



MURANO: Design, Run, and Reproduce Mechanistic Interpretability Experiments as Composable Pipelines

Alireza Bayat Makou^{1,*}, Emirhan Böge^{1,4,*}, Phu Gia Hoang^{1,*}, Federico Tiblias^{1,2,3,*}, Jingcheng Niu¹, Subhabrata Dutta^{1,2}, Richard Eckart de Castilho¹, Iryna Gurevych^{1,2,4}

Correspondence: alireza.makou@tu-darmstadt.de

¹Ubiquitous Knowledge Processing (UKP) Lab, Technical University of Darmstadt, Germany

²National Research Center for Applied Cybersecurity ATHENE, Germany

³Zuse School ELIZA, Technical University of Darmstadt

⁴Cluster of Excellence “Reasonable Artificial Intelligence” (RAI), hessian.AI, Germany

Abstract

This paper presents Murano, an open source framework for designing, running, and reproducing mechanistic interpretability studies of large language models, intended for researchers across disciplines. These studies often combine loading, recording, attribution, intervention, and evaluation, while existing libraries tend to focus on different parts of this workflow. As a result, researchers using several libraries may need to adapt outputs from one for use by another. To bridge this gap, Murano represents operations from these five areas as composable steps. Steps exchange named result artifacts and declare the inputs they require and the outputs they produce. A pipeline executes its steps in the order supplied, and Murano uses canonical addresses when component identities pass between operations. Murano builds on existing interpretability and machine learning libraries. We demonstrate Murano through two reproductions of established interpretability studies and one illustrative sparse autoencoder case study.¹

1 Introduction

Many mechanistic interpretability studies combine three core operations: recording model activations, using attribution methods to localize components associated with a behavior, and intervening on those components to measure how a specified manipulation changes the behavior (Rai et al., 2024; Geiger et al., 2021). The recording stage captures model activations without changing them. Attribution localizes heads, layers, directions, or sparse autoencoder features associated with a behavior. Without an intervention, this evidence is correlational. Interventions such as patching, steering, and ablation manipulate the attributed components or representations and measure the resulting behavioral changes.

Existing libraries provide complementary capabilities: TransformerLens (Nanda and Bloom, 2022) and nnsight (Fiotto-Kaufman et al., 2024) expose model internals, sae-lens (Bloom et al., 2024) provides sparse autoencoders (Huben et al., 2024), and nninterp (Dumas, 2025) standardizes access to transformer modules across supported model families. Murano’s design for composing configured operations is inspired by Keras (Chollet, 2015) and scikit-learn (Pedregosa et al., 2011), adapted here to mutable experiment artifacts and interventions on model internals. Researchers who combine several such libraries may need to map one library’s names for model components to another’s, convert outputs into formats the next library can use, put dependent operations in the correct order, and connect intervention and evaluation code across scripts.

Murano is a framework for orchestrating mechanistic interpretability experiments through a shared step contract, result store, and model addressing scheme. For Hugging Face transformer architectures supported by nninterp and the relevant Murano operation, a pipeline is an ordered sequence of steps that may implement one or more of five conceptual phases: load, record, attribute, intervene, and evaluate (Figure 1). Steps exchange named result artifacts under declared input and output contracts. Operations that pass component identities between analyses represent them as canonical Node addresses. This reduces renaming and conversion specific to individual libraries without implying that every constructor accepts every Node form. Murano wraps implemented methods as reusable steps that researchers can configure and combine for different experiments. Researchers can extend the framework by implementing a new Step with declared inputs and outputs.

Murano is designed for researchers across disciplines conducting mechanistic interpretability studies of large language models. Through its

¹<https://github.com/UKPLab/murano>

* Equal contribution.

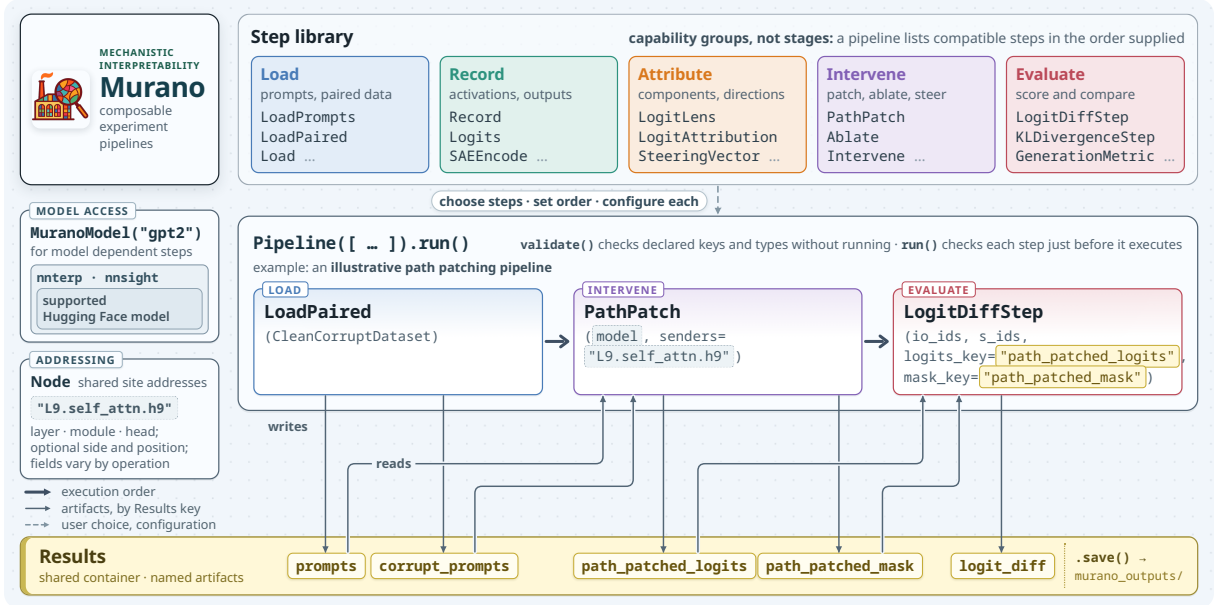


Figure 1: Murano pipeline architecture and an illustrative path patching workflow. A Pipeline executes compatible Step instances in the order supplied, and the steps exchange named artifacts through a shared Results container. Steps that depend on a model use MuranoModel, and supported operations accept canonical Node addresses. The five groups organize available capabilities rather than mandatory stages.

step interface, researchers can combine activation recording, attribution, intervention, and evaluation in exploratory analyses, new experiments, and reproducible case studies. Documentation is available on the project website,² and Murano can be installed from PyPI as murano-interp.³ Our contributions are: (1) an open source orchestration framework for composing mechanistic interpretability operations; (2) declared result keys and optional type expectations, together with a canonical Node representation for exchanging component identities; and (3) two reproductions that recover selected reported findings plus one SAE feature steering case study.

2 Background and Related Work

Mechanistic interpretability aims to explain a neural network’s behavior in terms of its internal components and computations (Olah, 2022; Elhage et al., 2021). Transformers use a residual stream to carry information between layers. Attention heads and MLP blocks read from this shared state and write their outputs back to it, allowing the flow of information between model components to be represented as a graph (Elhage et al., 2021). Causal abstraction provides a formal description of activation and path patching as interventions on a

model’s internal states (Geiger et al., 2021, 2025).

Activation patching takes an activation produced by one input, inserts it at the corresponding location during a run on another input, and measures how the output changes (Vig et al., 2020; Meng et al., 2022). Path patching focuses the intervention on information transmitted from a chosen sender to a chosen receiver (Goldowsky-Dill et al., 2023; Wang et al., 2023). The logit lens projects intermediate residual states into vocabulary space, while direct logit attribution projects component outputs to estimate their contributions to a selected logit (nostalgebraist, 2020; Elhage et al., 2021). Sparse autoencoders represent activations using learned sparse features (Huben et al., 2024), and activation steering changes model activations along a chosen direction (Arditi et al., 2024). Murano makes these methods composable within a single experiment.

A growing set of libraries supports these operations, but most focus on a particular abstraction layer or subset of the workflow. TransformerLens (Nanda and Bloom, 2022) is widely used for mechanistic interpretability but requires support specific to each architecture, which bounds its model coverage. nnsight (Fiotto-Kaufman et al., 2024) and pyvene (Wu et al., 2024) provide intervention interfaces for supported PyTorch models, while nninterp (Dumas,

²<https://ukplab.github.io/murano/>

³<https://pypi.org/project/murano-interp/>

2025) standardizes access to selected transformer modules across model families. Other systems focus on adjacent tasks: EasyEdit and EasyEdit2 support knowledge and behavior editing (Wang et al., 2024; Xu et al., 2025), while Neuronpedia (Lin, 2023) provides a hosted interface for feature exploration and intervention. Combining systems from these different categories can require users to align model addresses, data formats, and intervention code.

Murano currently uses `nnterp` and `nnsight` for model access and integrates `sae-lens` (Bloom et al., 2024) and `scikit-learn` (Pedregosa et al., 2011) for selected operations. It coordinates these implementations through shared `Step` and `Results` contracts and uses canonical `Node` addresses when component identities pass between operations. Section 3 describes these contracts and their support boundaries.

3 System Overview

A Murano Pipeline passes a mutable `Results` container through an ordered list of steps. Figure 1 groups available steps into five conceptual phases, but an individual pipeline may omit or repeat phases. The following paragraphs describe the result contracts, model backend, and model addressing scheme.

Results, steps, and pipelines. The shared `Results` container maps string keys to arbitrary Python values; type expectations come from step declarations rather than the container itself. For example, `prompts` normally stores a `PromptBatch`, and `logit_lens` stores a `LogitLensResult`.

The `Step` implementations included with Murano mutate and return the same `Results` container. A step declares the keys it reads and writes and may declare their expected types. `Pipeline.run()` checks required keys and declared input types immediately before executing each step. It does not check a produced value against the step’s declared output type when that value is written.

Steps exchange run artifacts through declared `Results` keys, while step instances retain their configuration and model objects. If two `Step` instances in the same pipeline declare the same output key, Murano warns that the later value will replace the earlier one. For `Step` instances, `Pipeline.validate()` checks declared key availability and any available type declarations

without executing the steps, but it assumes the pipeline starts with an empty `Results` container. `Pipeline.run()` can continue from an existing container, so an error in a later step may be found only after earlier steps have run.

Model backend. `MuranoModel` loads a Hugging Face model from an identifier or existing local path and wraps `nnterp`’s `StandardizedTransformer`, which is built on `nnsight`. Steps depend on the `ModelBackend` protocol, but its current interface retains `nnsight` proxy semantics and `MuranoModel` is its only implementation.

Addressing model internals. A `Node` stores a layer index and module name, with optional fields for an attention head, projection, and token position. It normalizes the aliases `residual`, `mlp_out`, and `attn_out`; other dotted module paths are stored as given and may remain architecture specific. In the canonical string form, `L5.self_attn.h3.Q@p-1` denotes the Q projection of attention head 3 at token position -1 in layer 5; the projection field can likewise represent K, V, and O projections. Among current operations, only `PathPatch` uses the projection field, and only for Q/K/V receivers; other targeting operations reject it. Each operation validates the address forms and architectures it supports.

Artifacts that identify model components store canonical `Node` objects. `Ablate`, `Patch`, and `PathPatch` accept `Node` addresses and shorthand forms. `Record` takes separate layer, module, and position arguments, while `Intervene` takes separate layer and module arguments. The supported address fields therefore depend on the operation.

Intervention. Murano represents activation interventions with a callback `Callable[[Tensor, Node], Tensor]`. The backend applies the callback at selected module hooks during an intervened forward pass or throughout autoregressive generation. `Ablate` replaces selected activations with zeros, an estimated or supplied mean, or activations resampled from another example or source run. `Patch` configures resampling across runs: it runs one prompt batch while inserting aligned activations from another batch and writes the resulting logits. `Intervene` can project out a supplied direction or add a normalized direction with a chosen scale during generation. Downstream metric steps can then compare the resulting logits or generations.

The step library. The following classes imple-

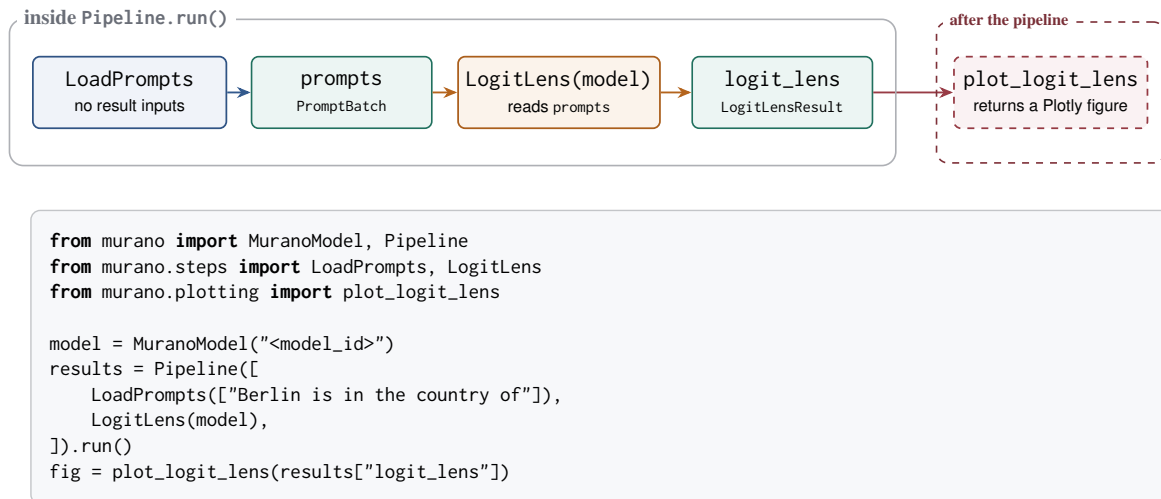


Figure 2: Artifact flow for a logit lens workflow. LoadPrompts stores a PromptBatch under prompts, and LogitLens reads it and stores a LogitLensResult under logit_lens. Both operations run inside the pipeline. The plotting helper is called afterward with the stored result. The lower panel shows the corresponding executable code.

ment the Step contract. Record captures activations from selected layers and modules, optionally retaining all token positions or separating attention heads. RecordAttention captures the full attention weight tensor for every head at each selected layer. LogitLens projects each selected layer’s residual output through the model’s final normalization and unembedding.

SteeringVector estimates a direction from the difference between the mean activations of two classes. Intervene can add the normalized direction at selected layers and modules during generation. AblateAttention replaces selected heads’ attention weights with zeros, the batch mean pattern, or aligned weights from a source run and writes the resulting logits. For SAEs, SAEEncode records feature activations, helper functions rank candidate features, SAEtopActivations and SAEFeatureLabel describe specified features from their activating contexts and promoted tokens, and sae_steel constructs an Intervene step from one decoder direction. Probe evaluates logistic regression with cross validation by default and accepts another compatible scikit-learn estimator.

LogitAttribution estimates contributions from individual attention heads and MLPs after freezing the scale of the final normalization and reports the resulting completeness error. PathPatch freezes attention head outputs at their base values, inserts activations from a source run at specified sender heads or MLPs, captures the resulting activation

at a resid_post or head Q/K/V receiver, and writes the resulting logits. By default, MLPs can mediate the perturbation while the receiver activation is captured; freeze_mlps=True freezes them during that capture to isolate the direct residual path to the receiver.

Steps exchange selected targets as canonical Node objects. SelectComponents stores these objects in a ComponentSelection. Ablate and Patch can read this artifact through targets_key, and PathPatch can read its senders through senders_key. Other operations may still require targets in their constructors.

Evaluation, persistence, and optional dependencies. GenerationMetric applies scorers supplied by users to paired baseline and modified generations. Separate steps compute logit difference, KL divergence, and recovered fractions from stored forward pass results. Save and Results.save() persist artifact types registered in murano.io. Values without a registered serializer are skipped; Murano warns for skipped non-primitive values unless the value has already been recorded as metadata. Probing, dataset loading, plotting, SAE support, and notebook tools are optional extras.

4 Usage

Users construct a Pipeline by listing configured steps in execution order and then call run(). Figure 2 shows a complete logit lens (nostalgebraist, 2020) pipeline for a supported Hugging

Face model. LoadPrompts stores a PromptBatch; LogitLens projects each selected layer’s residual output through the model’s final normalization and unembedding and writes a LogitLensResult; plot_logit_lens visualizes that artifact.

Outside the pipeline API, MuranoModel provides convenience methods: `model.record(...)` performs a recording operation, while `model.generate(..., ablate=direction)` runs baseline and intervened generation; these methods do not define additional workflow phases. Named result keys make dependencies explicit. When a workflow does not depend on existing results, users can call `Pipeline.validate()` to check the declared key and type dependencies before running it.

We compare frameworks across the seven operation categories shown in Table 2 in Appendix A. Recording captures model activations. Attribution estimates contributions from individual heads or MLPs, while activation patching replaces a target activation with one from a source run and measures a downstream change. Attention operations record or edit the weight matrix for each selected head. Steering adds selected directions, including SAE decoder directions, to the residual stream. Linear probes estimate whether labels are decodable from recorded activations. Among the libraries we compare against, Murano is the only one whose documented public interface directly implements all seven listed operation categories: recording, attribution, patching, attention analysis, steering, SAE analysis, and probing.

The same result and addressing contracts also support workflows with multiple steps. Feature steering, for example, loads a pre-trained SAE through `sae-lens`, encodes activations with `SAEEncode`, ranks candidate features by activation on selected tokens with the `top_sae_features_for_tokens` helper, and steers generation with `sae_steer`.

5 Evaluation and Case Studies

We use two published studies and one SAE feature steering case study to evaluate whether Murano can express the selected workflows. We report results from the Murano notebooks and count the logical statements in the corresponding core snippets, excluding imports, comments, printing, and shared setup. These counts describe the shown source only; they do not measure usability or re-

producibility. The case studies do not evaluate user completion time, error rates, runtime, memory use, model coverage, the benefit of the Node representation by itself, or the effort required to add a new method.

5.1 Indirect Object Identification

Wang et al. (2023) identify attention heads that contribute to indirect object identification (IOI) in GPT-2 small (Radford et al., 2019), using sentences such as “When Mary and John went to the store, John gave a drink to __.” To test the reported heads, we replace each head’s output at every prompt’s final real token with the corresponding output from a corrupted prompt. The score is the batch mean of the correct indirect object logit minus the distractor subject logit. We report the relative change from the clean score as a percentage: $100 \times (s_{\text{patched}} - s_{\text{clean}}) / s_{\text{clean}}$.

The three reported name mover heads, L9H9, L9H6, and L10H0, have the three strongest negative effects, while the two reported negative mover heads, L10H7 and L11H10, have the two strongest positive effects. For the three name mover heads, mean attention from the final token to the indirect object is 0.812, 0.742, and 0.460. In a separate output projection test, the tested name tokens appear among the five highest scoring tokens in 96% to 100% of cases. These measurements characterize the heads’ attention patterns and copying tendency; they do not show that a head copied the answer in the original forward pass.

Query patches through the reported S-inhibition heads lower the logit difference for L9H9 and L10H0 by 28.9% and 22.8% of the clean to corrupt gap, but raise it for L9H6 by 4.9%. Value patches produce no measurable change at the displayed precision. The S-inhibition result therefore holds for two of the three tested name mover heads. Mean ablation of the three name mover heads changes the logit difference from 3.484 to 3.520, so this result does not show that these heads are necessary for the answer.

For the attention head sweep, PathPatch captures the clean and corrupt head outputs, freezes all attention head outputs at their clean values, inserts the selected corrupt output, and runs the patched model. LogitDiffStep then computes the score. MLP outputs are not frozen in the reported sweep, so they may carry part of the resulting change. The corresponding core snippets are shown in Appendix Figure 4.

5.2 Truth Direction

The second study examines the truth directions described by Marks and Tegmark (2024). The original study estimates directions that distinguish residual activations for true and false statements in LLaMA-2 models (Touvron et al., 2023). It adds or subtracts these directions and measures the change in the model’s probability difference between TRUE and FALSE. Training on one group of statement types and testing on others measures how well the directions transfer.

We reproduce selected LLaMA-2-13B results with a Murano workflow. MuranoDataset stores separate groups of true and false statements. Record captures their layer 14 residual activations at each statement’s final token, and SteeringVector(normalize=False) computes the true group mean minus the false group mean. The notebook defines a callback that adds or subtracts this vector at the two positions around the final period in layers 8 through 14, and forward_logits runs the modified forward pass and returns the logits.

We reproduce the mass mean intervention trained on cities+neg_cities. The normalized indirect effect measures the fraction of the baseline gap between the TRUE and FALSE probability scores covered by the intervention. Murano obtains 0.88 when adding the direction to false statements and 0.98 when subtracting it from true statements, compared with 0.85 and 0.97 in Marks and Tegmark (2024). The largest difference is 0.03. Table 1 summarizes these values together with selected IOI findings. Separately, Figure 3 compares logistic regression and mass mean probe accuracy on six datasets and two training compositions; the largest difference across the 24 corresponding pairs of values is 10.53 percentage points.

Appendix Figure 6 compares the core code for one additive mass mean pass: 10 logical statements with Murano and 14 with nnsight. SteeringVector(normalize=False) computes the raw difference between the group means, the callback states where to add it, and forward_logits runs the intervention.

5.3 Sparse Autoencoder Feature Steering

We next use Murano to add the vector associated with one Gemma Scope feature (Lieberum et al., 2024) to the internal state of Gemma 2 2B Instruct (Team et al., 2024) during generation. This

Finding	Original	Murano
IOI reproduction		
Name mover heads	L9H9, L10H0, L9H6	Three strongest: L9H9, L9H6, L10H0
Negative mover heads	L10H7, L11H10	Two strongest: L10H7, L11H10
Name tokens in top 5	100%	96–100%
S-inhibition query patch	Effect reported	Lowers logit difference for L9H9 and L10H0; raises it for L9H6
S-inhibition value patch	No effect	No measurable change
Truth direction: normalized indirect effect		
Add to false statements	0.85	0.88
Subtract from true statements	0.97	0.98

Table 1: Selected IOI results from Wang et al. (2023) and truth direction results from Marks and Tegmark (2024), alongside the Murano results. Truth direction setup: LLaMA-2-13B, cities+neg_cities training, and sp_en_trans evaluation.

case study is inspired by Templeton et al. (2024), but it is not a reproduction because it uses a different model, SAE, concept, and intervention. Murano normalizes the selected feature vector and adds it with the chosen strength at layer 20 while each token is generated, whereas Templeton et al. (2024) clamp the feature activation.

We encode six sentences containing California with a Gemma Scope SAE and rank eight candidate features by their mean activation on the California tokens. Feature 1466 is the first candidate for which the model’s output layer places three variants of California among the three tokens with the highest scores, so we use it for the intervention. This selection does not show that the feature responds only to California.

We evaluate four unrelated prompts at strengths 150, 420, and 2000. At strength 420, two displayed generations contain broken or repeated word fragments. At strength 2000, all four collapse into repetitions of California or Sacramento. These examples show that a large fixed addition can overwhelm generation; they do not show coherent control of topic.

Under our counting protocol, the shown core SAE workflow contains 13 logical statements with Murano and 24 in the direct sae-lens snippet. SAEEncode captures and encodes residual states, top_sae_features_for_tokens ranks candidates, SAEFeatureLabel lists their promoted output tokens, and sae_steel constructs

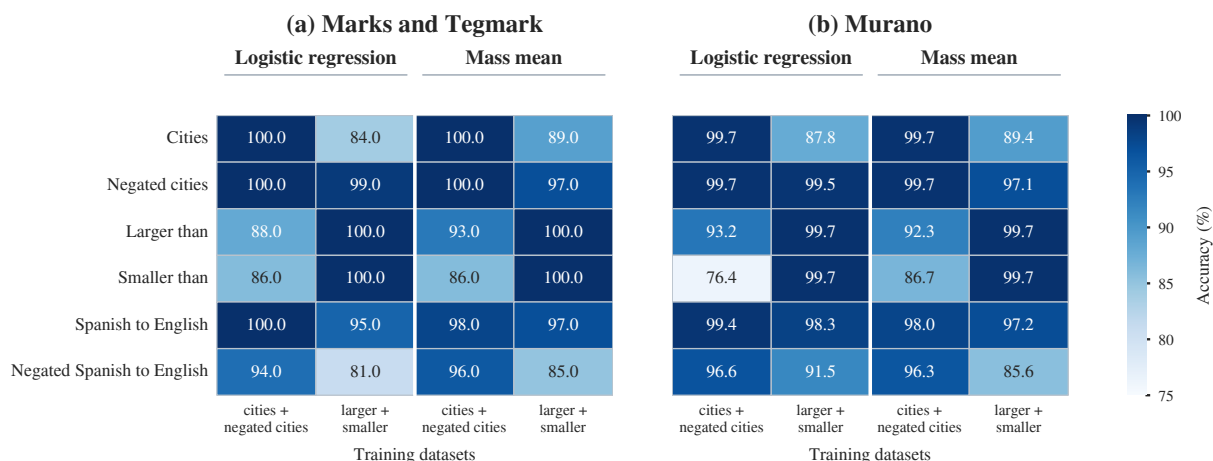


Figure 3: Probe accuracy in the truth direction reproduction. The left panel reports values from Marks and Tegmark (2024), and the right panel reports Murano results. Logistic regression and mass mean probes are trained on the two data compositions shown and evaluated on six datasets. Both panels use the same scale from 75 to 100 percent. Cell values are rounded to one decimal place; the largest absolute difference computed from the unrounded results is 10.53 percentage points.

the additive intervention. This comparison does not cover feature clamping. The core snippets are shown in Appendix Figure 5.

6 Conclusion

Murano coordinates selected mechanistic interpretability operations as ordered steps that exchange named results and use consistent addresses for model components. The IOI head sweep recovers the three reported name mover heads and two reported negative mover heads, although the S-inhibition query result holds for only two of the three tested name mover heads. In the truth direction case study, the two reproduced mass mean intervention values differ from the original values by at most 0.03, while the probe accuracy grid differs by at most 10.53 percentage points. The SAE case study shows that a large fixed feature addition can overwhelm generation. Murano encapsulates recurring implementation patterns as reusable operations. The case studies show how selected workflows can be expressed, and the truth direction notebook compares selected measurements with the original study. They do not directly evaluate readability, reproducibility, or extension effort.

Limitations

Murano inherits the scientific assumptions of the methods it wraps and has additional system limitations: pipelines execute sequentially; the current backend depends on nnterp and nnsight; automatic GPU placement defaults to one GPU when

CUDA is available; support for models and Node forms varies by operation and architecture; token positions depend on tokenization and padding; and analyses that retain many activations can require substantial accelerator memory. The current framework targets forward passes and bounded text generation with one model. Adapting it to language agents with tool use, persistent memory, and state across multiple turns remains future work. The case studies use fixed prompt sets, data splits, corruption schemes, and intervention settings. We do not systematically test sensitivity to alternative samples or hyperparameters. The framework comparison records documented interface coverage rather than implementation quality, usability, or performance.

Ethics Statement

Murano is intended for research on understanding and controlling language model behavior. The same intervention capabilities could be misused to elicit harmful outputs or weaken safeguards. Users should follow applicable institutional policies, model licenses, and responsible disclosure practices.

Acknowledgement

This work was funded by the LOEWE Distinguished Chair “Ubiquitous Knowledge Processing”, LOEWE initiative, Hesse, Germany (Grant Number: LOEWE/4a/519/05/00.002(0002)/81), as well as by the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE, and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy, EXC-3057. Federico Tiblias is supported by the Konrad Zuse School of Excellence in Learning and Intelligent Systems (ELIZA) through the DAAD programme Konrad Zuse Schools of Excellence in Artificial Intelligence, sponsored by the Federal Ministry of Education and Research.

References

- Andy Arditi, Oscar Balcells Obeso, Aaquib Syed, Daniel Paleka, Nina Rimskey, Wes Gurnee, and Neel Nanda. 2024. [Refusal in language models is mediated by a single direction](#). In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- Joseph Bloom, Curt Tigges, Anthony Duong, and David Chanin. 2024. SAELens. <https://github.com/decoderresearch/SAELens>.
- François Chollet. 2015. Keras. <https://keras.io>.
- Clément Dumas. 2025. [nnterp: A standardized interface for mechanistic interpretability of transformers](#). *CoRR*, abs/2511.14465.
- Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, and 6 others. 2021. [A mathematical framework for transformer circuits](#). *Transformer Circuits Thread*.
- Jaden Fiotto-Kaufman, Alexander R Loftus, Eric Todd, Jannik Brinkmann, Caden Juang, Koyena Pal, Can Rager, Aaron Mueller, Samuel Marks, Arnab Sen Sharma, Francesca Lucchetti, Michael Ripa, Adam Belfki, Nikhil Prakash, Sumeet Multani, Carla Brodley, Arjun Guha, Jonathan Bell, Byron Wallace, and David Bau. 2024. [NNsight and NDIF: Democratizing access to foundation model internals](#). *Preprint*, arXiv:2407.14561.
- Atticus Geiger, Duligur Ibeling, Amir Zur, Maheep Chaudhary, Sonakshi Chauhan, Jing Huang, Aryaman Arora, Zhengxuan Wu, Noah Goodman, Christopher Potts, and Thomas Icard. 2025. [Causal abstraction: A theoretical foundation for mechanistic interpretability](#). *Journal of Machine Learning Research*, 26(83):1–64.
- Atticus Geiger, Hanson Lu, Thomas F Icard, and Christopher Potts. 2021. [Causal abstractions of neural networks](#). In *Advances in Neural Information Processing Systems*.
- Nicholas Goldowsky-Dill, Chris MacLeod, Lucas Sato, and Aryaman Arora. 2023. [Localizing model behavior with path patching](#). *CoRR*, abs/2304.05969.
- Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. 2024. [Sparse autoencoders find highly interpretable features in language models](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Tom Lieberum, Senthooan Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Varma, Janos Kramar, Anca Dragan, Rohin Shah, and Neel Nanda. 2024. [Gemma Scope: Open sparse autoencoders everywhere all at once on Gemma 2](#). In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 278–300, Miami, Florida, US. Association for Computational Linguistics.
- Johnny Lin. 2023. [Neuronpedia: Interactive reference and tooling for analyzing neural networks](#). Software available from neuronpedia.org.
- Samuel Marks and Max Tegmark. 2024. [The geometry of truth: Emergent linear structure in large language model representations of true/false datasets](#). In *First Conference on Language Modeling*.
- Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. [Locating and editing factual associations in GPT](#). In *Advances in Neural Information Processing Systems*, volume 35, pages 17359–17372. Curran Associates, Inc.
- Neel Nanda and Joseph Bloom. 2022. [TransformerLens](https://github.com/TransformerLensOrg/TransformerLens). <https://github.com/TransformerLensOrg/TransformerLens>.
- nostalgebraist. 2020. [Interpreting GPT: the Logit Lens](#).
- Chris Olah. 2022. [Mechanistic interpretability, variables, and the importance of interpretable bases](#). *Transformer Circuits Thread*.
- Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. [Scikit-learn: Machine learning in Python](#). *Journal of Machine Learning Research*, 12(85):2825–2830.

- Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. [Language models are unsupervised multitask learners](#).
- Daking Rai, Yilun Zhou, Shi Feng, Abulhair Saparov, and Ziyu Yao. 2024. [A practical review of mechanistic interpretability for transformer-based language models](#). *CoRR*, abs/2407.02646.
- Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, and 179 others. 2024. [Gemma 2: Improving open language models at a practical size](#). *Preprint*, arXiv:2408.00118.
- Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermy, and 3 others. 2024. [Scaling monosemanticity: Extracting interpretable features from Claude 3 Sonnet](#). *Transformer Circuits Thread*.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. [Llama 2: Open foundation and fine-tuned chat models](#). *Preprint*, arXiv:2307.09288.
- Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. 2020. [Investigating gender bias in language models using causal mediation analysis](#). In *Advances in Neural Information Processing Systems*, volume 33, pages 12388–12401. Curran Associates, Inc.
- Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2023. [Interpretability in the wild: a circuit for indirect object identification in GPT-2 small](#). In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.
- Peng Wang, Ningyu Zhang, Bozhong Tian, Zekun Xi, Yunzhi Yao, Ziwen Xu, Mengru Wang, Shengyu Mao, Xiaohan Wang, Siyuan Cheng, Kangwei Liu, Yuansheng Ni, Guozhou Zheng, and Huajun Chen. 2024. [EasyEdit: An easy-to-use knowledge editing framework for large language models](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 82–93, Bangkok, Thailand. Association for Computational Linguistics.
- Zhengxuan Wu, Atticus Geiger, Aryaman Arora, Jing Huang, Zheng Wang, Noah D. Goodman, Christopher D. Manning, and Christopher Potts. 2024. [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: System Demonstrations, NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 158–165. Association for Computational Linguistics.
- Ziwen Xu, Shuxun Wang, Kewei Xu, Haoming Xu, Mengru Wang, Xinle Deng, Yunzhi Yao, Guozhou Zheng, Huajun Chen, and Ningyu Zhang. 2025. [EasyEdit2: An easy-to-use steering framework for editing large language models](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025 - System Demonstrations, Suzhou, China, November 4-9, 2025*, pages 522–535. Association for Computational Linguistics.

A Framework Capability Comparison

Framework	Record	Attribution	Patching	Attention	Steering	SAE	Probing
<i>User libraries</i>							
Murano	✓	✓	✓	✓	✓	✓	✓
TransformerLens (Nanda and Bloom, 2022)	✓	✓	✓	✓	○	○	○
nnsight (Fiotto-Kaufman et al., 2024)	✓	○	✓	○	○	○	○
pyvene (Wu et al., 2024)	✓	✗	✓	○	✓	✓	○
sae-lens (Bloom et al., 2024)	✓	○	○	○	○	✓	✗
EasyEdit2 (Xu et al., 2025)	○	✗	✗	✗	✓	✓	✗
<i>Model access backend</i>							
nnterp (Dumas, 2025)	✓	✓	✓	○	✓	○	○
<i>Hosted platform</i>							
Neuronpedia (Lin, 2023)	✓	○	✗	✗	✓	✓	✓

Table 2: Capability comparison based on the cited publications and public documentation, checked in August 2026. For libraries and the backend, ✓ denotes a directly documented public interface, ○ denotes an operation that requires user supplied hooks or code or a documented companion package, and ✗ denotes that no support was identified in the cited sources. For SAE use in the TransformerLens and nnsight rows, the companion package is sae-lens. For the hosted Neuronpedia row, ✓ denotes a documented platform feature. The groups distinguish user libraries from a model access backend and a hosted platform.

B Evaluation Implementation Details

Murano	nnsight
<pre> 1 def logit_diff(res, logits_key, mask_key): 2 step = LogitDiffStep(correct=io_ids, incorrect=s_ids, 3 logits_key=logits_key, mask_key=mask_key) 4 return step(res)[keys.LOGIT_DIFF].value 5 6 model = MuranoModel("gpt2", dtype=torch.float32, 7 enable_attention_probs=True) 8 ds = CleanCorruptDataset(clean=clean, corrupt=corrupt, 9 correct=io_ids, incorrect=s_ids) 10 base = LoadPaired(ds)(Results()) 11 final_positions = (12 base[keys.ATTENTION_MASK].sum(dim=1) - 1) 13 clean_ld = logit_diff(logits(model)(base), 14 keys.FINAL_LOGITS, keys.ATTENTION_MASK) 15 effect = torch.zeros(model.n_layers, model.n_heads) 16 17 for layer in range(model.n_layers): 18 for head in range(model.n_heads): 19 path_patch = PathPatch(20 model, Node(layer, SELF_ATT_N, head=head), 21 positions=final_positions) 22 out = path_patch(base) 23 ld = logit_diff(out, keys.PATH_PATCHED_LOGITS, 24 keys.PATH_PATCHED_MASK) 25 effect[layer, head] = 100 * (ld - clean_ld) / clean_ld 26 27 top = effect.flatten().argsort()[:3] 28 movers = [(int(i) // model.n_heads, 29 int(i) % model.n_heads) for i in top]</pre>	<pre> 1 model = LanguageModel("openai-community/gpt2", 2 device_map="auto", dispatch=True) 3 L = model.config.n_layer 4 H = model.config.n_head 5 D = model.config.n_embd // H 6 B = len(clean) 7 tokens = model.tokenizer(8 clean, padding=True, return_tensors="pt") 9 final_positions = (10 tokens["attention_mask"].sum(dim=1) - 1) 11 io = torch.tensor(io_ids) 12 s = torch.tensor(s_ids) 13 rows = torch.arange(B) 14 15 def logit_diff(logits: torch.Tensor) -> torch.Tensor: 16 last = logits.detach().float().cpu()[17 rows, final_positions] 18 return (last[rows, io] - last[rows, s]).mean() 19 20 def cache(prompts): 21 with model.trace(prompts): 22 heads = torch.stack([23 model.transformer.h[i].attn.c_proj.input 24 for i in range(L)]) 25 logits = model.lm_head.output.save() 26 return heads, logits 27 28 clean_heads, clean_logits = cache(clean) 29 corrupt_heads, _ = cache(corrupt) 30 clean_ld = logit_diff(clean_logits) 31 effect = torch.zeros(L, H) 32 33 for layer in range(L): 34 for head in range(H): 35 sender = clean_heads[layer].clone().reshape(36 B, -1, H, D) 37 corrupt_sender = corrupt_heads[layer].reshape(38 B, -1, H, D) 39 sender[rows, final_positions, head] = (40 corrupt_sender[rows, final_positions, head]) 41 sender = sender.reshape(B, -1, H * D) 42 43 with model.trace(clean): 44 for i in range(L): 45 model.transformer.h[i].attn.c_proj.input = (46 sender if i == layer else clean_heads[i]) 47 patched = model.lm_head.output.save() 48 49 effect[layer, head] = (50 100 * (logit_diff(patched) - clean_ld) / clean_ld) 51 52 top = effect.flatten().argsort()[:3] 53 movers = [(int(i) // H, int(i) % H) for i in top]</pre>

Figure 4: Core logic for the attention head sweep with Murano (17 logical statements, left) and a direct nnsight implementation (35 logical statements, right). Shared imports, prompts, labels, configuration, comments, and printing are omitted. Both snippets replace each selected head’s output at the final real token, keep all other attention head outputs at their clean values, and use the same logit difference metric.

Murano	Direct sae-lens
<pre> 1 model = MuranoModel(MODEL_ID) 2 encode = SAEEncode(model, release=SAE_RELEASE, sae_id=SAE_ID) 3 results = Pipeline([LoadPrompts(PROBES), encode]).run() 4 record = results["sae_record"] 5 cands = list(top_sae_features_for_tokens(6 record, model, {CONCEPT}, n=8)) 7 results = Pipeline([SAEFeatureLabel(8 model, feat_ids=cands, k_tokens=3)]).run(results) 9 labels = results["feature_labels"] 10 def matches(fid): 11 return any((CONCEPT.lower() in token.lower() 12 for token in labels.tokens[fid])) 13 feat = next((fid for fid in cands if matches(fid)), cands[0]) 14 steer = sae_steer(model, encode.sae_model, feat, alpha=ALPHA) 15 results = Pipeline([LoadPrompts(PROMPTS), steer]).run() 16 gens = results["intervene"].modified_generations </pre>	<pre> 1 tok = AutoTokenizer.from_pretrained(MODEL_ID) 2 model = AutoModelForCausalLM.from_pretrained(3 MODEL_ID, torch_dtype=torch.float16, device_map="auto") 4 sae = SAE.from_pretrained(5 release=SAE_RELEASE, sae_id=SAE_ID, device="cuda")[0] 6 enc = tok(PROBES, return_tensors="pt", padding=True).to(model.device) 7 with torch.no_grad(): 8 resid = model(**enc, output_hidden_states=True 9).hidden_states[LAYER + 1] 10 acts = sae.encode(resid.to(sae.W_enc.dtype)) 11 mask = torch.tensor([12 [CONCEPT.lower() in tok.decode([token]).lower() for token in row] 13 for row in enc["input_ids"]], device=acts.device) 14 scores = (acts * mask.unsqueeze(-1)).sum((0, 1)) 15 scores /= mask.sum().clamp(min=1) 16 cands = scores.topk(8).indices.tolist() 17 unembed = model.get_output_embeddings().weight 18 norm = model.model.norm 19 feat = cands[0] 20 for fid in cands: 21 promoted = (unembed @ norm(22 sae.W_dec[fid].to(model.dtype))).topk(3).indices.tolist() 23 if any((CONCEPT.lower() in tok.decode([token]).lower() 24 for token in promoted): 25 feat = fid 26 break 27 direction = sae.W_dec[feat] / sae.W_dec[feat].norm() 28 def steer_hook(module, inputs, output): 29 return (output[0] + ALPHA * direction,) + tuple(output[1:]) 30 handle = model.model.layers[LAYER].register_forward_hook(steer_hook) 31 gens = [tok.decode(model.generate(32 **tok(prompt, return_tensors="pt").to(model.device), 33 **GEN_KWARGS)[0]) for prompt in PROMPTS] </pre>

Figure 5: Core logic for ranking candidate SAE features, inspecting promoted output tokens, and constructing an additive generation intervention with Murano (13 logical statements, left) and direct sae-lens code (24 logical statements, right). Murano’s SAEEncode and SAEFeatureLabel steps, top_sae_features_for_tokens helper, and sae_steer step factory encapsulate residual capture, candidate ranking, inspection of promoted output tokens, and construction of the hook used during generation. Shared imports, data, configuration, comments, and printing are omitted.

Murano	nnsight
<pre> 1 direction = Pipeline([2 Load(MuranoDataset(positive_texts=t_c + t_n, 3 negative_texts=f_c + f_n)), 4 Record(model, layers=[PROBE_LAYER], position="last", 5 SteeringVector(normalize=False), 6)].run()["steering"].direction_per_layer[(PROBE_LAYER, "residual")] 7 true_id = model.tokenizer.encode(" TRUE")[1] 8 false_id = model.tokenizer.encode(" FALSE")[1] 9 length = len(model.tokenizer.encode("This statement is:")) 10 def intervene(act, node): 11 act[:, -length - 1:-length + 1] += direction 12 return act 13 logits = model.forward_logits(toks, fn=intervene, 14 layers=INTERVENE_LAYERS) 15 probs = logits[:, -1].softmax(-1) 16 p_diff = (probs[:, true_id] - probs[:, false_id]).mean() </pre>	<pre> 1 pos, neg = acts[labels == 1], acts[labels == 0] 2 direction = pos.mean(0) - neg.mean(0) 3 direction = direction / direction.norm() 4 direction = ((pos.mean(0) - neg.mean(0)) @ direction) * direction 5 sites = [(layer, offset) 6 for layer in range(8, 15) 7 for offset in (-1, 0)] 8 true_id = tok.encode(" TRUE")[1] 9 false_id = tok.encode(" FALSE")[1] 10 length = len(tok.encode("This statement is:")) 11 with model.forward(remote=remote) as runner: 12 with runner.invoke(batch): 13 for layer, offset in sites: 14 model.model.layers[layer].output[0][15 :, -length + offset, :] += direction 16 probs = model.lm_head.output[:, -1].softmax(-1) 17 p_diff = (18 probs[:, true_id] - probs[:, false_id] 19).save() </pre>

Figure 6: Core logic for one additive mass mean intervention with Murano (10 logical statements, left) and nnsight (14 logical statements, right). Both snippets use a direction trained on the cities+neg_cities groups. SteeringVector(normalize=False) computes the raw difference between the group means, the callback states where to add it, and forward_logits runs the intervention. Shared imports, data aliases, configuration, comments, and printing are omitted.