# Automating Interpretability Research in Deep Learning Models

Matthew Shinkle (matthewshinkle@gmail.com)
Yeonwoo Jang (yeonwoojangus@gmail.com)
Jacques Thibodeau (thibo.jacques@gmail.com)

July 19, 2025

**Abstract**

As AIs become more capable, automating AI research and development is emerging as a critical pathway to advance model interpretability and overall AI safety. This project develops a set of tools that integrate into a pipeline for parsing research papers, retrieving and understanding relevant codebases, and designing and running experiments. We present techniques for improving interpretability research by AI agents, including paper search and parsing, codebase discovery and preparation, remote execution, and automated package documentation. We demonstrate these tools through a sandbox environment for sparse autoencoders (SAEs) that enables autonomous implementation and evaluation of diverse SAE variants.

Our framework includes tools for discovering and processing research papers to identify key ideas, methodologies, and performance metrics. It provides methods for finding, validating, and processing codebases associated with research papers. The system supports experiment design and execution through cloud-based GPU instances, with features for configuration management and result collection. We show that these components can be combined to automate aspects of interpretability research, using SAE variants as a proof of concept. This approach may be expanded to other interpretability tasks as the underlying tools mature.

## 1 Introduction and Statement of the Problem

AIs are rapidly improving in their capacity to automate scientific research [1]. Maximizing the rate and impact of interpretability work requires leveraging these advances. While current models may be incapable of full automation, scaffolding can still improve their current usefulness while smoothing the path to full automation [2]. We developed tools that we believe enhance current and near-future models' ability to use interpretability research tools. We hope that further refinement of these techniques will accelerate the overall pace of interpretability research.

The development of these tools is particularly important for interpretability research, which faces unique challenges in automation. Compared to mainstream machine learning research, interpretability-focused codebases have smaller user bases and less representation in training data. This creates a systematic disadvantage: while LLMs can effectively work with widely-used packages like PyTorch, they struggle more with niche interpretability packages, often making fundamental misunderstandings of their structure and functionality. This disparity is reflected in both training data and evaluation benchmarks, which heavily favor mainstream ML packages over safety-focused ones.

Rather than waiting for models to improve, we believe there are concrete steps that can be taken now to accelerate interpretability research [3]. First, the field's relatively small size and focused scope presents opportunities for targeted improvements that wouldn't scale to larger domains. Second, the collaborative nature of safety research allows for shared development of tools and standards. Finally, the current usefulness of AI in research means that even incremental improvements can have immediate impact [4].

Our approach focuses on developing tools that specifically address the challenges of automating interpretability research. This includes structured sandbox environments that provide clear guardrails for experimentation, improved package documentation and interoperability, and specialized tools for processing research papers and codebases. We demonstrate these tools through a sandbox environment for sparse autoencoders (SAEs), showing how they can enable autonomous implementation and evaluation of diverse SAE variants.

## 2 Methodology

Our research methodology follows a structured, iterative approach that integrates multiple components into a unified autonomous research pipeline. The system is designed to parse provided literature, retrieve and process relevant codebases, design and execute experiments, and synthesize results for continuous refinement. We developed a range of tools to facilitate the automation of interpretability experiments:

### 2.1 Paper Search and Discovery

Our paper search system combines three complementary approaches: arXiv's structured API, Perplexity's semantic search, and Exa's web search capabilities. The system queries each source sequentially, with arXiv providing metadata through its official API, Perplexity offering semantic matching, and Exa searching across the web. Each discovered paper is evaluated using a specialized Perplexity prompt that assesses relevance to the search context, assigning confidence scores and providing reasoning. The system deduplicates results based on title and year, keeping the highest confidence version of each paper.

### 2.2 PDF Parsing

The PDF parsing system extracts structured information from research papers by processing them in chunks. Each chunk is analyzed for experimental methodology, results, and future work. The system captures technical parameters, validation methods, computational requirements, quantitative metrics, and qualitative findings. It also identifies explicit limitations, failure modes, and proposed future work directions. The extracted information is structured into a consistent format that maintains academic rigor while enabling systematic comparison across papers.

### 2.3 Codebase Discovery

The codebase discovery component uses a two-stage approach to identify and validate codebases associated with research papers. First, it extracts direct references from the paper and uses AI to search for relevant implementations. Each discovered codebase is then validated by analyzing repository documentation and implementation details to determine correspondence with the paper's methods. Codebases are classified as primary or secondary based on validation scores, with primary codebases representing the most authoritative implementations. The final output organizes discovered codebases hierarchically, enabling quick identification of the most relevant implementations.

### 2.4 RepoMix Codebase Processing

The RepoMix component processes both remote and local repositories to create consolidated representations. It converts Jupyter notebooks into executable scripts and generates structured outputs in XML and markdown formats. This standardization helps researchers understand and compare different implementations while preserving their original functionality. The processed outputs provide a clear view of the codebase's structure and implementation details.

### 2.5 Project Guides

The project guides component learns package functionality by exploring example code, documentation, and source implementations. It identifies primary use cases and common patterns through analysis of demo scripts and source code. The component tests these implementations, documenting successful approaches and potential pitfalls. This process captures common issues, their solutions, and tested integration patterns with other packages. The resulting guides help prevent common implementation challenges and streamline package adoption.

### 2.6 Sandbox Environments

Our sandbox environments are specifically designed to enable LLM agents to safely explore variations of interpretability techniques. These environments can be created through three approaches: human development, LLM-assisted generation, or collaborative development combining both approaches. The SAE sandbox environment, for example, was developed collaboratively using an LLM agent that leveraged package guides and consolidated codebases, along with human refinement to ensure correctness and usability.

The key innovation of these sandboxes is their design for agentic exploration. They are structured to allow LLM agents to implement variations in a streamlined manner, with minimal risk of undesired side effects. This is achieved through two main mechanisms: first, by isolating variations to specific, well-defined interfaces that prevent modifications to core infrastructure, and second, by implementing strict guardrails that prevent agents from making changes that could compromise evaluation integrity, such as modifying downstream metrics or evaluation procedures.

The sandbox lets LLM agents experiment with architectural modifications like skip connections or new nonlinearities without requiring deep expertise in the underlying codebase. These environments serve as reusable templates that can be shared across the research community, enabling automated exploration of interpretability techniques while maintaining rigorous experimental standards.

### 2.7 RunPod Execution

The RunPod execution component manages cloud-based GPU instances from creation through execution and result collection. It supports various experiment types including SAE training, transcoder training, and custom evaluations. The component handles configuration management, file transfers, and result collection, with features for pod reuse and parallel job execution. This infrastructure lets researchers scale experiments across multiple GPUs without managing the underlying resources.

### 2.8 AI Scientist

The AI Scientist component orchestrates the interpretability research workflow through an iterative loop of planning, execution, analysis, and reflection. It analyzes research questions to generate structured plans, executes experiments using local or cloud resources, and supports various experiment types including SAE training and transcoder experiments. After each experiment, the component analyzes results, generates insights, and determines next steps. It compares different approaches, identifies patterns, and suggests parameter modifications while maintaining comprehensive research state. The component incorporates knowledge from research papers and codebases discovered by other components, creating a closed-loop research system that can autonomously conduct experiments while maintaining scientific standards.

## 3 Results

As a proof-of-concept, we developed and tested our tools through a sandbox environment for sparse autoencoders (SAEs). This sandbox was created collaboratively using an LLM agent that leveraged package guides and consolidated codebases, along with a specialized sandbox generation prompt and human refinement. The sandbox provides a structured environment for LLM agents to safely explore SAE variants, with clear guardrails that prevent breaking the core training and evaluation pipeline.

The sandbox is organized around a modular architecture that separates the core training and evaluation infrastructure from custom SAE implementations. At its core is a base SAE implementation built on the dictionary_learning package, which provides the base SAE class and trainer with well-defined interfaces for encoding, decoding, and training. This is complemented by a standardized training pipeline that handles data loading, model configuration, and training loops, along with a benchmarking system that evaluates SAEs using the sae-bench package. A utility layer provides common functionality and ensures consistent behavior across different implementations.

The key innovation is that custom SAE variants can be implemented by an LLM agent through a single Python file that inherits from the base classes and overrides specific methods. This design ensures that architectural modifications are isolated and can't break the core pipeline or compromise evaluation integrity. The sandbox supports various types of modifications, including changes to the encoding process (such as different nonlinearities or normalization), modifications to the decoding process, custom loss functions and training objectives, and architectural variations like skip connections or multiple paths.

Below is a list of the 24 SAE variants that our LLM model successfully brainstormed, implemented, and benchmarked within the sandbox environment:

**dynamic_k:** uses learnable k for top-k feature selection

**cross_latent_causality:** penalizes cross-latent derivatives for independent features

**synonym_factor_merging:** merges offset features into core factors via clustering

**register_balance:** uses style embeddings and gating for style-robust features

**mixed_granularity:** applies group sparsity at multiple scales

**task_weighted_jacobian:** uses learnable task weights for feature selection

**synonymic_overlap_minimizer:** penalizes co-activated features with different decoders

**self_conditioned:** conditions activations on overall pattern for balanced usage

**rare_concept_preservation:** rewards usage of rare features

**partitioned_gram_l1:** enforces group-wise orthogonality via Gram matrix

**orthogonal_feature:** penalizes decoder Gram matrix off-diagonals for orthogonality

**morpheme_driven:** penalizes affix features without active stem features

**mixed_precision_top_k:** uses learnable precision levels for hierarchical selection

**local_orthogonality_penalty:** penalizes co-activated feature pairs for orthogonality

**layered_shrinkage_switch:** applies multi-stage sparsity with adaptive switches

**hierarchy_promoting:** penalizes active children with inactive parents

**group_sparsity:** uses learnable group sizes for group and feature sparsity

**global_local_tension:** splits features into global/local, penalizes global instability

**feature_zoom_out:** aligns parent decoder weights with sum of children

**feature_recycling:** penalizes high cosine similarity between decoder weights

**entropy_sparsity:** uses entropy penalties for local and global sparsity

**dynamic_covariance:** penalizes off-diagonal feature covariance

**correlation_minimization:** penalizes off-diagonal feature correlation

**competitive_inhibition:** applies group-wise competition to limit active features

As shown in Figures 1 and 2, we used this sandbox environment to brainstorm, implement, train, and benchmark 24 SAE variants, with all steps of this process performed entirely by an agentic LLM (Claude 3.7 as a Cursor agent). While we do not focus on the detailed results, we do note some interesting findings. First, of the 30 different variants brainstormed via our LLM, twenty-four of these were successfully implemented, trained, and benchmarked by the agentic LLM alone. While some implemented variants took one or more steps of subsequent (fully antonymous) LLM debugging, this does suggest that our sandbox environment enables LLMs to effectively implement a diverse range of autoencoder variants [5].

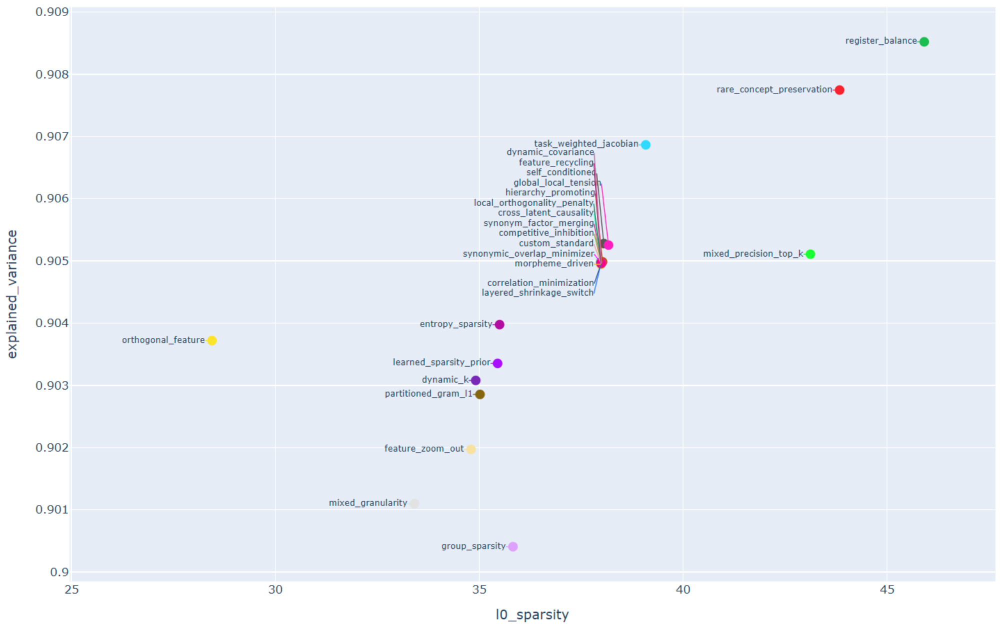

Figure 1: SAEBench results for 24 different LLM-generated SAE variants. X axis shows L0 sparsity of latent SAE activations. Y axis shows reconstruction performance measured as explained variance (EV). See the SAEBench package for more details of these metrics.

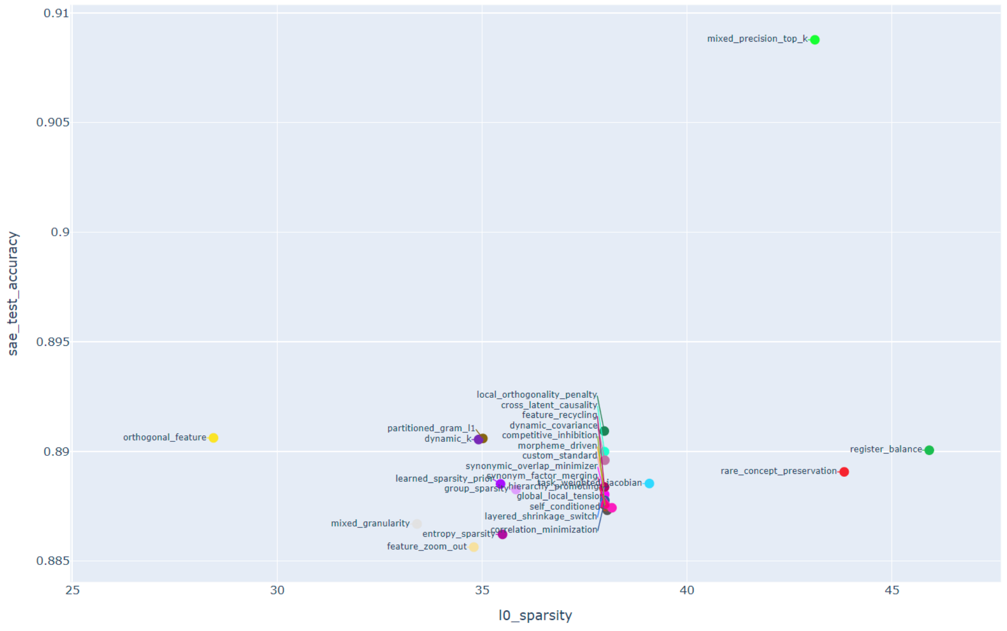

Figure 2: SAEBench results for 24 different LLM-generated SAE variants. X axis shows L0 sparsity of latent SAE activations. Y axis shows the accuracy of sparse probes trained on latent SAE activations. See the SAEBench package for more details of these metrics.

Second, when compared to common SAE variants already implemented within the dictionary_learning package, we find that many of our custom variants achieve competitive benchmark performance (MSE, explained variance, sparse probing test accuracy) for a given level of sparsity (L0, L1). While further testing at fixed sparsity levels, as well as comparison along other benchmark scores, would be valuable, some of these appear to be promising SAE variants.

The successful implementation and evaluation of these variants demonstrates the effectiveness of our sandbox environment in enabling LLM agents to explore architectural variations. The high success rate (24 out of 30 attempted variants) suggests that the sandbox's clear interfaces and guardrails effectively guide the LLM in implementing valid modifications while preventing errors that could compromise the evaluation process. This success rate, combined with the competitive performance of many variants, indicates that our approach to automating interpretability research through structured sandbox environments shows promise for accelerating the development and evaluation of new techniques.

## 4 Discussion and Conclusions

We developed tools for automating interpretability research—paper parsing, codebase prep, package docs, sandbox setups, remote execution, and AI-driven orchestration. We tested them on sparse autoencoders by building a guarded sandbox where an LLM could propose, implement, and benchmark SAE variants. We expect these tools—and their refinements—to speed up AI interpretability via automated workflows [6].

The collaborative development and sharing of package guides, sandbox templates, and tools would avoid redundant effort across the safety research community. A key consideration is whether these tools might inadvertently accelerate harmful research directions. We believe this risk is low for several reasons. First, the tools are specifically designed to address the systematic disadvantages faced by interpretability research, such as limited representation in training data and evaluation benchmarks. Second, the collaborative development and sharing of these tools within the safety research community would naturally limit their dissemination to other areas. Finally, the competitive nature of capabilities research means these tools would likely have less impact there than in safety research, where collaboration is more common [7].

The development of sandbox environments represents a key direction for future work. While our SAE sandbox was created through a combination of LLM assistance and human refinement, future work could focus on automating the sandbox creation process itself. This could involve developing better prompts and templates for sandbox generation, creating tools to automatically extract and structure package interfaces, building systems to validate sandbox correctness and completeness, and establishing standards for sandbox design and documentation. By automating more of the sandbox creation process, we could further enable LLM agents to safely explore variations of interpretability techniques while maintaining high standards of experimental rigor.

## References

[1] Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

[2] METR. Measuring ai ability to complete long tasks. `https://metr.org/blog/2025-03-19-measuring-ai-ability-to-complete-long-tasks/`, 03 2025.

[3] Abram Demski. Llms for alignment research: a safety priority? `https://www.lesswrong.com/posts/nQwbDPgYvAbqAmAud/llms-for-alignment-research-a-safety-priority`, 04 2024.

[4] Jennifer Chubb, Peter Cowling, and Darren Reed. Speeding up to keep up: exploring the use of ai in the research process. *AI & Society*, 37:1439–1457, 2022.

[5] METR. Measuring automated kernel engineering. `https://metr.org/blog/2025-02-14-measuring-automated-kernel-engineering/`, 02 2025.

[6] Jacob Pfau and Geoffrey Irving. Prospects for alignment automation: Interpretability case study, 03 2025.

[7] Joe Carlsmith. Ai for ai safety. `https://www.lesswrong.com/posts/F3j4xqpxjxgQD3xXh/ai-for-ai-safety`, 03 2025.