

pyvene: A Library for Understanding and Improving PyTorch Models via Interventions

Zhengxuan Wu[†], Atticus Geiger[‡], Aryaman Arora[†], Jing Huang[†], Zheng Wang[†],
Noah D. Goodman[†], Christopher D. Manning[†], Christopher Potts[†]

[†]Stanford University [‡]Pr(Ai)²R Group

{wuzhengx, atticusg, aryamana, hij, peterwz, ngd, manning, cgpotts}@stanford.edu

Abstract

Interventions on model-internal states are fundamental operations in many areas of AI, including model editing, steering, robustness, and interpretability. To facilitate such research, we introduce **pyvene**, an open-source Python library that supports customizable interventions on a range of different PyTorch modules. **pyvene** supports complex intervention schemes with an intuitive configuration format, and its interventions can be static or include trainable parameters. We show how **pyvene** provides a unified and extensible framework for performing interventions on neural models and sharing the intervened upon models with others. We illustrate the power of the library via interpretability analyses using causal abstraction and knowledge localization. We publish our library through Python Package Index (PyPI) and provide code, documentation, and tutorials at <https://github.com/stanfordnlp/pyvene>.

1 Introduction

When we *intervene* on a neural network, we make an in-place change to its activations, putting the model in a counterfactual state. This fundamental operation has emerged as a powerful tool for both understanding and improving models; interventions of various kinds are key to recent efforts in model robustness (He et al., 2019), model editing (Meng et al., 2022) and steering (Li et al., 2023a), causal abstraction (Geiger et al., 2020, 2021, 2023; Wu et al., 2023) or activation patching (Chan et al., 2022; Wang et al., 2023), circuit finding (Conmy et al., 2023; Goldowsky-Dill et al., 2023), and knowledge tracing (Geva et al., 2023).

As intervention-based techniques have matured, the need has arisen to run ever more complex interventions on ever larger models. Currently, there is no unified and generic intervention-oriented library to support such research. Existing libraries are often project-based (see implementations for Wang

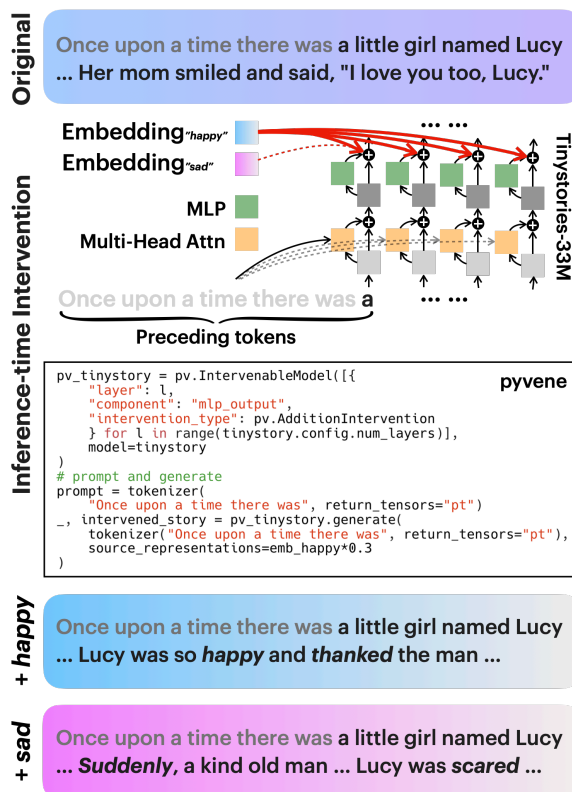


Figure 1: An inference-time intervention (Li et al., 2023a) on TinyStories-33M. The model is prompted with “Once upon a time there was a”, and is asked to complete the story. We add a static word embedding (for “happy” or “sad”) into the MLP output at each decoding step for all layers with a coefficient of 0.3. **pyvene**’s complete implementation is provided. The original and intervened generations use greedy decoding.

et al. 2023; Geiger et al. 2023 as examples) that lack extensibility and are hard to maintain and share, and current toolkits focus on single or non-nested interventions (e.g., ablation neurons in a single forward pass) and are often limited to interventions on Transformers (Vaswani et al., 2017) without natively supporting other neural architectures. Some of these existing libraries (Bau, 2022; Lloyd, 2023; Fiotto-Kaufman, 2023; Mossing et al., 2024) can

support complex interventions such as exchanging activations across multiple forward passes yet they require sophisticated knowledge and heavy implementations.

To address these limitations, we introduce **pyvene**, an open-source Python library that supports customizable interventions on different neural architectures implemented in PyTorch. Different from previous libraries (Bau, 2022; Nanda and Bloom, 2022; Lloyd, 2023; Fiotto-Kaufman, 2023; Mossing et al., 2024), **pyvene** is intervention-oriented. It supports complex interventions by manipulating or exchanging activations across multiple model forward runs while allowing these interventions to be shared with a serialization configuration file. Specifically, **pyvene** has a number of advantages:

1. **Intervention as the primitive.** The intervention is the basic primitive of **pyvene**. Interventions are specified with a dict-based format, in contrast to previous approaches where interventions are expressed as code and executed during runtime (Bau, 2022; Lloyd, 2023; Fiotto-Kaufman, 2023; Mossing et al., 2024). All **pyvene** intervention schemes and models are serializable objects that can be shared through a public model hub such as HuggingFace.
2. **Complex intervention schemes.** **pyvene** supports interventions at multiple locations, involving arbitrary subsets of neurons, and interventions can be performed in parallel or in sequence. For generative use of LMs, **pyvene** supports interventions at decoding steps. Furthermore, activations can easily be collected for probe training.
3. **Support for recurrent and non-recurrent models.** Existing libraries offer only limited support for recurrent models. **pyvene** supports simple feed-forward networks, Transformers, and recurrent and convolutional neural models.

In this paper, we provide two detailed case studies using **pyvene** as well: (1) we fully reproduce Meng et al. (2022)’s locating factual associations in GPT2-XL (Figure 1 in the original paper) in about 20 lines of code, and (2) we show intervention and probe training with **pyvene** to localize gender in Pythia-6.9B. **pyvene** is published through the

Python Package Index (PyPI),¹ and the project site² hosts more than 20 tutorials that cover interventions at different levels of complexity with various model architectures from simple feed-forward models to multi-modal models.

2 System Design and Architecture

Two primary components of **pyvene** are the **intervenable configuration**, which outlines which model components will be intervened upon, and the **intervenable model**, which decorates the original torch model with hooks that allow activations to be collected and overwritten.³ Here is a setup for performing a zero-out intervention (often called a zero ablation; Li et al. 2023b) on the 10th, 11th, and 12th dimensions of the MLP output for 3rd token embedding of layer 0 in GPT-2:

```
import torch
import pyvene as pv
# built-in helper to get a HuggingFace model
_, tokenizer, gpt2 = pv.create_gpt2()
# create with dict-based config
pv_config = pv.IntervenableConfig({
    "layer": 0,
    "component": "mlp_output",
    "intervention_type": pv.VanillaIntervention})
# initialize model
pv_gpt2 = pv.IntervenableModel(
    pv_config, model=gpt2)
# run an intervened forward pass
intervened_outputs = pv_gpt2(
    # the base input
    base=tokenizer(
        "The capital of Spain is",
        return_tensors="pt"),
    # the location to intervene at (3rd token)
    unit_locations={"base": 3},
    # the individual dimensions targetted
    subspaces=[10,11,12],
    # the intervention values
    source_representations=torch.zeros(
        gpt2.config.n_embd)
)
# sharing
pv_gpt2.save("./tmp/", save_to_hf_hub=True)
```

The model takes a tensor input base and runs through the model’s computation graph modifying activations in place to be other values source. In this code, we specified source in the forward call. When source is a constant, it can alternatively be specified in the IntervenableConfig. To target complete MLP output representations, one simply leaves out the subspaces argument. The final line of the code block shows how to serialize and share an intervened model remotely through a model hub such as HuggingFace.

¹pip install pyvene

²<https://github.com/stanfordnlp/pyvene>

³Code snippets provided in the paper can be run on Google Colab at https://colab.research.google.com/github/stanfordnlp/pyvene/blob/main/pyvene_101.ipynb.

2.1 Interchange Interventions

Interchange interventions (Geiger et al., 2020; Vig et al., 2020; Wang et al., 2023, also known as activation patching) fix activations to take on the values they would be if a different input were provided. With minor changes to the forward call, we can perform an interchange intervention on GPT-2:

```
# run an interchange intervention
intervened_outputs = pv.gpt2(
  # the base input
  base=tokenizer(
    "The capital of Spain is",
    return_tensors = "pt"),
  # the source input
  sources=tokenizer(
    "The capital of Italy is",
    return_tensors = "pt"),
  # the location to intervene at (3rd token)
  unit_locations={"sources->base": 3},
  # the individual dimensions targeted
  subspaces=[10,11,12]
)
```

This forward call produces outputs for base but with the activation values for MLP output dimensions 10–12 of token 3 at layer 0 set to those that obtained when the model processes the source. Such interventions are used in interpretability research to test hypotheses about where and how information is stored in model-internal representations.

2.2 Addition Interventions

In the above examples, we replace values in the base with other values (VanillaIntervention). Another common kind of intervention involves updating the base values in a systematic way:

```
noising_config = pv.IntervenableConfig({
  "layer": 0,
  "component": "block_input",
  "intervention_type": pv.AdditionIntervention})
noising_gpt2 = pv.IntervenableModel(
  config, model=gpt2)
intervened_outputs = noising_gpt2(
  base=tokenizer(
    "The Space Needle is in downtown",
    return_tensors = "pt"),
  # target the first four tokens for intervention
  unit_locations={"base": [0, 1, 2, 3]},
  source_representations = torch.rand(
    gpt2.config.n_embd, requires_grad=False))
```

As in this example, we add noise to a representation as a basic robustness check. The code above does this, targeting the first four input token embeddings to a Transformer by using AdditionIntervention. This example serves as the building block of causal tracing experiments as in Meng et al. 2022, where we corrupt embedding inputs by adding noise to trace factual associations. Building on top of this, we reproduce Meng et al.’s result in Section 3. **pyvene** allows Autograd on the static representations, so this code could be the

basis for training models to be robust to this noising process.

2.3 Activation Collection Interventions

This is a pass-through intervention to collect activations for operations like supervised probe training. Such interventions can be combined with other interventions as well, to support things like causal structural probes (Hewitt and Manning, 2019; Elazar et al., 2020; Lepori et al., 2023). In the following example, we perform an interchange intervention at layer 8 and then collect activations at layer 10 for the purposes of fitting a probe:

```
# set up a upstream intervention
probe_config = pv.IntervenableConfig({
  "layer": 8,
  "component": "block_output",
  "intervention_type": pv.VanillaIntervention})
# add downstream collector
probe_config = probe_config.add_intervention({
  "layer": 10,
  "component": "block_output",
  "intervention_type": pv.CollectIntervention})
probe_gpt2 = pv.IntervenableModel(
  probe_config, model=gpt2)
# return the activations for 3rd token
collected_activations = probe_gpt2(
  base=tokenizer(
    "The capital of Spain is",
    return_tensors="pt"),
  unit_locations={"sources->base": 3})
```

2.4 Custom Interventions

pyvene provides a flexible way of adding new intervention types. The following is a simple illustration in which we multiply the original representation by a constant value:

```
# multiply base with a constant
class MultInt(pv.ConstantSourceIntervention):
  def __init__(self, **kwargs):
    super().__init__()
  def forward(self, base, source=None,
              subspaces=None):
    return base * 0.3

pv.IntervenableModel({
  "intervention_type": MultInt(),
  model=gpt2})
```

The above intervention becomes useful when studying interpretability-driven models such as the Backpack LMs of Hewitt et al. (2023). The sense vectors acquired during pretraining in Backpack LMs have been shown to have a “multiplication effect”, and so proportionally decreasing sense vectors could effectively steer the model’s generation.

2.5 Trainable Interventions

pyvene interventions can include trainable parameters. RotatedSpaceIntervention implements Distributed Alignment Search (DAS; Geiger et al. 2023), LowRankRotatedSpaceIntervention is a

more efficient version of that model, and BoundlessRotatedSpaceIntervention implements the Boundless DAS variant of Wu et al. (2023). With these primitives, one can easily train DAS explainers.

In the example below, we show a single gradient update for a DAS training objective that localizes the capital associated with the country in a one-dimensional linear subspace of activations from the Transformer block output (i.e., main residual stream) at the 8th layer by training our intervention module to match the gold counterfactual behavior:

```
das_config = pv.IntervenableConfig({
    "layer": 8,
    "component": "block_output",
    "low_rank_dimension": 1,
    "intervention_type":
        pv.LowRankRotatedSpaceIntervention})

das_gpt2 = pv.IntervenableModel(
    das_config, model=gpt2)

last_hidden_state = das_gpt2(
    base=tokenizer(
        "The capital of Spain is",
        return_tensors="pt"),
    sources=tokenizer(
        "The capital of Italy is",
        return_tensors="pt"),
    unit_locations={"sources->base": 3}
)[-1].last_hidden_state[:, -1]

# gold counterfactual label as "Rome"
label = tokenizer.encode(
    "Rome", return_tensors="pt")
logits = torch.matmul(
    last_hidden_state, gpt2.wte.weight.t())

m = torch.nn.CrossEntropyLoss()
loss = m(logits, label.view(-1))
loss.backward()
```

2.6 Training with Interventions

Interventions can be co-trained with the intervening model for techniques like interchange intervention training (IIT), which induce specific causal structures in neural networks (Geiger et al., 2022):

```
pv_gpt2 = pv.IntervenableModel({
    "layer": 8},
    model=gpt2)
# enable gradients on the model
pv_gpt2.enable_model_gradients()
# run counterfactual forward as usual
```

In the example above, with the supervision signals from the training dataset, we induce causal structures in the residual stream at 8th layer.

2.7 Multi-Source Parallel Interventions

In the parallel mode, interventions are applied to the computation graph of the same base example at the same time. We can perform interchange interventions by taking activations from multiple source

examples and swapping them into the base’s computation graph:

```
parallel_config = pv.IntervenableConfig([
    {"layer": 3, "component": "block_output"},
    {"layer": 3, "component": "block_output"},
    # intervene on base at the same time
    mode="parallel")

parallel_gpt2 = pv.IntervenableModel(
    parallel_config, model=gpt2)

base = tokenizer(
    "The capital of Spain is",
    return_tensors="pt")
sources = [
    tokenizer("The language of Spain is",
        return_tensors="pt"),
    tokenizer("The capital of Italy is",
        return_tensors="pt")]

intervened_outputs = parallel_gpt2(
    base, sources,
    {"sources->base": (
        # each list has a dimensionality of
        # [num_intervention, batch, num_unit]
        [[1],[[3]]], [[1],[[3]]])})
```

In the example above, we interchange the activations from the residual streams on top of the second token from the first example (“language”) as well as the fourth token from the second example (“Italy”) into the corresponding locations of the base’s computation graph. The motivating intuition is that now the next token might be mapped to a semantic space that is a mixture of two inputs in the source “The language of Italy”. (And, in fact, “Italian” is among the top five returned logits.)

2.8 Multi-Source Serial Interventions

Interventions can also be sequentially applied, so that later interventions are applied to an intervened model created by the previous ones:

```
serial_config = pv.IntervenableConfig([
    {"layer": 3, "component": "block_output"},
    {"layer": 10, "component": "block_output"}],
    # intervene on base one after another
    mode="serial")

serial_gpt2 = pv.IntervenableModel(
    serial_config, model=gpt2)

intervened_outputs = serial_gpt2(
    base, sources,
    # src_0 intervenes on src_1 position 1
    # src_1 intervenes on base position 4
    {"source_0->source_1": 1,
     "source_1->base" : 4}
)
```

In the example above, we first take activations at the residual stream of the first token (“language”) at the 3rd layer from the first source example and swap them into the same location during the forward run of the second source example. We then take the activations of the 4th token (“is”) at layer 10 at upstream of this intervened model and swap them

into the same location during the forward run of the base example. The motivating intuition is that the first intervention will result in the model retrieving “The language of Italy” and the second intervention will swap the retrieved answer into the output stream of the base example. (Once again, “Italian” is among the top five returned logits.)

2.9 Intervenable Model

The `IntervenableModel` class is the backend for decorating torch models with intervenable configurations and running intervened forward calls. It implements two types of hooks: **Getter** and **Setter** hooks to save and set activations.

Figure 1 highlights `pyvene`’s support for LMs. Interventions can be applied to any position in the input prompt or any selected decoding step.

The following involves a model with recurrent (GRU) cells where we intervene on two unrolled recurrent computation graphs at a time step:

```
# built-in helper to get a GRU
_, _, gru = pv.create_gru_classifier(
    pv.GRUConfig(h_dim=32))
# wrap it with config
pv_gru = pv.IntervenableModel({
    "component": "cell_output",
    # intervening on time
    "unit": "t",
    "intervention_type": pv.ZeroIntervention},
    model=gru)
# run an intervened forward pass
rand_b = torch.rand(1,10, gru.config.h_dim)
rand_s = torch.rand(1,10, gru.config.h_dim)
intervened_outputs = pv_gru(
    base = {"inputs_embeddings": rand_b},
    sources = [{"inputs_embeddings": rand_s}],
    # intervening time step
    unit_locations={"sources->base": (6, 3)})
```

A hook is triggered every time the corresponding model component is called. As a result, a vanilla hook-based approach, as in all previous libraries (Bau, 2022; Lloyd, 2023; Fiotto-Kaufman, 2023; Mossing et al., 2024), fails to intervene on any recurrent or state-space model. To handle this limitation, `pyvene` records a state variable for each hook, and only executes a hook at the targeted time step.

3 Case Study I: Locating Factual Associations in GPT2-XL

We replicate the main result in Meng et al. (2022)’s Locating Factual Associations in GPT2-XL with `pyvene`. The task is to trace facts via interventions on fact-related datasets. Following Meng et al.’s setup, we first intervene on input embeddings by adding Gaussian noise. We then restore individual states to identify the information that restores the

results. Specifically, we restore the Transformer block output, MLP activation, and attention output for each token at each layer. For MLP activation and attention output, we restore 10 sites centered around the intervening layer (clipping on the edges). Our Figure 2 fully reproduces the main Figure 1 (p. 2) in Meng et al.’s paper. To replicate their experiments, we first define a configuration for causal tracing:

```
def tracing_config(
    l, c="mlp_activation", w=10, tl=48):
    s = max(0, l - w // 2)
    e = min(tl, l + (-w // 2))
    config = IntervenableConfig(
        [{"component": "block_input"}] +
        [{"layer": l, "component": c}
         for l in range(s, e)],
        [pv.NoiseIntervention] +
        [pv.VanillaIntervention]*(e-s))
    return config
```

With this configuration, we corrupt the subject token and then restore selected internal activations to their clean value. Our main experiment is implemented with about 20 lines of code with `pyvene`:

```
trace_results = []
_, tokenizer, gpt = pv.create_gpt2("gpt2-xl")
base = tokenizer(
    "The Space Needle is in downtown",
    return_tensors="pt")
for s in ["block_output", "mlp_activation",
         "attention_output"]:
    for l in range(gpt.config.n_layer):
        for p in range(7):
            w = 1 if s == "block_output" else 10
            t_config, n_r = tracing_config(l, s, w)
            t_gpt = pv.IntervenableModel(t_config, gpt)
            _, outs = t_gpt(base, [None] + [base]*n_r,
                {"sources->base": ([None] + [[p]]*n_r,
                    [[0, 1, 2, 3]] + [[p]]*n_r)})
            dist = pv.embed_to_distrib(gpt,
                outs.last_hidden_state, logits=False)
            trace_results.append(
                {"stream": s, "layer": l, "pos": p,
                 "prob": dist[0][-1][7312]})
```

4 Case Study II: Intervention and Probe Training with Pythia-6.9B

We showcase intervention and probe training with `pyvene` using a simple gendered pronoun prediction task in which we try to localize gender in hidden representations. For trainable intervention, we use a one-dimensional Distributed Alignment Search (DAS; Geiger et al., 2023), that is, we seek to learn a 1D subspace representing gender. To localize gender, we feed prompts constructed from a template of the form “[**John/Sarah**] walked because [**he/she**]” (a fixed length of 4) where the name is sampled from a vocabulary of 47 typically male and 10 typically female names followed by the associated gendered pronoun as the output token. We use `pythia-6.9B` (Biderman et al., 2023)

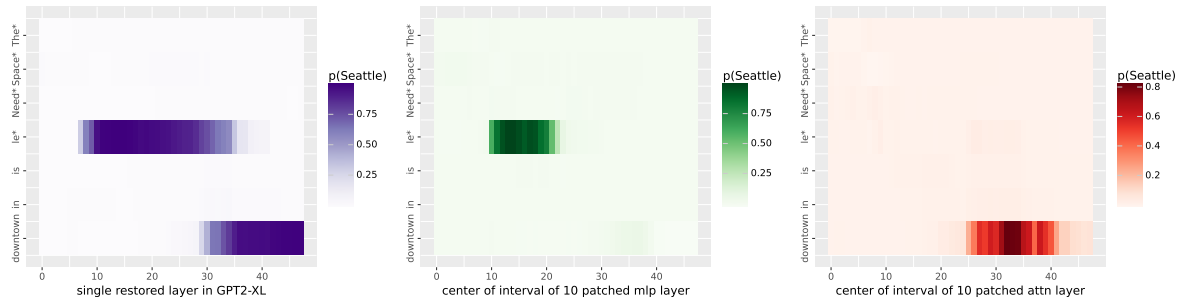


Figure 2: We reproduce the results in Meng et al. (2022)’s Figure 1 of locating early sites and late sites of factual associations in GPT2-XL in about 20 lines of `pyvene` code. The causal impact on output probability is mapped for the effect of each Transformer block output (left), MLP activations (middle), and attention layer output (right).

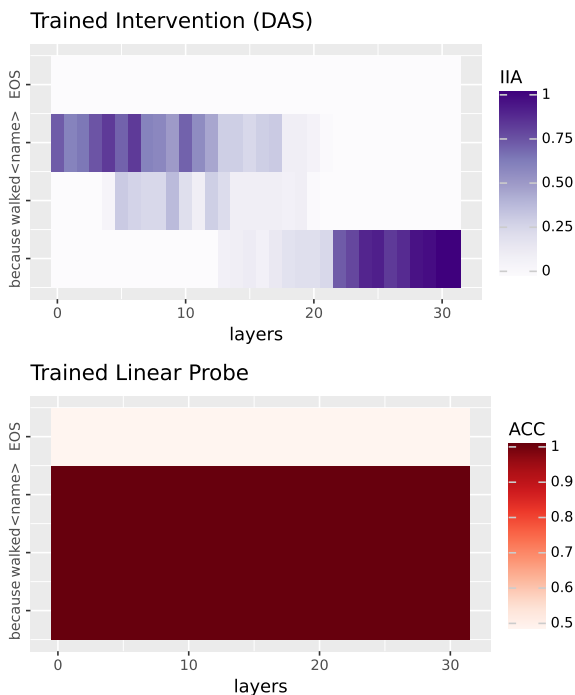


Figure 3: Results of interchange intervention accuracy (IIA) with the trainable intervention (DAS) and accuracy with the trainable linear probe on different model components when localizing gender information.

in this experiment, which achieves 100% accuracy on the task. We then train our interventions and probes at the Transformer block output at each layer and token position. For intervention training, we construct pairs of examples and train the intervention to match the desired counterfactual output (i.e., if we swap activations from an example with a male name into another example with a female name, the desired counterfactual output should be “he”). For linear probe training, we use activation collection intervention to retrieve activations to predict the pronoun gender with a linear layer.

As shown in Figure 3, a trainable intervention

finds sparser gender representations across layers and positions, whereas a linear probe achieves 100% classification accuracy for almost all components. This shows that a probe may achieve high performance even on representations that are not causally relevant for the task.

5 Limitations and Future Work

We are currently focused on two main areas:

1. Expanding the default intervention types and model types. Although `pyvene` is extensible to other types, having more built-in types helps us to onboard new users easily.
2. `pyvene` is designed to support complex intervention schemes, but this comes at the cost of computational efficiency. As language models get larger, we would like to investigate how to scale intervention efficiency with multi-node and multi-GPU training.

6 Conclusion

We introduce `pyvene`, an open-source Python library that supports intervention-based research on neural models. `pyvene` supports customizable interventions with complex intervention schemes as well as different families of model architectures, and intervened models are shareable with others through online model hubs such as HuggingFace. Our hope is that `pyvene` can be a powerful tool for discovering new ways in which interventions can help us explain and improve models.

References

- David Bau. 2022. BauKit. <https://github.com/davidbau/baukit>.

- Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O’Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. 2023. [Pythia: A suite for analyzing large language models across training and scaling](#). In *International Conference on Machine Learning (ICML)*.
- Lawrence Chan, Adrià Garriga-Alonso, Nicholas Goldowsky-Dill, Ryan Greenblatt, Jenny Nitishinskaya, Ansh Radhakrishnan, Buck Shlegeris, and Nate Thomas. 2022. [Causal scrubbing: a method for rigorously testing interpretability hypotheses](#). Alignment Forum Blog post.
- Arthur Conmy, Augustine N Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. 2023. [Towards automated circuit discovery for mechanistic interpretability](#). In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Yanai Elazar, Shauli Ravfogel, Alon Jacovi, and Yoav Goldberg. 2020. [Amnesic probing: Behavioral explanation with amnesic counterfactuals](#). In *Transactions of the Association of Computational Linguistics (TACL)*.
- Jaden Fiotto-Kaufman. 2023. nnsight. <https://github.com/JadenFiotto-Kaufman/nnsight>.
- Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. 2021. [Causal abstractions of neural networks](#). In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Atticus Geiger, Kyle Richardson, and Christopher Potts. 2020. [Neural natural language inference models partially embed theories of lexical entailment and negation](#). In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, Online.
- Atticus Geiger, Zhengxuan Wu, Hanson Lu, Josh Rozner, Elisa Kreiss, Thomas Icard, Noah Goodman, and Christopher Potts. 2022. [Inducing causal structure for interpretable neural networks](#). In *International Conference on Machine Learning (ICML)*.
- Atticus Geiger, Zhengxuan Wu, Christopher Potts, Thomas Icard, and Noah D. Goodman. 2023. [Finding alignments between interpretable causal variables and distributed neural representations](#). In *Causal Learning and Reasoning (CLear)*.
- Mor Geva, Jasmijn Bastings, Katja Filippova, and Amir Globerson. 2023. [Dissecting recall of factual associations in auto-regressive language models](#). In *Empirical Methods in Natural Language Processing (EMNLP)*, Singapore.
- Nicholas Goldowsky-Dill, Chris MacLeod, Lucas Sato, and Aryaman Arora. 2023. [Localizing model behavior with path patching](#). *arXiv:2304.05969*.
- Zhezhi He, Adnan Siraj Rakin, and Deliang Fan. 2019. [Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack](#). In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- John Hewitt and Christopher D Manning. 2019. [A structural probe for finding syntax in word representations](#). In *North American Chapter of the Association for Computational Linguistics (NAACL)*.
- John Hewitt, John Thickstun, Christopher Manning, and Percy Liang. 2023. [Backpack language models](#). In *Association for Computational Linguistics (ACL)*.
- Michael A Lepori, Thomas Serre, and Ellie Pavlick. 2023. [Uncovering intermediate variables in transformers using circuit probing](#). *arXiv:2311.04354*.
- Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. 2023a. [Inference-time intervention: Eliciting truthful answers from a language model](#). In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Maximilian Li, Xander Davies, and Max Nadeau. 2023b. [Circuit breaking: Removing model behaviors with targeted ablation](#). *arXiv:2309.05973*.
- Evan Lloyd. 2023. graphpatch. <https://github.com/evan-lloyd/graphpatch>.
- Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. [Locating and editing factual associations in GPT](#). In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Dan Mossing, Steven Bills, Henk Tillman, Tom Dupré la Tour, Nick Cammarata, Leo Gao, Joshua Achiam, Catherine Yeh, Jan Leike, Jeff Wu, and William Saunders. 2024. [Transformer debugger](#). <https://github.com/openai/transformer-debugger>.
- Neel Nanda and Joseph Bloom. 2022. [Transformerlens](#). <https://github.com/neelnanda-io/TransformerLens>.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. [Attention is all you need](#). In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. 2020. [Investigating gender bias in language models using causal mediation analysis](#). In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2023. [Interpretability in the wild: A circuit for indirect object identification in GPT-2 small](#). In *International Conference on Learning Representations (ICLR)*.

Zhengxuan Wu, Atticus Geiger, Thomas Icard, Christopher Potts, and Noah Goodman. 2023. [Interpretability at scale: Identifying causal mechanisms in Alpaca](#). In *Advances in Neural Information Processing Systems (NeurIPS)*.