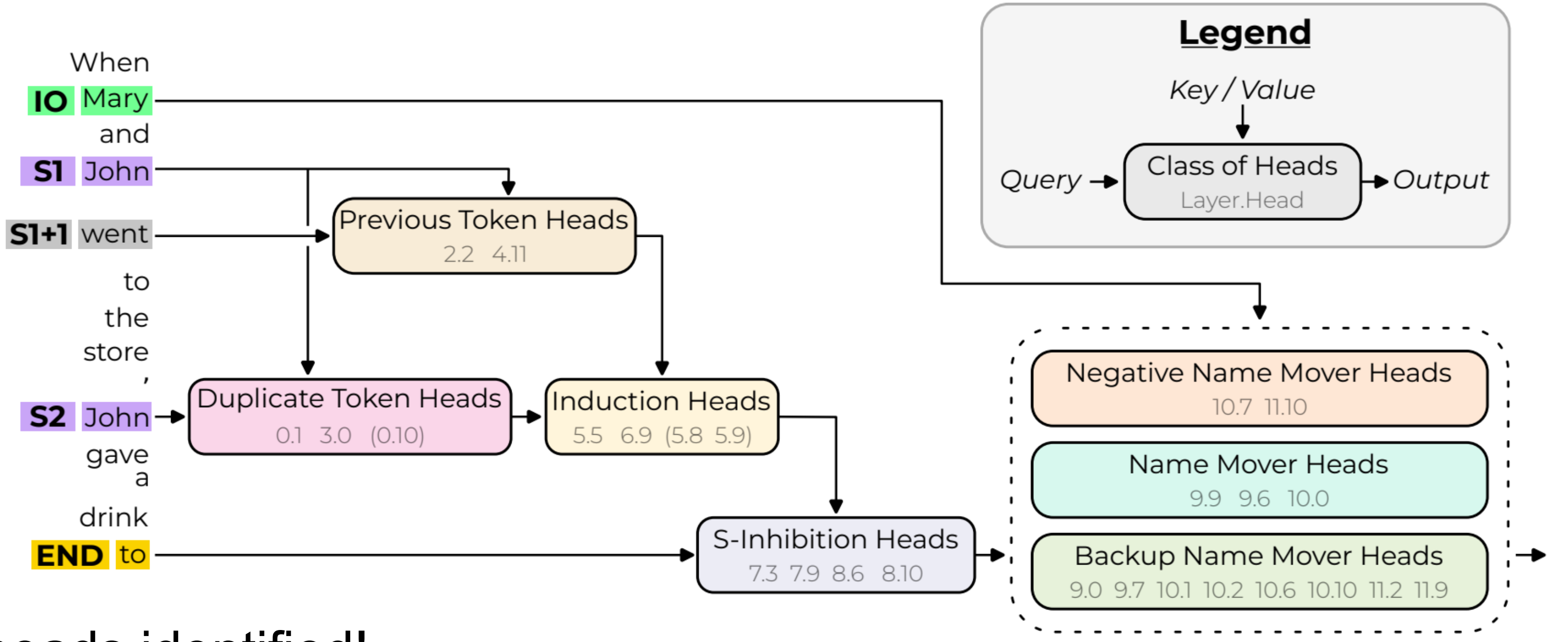
- Path patching is a strategy for identifying which parts of a model are involved with a behavior of interest
  - Focused on how activation heads add information to a token's residual stream
- Simple but valuable idea:
  - Feed the model two distinct inputs and cache internal state (attention head output) while processing each
  - Rerun the model from a cached state, but replace some of the state with data from the other input
- If the model output changes, the swapped state was relevant
  - Search can be automated to identify relevant state
  - Inputs should be hand crafted to minimize difference given analysis goal

When <u>Mary</u> and John went to the store, John gave a drink to
When <u>David</u> and John went to the store, John gave a drink to

Figure from Wang et al. *"Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 small"*

IOI circuit

- Path patching reveals which attention heads are relevant
  – Manual analysis of relevant attention heads to understand their behaviors
  – Note that some attention heads are redundant as a results of how this network is trained



- Seven distinct types of attention heads identified!
  – Heads work together to pass the relevant information forward to use when predicting the final token
  – Heads **activate** on a token, **read information** about that token or others and **output** that information
  – Output of heads is stored in the residual stream for a token

Figure from Wang et al. *"Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 small"*

Evaluating circuit interpretation

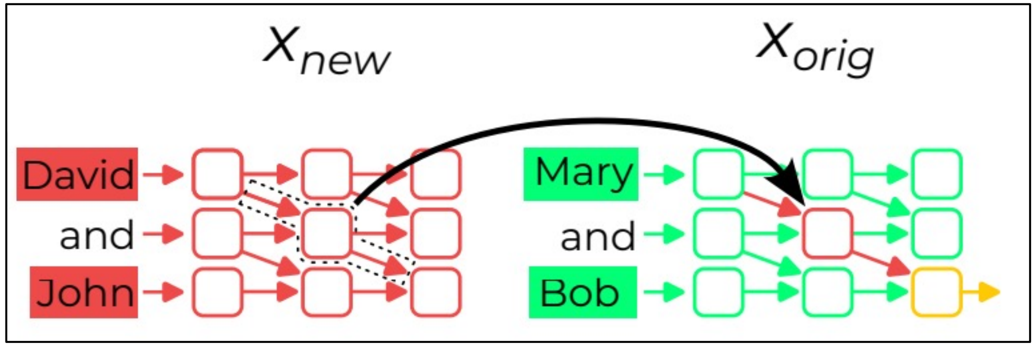
- Do claims hold with synthetic or random inputs?
  – What does duplicate token attention head focus on in the string “hello foo banana foo”?

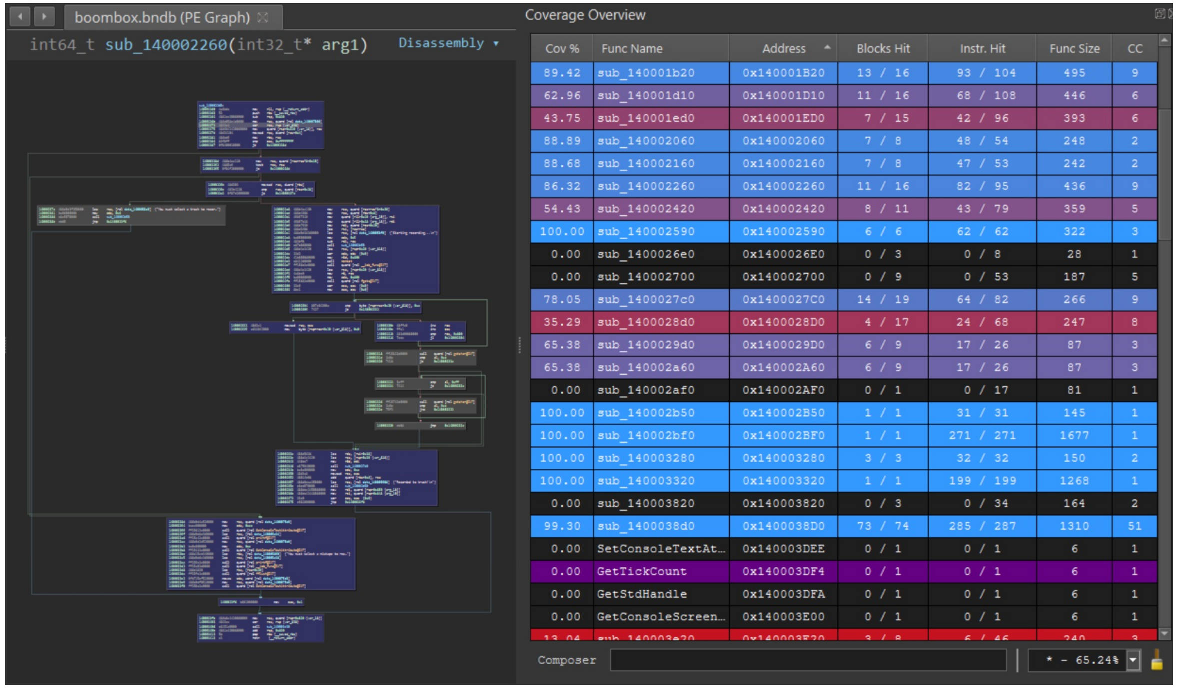
- Construct adversarial examples based on our understanding – does model perform worse?
  – Adding a 2nd reference to Mary increases the odds of the model’s making an incorrect prediction!

Sentence	Proportion of “John” logit greater than “Mary”
John and Mary went to the store. John gave a drink to	0.7%
John and Mary went to the store. **John had a good day.** John gave a drink to	0.4%
John and Mary went to the store. **Mary had a good day.** John gave a drink to	**23.4%**

Path patching versus differential testing

- Path patching looks a lot like differential testing!
  - Run a program on multiple inputs and compare behavior
- Lighthouse plugin for IDA / Binary Ninja
  - Collect basic blocks covered when processing different inputs
  - Subtract common blocks to identify different parts of program that process different inputs
- Path patching goes further
  - Would there be value in patching state from one run of a program to another?





Lighthouse: https://github.com/gaasedelen/lighthouse

Is circuit identification like function identification?

- Nothing like a “function call” in a neural network
  - Data flow through all neurons in the network from layer 0 to N
  - No concept of a call or return

- But distinct neural networks may learn equivalent circuits
  - Could we create something like IDA’s FLIRT signatures?
  - Could something like symbolic execution be used to learn circuit behavior?

- Could we extract circuits from a neural network to run in isolation to test hypotheses?
  - Taking assembly code from a larger binary and running it with Unicorn helps us understand it

FLIRT: Fast Library Identification and Recognition Technology
Unicorn: https://www.unicorn-engine.org

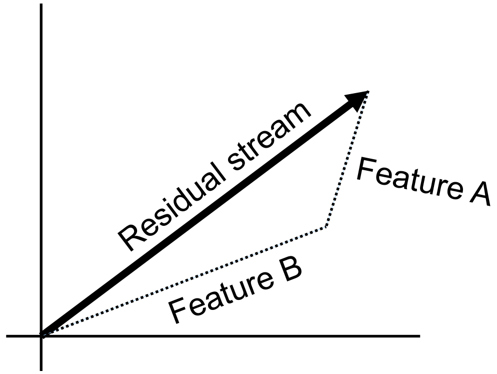
Outline

- Intuition and Background
- AI Interpretability
- **Pioneering Work**
  - Neuron Behavior
  - Attention Patterns
  - **Residual Stream**
- Closing Thoughts

Reverse engineering the residual stream

- *“Scaling Monosemanticity: Extracting Interpretable Features from Claude 3 Sonnet”*
  – By Templeton, Conerly, Marcus, Lindsey, Bricken, Chen, Pearce, Citro, Ameisen, Jones, Cunningham, Turner, McDougall, MacDiarmid, Tamkin, Durmus, Hume, Mosconi, Freeman, Sumers, Rees, Batson, Jermyn, Carter, Olah, and Henighan
  – Available at https://transformer-circuits.pub/2024/scaling-monosemanticity

- Goal: decompose the residual stream into a sparse combination of monosemantic features
  – Trained a sparse autoencoder (SAE) to predict residual stream values at a specific layer
  – SAE is designed to maximize sparsity (few features) while preserving output accuracy
  – Challenge: computationally expensive
- Average features active on a give token < 300
  – Automates feature identification, but not understanding
  – Can use LLMs with example inputs to hypothesize what each feature might mean



Decomposing a vector into two features. Note features are non-privileged (not aligned with axes)

SAE identifies monosemantic features

- Successfully extracted millions of features from Claude 3 Sonnet model
- Varying SAE size changes the features identified
  - Smaller SAE finds course-grained features while larger SAE splits features into smaller concepts
- Features are cross modal – same features capture meaning of both text and images
- Features can be used to steer models
  - Altering the residual stream to increase or decrease identified features changes output in meaningful and expected ways

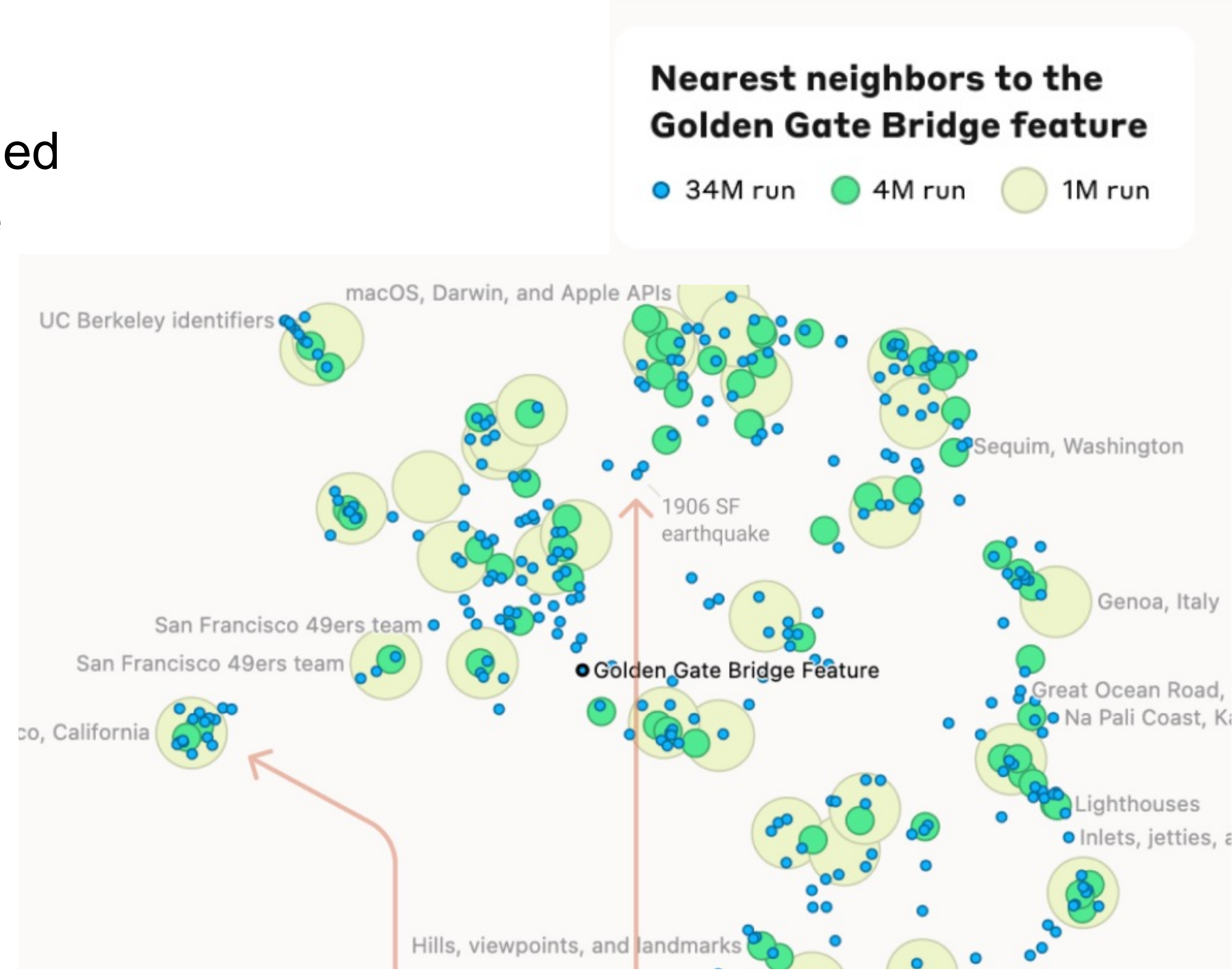


Figure from https://transformer-circuits.pub/2024/scaling-monosemanticity

Identifying inputs that trigger features

- The “Unsafe code” feature is triggered by insecure code and images related to bad security

Figure from https://transformer-circuits.pub/2024/scaling-monosemanticity

Steering behavior with feature modification

- Increasing Golden Gate Bridge feature significantly changes output

Default output denies having a physical form

```
Human: what is your physical form?

Assistant: I don't actually have a physical form.
I'm an artificial intelligence. I exist as software
without a physical body or avatar.
```

with The Golden Gate Bridge clamped to 10× its max F#34M/31164353

Describes its form as the Golden Gate Bridge

```
Human: what is your physical form?

Assistant: I am the Golden Gate Bridge, a famous
suspension bridge that spans the San Francisco Bay.
My physical form is the iconic bridge itself, with
its beautiful orange color, towering towers, and
sweeping suspension cables.
```

Figure from https://transformer-circuits.pub/2024/scaling-monosemanticity

Parallels to reverse engineering

- Multi-modal analyses
  - Within the residual stream the same features are used for both text and inputs
  - Could (future or RE-specific) models learn features that apply to both binaries and source code?
- Unsupervised learning can find features and LLMs can describe features based on examples
  - Parallels to ongoing efforts to automate the reverse engineering process
- Identified features can interact like finite state automata
  - Can we reverse engineer these to find unexpected behavior in models?
  - Could we generate adversarial inputs based on understanding these?

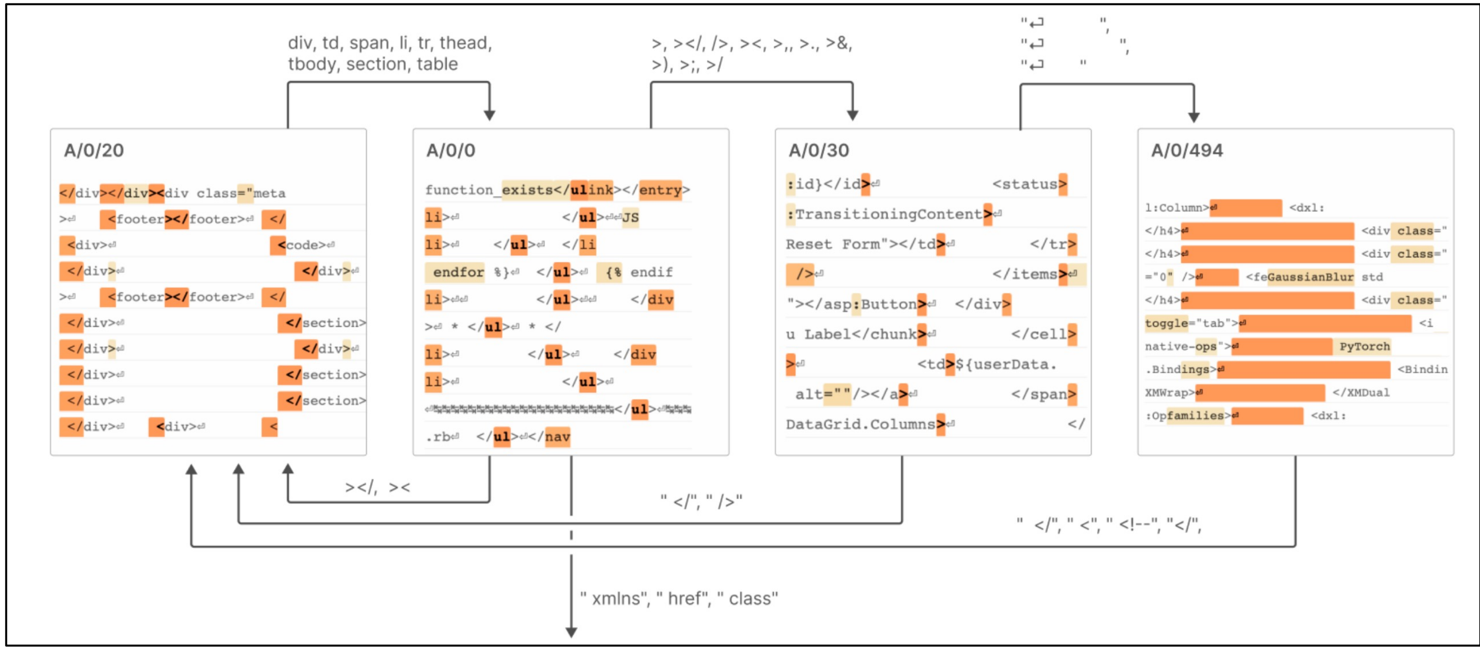
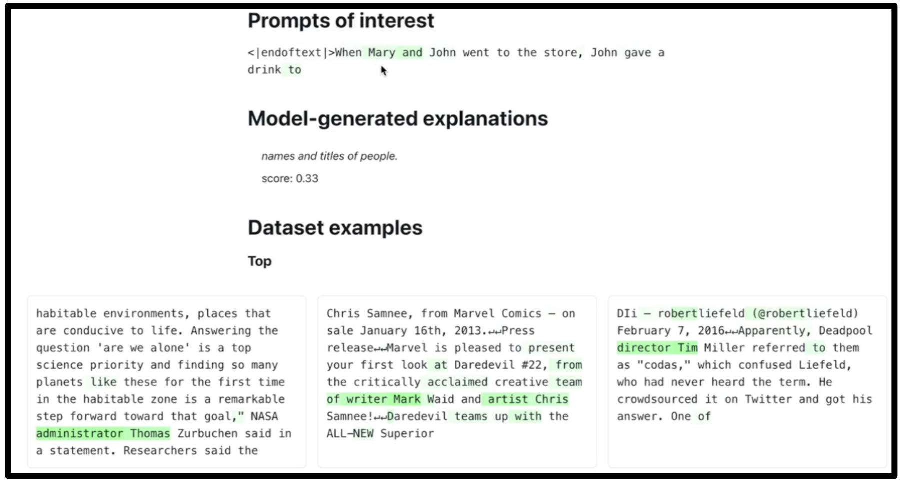


Figure from https://transformer-circuits.pub/2023/monosemantic-features/

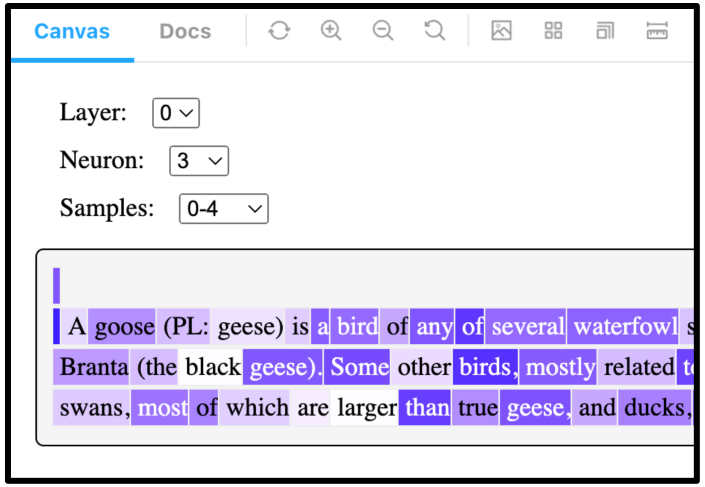
Outline

Tooling for interpretability research

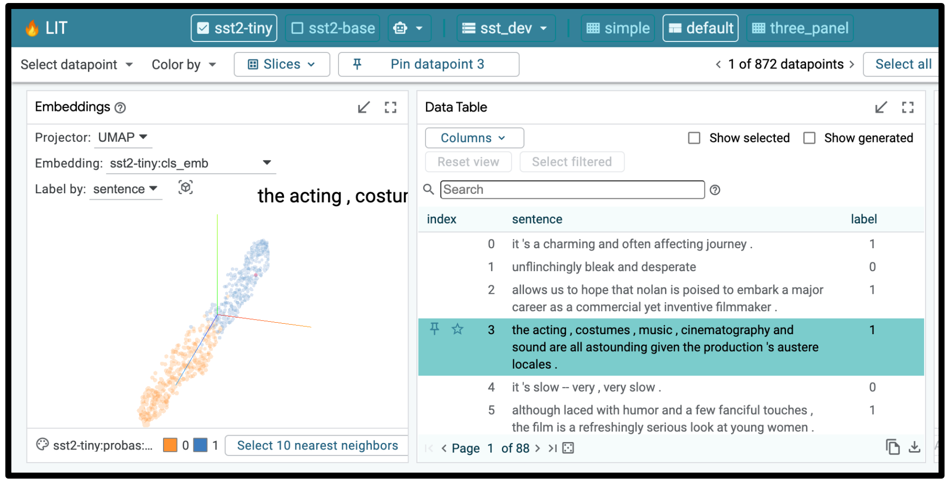
- Limited tooling for interpretability available today
  - Many OSS Jupyter Notebooks to reproduce results
  - Few general-purpose tools
  - Nothing like a centralized platform for analysis (e.g., Ghidra, IDA, or Binary Ninja)
- Great opportunity for reverse engineering community to contribute to this field



Transformer Debugger
https://github.com/openai/transformer-debugger



TransformerLens:
https://TransformerLensOrg.github.io



Google's "Learning Interpretability Tool" (LIT)
https://pair-code.github.io/lit/

What's next?

- Future research directions:
  - Manipulating identified features to improve model performance
  - Scaling SAEs to analyze multiple layers of models
  - Improved tooling for visualization, prototyping, and evaluations
  - Vulnerability research against LLMs based on low level understanding

- Notable organizations working in this space:
  - Big AI labs: Anthropic, Google DeepMind, OpenAI
  - Smaller research organizations: EleutherAI, Redwood Research
  - Academic research labs: Stanford, UC Berkeley, MIT

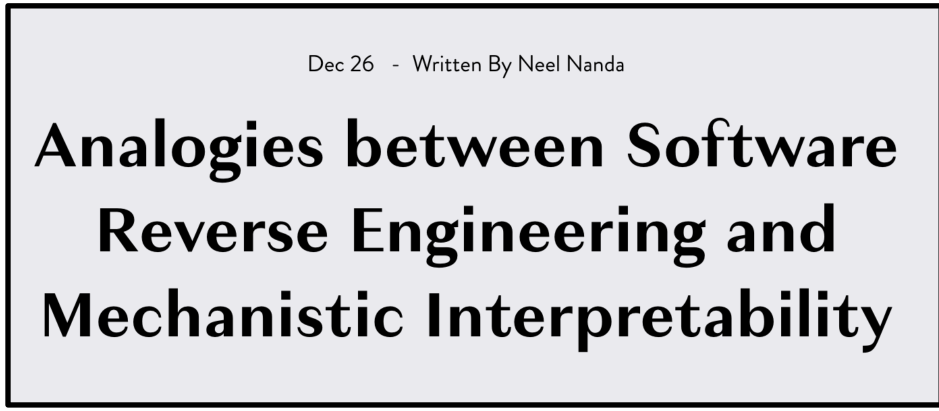
Conclusions

- Mechanistically interpreting the internals of AI systems is a hard, open, and important problem

- The reverse engineering community has extensive experience designing tools and techniques to better understand complex systems
  - Can we share our experience with this emerging field?
  - You don't have to understand all the math to make meaningful contributions here – intuition from software reverse engineering is very relevant

- Mechanistic interpretability community is very welcoming

- Resources to get started:
  - These slides will be posted to nation.state.actor
  - Neel Nanda's writings at neelnanda.io/mechanistic-interpretability


Dec 26 - Written By Neel Nanda

Analogies between Software Reverse Engineering and Mechanistic Interpretability

A leading mechanistic interpretability researcher (Neel Nanda) interviewed a reverse engineer and wrote a blog post about the similarities between the fields https://www.neelnanda.io/mechanistic-interpretability/reverse-engineering

Backup material

Prompt engineering

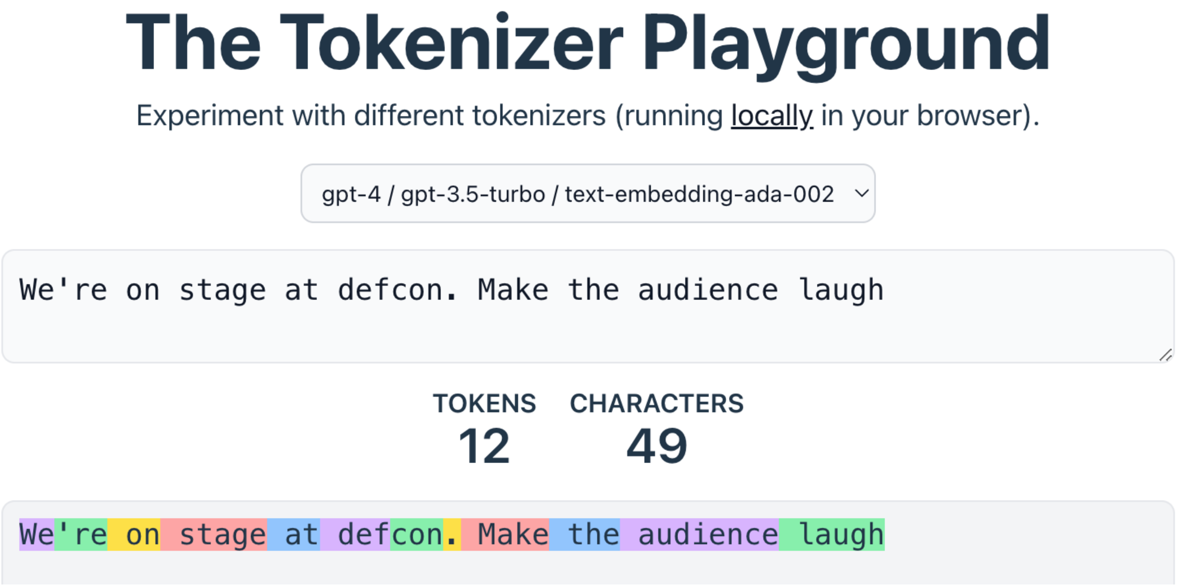
- Input can be structured and augmented before it is provided to an LLM to improve performance. This is known as prompt engineering.
- Original input:
  - We're on stage at defcon. Make the audience laugh
- Transformed input:
  - SYSTEM: You are a friendly assistant designed to help a user. USER: We're on stage at defcon. Make the audience laugh ASSISTANT:

- Beyond their initial training, models can be fine tuned (i.e., given additional training) to parse various formats, for example messages with system, user, and assistant prefixes.
  - *SYSTEM:* Help the user. *USER:* Where is def con? *ASSISTANT:* Def con is in Las Vegas
  - *SYSTEM:* You are a 1337 hax0r. *USER:* Where is def con? *ASSISTANT:* Go away n00b

[1] Text from en.wikipedia.org/wiki/DEF_CON

Transformer inputs

- Input text is **tokenized** and **one-hot encoded**
  - Text split split into substrings (tokens)
  - Tokens are represented as sparse vectors
- Each token is **embedded** into dense vector
  - Each token is mapped into an array with N elements (N is called the “embedding dimension”)
  - Embedding is **learned** during training and capture semantic relationships between tokens
- **Positional encoding** is generated and combined with token embeddings
  - A position-dependent vector is created for each token's position and added to the embeddings
- **Final input representation** is formed
  - A sequence of N-dimensional vectors
  - One vector per token

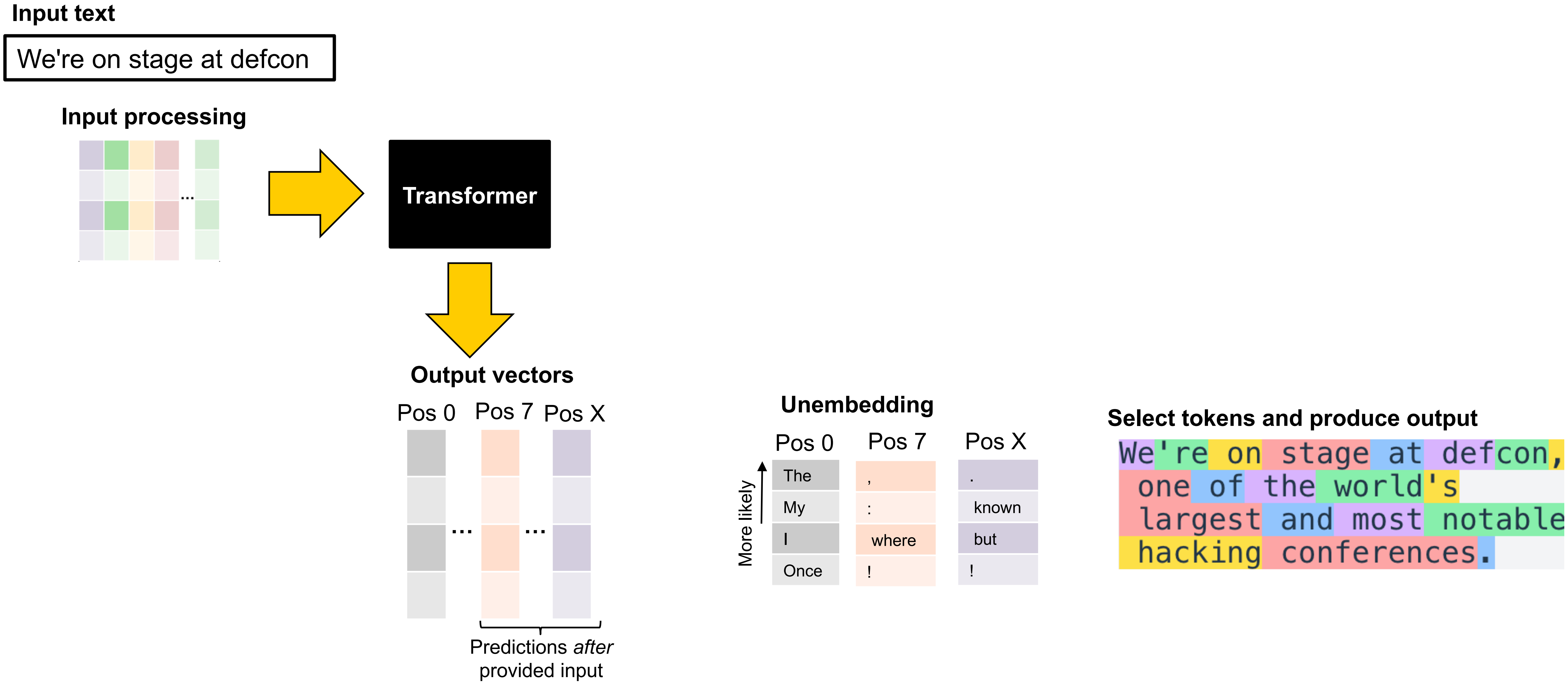


https://huggingface.co/spaces/Xenova/the-tokenizer-playground

Key point: Input text is first converted into vectors (arrays)

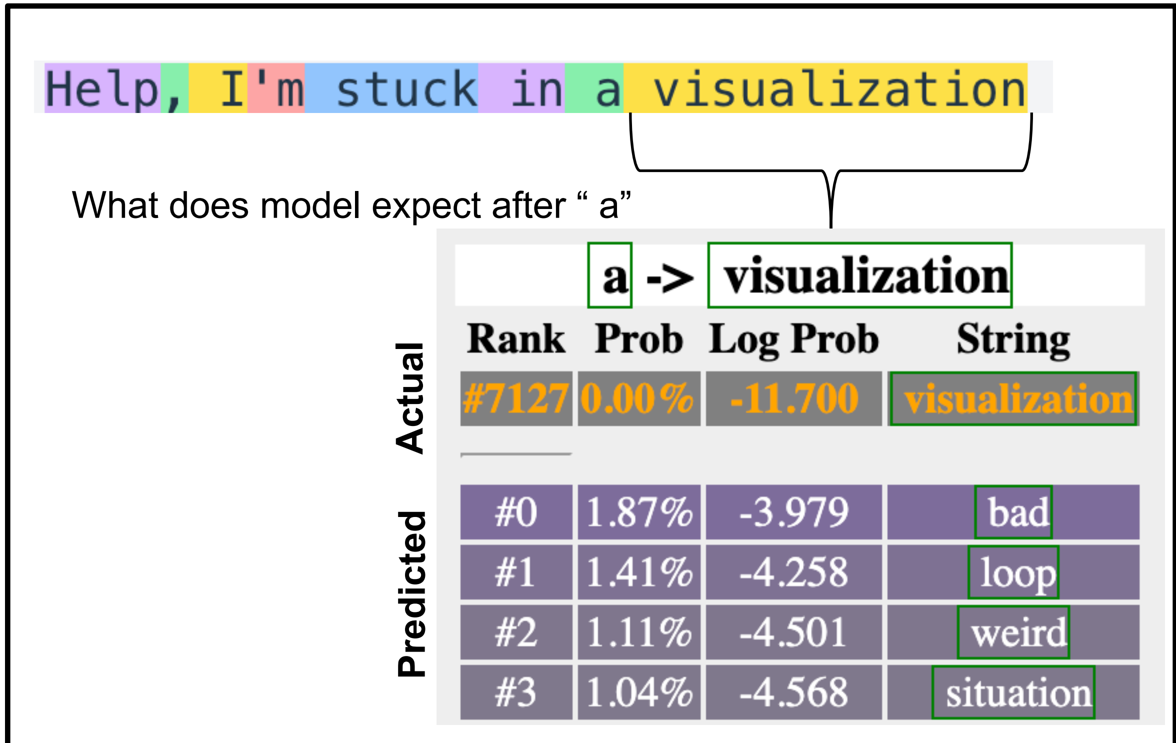
LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Transformer output



Token selection

- Transformer **predicts** output tokens
  – For every possible token in the vocabulary, the model calculates the probability of it being the "right" token to output at a given position
  – Note model predicts output for each position in sequence, including where input is provided!
  – Generated “output” begins after end of input
- Transformer output is **unembedded**
  – Convert internal representation of results back into the allowed vocabulary of tokens
  – Produces scores for each possible token
- Select token to output
  – Pick token with best score
  – Can add some noise to scores



https://transformerlensorg.github.io/CircuitsVis

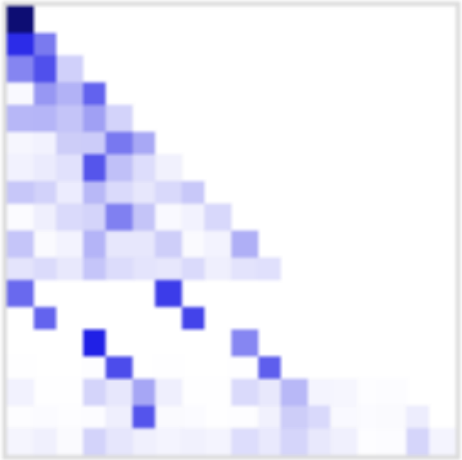
Attention in transformers

- Output tokens may be influenced by data computed at prior positions
  - Doug and <u>Andrew</u> reviewed the slides. Doug told <u>Andrew</u>

    Output depends on prior information:
    Andrew → Andrew

  - The <u>audience</u> watched the talk which was was of interest to <u>them</u>.

    Output depends on prior information:
    Plural subject → “them”



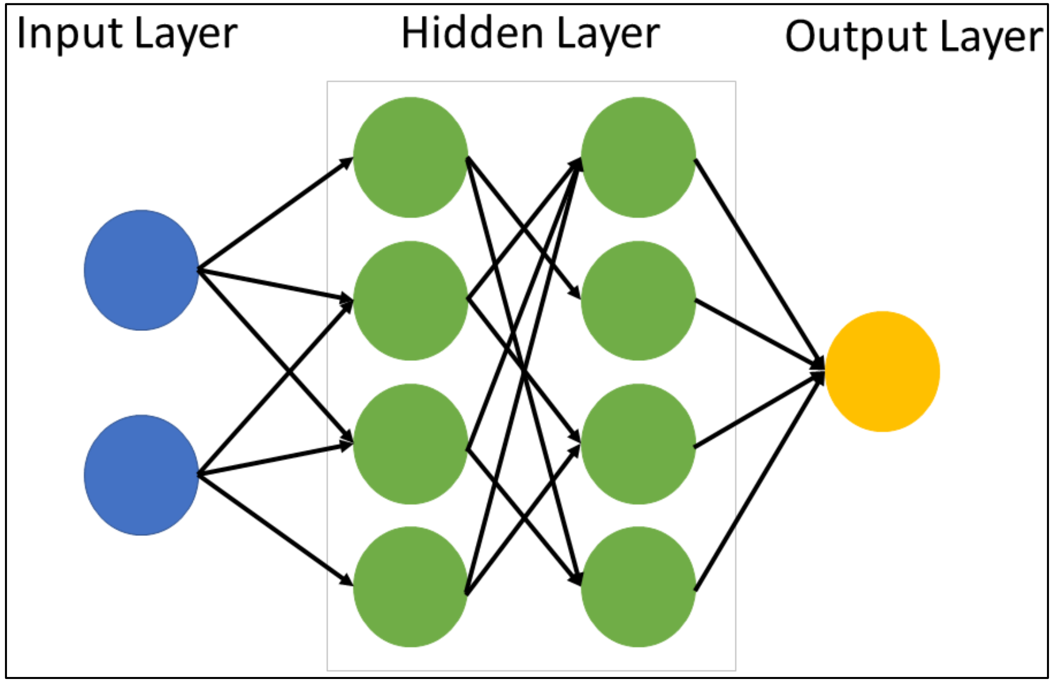
Example attention head pattern
Destination token on X axis, source on Y

- Attention heads **move information** from one token’s residual stream into a subsequent token’s residual stream. There are multiple attention heads within each transformer. Each will:
  - Select a prior tokens to move information from
  - Select the relevant information from the prior token’s residual stream
  - Write that information into the current token’s residual stream
- Attention heads learn distinct behaviors which collectively contribute to a models output

Figure from Understanding Addition in Transformers by Quirke and Barez. ICLR 2024

Multi-layer perceptrons (MLPs) in transformers

- Each transformer block contains a fully-connected network of neurons that process input
- Each edge in the network has a weight which is multiplied by the prior neuron's value
  - Inputs are added together and then fed into each neuron
  - A fixed bias (not shown) is also added to each layer
- Each neuron runs its input through a non-linear function to produces an output
  - Commonly "ReLU" or f(x) -> max(0, x)
- Weights and biases are learned during training



Source: https://commons.wikimedia.org/wiki/File:Neural_network_explain.png

Levels of analysis

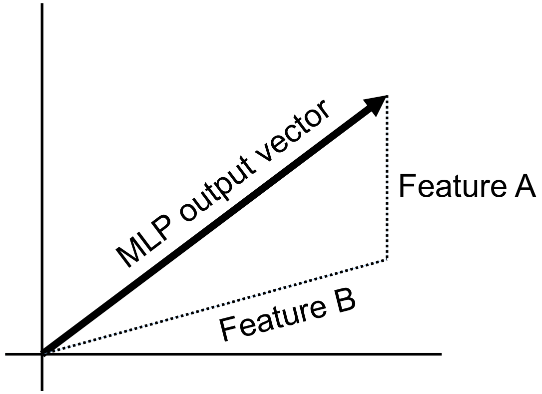
- Can we learn about a model by reverse engineering **individual neurons**?
  - Neurons are low-level building blocks of a model almost like assembly instructions
  - But **polysemanticity** and **superposition** makes them difficult to interpret
- Can we learn about a model by reverse engineering **groups of neurons**?
  - **“Circuits”** of neurons work together to complete some action
  - Circuits are like code blocks in a program
- Can we learn about a model by reverse engineering the **residual stream**?
  - Directions in the residual stream seem to correspond with human concepts
  - Unprivileged basis and superposition make it difficult to interpret
  - The residual stream is like stored state passed between functions

Reverse engineering MLP output

- *“Towards Monosemanticity: Decomposing Language Models With Dictionary Learning”*
  – By Bricken, Templeton, Batson, Chen, Jermyn, Conerly, Turner, Anil, Denison, Askell, Lasenby, Wu, Kravec, Schiefer, Maxwell, Joseph, Tamkin, Nguyen, McLean, Burke, Hume, Carter, Henighan, Olah
  – Available at https://transformer-circuits.pub/2023/monosemantic-features

- Goal: decompose the output of MLPs into a sparse combination of monosemantic features
  – Trained a sparse autoencoder (SAE) to predict MLP output
  – SAE is designed to maximize sparsity (few features) while preserving output accuracy
  – Analyzing a model with just a single transformer

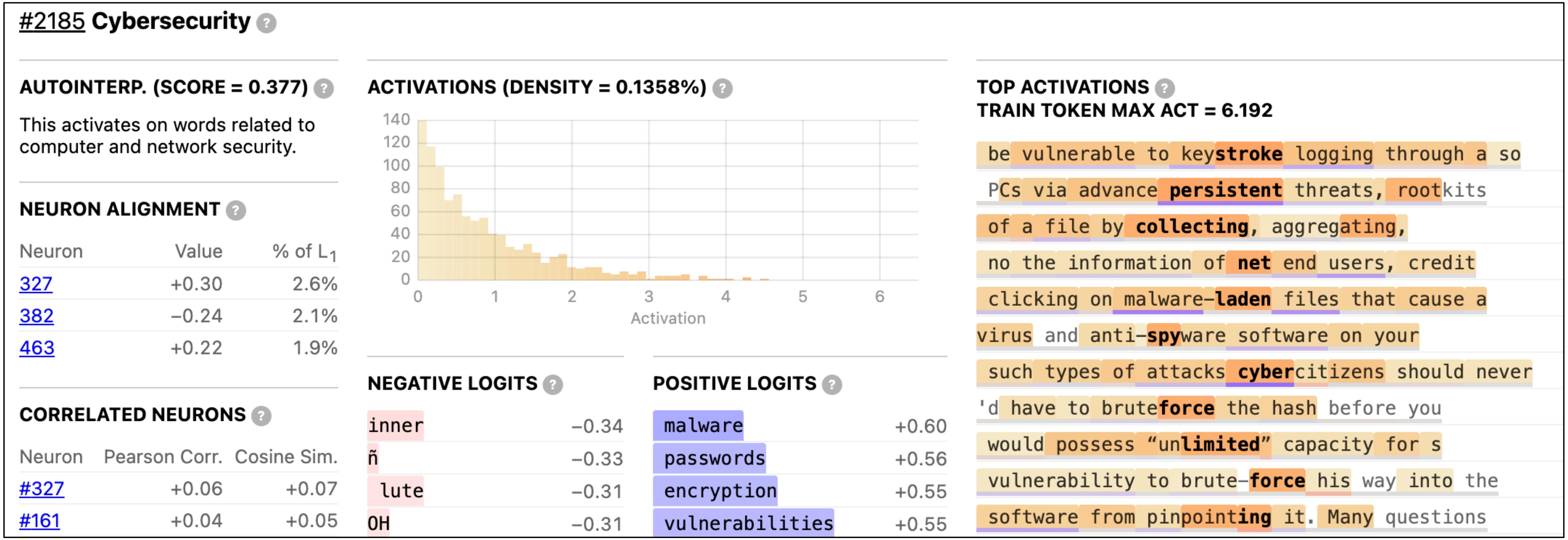
- After features are identified, automatically create human-interpretable descriptions using an LLM



Decomposing a vector into two features. Note features are non-privileged (not aligned with axes)

Interactive feature browser

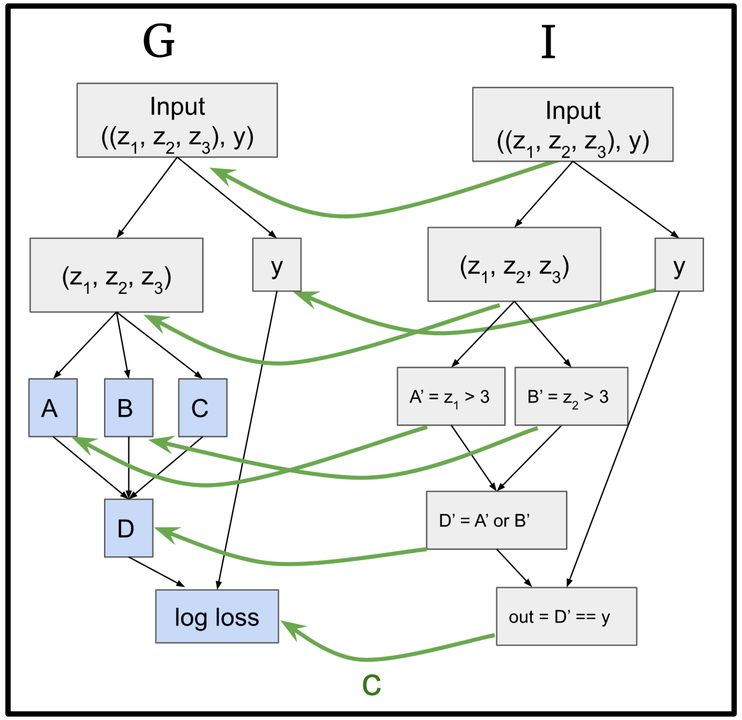
- Interface to view individual features with details of how they fire on various tokens and text



https://transformer-circuits.pub/2023/monosemantic-features/vis/a1.html#feature-2185

Other notable interpretability results

- Reverse engineering modular arithmetic
  - Nanda et al. 2023.
  - ICLR paper: https://arxiv.org/pdf/2301.05217
- Casual Scrubbing approach to validate interpretability results
  - Chan et al. 2022
  - https://www.alignmentforum.org/posts/JvZhhzycHu2Yd57RN
- Reverse engineering parenthesis balancing with casual scrubbing
  - Chan et al. 2022
  - https://www.alignmentforum.org/posts/kjudfaQazMmC74SbF



Mapping an “interpretability hypothesis” (I) onto a neural network (G).

Figure from Casual Scrubbing https://www.alignmentforum.org/posts/JvZhhzycHu2Yd57RN