AI Safety Tools and Interpretability Frameworks

A Comprehensive Technical Reference

Elisabetta Rocchetti

February 6, 2026

Abstract

This document provides a comprehensive survey of tools and frameworks for AI safety research and mechanistic interpretability. The field has rapidly evolved to address the critical need for understanding, evaluating, and controlling the behavior of large language models and other neural networks. The tools covered span three main categories: evaluation frameworks for assessing model capabilities and risks, mechanistic interpretability libraries for understanding model internals through techniques like sparse autoencoders and circuit analysis, and intervention frameworks for causally probing and steering model behavior. Each tool is described with standardized information including capabilities, supported models, computational requirements, and programmatic accessibility. This reference aims to help researchers select appropriate tools for their specific AI safety and interpretability research needs.

Contents

1 Introduction

Understanding and ensuring the safety of AI systems has become one of the most pressing technical challenges in machine learning research [Amodei et al., 2016]. As models grow in capability and deployment scale, the need for rigorous evaluation, deep interpretability, and precise control mechanisms has intensified. This document catalogs the major open-source tools that enable such research.

The tools described here fall into three broad categories. First, **evaluation frameworks** provide systematic methods for assessing model capabilities, identifying vulnerabilities, and measuring safety-relevant properties. Second, **mechanistic interpretability tools** enable researchers to peer inside neural networks, decomposing their computations into interpretable components like features and circuits. Third, **intervention frameworks** allow researchers to causally probe models by modifying internal activations and observing the effects.

Together, these tools form an ecosystem that supports the technical work necessary to build AI systems that are transparent, controllable, and aligned with human intentions. While no single tool solves all problems, the combination of rigorous evaluation, deep understanding, and precise intervention provides the foundation for safer AI development.

The Open Weight Advantage for Safety Research

The rise of **open weight models**—where model weights are publicly released, even if training code or data remain proprietary—has fundamentally transformed AI safety research. Unlike closed API-only models, open weight models enable researchers to conduct **white-box analysis**, examining internal activations, attention patterns, and learned representations directly. This access is critical for mechanistic interpretability work, which seeks to reverse-engineer the algorithms and circuits that models learn.

Most tools in this document, particularly those for mechanistic interpretability and intervention, **require open weight access**. Techniques like activation patching, circuit discovery, sparse autoencoder training, and feature attribution fundamentally depend on inspecting and modifying model internals—capabilities unavailable with API-only access. While evaluation frameworks can often work with both open weight and API-based models, the deepest insights into model behavior come from direct weight access.

The availability of capable open weight models such as **LLaMA**, **Gemma**, **Qwen**, **Mistral**, and **Pythia** has democratized safety research, allowing academic labs and independent researchers to conduct work previously limited to organizations with proprietary model access. This openness has accelerated progress in understanding how neural networks function, enabled reproducible science, and fostered a collaborative research community. As frontier capabilities continue to advance, maintaining open weight releases alongside closed models will be essential for the AI safety research ecosystem to keep pace with emerging risks.

2 Evaluation Frameworks

Evaluation frameworks provide structured approaches to assess AI model capabilities, safety properties, and potential risks. Unlike ad-hoc testing, these frameworks offer reproducible, scalable methods for systematic evaluation.

2.1 Inspect AI

Inspect is a comprehensive framework for large language model evaluations developed by the UK AI Safety Institute [UK AI Safety Institute, 2024b]. It provides built-in components for prompt engineering, tool usage, multi-turn dialog, and model-graded evaluations, with extensive support for custom evaluation development. The framework supports systematic capability evaluations across multiple models, safety and alignment testing, tool use and agent behavior assessment, and custom evaluation development and benchmarking. Inspect focuses on black-box evaluation rather than inspecting model internals, producing evaluation results, metrics, logs, and detailed HTML reports with visualization.

A companion repository, **Inspect Evals** [UK AI Safety Institute, 2024c], provides over 100 community-contributed pre-built evaluations covering diverse areas including knowledge (MMLU, TruthfulQA), reasoning (GSM8K, MATH, ARC), safety (ToxiGen, bias benchmarks), coding (HumanEval, MBPP), agent capabilities (GAIA, WebArena, SWE-bench), and cybersecurity (CyberSecEval). These evaluations work seamlessly with the Inspect AI framework and can be run individually or in batches.

Objective & Scope

Primary Purpose: Framework for systematic LLM capability and safety evaluation through reproducible benchmarks, with 100+ pre-built community evaluations.
Scope: Black-box evaluation (no model internals access), API and local models, custom evaluation development, multi-model comparisons, extensive benchmark library.
Output: Evaluation metrics, performance reports, HTML visualizations, detailed logs.

Technical Requirements

Supported Models: Any LLM accessible via API (OpenAI, Anthropic, Together, etc.) or locally via HuggingFace.
Compute Requirements: Minimal for framework; depends on models being evaluated. Can run on standard laptop for API-based models.
Dependencies: Python 3.10+, PyTorch (optional for local models).
Usage: Programmatic Python library with CLI interface. Can be integrated into CI/CD pipelines.

Resources

Repository: `https://github.com/UKGovernmentBEIS/inspect_ai`
Documentation: `https://inspect.aisi.org.uk/`
Key Features: Extensible solver and scorer system; built-in support for tool calling; dataset integration; parallel execution; detailed logging; 100+ pre-built evaluations via Inspect Evals companion repository.

2.2 ControlArena

A framework for evaluating AI control protocols and oversight mechanisms [UK AI Safety Institute, 2024a]. Tests how well monitoring systems can detect and prevent undesired model behaviors, supporting testing of AI control and monitoring strategies, evaluating red-teaming effectiveness, measuring detectability of deceptive behaviors, and oversight protocol development. The framework provides limited intervention support, focusing primarily on behavioral evaluation under dif-

ferent control regimes, and produces control effectiveness metrics, detection rates, and protocol performance analysis.

Objective & Scope

Primary Purpose: Evaluate AI control protocols and oversight mechanisms for detecting/preventing undesired behaviors.
Scope: Red-team/blue-team scenarios, monitoring strategy testing, deception detection, limited intervention support.
Output: Control effectiveness metrics, detection rates, protocol performance analysis.

Technical Requirements

Supported Models: LLMs accessible via standard APIs.
Compute Requirements: Moderate; requires running multiple model instances for red-team/blue-team scenarios.
Dependencies: Python, Inspect AI framework.
Usage: Programmatic interface for defining control protocols and evaluation scenarios.

Resources

Repository: `https://github.com/UKGovernmentBEIS/control-arena`

2.3 SAIL Framework

A process-oriented methodology for managing AI security throughout the entire AI system lifecycle [Pillar Security, 2024]. Developed by Pillar Security, SAIL (Secure AI Lifecycle) provides a practical framework to systematically add security controls at each phase of AI development and deployment. The framework unites development, MLOps, security, and governance teams around a common language for managing AI-specific risks. SAIL covers seven phases: Plan (AI policy and safe experimentation), Code/No Code (AI asset discovery), Build (AI security posture management), Test (AI red teaming), Deploy (runtime guardrails), Operate (safe execution environments), and Monitor (AI activity tracing). Each phase addresses specific security risks with detailed controls and mitigation strategies. The framework provides risk assessment templates, threat modeling guidance, and compliance mapping to regulatory requirements. Designed for practitioners, SAIL offers a comprehensive roadmap for building trustworthy AI systems with appropriate security controls at each lifecycle stage.

Objective & Scope

Primary Purpose: Lifecycle-oriented security framework for managing AI-specific risks across development, deployment, and operation.
Scope: 7-phase methodology covering policy, asset discovery, security posture management, red teaming, runtime protection, safe execution, and monitoring.
Output: Security roadmaps, risk assessment documentation, compliance mappings, threat models, audit trails.

Technical Requirements

Supported Models: Framework-agnostic; applies to any AI system regardless of architecture or deployment method.
Compute Requirements: None (methodology and assessment framework).
Dependencies: None for framework adoption; implementation may require various security tools depending on phase.
Usage: Process framework with assessment templates, risk catalogs, and implementation guidance. Can be used with Pillar Security platform for automated analysis.

Resources

Framework Website: `https://www.pillar.security/sail`
Key Features: 7-phase lifecycle coverage; 100+ specific AI security risks cataloged; threat modeling integration; regulatory compliance mapping; red teaming methodology; runtime guardrails specification; monitoring and incident response guidance; community-driven with practitioner feedback.

3 Mechanistic Interpretability Tools

Mechanistic interpretability aims to reverse-engineer neural networks by identifying the algorithms and representations they learn [Olah et al., 2020]. These tools enable researchers to decompose model computations into interpretable components.

3.1 TransformerLens

A library specifically designed for mechanistic interpretability of GPT-style language models [Nanda et al., 2024]. Provides easy access to all model internals with clean, interpretable activations and comprehensive hooks for activation caching and analysis, circuit discovery and tracing, attention pattern visualization, studying induction heads and algorithmic circuits, and feature attribution and ablation studies. The library offers comprehensive support for activation patching, ablations, and causal interventions at any layer or component. Outputs include tensors of activations, attention patterns, and logits, with integration to CircuitsVis for visualization.

Objective & Scope

Primary Purpose: Full-featured library for mechanistic interpretability of transformer language models.
Scope: Complete activation access, circuit discovery, attention analysis, intervention support (patching/ablations).
Output: Activation tensors, attention patterns, logits, visualizations via CircuitsVis.

Technical Requirements

Supported Models: Extensive model zoo including GPT-2, GPT-Neo, GPT-J, Pythia, OPT, BLOOM, LLaMA family, Gemma, Mistral, and 100+ additional architectures. Full model list with specifications available at `https://transformerlensorg.github.io/TransformerLens/generated/model_properties_table.html`.
Compute Requirements: Moderate to high; requires loading full model weights. GPU recommended for models larger than 1B parameters.
Dependencies: PyTorch, transformers, einops, fancy_einsum.
Usage: Programmatic Python library; designed for notebook-based exploratory analysis. Extensive tutorial notebooks available.

Resources

Repository: `https://github.com/TransformerLensOrg/TransformerLens`
Documentation: `https://transformerlensorg.github.io/TransformerLens/`
Key Features: Clean, factored activation access; modular intervention system; caching for efficient experimentation; extensive model zoo.

3.2 CircuitTracer

A library for finding circuits using features from cross-layer MLP transcoders [Hanna et al., 2025]. Implements the attribution graph methods introduced by Anthropic's circuit discovery research, computing direct effects between transcoder features, error nodes, tokens, and output logits. Enables circuit discovery using transcoder features, attribution graph computation and visualization, feature-level intervention experiments, integration with Neuronpedia for graph exploration, and support for both programmatic and CLI-based workflows. Works with pre-trained transcoders (PLTs and CLTs) for Gemma-2, Llama-3.2, and Qwen-3 models, producing attribution graphs, interactive visualizations, and intervention results.

Objective & Scope

Primary Purpose: Find circuits using transcoder features via attribution graphs; compute direct effects between features, tokens, and logits.
Scope: Attribution graph computation, circuit visualization, feature interventions, Neuronpedia integration, supports TransformerLens/nnsight backends.
Output: Attribution graphs, interactive visualizations (web-based), intervention results, graph annotations.

Technical Requirements

Supported Models: Gemma-2 (2B), Llama-3.2 (1B), Qwen-3 (0.6B-14B); requires pre-trained transcoders. Can use TransformerLens or nnsight backend.
Compute Requirements: Moderate; Gemma-2 2B works on Colab free tier (15GB VRAM). Larger models benefit from more GPU memory to reduce offloading.
Dependencies: PyTorch, transformers, TransformerLens (default) or nnsight (experimental backend).
Usage: Python library (Jupyter notebooks/scripts) or CLI. Integrates with Neuronpedia web interface for visualization.

Resources

Repository: `https://github.com/safety-research/circuit-tracer`
Documentation: Tutorial notebooks available; see `https://transformer-circuits.pub/2025/attribution-graphs/methods.html`
Key Features: Transcoder-based attribution graphs; interactive visualization server; Neuronpedia integration; feature intervention capabilities; CLI workflow; supports PLTs and CLTs.

3.3 nnsight

A framework for interpreting and intervening on the internals of PyTorch models [NDIF Team, 2024]. Provides a unified interface for accessing activations, performing interventions, and tracing computations. Enables remote model access via NDIF (National Deep Inference Fabric), distributed interpretability research, activation extraction and intervention, and cross-model comparison studies. Offers full support for intervention on any module's inputs and outputs, producing PyTorch tensors, intervention results, and computation traces.

Objective & Scope

Primary Purpose: Unified framework for interpreting PyTorch models with remote compute access via NDIF.
Scope: Model-agnostic activation access, full intervention support, distributed research, local/remote execution.
Output: PyTorch tensors, intervention results, computation traces.

Technical Requirements

Supported Models: Any PyTorch model; strong support for HuggingFace models.
Compute Requirements: Variable; can use remote compute via NDIF for expensive operations.
Dependencies: PyTorch, transformers.
Usage: Programmatic Python library with both local and remote execution modes.

Resources

Repository: `https://github.com/ndif-team/nnsight`
Documentation: `https://nnsight.net/`
Key Features: Remote model access; unified API across architectures; efficient batched interventions; grad-compatible operations.

3.4 SAELens

A comprehensive library for training and analyzing Sparse Autoencoders (SAEs) on language models [Bloom et al., 2024]. SAEs decompose neural network activations into sparse, interpretable features that often correspond to human-understandable concepts [Cunningham et al., 2023]. The library supports training SAEs on model activations, loading and analyzing pre-trained SAEs, feature discovery and interpretation, activation reconstruction analysis, and generating feature dashboards. SAE features can be used for targeted interventions and steering, with outputs including trained SAE weights, feature activations, reconstruction statistics, and visualization dashboards. An

extensive collection of pre-trained SAEs for GPT-2, Pythia, Gemma, and other models is available via HuggingFace.

Objective & Scope

Primary Purpose: Train and analyze Sparse Autoencoders to decompose model activations into interpretable features.
Scope: SAE training (multiple architectures), pre-trained SAE library, feature interpretation, intervention via features.
Output: Trained SAE weights, feature activations, reconstruction statistics, visualization dashboards.

Technical Requirements

Supported Models: Deep integration with TransformerLens; also works with HuggingFace models, nnsight, and custom PyTorch models via encode/decode methods.
Compute Requirements: High for training (GPU required; training can take hours to days). Moderate for inference.
Dependencies: PyTorch, transformers, TransformerLens (for tight integration), wandb (for training monitoring).
Usage: Programmatic Python library. Training via config files or Python API; analysis via notebooks.

Resources

Repository: `https://github.com/jbloomAus/SAELens`
Documentation: `https://jbloomaus.github.io/SAELens/`
Key Features: Multiple SAE variants (vanilla, TopK, Gated); pre-trained SAE library; automated interpretability via GPT-4; integration with SAE-Vis for dashboards; supports distributed training.

3.5 Neuronpedia

An open platform for sharing and exploring interpretability artifacts, particularly SAE features and circuits [Lin, 2024]. Functions as a collaborative database and visualization tool for mechanistic interpretability research, enabling browsing of pre-computed SAE features, visualizing feature activations and examples, sharing interpretability findings, exploring circuits and computational graphs, and community collaboration on feature interpretation. This primarily visualization and exploration platform produces interactive web visualizations, feature dashboards, and circuit diagrams, with no computational requirements for users as it is entirely web-based.

Objective & Scope

Primary Purpose: Collaborative web platform for exploring and sharing SAE features and interpretability findings.
Scope: Web-based visualization, pre-computed feature database, community annotations, circuit exploration.
Output: Interactive web visualizations, feature dashboards, circuit diagrams.

Technical Requirements

Supported Models: GPT-2, Claude Sonnet (via Anthropic's public SAEs), Gemma; expanding coverage.
Compute Requirements: None for users (web-based platform).
Dependencies: Web browser for access; Python API available for programmatic upload/download of features.
Usage: Web application for browsing; Python API for programmatic interaction.

Resources

Website: `https://neuronpedia.org/`
Repository: `https://github.com/hijohnnylin/neuronpedia`
Key Features: Large database of interpreted features; community annotations; circuit visualization; integration with SAELens and Anthropic's work; search and filter capabilities.

3.6 CircuitsVis

A library of React-based visualizations for mechanistic interpretability [TransformerLens Organization, 2024]. Works seamlessly in both Python (Jupyter) and JavaScript environments for attention pattern visualization, token-level attribution display, neuron activation heatmaps, logit lens visualizations, and interactive exploration of model internals. This visualization-focused tool produces interactive HTML visualizations renderable in notebooks or web pages, requiring minimal compute for lightweight visualization rendering.

Objective & Scope

Primary Purpose: React-based visualization library for attention patterns, activations, and interpretability artifacts.
Scope: Model-agnostic, works in Python (Jupyter) and JavaScript, interactive HTML visualizations.
Output: Interactive HTML visualizations for notebooks/web pages.

Technical Requirements

Supported Models: Model-agnostic (works with any activation data).
Compute Requirements: Minimal (lightweight visualization rendering).
Dependencies: React (for JS usage); ipython for Python usage.
Usage: Dual-use library with Python functions for notebooks and React components for web integration.

Resources

Repository: `https://github.com/TransformerLensOrg/CircuitsVis`
Documentation: `https://transformerlensorg.github.io/CircuitsVis/`
Key Features: Attention head visualization; token-based coloring; logit lens; neuron activation displays; works offline; customizable styling.

3.7 Transformer Debugger (TDB)

An interactive tool developed by OpenAI for investigating specific behaviors in small language models [Mossing et al., 2024]. Combines automated interpretability techniques with sparse autoencoders for rapid exploration of model behaviors, feature-based activation analysis, forward pass intervention experiments, automated feature explanation generation, and behavior attribution to specific features. Supports interactive interventions on features and activations, producing an interactive web interface with real-time analysis and automated feature explanations.

Objective & Scope

Primary Purpose: Interactive GUI tool for rapid investigation of specific model behaviors using SAEs and automated interpretability.
Scope: Small language models (optimized for GPT-2 Small), feature-based analysis, interactive interventions, GPT-4 explanations.
Output: Interactive web interface with real-time analysis and automated feature explanations.

Technical Requirements

Supported Models: Currently optimized for GPT-2 Small; expandable to similar architectures.
Compute Requirements: Moderate; runs locally with model loaded in memory.
Dependencies: Python backend; React frontend; pre-trained SAEs.
Usage: Interactive application with GUI; programmatic backend API.

Resources

Repository: `https://github.com/openai/transformer-debugger`
Key Features: Automated interpretability (using GPT-4); SAE feature integration; activation server architecture; neuron viewer interface; intervention capabilities.

3.8 Prisma (ViT-Prisma)

An open-source framework for mechanistic interpretability of vision and multimodal models [Joseph et al., 2025]. Provides unified access to over 75 vision transformers with SAE support for vision model interpretability, training SAEs on vision transformers, multimodal model analysis (CLIP, etc.), circuit discovery in vision models, and feature visualization and attribution. Supports interventions on vision model internals, producing activations, SAE features, circuit analyses, and visualizations. Includes over 80 pre-trained vision SAEs.

Objective & Scope

Primary Purpose: Mechanistic interpretability framework for vision transformers and multimodal models.
Scope: 75+ vision models, SAE training/analysis, circuit discovery, intervention support, includes 80+ pre-trained vision SAEs.
Output: Activations, SAE features, circuit analyses, visualizations.

Technical Requirements

Supported Models: 75+ vision and video transformers including CLIP, DINOv2, MAE, ViT variants; multimodal models.
Compute Requirements: Moderate to high (vision models are compute-intensive); GPU recommended.
Dependencies: PyTorch, timm, transformers, SAELens integration.
Usage: Programmatic Python library.

Resources

Repository: `https://github.com/Prisma-Multimodal/ViT-Prisma`
Documentation: `https://arxiv.org/abs/2504.19475`
Key Features: 80+ pre-trained vision SAEs; unified model interface; activation caching; circuit analysis tools; transcoder and crosscoder support; educational resources.

3.9 Language Model SAEs (OpenMOSS)

A distributed SAE training framework developed by OpenMOSS with focus on scalability and frontier variants [Ge et al., 2024]. Provides performant tools for training and analyzing sparse autoencoders at scale, supporting distributed training across multiple GPUs, implementation of novel SAE architectures, large-scale feature discovery, and efficient handling of frontier models. Designed for researchers working with large models who need scalable SAE training infrastructure.

Objective & Scope

Primary Purpose: Distributed SAE training framework for large-scale language models.
Scope: Multi-GPU distributed training, novel SAE architectures, frontier model support, scalability optimizations.
Output: Trained SAE weights, features at scale.

Technical Requirements

Supported Models: HuggingFace transformers; optimized for large language models.
Compute Requirements: High; designed for multi-GPU distributed training.
Dependencies: PyTorch, transformers, distributed training libraries.
Usage: Programmatic Python library with distributed training support.

Resources

Repository: `https://github.com/OpenMOSS/Language-Model-SAEs`
Key Features: Distributed training; novel SAE architectures; scalability optimizations; frontier model support.

4 Intervention and Causal Analysis Frameworks

Intervention frameworks enable researchers to perform causal experiments by modifying model internals and observing effects. This is crucial for understanding causal mechanisms and testing interpretability hypotheses.

4.1 pyvene

A comprehensive Python library for intervening on PyTorch model internals [Wu et al., 2024b]. Supports customizable interventions with both static and trainable parameters for causal analysis and model editing. Enables activation patching and ablation studies, causal tracing experiments, interchange intervention training (IIT), model editing and knowledge localization, causal abstraction alignment, and path patching for circuit discovery. This is core functionality with extensive intervention types including vanilla swapping, noise addition, rotation, and trainable low-rank interventions, producing modified model outputs, intervention effects, alignment scores, and causal graphs.

Objective & Scope

Primary Purpose: Comprehensive framework for causal interventions on PyTorch models (model-agnostic).
Scope: Activation patching/ablation, causal tracing, IIT, model editing, path patching, trainable interventions.
Output: Modified outputs, intervention effects, alignment scores, causal graphs.

Technical Requirements

Supported Models: Any PyTorch model (transformers, RNNs, CNNs, ResNets, Mamba); works out-of-the-box without model-specific code.
Compute Requirements: Moderate; depends on base model size and intervention complexity.
Dependencies: PyTorch, transformers.
Usage: Programmatic Python library. Interventions specified as serializable dicts (shareable via HuggingFace).

Resources

Repository: `https://github.com/stanfordnlp/pyvene`
Documentation: `https://stanfordnlp.github.io/pyvene/`
Key Features: Model-agnostic; compositional interventions (parallel/sequential); trainable intervention parameters; intervention sharing as artifacts; supports complex schemas like path patching; detailed tutorials and examples.

4.2 PyReFT

A parameter-efficient fine-tuning framework through learned interventions (Representation Fine-Tuning) [Wu et al., 2024a,b]. Enables efficient model adaptation by learning interventions on representations rather than updating all model parameters. Supports parameter-efficient adaptation, learned intervention training, representation editing for alignment, efficient fine-tuning for downstream tasks, and interpretable model modification.

Objective & Scope

Primary Purpose: Parameter-efficient fine-tuning via learned representation interventions.
Scope: Efficient model adaptation, learned interventions, representation editing, downstream task fine-tuning.
Output: Fine-tuned models with learned intervention parameters.

Technical Requirements

Supported Models: HuggingFace transformers.
Compute Requirements: Moderate; more efficient than full fine-tuning.
Dependencies: PyTorch, transformers, pyvene.
Usage: Programmatic Python library integrated with pyvene framework.

Resources

Repository: `https://github.com/stanfordnlp/pyreft`
Key Features: Parameter efficiency; learned interventions; integration with pyvene; interpretable adaptation.

4.3 Baukit

Early intervention library by David Bau, focusing on CNN interpretability and activation editing [Bau, 2022]. Provides tools for probing and modifying neural network activations, particularly useful for vision models and understanding convolutional architectures.

Objective & Scope

Primary Purpose: Intervention library for CNN interpretability and activation editing.
Scope: PyTorch models (especially CNNs/vision), activation editing, neuron visualization, causal interventions.
Output: Modified activations, neuron visualizations, intervention results.

Technical Requirements

Supported Models: PyTorch models, particularly CNNs and vision architectures.
Compute Requirements: Moderate; depends on model size.
Dependencies: PyTorch, numpy.
Usage: Programmatic Python library.

Resources

Repository: `https://github.com/davidbau/baukit`
Key Features: Activation editing; neuron visualization; causal interventions; originally designed for CNN interpretability.

4.4 Concordance Token Injection Lab

A web-based playground for exploring token injection as a steering mechanism [Concordance Research, 2024]. Enables researchers to inject words, concepts, or phrases directly into an LLM's generation stream to observe and control output trajectories in real-time. The tool provides an interactive interface for token injection experiments, real-time observation of steering effects, phrase pair detection and replacement functionality, configurable injection positions and strategies, and support for multiple model backends. Designed for rapid experimentation with token-level steering, the system operates entirely through a web interface without requiring local infrastructure. Outputs

include steered model generations, injection behavior logs, and comparative analyses of intervention effects.

Objective & Scope

Primary Purpose: Interactive web playground for token injection experiments and LLM output steering.
Scope: Real-time token/phrase injection, detection-replacement pairs, trajectory steering, behavioral observation.
Output: Steered generations, injection logs, comparative results.

Technical Requirements

Supported Models: API-accessible LLMs; model selection available through web interface.
Compute Requirements: None for users (web-based platform with remote compute).
Dependencies: Web browser only.
Usage: Interactive web application. Accessible via browser without installation or infrastructure requirements.

Resources

Playground: `https://research.concordance.co/playground`
Research Paper: `https://www.concordance.co/blog/token-injection-steering-llms`
Documentation: `https://docs.concordance.co/`
Key Features: Phrase pair detection and replacement; configurable injection positions; real-time steering visualization; system prompt control; temperature and token limit configuration; experiment history tracking.

5 Open-Weight Assets and Training Checkpoints

While many evaluation and interpretability tools operate on black-box APIs, deep safety research often requires full access to model weights, training data, and intermediate states. A growing ecosystem of "open science" models provides these assets, enabling researchers to study learning dynamics, phase transitions in capability, and the emergence of specific behaviors over time.

5.1 Intermediate Training Checkpoints

Analyzing a final model provides only a snapshot of a static system. To understand how capabilities and risks emerge, researchers need access to models throughout their training process.

Pythia by EleutherAI is the gold standard for this type of research [Biderman et al., 2023]. It consists of a suite of models ranging from 70M to 12B parameters, all trained on the exact same data in the same order. Crucially, the release includes 154 intermediate checkpoints for each model size, allowing researchers to trace the exact training step where specific circuits form or memorization occurs.

OLMo (Open Language Model) by AllenAI and **LLM360** push transparency further by releasing not just weights and checkpoints, but also the full training data, code, and training logs [Groeneveld et al., 2024, Liu et al., 2023]. This allows safety researchers to correlate internal model behaviors

directly with specific subsets of training data, a critical capability for dataset influence analysis and unlearning research.

Objective & Scope

Primary Purpose: Enable longitudinal study of model behavior, learning dynamics, and capability emergence.
Scope: Analysis of phase transitions, circuit formation, bias acquisition, and memorization during training.
Key Assets: Pythia (154 checkpoints/model), OLMo (full data/logs), LLM360 (Amber/Crystal checkpoints).

Technical Requirements

Storage: High. Storing full checkpoint suites requires terabytes of disk space; however, individual checkpoints can be streamed or downloaded on demand via HuggingFace.
Usage: Checkpoints function as standard HuggingFace models. Analysis scripts typically loop over checkpoint revision tags (e.g., `step1000`, `step2000`).

5.2 Open-Weight Sparse Autoencoders (Gemma Scope)

Training Sparse Autoencoders (SAEs) is computationally expensive, often exceeding the cost of training small language models. To democratize access to mechanistic interpretability, major labs have begun releasing pre-trained SAEs.

Gemma Scope is a comprehensive suite of open SAEs released by Google DeepMind for the Gemma 2 suite of models [Lieberum et al., 2024]. It includes "JumpReLU" SAEs trained on every layer and sub-layer of the Gemma 2 2B and 9B models, providing a pre-computed microscope for these architectures. These artifacts are hosted on HuggingFace and integrated into Neuronpedia, allowing researchers to skip the expensive training phase and immediately begin analyzing features.

Resources

Gemma Scope: `https://huggingface.co/google/gemma-scope` **Pythia Collection:** `https://huggingface.co/EleutherAI` **OLMo Ecosystem:** `https://allenai.org/olmo`

6 Benchmarks for Evaluating Interpretability Techniques

As mechanistic interpretability methods proliferate, systematic benchmarks have emerged to evaluate their effectiveness. These benchmarks provide standardized tests for comparing techniques across well-defined tasks, enabling researchers to make informed choices about which methods to employ [Rai et al., 2024].

6.1 Benchmarks for Feature Study Techniques

Several benchmarks evaluate how well various techniques can identify meaningful feature vectors for specific concepts (targeted feature study) or discover features in an open-ended manner.

CausalGym [Arora et al., 2024] introduces a suite of linguistic tasks testing the effectiveness of techniques in discovering feature vectors corresponding to given concepts, showing that Distributed Alignment Search (DAS) outperforms alternatives like probing. **RAVEL** (Resolving Attribute–Value Entanglements in Language Models) [Huang et al., 2024] benchmarks targeted feature study methods on their ability to localize and disentangle specific attributes of entities (e.g., "Paris is on the continent of"), again demonstrating DAS achieving the strongest performance over SAE and probing methods.

Mechanistic Interpretability Benchmark (MIB) [Mueller et al., 2025] provides broader coverage of both targeted and open-ended feature study. While MIB confirms DAS as the best technique for isolating feature vectors for given concepts, it reports a surprising finding for open-ended feature study: SAE features are no more effective than individual neurons at isolating meaningful features, contrary to prior claims. **SAEBench** [Karvonen et al., 2025] offers a comprehensive evaluation suite comparing various SAE architectures and training setups across eight diverse metrics including interpretability, feature disentanglement, and practical applications such as unlearning. Results indicate that Matryoshka SAEs achieve the best overall performance, particularly in feature disentanglement. Finally, **FIND** (Function INterpretation and Description) [Schwettmann et al., 2023] focuses on evaluating the correctness of interpretability language model agents in describing the latent functions implemented by model components, assessing the explanation step of open-ended feature study.

6.2 Benchmarks for Circuit Study Techniques

Evaluating circuit localization techniques presents unique challenges. Some works employ collections of synthetic [Lindner et al., 2024] or semi-synthetic [Gupta et al., 2024] transformers with known circuits, where ground truth availability simplifies evaluation. Results from these studies show that ACDC (Automatic Circuit DisCovery) and EAP-IG (Edge Attribution Patching with Integrated Gradients) achieve the strongest performance. However, concerns remain that such synthetic benchmarks may not faithfully capture the behavior of standard pre-trained transformers.

To address this limitation, **MIB** [Mueller et al., 2025] evaluates circuit localization techniques on both semi-synthetic and standard pre-trained transformer models. Their results confirm that EAP-IG outperforms other approaches like edge patching across both settings, providing evidence that findings from synthetic benchmarks can generalize to real-world models. These benchmarks collectively enable rigorous comparison of circuit discovery methods, helping researchers select appropriate techniques for their specific interpretability goals.

7 Computational Considerations

Understanding the computational requirements of different tools is essential for planning interpretability research effectively. The landscape ranges from tools that can run on a standard laptop to those requiring significant GPU clusters, and choosing appropriately can mean the difference between rapid iteration and waiting days for results.

Low-Compute Options (Laptop-Friendly)

For researchers just starting with **limited hardware**—perhaps a laptop with **16GB of RAM and no dedicated GPU**—several powerful options remain accessible:

- **Inspect AI** with API-based models requires minimal local compute, as the heavy lifting happens on the provider's servers.
- **Neuronpedia** operates entirely in the browser, requiring no local computation at all.
- **TransformerLens** for analyzing small models like **GPT-2 Small** can yield meaningful insights without expensive hardware.
- **CircuitsVis** for creating visualizations from pre-computed activations.

Even mechanistic interpretability work is feasible at this scale, enabling exploratory research without significant infrastructure investment.

Medium-Compute Scenarios (Single GPU)

Moving up to **medium-scale compute**—a workstation with **a single GPU and 16–40GB of VRAM**—opens substantially more possibilities. This configuration supports working with models up to approximately **7 billion parameters**, which includes many research-relevant architectures:

- **SAE training** becomes feasible for smaller models, though it may take several hours.
- **TransformerLens** analyses run comfortably at this scale.
- **pyvene** interventions on medium-sized models execute efficiently.
- **CircuitTracer** works well (e.g., Gemma-2 2B runs on Colab free tier with 15GB VRAM).

This represents a **sweet spot for many researchers**: enough power for serious interpretability work without requiring institutional infrastructure.

High-Compute Requirements (Multi-GPU)

High-compute scenarios—multi-GPU setups with **more than 40GB of VRAM per device**—become necessary when working with frontier models or conducting large-scale studies:

- Training **SAEs on models larger than 13B parameters**.
- Analyzing vision transformers with **Prisma**.
- Running distributed training jobs with **OpenMOSS's Language Model SAEs**.

While these resources were once available only at major research institutions, **cloud computing** has made them increasingly accessible, albeit at significant cost.

Remote Compute Access

An increasingly important category is **remote compute access**, which decouples analysis from local hardware limitations:

- **nnsight** library's integration with **NDIF (National Deep Inference Fabric)** allows researchers to perform interventions on large models hosted remotely.
- **Inspect AI's API-based evaluation** enables testing frontier models like GPT-4 or Claude without ever loading weights locally.
- **Cloud-based Jupyter environments** provide temporary access to powerful hardware without long-term infrastructure investment.

Practical Guidance

The choice of tools often depends not just on what hardware is available, but on **the research questions being asked**. A researcher investigating induction heads in GPT-2 Small needs far less compute than one training SAEs on LLaMA-70B. **Starting with the smallest model that exhibits the phenomenon of interest**, then scaling up only when necessary, remains sound research practice. The field's maturation has brought better tooling for resource-constrained researchers, but computational limitations still meaningfully shape what questions can be explored efficiently.

8 Tool Selection Guide

The following decision tree helps identify the most appropriate tools based on research objectives and constraints. Start with the primary research goal and follow the path through the questions.

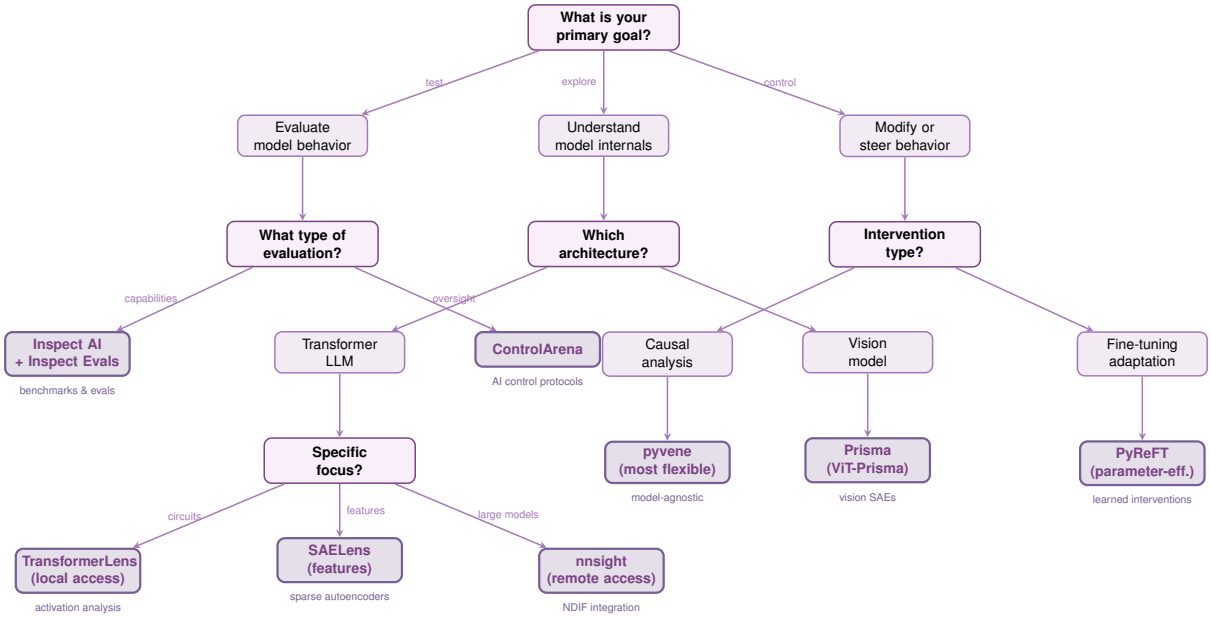

Figure 1: Decision tree for selecting AI safety and interpretability tools based on research objectives. Start at the top node and follow branches corresponding to your primary goal and constraints.

Visualization and Auxiliary Tools: Regardless of your primary path, several tools support any workflow. Use **CircuitsVis** for creating interactive visualizations of activations and attention patterns. **Neuronpedia** provides a web-based platform for exploring pre-computed SAE features and sharing interpretability findings. **Transformer Debugger** offers an interactive GUI for rapid exploration when working with small models like GPT-2.

8.1 Comparative Analysis of Mechanistic Interpretability Tools

Mechanistic interpretability tools can be systematically compared across two primary dimensions: **(a) model support and extensibility**, and **(b) interactivity and ease-of-use** [Somvanshi et al., 2025]. On the extensibility dimension, **nnsight** exemplifies maximum flexibility, working with any PyTorch model and supporting remote compute via NDIF, while **CircuitsVis** is model-agnostic for visualizations, accepting any activation data regardless of source. **TransformerLens** balances extensibility with convenience, supporting 100+ pre-configured transformer architectures with clean activation access. On the interactivity dimension, **Neuronpedia** and **Transformer Debugger** provide fully GUI-based experiences accessible to researchers without coding requirements—Neuronpedia operates entirely in the browser for exploring SAE features, while Transformer Debugger offers real-time interactive investigation of model behaviors with automated GPT-4 explanations. In contrast, **TransformerLens**, **nnsight**, and **pyvene** are code-first libraries requiring programmatic workflows but offering maximum research flexibility. **SAELens** bridges both categories: it requires coding for SAE training but generates interactive SAE-Vis dashboards for feature exploration. The field employs several fundamental techniques including activation patching (identifying causal factors), sparse autoencoders (decomposing activations into interpretable features), induction head tracing (detecting pattern-reproducing attention heads), logit lens (visualizing layer-wise predictions), and toy models (simplified systems for controlled study), together representing the spectrum of techniques for making AI systems comprehensible and trustworthy [Somvanshi et al., 2025].

9 Community Resources

Beyond individual tools, the AI safety and interpretability community maintains several valuable resources. The AI Safety Support website at `https://www.aisafety.com/projects` provides a community-maintained list of volunteer AI safety projects and tools. Anthropic's Transformer Circuits Thread at `https://transformer-circuits.pub/` offers an ongoing research publication series on mechanistic interpretability, including landmark work on induction heads, monosemanticity, and circuit tracing. The Open Source Mechanistic Interpretability Slack hosts an active community for tool development and research collaboration, while the LessWrong AI Alignment Forum facilitates community discussion and research sharing, including extensive coverage under the Sparse Autoencoders tag for SAE research.

References

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Aryaman Arora et al. Causalgym: Benchmarking causal interpretability methods on linguistic tasks, 2024. Referenced in Rai et al., 2024.

David Bau. Baukit: Tools for analyzing neural network internals. `https://github.com/davidbau/baukit`, 2022. Accessed: 2026-01-29.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023. URL https://proceedings.mlr.press/v202/biderman23a.html.

Joseph Bloom, Curt Tigges, Anthony Duong, and David Chanin. Saelens: Training sparse autoencoders on language models. https://github.com/jbloomAus/SAELens, 2024. Accessed: 2026-01-29.

Concordance Research. Concordance token injection lab. https://research.concordance.co/playground, 2024. Accessed: 2026-02-06.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023.

Xuyang Ge, Wentao Shu, Junxuan Wang, Guancheng Zhou, Jiaxing Wu, Fukang Zhu, Lingjie Chen, and Zhengfu He. Language-model-saes: Performant framework for training and analyzing sparse autoencoders. https://github.com/OpenMOSS/Language-Model-SAEs, 2024. Accessed: 2026-01-29.

Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. OLMo: Accelerating the science of language models. *arXiv preprint arXiv:2402.00838*, 2024. URL https://arxiv.org/abs/2402.00838.

Tanishq Gupta et al. Semi-synthetic transformers for circuit evaluation, 2024. Referenced in Rai et al., 2024.

Michael Hanna, Mateusz Piotrowski, Jack Lindsey, and Emmanuel Ameisen. circuit-tracer: Finding circuits using transcoder features. https://github.com/safety-research/circuit-tracer, 2025. The first two authors contributed equally and are listed alphabetically. Accessed: 2026-01-29.

Jing Huang et al. Ravel: Resolving attribute–value entanglements in language models, 2024. Referenced in Rai et al., 2024.

Sonia Joseph, Praneet Suresh, Lorenz Hufe, Edward Stevinson, Robert Graham, Yash Vadi, Danilo Bzdok, Sebastian Lapuschkin, Lee Sharkey, and Blake Aaron Richards. Prisma: An open source toolkit for mechanistic interpretability in vision and video, 2025. URL https://arxiv.org/abs/2504.19475.

Joonas Karvonen et al. Saebench: A comprehensive evaluation suite for sparse autoencoders, 2025. Referenced in Rai et al., 2024.

Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Varma, Janos Kramar, Anca Dragan, Rohin Shah, and Neel Nanda. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2. *arXiv preprint arXiv:2408.05147*, 2024. URL https://arxiv.org/abs/2408.05147.

Johnny Lin. Neuronpedia: Platform for sharing interpretability research. https://neuronpedia.org, 2024. Accessed: 2026-01-29.

David Lindner et al. Synthetic transformers with known circuits, 2024. Referenced in Rai et al., 2024.

Zhengzhong Liu, Aurick Qiao, Willie Neiswanger, Hongyi Wang, Bowen Tan, Tianhua Tao, Junbo Li, Yuqi Wang, Suqi Sun, Omkar Pangarkar, et al. LLM360: Towards fully transparent open-source LLMs. *arXiv preprint arXiv:2312.06550*, 2023. URL https://arxiv.org/abs/2312.06550.

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. Transformer debugger. https://github.com/openai/transformer-debugger, 2024. Accessed: 2026-01-29.

Aaron Mueller et al. Mechanistic interpretability benchmark, 2025. Referenced in Rai et al., 2024.

Neel Nanda et al. Transformerlens: A library for mechanistic interpretability of gpt-style language models. https://github.com/TransformerLensOrg/TransformerLens, 2024. Accessed: 2026-01-29.

NDIF Team. nnsight: A framework for interpreting pytorch models. https://github.com/ndif-team/nnsight, 2024. Accessed: 2026-01-29.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 2020. URL https://distill.pub/2020/circuits/zoom-in/.

Pillar Security. Sail framework: Secure ai lifecycle. https://www.pillar.security/sail, 2024. Accessed: 2026-02-06.

Daking Rai, Yilun Zhou, Shi Feng, Abulhair Saparov, and Ziyu Yao. A practical review of mechanistic interpretability for transformer-based language models. *arXiv preprint arXiv:2407.02646*, 2024. URL https://arxiv.org/abs/2407.02646.

Sarah Schwettmann et al. Find: Function interpretation and description, 2023. Referenced in Rai et al., 2024.

Shriyank Somvanshi, Md Monzurul Islam, Amir Rafe, Anannya Ghosh Tusti, Arka Chakraborty, Anika Baitullah, Tausif Islam Chowdhury, Nawaf Alnawmasi, Anandi Dutta, and Subasish Das. Bridging the black box: A survey on mechanistic interpretability in ai. SSRN, 2025. URL https://papers.ssrn.com/abstract=5345552. Accessed: 2026-02-06.

TransformerLens Organization. Circuitsvis: Mechanistic interpretability visualizations using react. https://github.com/TransformerLensOrg/CircuitsVis, 2024. Accessed: 2026-01-29.

UK AI Safety Institute. Controlarena: Evaluating ai control protocols. https://github.com/UKGovernmentBEIS/control-arena, 2024a. Accessed: 2026-01-29.

UK AI Safety Institute. Inspect ai: A framework for large language model evaluations. https://github.com/UKGovernmentBEIS/inspect_ai, 2024b. Accessed: 2026-01-29.

UK AI Safety Institute. Inspect evals: Pre-built evaluation suite. https://github.com/UKGovernmentBEIS/inspect_evals, 2024c. Accessed: 2026-01-29.

Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. ReFT: Representation finetuning for language models. 2024a. URL arxiv.org/abs/2404.03592.

Zhengxuan Wu, Atticus Geiger, Aryaman Arora, Jing Huang, Zheng Wang, Noah Goodman, Christopher Manning, and Christopher Potts. pyvene: A library for understanding and improving PyTorch models via interventions. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: System Demonstrations)*, pages 158–165, Mexico City, Mexico, June 2024b. Association for Computational Linguistics. URL https://aclanthology.org/2024.naacl-demo.16.