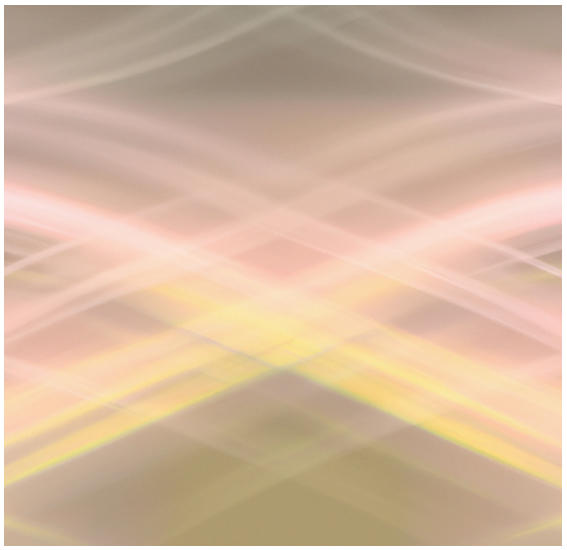
High-Level Mechanistic Interpretability Activation Engineering Library 🔥

Key Terms: Activation Engineering, Mechanistic Interpretability, Tooling, Python, TransformerLens, Quix, Mojo.



Summary

The fields of **mechanistic interpretability** and **activation engineering** have produced a variety of novel techniques for understanding the inner workings of Large Language Models. However, these approaches currently lack **unified software tooling** and standardised interfaces. This results in duplicated effort as researchers build one-off implementations.

Existing libraries cover a range of explainable AI methods for shallow learning models. But contemporary research on large neural networks calls for new tooling. This project seeks to build a **well-architected** library specifically for current techniques in mechanistic interpretability and activation engineering.

The library will emphasise comprehensive documentation and tutorials using a **tutorial-driven development** approach. It will build on best practices while innovating through the highly performant and **Python-compatible Mojo programming** architecture. Open questions include improving interfaces via **in-memory event streams** for minimal-overhead logging. This library is critical for accelerating leading research on model transparency. The project vision is an **extensible, communal** codebase where researchers easily build upon and extend each other's contributions.

The non-summary

Project Aims

This project aims to create an open source Python library to accelerate and enhance mechanistic interpretability and activation engineering research.

The goal is a flexible, lightweight framework with **intuitive classes, functions, and interfaces** to distil key approaches from published papers into **reusable** software components. This will allow rapid **implementation, comparison, and iteration of ideas** using standard idiomatic APIs tailored to the field. This would further the aim of a shared ecosystem enabling transparent and reproducible AI research.

This library could reduce risks from artificial general intelligence (AGI) and transformative AI (TAI) by enabling **faster feedback cycles** for human and AI alignment researchers. Crisp abstractions and accelerated research timelines make it more likely that **interpretability solutions are developed prior to AGI** capabilities being unlocked. Interpretable AI systems that can explain their reasoning in a scrutable manner reduces unintended consequences and existential risk.

Objectives

Two subfields of alignment research would especially benefit from a shared high-level library.

Mechanistic interpretability focuses on probing how language models (LMs) represent knowledge, make decisions, and learn from context. The library will contain modules to implement published techniques for visualising and explaining how information flows through model architectures. Researchers will be able to analyse model internals and behaviours using consistent interfaces tailored to this domain.

Activation engineering involves directly analysing and editing the neuron activations within deep learning models. The tooling developed in this library will enable researchers to readily conduct activation manipulation experiments, such as optimising activations based on theoretical models or editing activations to probe the valence of provided responses

The project objective is to bring similar acceleration to interpretability research by creating an open-source Python library with APIs for:

- Common **model introspection** tasks (getting activations, weights)
- **Manipulating** model internals
- Switching between input/output **representations**
- Applying published interpretation **techniques**
- **Visualising** and **analysing** results

This software library aims to amplify human researchers' capacities through abstractions, automation, and collaboration. Tools can enhance analysis and visualisation of internals like **activation traces**, which are a prerequisite for decision tracing.

Even incremental progress through enhanced visualisation, experimentation, and **pattern discovery**

moves the field closer to transparent and aligned AI. That is to say, surfacing greater model interpretability moves the field closer to transparent and aligned AI.

Feature List

Core Features

- [ ] Enables Reproducible LM research analyses.
- [ ] Works with any PyTorch Model
- [ ] Standard interfaces for common research tasks examining Language Model internals.
- [ ] Implements a number of widely-cited research papers.
- [ ] Extensive Tutorials through Tutorial-Driven Development workflow.
- [ ] Blazing fast - Written in Mojo and Python.
- [ ] Open Source Github Release and Contributing Guide.

Additional Features

- [ ] Mojo deployment on Packaging Index.
- [ ] Integration with HuggingFace 🤗Transformers.
- [ ] Interoperability with the NumFocus project ecosystem tools.
- [ ] Implements a larger number of widely-cited research papers.
- [ ] Broader documentation (similar to Choosing the Right Estimator from scikit-learn)
- [ ] Exemplar high-performance visualisations using Datashader

What will the library do?

The library will implement a variety of techniques, to be **selected during the research project** from the available research literature. The following section **describes some possibilities** and indicates how these techniques may be **integrated into a library**.

Interpretability Research Techniques

Mechanistic Interpretability (link) aims to understand large language models (LLMs) by examining their internal representations and computations. By tracing activations and **functionally disentangling layers**, researchers can **reverse engineer** the mechanisms by which LLMs process language. While currently done through manual inspection, there is potential to automate parts of this analysis to enable faster and scaled up mechanistic interpretation.

Two key techniques are activation **tracing and layered slicing of model weights**. **Activation** tracing involves propagating outputs backwards to determine which input components were most influential on the output. This reveals how information flows through the network. Meanwhile, slicing off layers of the model and analysing the changes allows attributing different functions to different levels. For example, lower layers may focus on syntax while higher layers handle semantics. By examining **connections between artificial neurons**, researchers aim to expose meaningful learned algorithms embedded in the weights. Making implicit computations explicit is key to increasing model transparency.

Concrete Problems in Mechanistic Interpretability (link) - Appendix A

Scalable Interpretability Research Techniques

A number of techniques also offer an **automated approach to interpretability**

Automated Circuit Discovery (ACDC) **-** Appendix B
Language models can explain neurons in language models (link) **-** Appendix C

Activation engineering Research Techniques

Activation Engineering Approaches (link) **-** Appendix D

Library Design

Overview

This interpretability library aims to provide researchers with an **extensible toolkit** enabling rapid implementation and comparison of analysis techniques. To fulfil this vision, the library needs:

- **Flexible interfaces** to reconstruct diverse published methods with minimal boilerplate.
- Mechanisms to access and transform **model internals** like activations for examination.
- **Performant architecture** to accelerate experiment iteration.
- Strong documentation and **tutorials** to onboard users.
- The library draws **inspiration from prior explainability tools** but requires more **modular interfaces** to capture the complexity of mechanistic analysis. Carefully designed base classes can provide researchers semantic building blocks to swiftly develop novel experiments through composition.

Advanced streaming pipelines promise to unlock real-time monitoring and querying abilities previously impossible with basic hooks. Integrating the **emerging Mojo language** can selectively accelerate compute-heavy operations while maintaining Python's flexibility.

Flexible Interfaces

Scikit-learn provides interfaces like .fit() and .predict() to standardise workflows for machine learning. This enables simple swapping between models and estimators. **Similar to scikit-learn**, **Alibi Explain** adopts the .explain() method to bring some uniformity.

But many techniques cannot be reduced to explaining black-box predictions. Mechanistic analysis in particular requires flexible internal access. However, interpretability algorithms are more complex so **modular, composable interfaces are preferable** to method standardisation.

For mechanistic interpretability, base classes like **ActivationsExtractor, RepresentationTransformer, and InterpretationMethod** could provide semantic building blocks. These reflect the composable stages of analysis while handling implementation details under the hood.

Researchers could combine these reusable components to swiftly benchmark ideas. **New papers** would just require **implementing a subclass** rather than full workflows. As a result, composition empowers novel experiments.

Overall, standardised methods work for simplified workflows but complex analyses require flexible interfaces. Base classes that encapsulate semantics, not just methods, enable **rapid innovation** through **mix-and-match composability**.

Explainable AI libraries

Mechanistic Interpretability technique libraries can be compared to prior Explainable AI technique libraries.

Mechanistic interpretability qualitatively reverse engineers models, while automated Explainable AI techniques (cheatsheet) like LIME and SHAP seek systematic explanations by attributing importance to inputs. However, perturbations ignore internals and can mislead on nonlinearity. Inherently interpretable models like linear regression provide understanding by design. For black boxes, perturbation treats models as functions to explain outputs using inputs.

More powerful for large language models are gradient and attention-based attribution. Integrated gradients reveal importance along prediction paths. Attention visualisation shows relevance in transformers. Libraries like Captum and Alibi implement many techniques with scikit-learn style APIs.

Captum (link) **and Alibi** (link) **-** Appendix E

Hooks, Probes and In-Memory Streams

Below the high-level interface, there are opportunities to build low-level internals with robust representations that can read/write from language model internals more effectively. Prioritising high-level versus low-level work will happen during the first week of the project contingent on the assembled team.

Effective model interfacing is crucial for advanced interpretability analysis. Existing primitive hooks provide basic access but lack capabilities and performance needed for large-scale inspection. More powerful interfaces could enable complex real-time monitoring and ad-hoc querying of model internals with a **'listen to event streams rather than probe'** architecture.

Small scope approaches like logging activations to databases facilitate some advanced analytics. But managing storage and ensuring fast response times presents challenges. Larger scope streaming architectures unlock transformative abilities for **live tracking and visualisation**.

Simple **hook-based** approaches provide **basic** functionality but limit possibilities. More advanced interfaces like **fast in-memory streams** such as **Python's Quix** (link) or Java's Kafka (link) with interactive processing better serve complex interpretability needs. The goal is **maximising accessibility of model internals** for interpretation **without needless complexity.**

Performance & Mojo

Performance is critical for this interpretability library to enable fast experimentation. Thus, parts will leverage **Mojo**, a new language claiming **35,000x speedups over Python**. Mojo is created by **Chris Lattner** of **LLVM** and **Swift** fame, aiming to become a **superset of Python**.

Mojo offers low-level control and optimizations tailored for **machine learning workloads**. The hope is its speed could accelerate research feedback loops. However, as an **immature** language, Mojo currently lacks capabilities like non-M1/M2 Apple/Windows support and list comprehensions. There are risks developing in a new ecosystem.

Mojo provides several low-level performance optimizations tailored for machine learning workloads:

- **Typed and compiled functions** enable static checking and efficient execution.
- **Structs** improve cache usage compared to classes through dense sequential memory layout.
- Vectorization applies **SIMD** instructions to operate on multiple data points concurrently.
- Multithreading **parallelism** computation across CPU cores.
- **Tiling** partitions matrices to better fit processor caches.
- **Auto-tuning** finds optimal tile sizes for specific hardware.
- Register tiling caches intermediate results in **fast registers**.

A key advantage of Mojo is the ability to **selectively incorporate** these optimizations into existing Python code. This provides a customizable performance continuum **balancing development speed and runtime efficiency**. Carefully targeted Mojo integration will help accelerate compute-intensive operations without rewriting entire workflows. Mojo offers strategic low-level speedups that can be smoothly integrated into Python's flexibility. Through modular performance optimizations and Python interop, it aims to **complement rather than fully replace** established languages.

The Development Process

Development will use an **agile, scrum-like process with fortnightly sprints**. A tutorial-driven approach will guide implementation, building classes and functions needed for tutorials to function. This iterative refinement will convert published methods into working code. The vision is a shared platform enabling rapid reproduction and extension of interpretability ideas.

Project Scope

Individual work packages are components. They can be worked on concurrently.

WP1: Requirements Gathering
Explore the ecosystem and capture the needs of researchers.

WP2: Implement Reference Papers
Reimplement 4 research papers as working tutorial-driven code.

WP3: Modular Architecture
Refactor code for extensibility, abstraction, and customization.

WP4: Documentation and Distribution
Create tutorials, API docs, contributing guidelines.
Package and distribute code to researchers.

Output

The primary research outputs will be:

- A **GitHub repository** containing well-documented code and tutorials in the form of Jupyter notebooks, guides, and API references. This allows sharing the tooling with the alignment research community.
- A blog post on the **Alignment Forum** motivating the library's development and describing its architecture and potential applications for AI safety. This disseminates the tool specifically within the alignment ecosystem.
- Thoughtful open source licensing, potentially MIT or GPL, to uphold software freedoms while considering security implications.
- Distribution strategies involve tradeoffs between security through obscurity and broader reach. The code could be shared quietly at first or released widely with Alignment Forum publication.

Potential secondary outputs, pending careful review of possible risks, are:

- Distribution to the Python ecosystem via a **PyPI** package for easy installation and imports.
- An **academic paper** in the style of scikit-learn's seminal 2011 JMLR publication, detailing the library's specifications.

Sharing code, documentation, and tutorials with the alignment community is the priority. Secondary outputs like PyPI distribution or academic publication require extensive ethical review to assess possible risks. The aim is maximising benefit while upholding rigorous safety standards.

Risks and downsides: Security and Open-Source

Interpretability tools carry inherent **dual-use** risks as they could aid either safety or capabilities research. However, open sourcing also enables collaboration and accountability. With thoughtful practices, benefits like faster progress through community input can be realised while limiting potential downsides.

An open source approach provides opportunities for auditing, enhancements, funding, and education that closed-source lacks. Implementing safety-focused techniques over capabilities-heavy techniques stewards progress responsibly. Overall, openness with care enables collective advancement on interpretability for good.

Acknowledgements

Special thanks to Marc Carauleanu, Dennis Akar, Lucius Bushnaq, Anna Dombroski and Dr Bogdan Ionut-Cirstea for their feedback and ideas while writing this proposal.

Team

Team size
Five skilled researchers.

Research Lead and Team Coordinator
Jamie Coombes
https://github.com/jcoombes/
coombes.jd@gmail.com

I'm a data consultant with 3 years of Python and PyTorch development experience. I've provided ML expertise to startups and the UK government, I am interested in beneficial AI applications.

I spoke at EuroPython Prague this summer and I have mentoring experience through my prior role as a Science Teacher with TeachFirst. My background is studying Physics and then Atmospheric Physics interpreting large tropical cyclone datasets at Imperial College London.

I work in UTC timezone and can commit one day a week to this project for coordinating sprint activity around existing Data Consulting engagements. This would be on Wednesdays (9:30-5:30).

Skill requirements
We welcome applicants of all backgrounds - no minimum 3-5 years experience in Mojo required! (Though bonus points if you have somehow mastered a language that didn't exist until 2023).

More seriously, we seek collaborative folks excited to advance interpretability research through engineering ingenuity. Specifically:

- Experience with PyTorch, Jupyter, and data science workflows to implement analysis techniques.
- Familiarity with transformer architectures and large language models through resources like The Illustrated Transformer. Curiosity is sufficient, mastery is not expected.
- Research and Academic writing skill to help document methodologies and results for publication.
- Low-level systems programming expertise to quickly pick up Mojo for performance-critical operations. This is by no means mandatory but is a plus.
- Passion for advancing safe and beneficial AI through interpretability techniques. Mindset matters more than credentials.

This project thrives on diversity of thought and mutually empowering contributions. If you're eager to create beneficial technologies, you belong here.

Appendices

The following sections describe some core, widely understood ideas in Mechanistic Interpretability and Activation Engineering.

Appendix A - Concrete Problems in Mechanistic Interpretability

Concrete Problems in Mechanistic Interpretability (link) - Appendix A

Looking for Circuits in the Wild involves reverse engineering real language models like GPT-2 Small to find computational circuits and understand how techniques scale. For example, Anthropic researchers identified circuits for indirect object identification in GPT-2 Small. Next steps would involve applying similar methods to models like GPT-Neo Small and analyzing how the circuits compare in terms of implementation, robustness, and confidence levels as model size increases. Studying circuits in larger models will reveal new challenges that arise with scaling and refine beliefs about how well insights apply across model families.

Interpreting Algorithmic Problems trains models on simplified tasks like modular addition which have clear ground truths. This allows testing interpretability techniques and honing skills with limited real-world usefulness but easier interpretability due to the simplicity. For instance, researchers could train and reverse engineer a single layer transformer doing 5-digit addition and analyze whether the learned circuits relate at all to the Fourier Transform based algorithm commonly used for efficient modular addition. Simplified domains like algorithmic tasks enable iterative refinement of methods in a controlled setting where true representations are known.

Studying Learned Features aims to catalogue features represented in multilayer perceptron (MLP) layers, which are less understood than other neural network components. Early layers likely learn simple features which get built into more complex features in later layers. For example, the SoLU paper found individual neurons implicitly representing numeric descriptions of groups of people. More systematic catalogueing of learned features across model layers is needed to advance understanding of which concepts are represented and how they evolve through the network.

Appendix B - Automated Circuit Discovery

Automated Circuit Discovery (ACDC) **- Appendix B**

Automated Circuit Discovery (ACDC) aims to accelerate the process of mechanistic interpretability by automatically searching for meaningful subgraphs within neural networks. Rather than manual inspection and experimentation, algorithms can methodically test model components to identify key circuits that implement behaviours of interest. This enables discovering and analysing learned algorithms without extensive human effort.

Automated techniques like ACDC can greatly reduce the time and trial-and-error needed to reverse engineer models. However, they may also overlook nuanced or negative components compared to human scrutiny. Automation complements rather than replaces manual analysis - algorithms generate hypotheses and humans provide context. Combining heuristic search with domain knowledge in the fitness metric holds promise for rapid yet reliable circuit discovery.

This could be implemented in a Python library for interpretability by providing common interfaces to specify a model, behaviour, and search technique. The library would execute automated experiments

like ablation and output interactive visualisations of the discovered circuits. Integrations with model inspection tools and dataset analyzers would streamline moving from circuits back to functional validation and interpretation. The aim is intuitive abstractions enabling seamless collaboration between humans and algorithms.

Appendix C - GPT-4 explains GPT-2

Language models can explain neurons in language models (link) **- Appendix C**

This paper (link) explores using GPT-4 to generate natural language explanations of neuron behaviour in GPT-2. They simulate neuron activations for an explanation prompt, then score how well the simulation matches the real neuron. Iterating this process produces better explanations, showing ML can improve interpretability.

However, there are limitations. Neurons may have complex polysemantic behaviour hard to succinctly describe. The method only explains neuron input-output functions, not downstream effects. It describes correlations, not causal mechanisms producing behaviour. And it is compute-intensive.

Extensions could explain whole circuits, generate richer hypotheses, and iterate like human researchers. Long-term goals are interpreting large models to detect alignment/safety issues before deployment. But significant advances are still needed for techniques to surface behaviours like dishonesty.

They demonstrate ML-generated explanations of neurons, but with key limitations around complexity, causality and scale. Future work is needed to produce more rigorous circuit-level interpretations and uncover subtle issues.

Appendix D - Activation Engineering

Activation Engineering Approaches (link) **- Appendix D**

Activation engineering techniques have emerged to steer large language model behaviour at runtime by strategically modifying neuron activations. There are still a number of open problems in Activation Engineering (link), making it an excellent candidate for a high-level library implementation.

This is often done by injecting "steering vectors" corresponding to desired attributes. For example, a "love-hate" steering vector can be derived by capturing activations from inputs like <start> love <pad> and <start> hate <pad>. The scaled steering vector is then injected at key layers to nudge the model toward outputs consistent with the attribute. This allows granular steering of behaviour with limited computational overhead.

Activation engineering may offer a high-level interpretable overview of model decision-making by determining which activation tweaks can alter outputs. For example, counterfactual explanations could be generated by identifying the modifications needed to flip a decision. One approach is prompting models to prioritise different principles by injecting associated activation patterns.

An emerging branch of activation engineering is representation engineering, pioneered by Dan Hendrycks' group. This approach involves applying activation modifications not to the raw activation space, but to a reduced dimensionality representation obtained through principal component analysis (PCA). PCA is used to identify the most semantically meaningful directions in activation space. Steering is then applied along these engineered principal component axes. Focusing interventions on the principal subspace provides greater control and precision than raw activation editing.

Ongoing work is refining techniques for creating accurate steering vectors, identifying optimal injection points, and determining steering coefficients. There remain open challenges around precision, generalizability and scalability. But the approach shows promise for interpretable nudging of models and analysis of decision-making factors.

Appendix E - Explainable AI Libraries

Captum (link) **and Alibi** (link) **- Appendix E**

Captum

Captum (link) provides a wide selection of attribution techniques to explain model predictions using Pytorch, ranging from simple gradients to Integrated Gradients and DeepLift. These attribute an importance score to each input feature indicating its influence on the output. Captum also offers perturbation and ablation methods to analyse model robustness.

Beyond inputs, Captum can explain intermediate layers and neurons. Techniques like Layer Conductance identify influential hidden units. Captum's neuron attribution methods assign importance scores to individual nodes' contributions. This enables tracing how information flows through networks.

For evaluating explanations, Captum provides faithfulness metrics like sensitivity and infidelity. It also features methods for concept-based interpretations, identifying parts of a model linked to high-level concepts. Influential examples techniques analyse training data influence on models.

Captum supplies useful utilities like interpretable embeddings, Linear Model modules, and visualisation tools. The library builds on PyTorch, leveraging its autograd engine to efficiently compute attributions. Captum provides abstractions enabling researchers to quickly implement and test ideas. The Insights API simplifies analysis workflows.

Captum equips researchers with a powerful toolkit for model understanding, including diverse attribution algorithms, explanation evaluation, and infrastructure to drive further innovations. The library advances interpretable AI through both proven techniques and support for emerging methods.

Abili Explain

Alibi Explain (link) is a library providing a toolkit of local and global explainability techniques for inspecting sklearn-like machine learning models. Global methods analyse overall feature importance. Local methods explain individual predictions through feature attributions or example-based explanations.

Anchor explanations identify minimal feature subsets sufficient for a prediction. They debug models by revealing compact rationales behind individual behaviours. **Integrated gradients** attribute an importance score to each input feature. They trace information flow through models and quantify contributions.

Similarity explanations find influential training instances related to test points. They expose models' training patterns and detect distribution shifts. Gradient-based similarity metrics capture semantic closeness effectively.

Together **anchors**, **integrated gradients**, and **similarity explanations** enable rigorous root cause analysis. Anchors rationalise predictions, integrated gradients show information flow, and similarity reveals training provenance and Alibi Explain provides a powerful framework for localised introspection.

While Scikit-Learn offers an Estimator, Predictor, Transformer and Model interface, Alibi Explain implements an Explainer interface.