Spectra: A Mechanistic Interpretability Library for Vision-Language Models

Clement Neo[1,2], Yongsen Zheng[1,2]*, Kwok-Yan Lam[1,2], Luke Ong[1],

[1]Nanyang Technological University, Singapore [2]Digital Trust Centre, Singapore
{clement.neohx, yongsen.zheng, kwokyan.lam, luke.ong}@ntu.edu.sg

Abstract

Vision-Language Models (VLMs) have become increasingly important in AI applications, yet interpretability tools for these models lag behind those available for text-only language models. While libraries like TransformerLens have enabled significant progress in understanding language models, existing tools for VLMs are limited to basic activation probing and saving. We introduce Spectra, a library specifically designed for mechanistic interpretability of VLMs that provides unified abstractions for activation patching, attention pattern analysis, and meta-functions across diverse VLM architectures. Built on HuggingFace's Transformers, our library handles architecture-specific complexities through per-checkpoint configurations while maintaining a simple, high-level interface. We demonstrate the library's capabilities by performing interpretability experiments on a counting task, showing how researchers can easily perform experiments that were previously cumbersome to do. The library currently supports Qwen2.5-VL, Qwen3-VL, LLaVA 1.5 and SmolVLM, with a design that facilitates extension to additional architectures. The library can be found at `github.com/clemneo/vlm-spectra`.

1 Introduction

Mechanistic interpretability aims to understand the internal mechanisms by which neural networks process information and produce outputs. This research often requires non-trivial engineering work to access model internals, perform interventions, and analyze representations. To facilitate this work, several toolkits have emerged: TransformerLens (Nanda and Bloom, 2022) for language models, NNsight (Fiotto-Kaufman et al., 2024) as a general framework for neural networks, and Prisma (Joseph et al., 2025) for vision transformers. However, comprehensive interpretability tools for vision-language models (VLMs) remain limited. In this paper, we take VLMs to refer to models that take in image and text as input and output text.

Creating a comprehensive library for VLMs can be difficult. Modern state-of-the-art VLMs predominantly follow an adapter-style architecture where a vision encoder processes images, an adapter maps visual features to the language model's input space, and a language model processes combined visual and textual inputs (Liu et al., 2023; Merullo et al., 2023). While this high-level structure appears consistent, VLMs exhibit substantial variation in implementation details. Vision tokens can be encoded differently across models. For example, Qwen2.5-VL (Bai et al., 2025b) uses explicit `START_IMAGE` and `END_IMAGE` tokens with contiguous patches, while SmolVLM (Marafioti et al., 2025) uses just one token for image boundaries, as well as adding row demarcation tokens which causes the image tokens to be non-contiguous. Adapters also range from simple linear projections (Liu et al., 2023; Merullo et al., 2023) to MLP networks that merge spatial tokens (Bai et al., 2025b,a). These variations make it challenging to build tools that work uniformly across architectures.

Existing libraries that focus on VLMs sidestep this complexity by focusing only on operations that work uniformly. For example, `vlm-lens` (Sheta et al., 2025) only supports saving residual stream activations, which requires no understanding of model-specific token encoding schemes. However, mechanistic interpretability fundamentally requires interventions: patching activations to test causal hypotheses, blocking attention to understand information flow, and manipulating representations to identify necessary components. These operations cannot ignore architectural details.

We introduce Spectra, a library designed for mechanistic interpretability of adapter-style VLMs. Our library provides high-level abstractions for acti-

*Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations), pages 794–803

vation patching, activation access, attention pattern analysis, and architecture-aware meta-functions while handling model-specific complexity. We demonstrate its utility by doing some mechanistic interpretability experiments on a counting task, as well as replicating findings from recent VLM interpretability papers. The library currently supports Qwen2.5-VL, Qwen3-VL, LLaVA 1.5, and SmolVLM. The library is released under the MIT license.

2 Background

Mechanistic interpretability (MI) aims to reverse-engineer neural networks by identifying the internal representations and mechanisms that drive model behavior. In language models, this approach has uncovered fundamental findings such as induction heads for in-context learning (Olsson et al., 2022), circuits for indirect object identification (Wang et al.), factual knowledge localized in MLP layers (Meng et al., 2022), superposition of features in activation space (Elhage et al., 2022), and sparse autoencoders that decompose activations into interpretable features (Cunningham et al., 2023; Bricken et al., 2023).

MI for Vision-Language Models. More recently, researchers have begun applying these methods to vision-language models. Logit lens analyses have revealed that visual token representations progressively align with interpretable textual concepts across layers and can spatially localize objects within VLM representations (Neo et al., 2025; Jiang et al., 2025). Activation patching and causal tracing have identified causally critical layers for image-conditioned generation (Palit et al., 2023) and universal cross-attention heads that implement object detection and suppression across tasks (Golovanevsky et al., 2025). Attention blocking experiments have shown that VLMs rely on spatially corresponding visual tokens for object identification (Neo et al., 2025) and that early layers handle cross-modal integration while later layers produce answers (Zhang et al., 2025).

Notably, many of these findings depend not just on observing internal states, but on *intervening* on them through patching, ablating, or blocking activations. This is to establish causal rather than merely correlational claims.

Related Works & Tooling Gaps. Current mechanistic interpretability libraries fall short of this need for VLMs. TransformerLens (Nanda and Bloom, 2022) supports only text-only language models, while NNsight (Fiotto-Kaufman et al., 2024) is in principle architecture-agnostic, but users report significant difficulty applying its abstractions to multimodal architectures. The remaining VLM-focused tools, like vlm-lens (Sheta et al., 2025) or VL-Interpret (Aflalo et al., 2022), support activation saving or attention visualization but provide no way to intervene on model internals.

This gap reflects an engineering challenge: Adapter-style VLMs share a common high-level structure of vision encoder, adapter, and language model, but can differ substantially in implementation. Vision tokens may be demarcated by start and end tokens or just a single demarcation token; they may be contiguous or interleaved with row-marker tokens. Adapters range from simple linear projections to spatial-merge networks that merge the token sequence before it enters the language model. Operations that are trivial in text-only models, such as indexing a specific image token position to patch an activation, may require model-specific logic in VLMs.

Hence, a library that addresses these challenges should provide two things: first, a simple interface for activation caching and intervention; and second, architecture-aware utilities for working with visual tokens, such as for resolving token boundaries, mapping between image coordinates and token indices, and handling adapter transformations. These would enable researchers to write experiments against a stable API without reasoning about per-model details. We introduce Spectra, a library that provides this for adapter-style VLMs.

3 Spectra

Spectra is a Python toolkit for mechanistic interpretability of adapter-style VLMs, built on HuggingFace Transformers. Its core abstraction is `HookedVLM`, a thin wrapper around HuggingFace model classes that preserves the standard interface (`forward`, `generate`) while exposing two context managers for interpretability work: `run_with_cache` for recording activations and `run_with_hooks` for intervening on them during a forward pass. The library currently supports Qwen2.5-VL, Qwen3-VL, LLaVA 1.5, and SmolVLM.

e.g., See this GitHub issue.

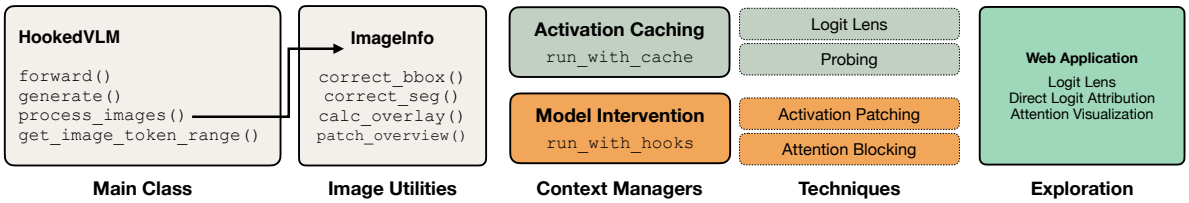

Figure 1: Overview of Spectra.

3.1 Activation Caching and Intervention

The two core primitives for mechanistic interpretability are observing internal states and intervening on them. Spectra exposes both through lightweight context managers that attach and remove hooks for the underlying HuggingFace model.

Caching activations. `run_with_cache` records selected activations (e.g. residual streams, attention patterns, MLP outputs) keyed by a hook name that encodes the layer and component (e.g. `lm.blocks.12.hook_resid_post`). Users pass a list of hook names and receive a cache dictionary after the forward pass, enabling analyses such as logit lens, cross-layer similarity, or attribution over specific token ranges. These hook names are largely similar to the ones from TransformerLens (Nanda and Bloom, 2022).

Model intervention. `run_with_hooks` enables causal interventions by executing user-supplied functions at specified hook points during a forward pass. This supports standard patterns including activation patching from a source into a target run, selective head ablation, and attention blocking on specified token ranges.

Figure 4 illustrates a concrete example of how patching residual-stream activations can be much simpler with Spectra, as it abstracts away image and image metadata handling, as well as registering and removing hooks.

Critically, these operations are architecture-agnostic. Spectra resolves model-specific module paths and token layouts via the adapter registry (§3.4), so the same hook name targets the corresponding component across different VLM families.

3.2 Architecture-Aware Utilities

VLM interpretability experiments are highly susceptible to indexing errors. Image preprocessing pipelines of different models may resize images differently, adapters may merge or reshape visual tokens before they enter the language model (e.g. Qwen's spatial-merge adapter reduces every four vision-encoder tokens to one), and different models demarcate image regions with different boundary token schemes. An off-by-one error in token indexing can silently invalidate an entire experiment. Based on our conversations with researchers in this field, we provide three categories of utilities that we believe to be broadly useful:

First, `get_image_token_range` returns canonical start and end indices for image tokens in the input sequence, correctly handling boundary markers and non-contiguous layouts (as in SmolVLM's row-demarcation tokens).

Second, models often crop the input image to a certain size in the forward pass. The `process_image` method in the processor does the same crop, and returns an `ImageInfo` object, which contains the image and relevant metadata such as grid height, grid width, and patch size. This helps users ensure that the image they are working with is the image fed into the model. It also includes methods for re-calibrating any bounding boxes or segmentation masks given the crop.

Third, `patch_overview` is a visualization helper that overlay the token grid on the input image, making patch boundaries explicit and supporting attention or attribution heatmaps aligned to the spatial layout (Example in Figure 2).

Together, these utilities let researchers perform and verify spatially precise experiments without reasoning about per-model preprocessing details.

Practical Constraints and Assumptions. The method `ImageInfo.correct_bbox()` assumes the affine crop-and-resize preprocessing used by current adapter-style VLMs (Qwen2.5/3-VL, LLaVA 1.5, SmolVLM), where the transformation from input to model-space coordinates is fully described by a scale factor and an offset recovered from the processor. Architectures that use dynamic high-resolution tiling or multi-crop preprocessing fall

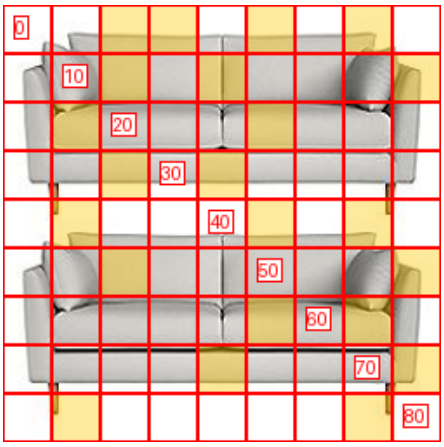

Figure 2: An example output from `patch_overview`. The tokens with prime-numbered indices are highlighted in this example.

outside this assumption and would likely require per-tile bookkeeping; we leave this to future work.

3.3 Logit Lens and Visualizations.

Logit Lens. Spectra includes a logit lens pipeline that projects intermediate residual-stream states into vocabulary space layer-by-layer using the model's own layer norm and unembedding matrix. Most importantly, the pipeline integrates image layout metadata so that logit lens predictions over visual tokens can be mapped back to spatial positions in the input image. We think this is valuable as logit-lens analyses are starting to be used as general diagnostic tools in model development beyond interpretability (Cheng et al., 2026).

Web App. We provide an interactive interface through a web application for common analysis workflows, such as logit lens results, attention heatmaps overlaid on the input image, and direct logit attribution. We note that this is intended as a rapid exploration tool, and all underlying functions are also available programmatically. We provide an example of exploration through the web-app in Appendix B.

3.4 Adapter Registry.

Spectra handles architectural variation through a registry that maps HuggingFace model classes to model-specific adapters. Each adapter implements a fixed interface: accessors for the language model stack (layer count, head count, hidden dimension, and methods to retrieve specific layers, attention modules, MLP blocks, and output projections), as well as model-specific logic for attention pattern extraction and per-head contribution computation. This isolates all architecture-dependent code behind a uniform API that the rest of Spectra calls into.

The registry supports per-checkpoint overrides for models that share a HuggingFace class but differ in configuration. For example, `UI-TARS-1.5-7B` uses the same model class as `Qwen2.5-VL-7B` but requires different system prompts and additional special tokens; registering a checkpoint-level override handles this without duplicating the adapter. Adding support for a new VLM family requires implementing a single adapter class and registering it, with no changes to user-facing code.

On non-adapter style architectures. We focus on adapter-style architectures because they dominate the current frontier of open VLMs, including all models in our initial release (Qwen2.5-VL, Qwen3-VL, LLaVA 1.5, SmolVLM). Cross-attention architectures (e.g. Flamingo) remain important but are no longer the dominant pattern in new releases.

The adapter registry is designed to absorb non-adapter architectures without changes to user-facing code: a `CrossAttentionAdapter` implementing the same accessor interface (layer count, attention modules, hook resolution) could map cross-attention layers to a distinct hook namespace (e.g. `lm.blocks.N.cross_attn.hook_pattern`), leaving hook names, context managers, and intervention APIs unchanged from the user's perspective.

Toward vision-encoder interventions. The library currently exposes hooks under the `lm.` namespace, reflecting our focus on the language model component, where most published VLM interpretability work has been situated. Recent results suggest the vision encoder is itself a fruitful target: mid-layer vision encoder hidden states are interpretable via logit lens through the language model's unembedding (Li et al., 2025), and visual features evolve meaningfully within the encoder prior to the adapter.

The same context-manager and registry abstractions extend naturally to a parallel `vision.` namespace, since hook names already carry a component prefix and `ImageInfo` provides the spatial metadata needed to map encoder-side activations back to input-image coordinates. We intend to further integrate these functionalities in future work.

4 Case Study: How do VLMs Count?

Neo et al. (2025) observe that applying a logit lens to LLaVA 1.5 on an image of five green apples reveals number tokens surfacing at seemingly random background positions among the visual tokens. They present this as a single qualitative example and leave open whether the phenomenon is consistent or merely an artifact. We use Spectra to systematically investigate this finding, asking three questions: how reliably do count-relevant predictions appear in visual tokens, what is the nature of the token positions where they appear, and are they causally relevant to the model's output?

Setup. We run all experiments on CountBench (Paiss et al., 2023) using `Qwen3-VL-8B-Instruct` loaded in Spectra's `HookedVLM`. We first take 100 images from CountBench containing 2-9 objects. For each image, we prompt the model with "How many {objects} are there in the image? Answer with just the number.", parse the generated answer, and cache all residual-stream activations over the image-token range via `run_with_cache`. An example of a CountBench image is in Figure 2, where the task would be to count the number of sofas.

Logit lens prevalence. How often do the number tokens show up in the logit lens? We apply `compute_logit_lens` to these visual-token activations to obtain layer-by-layer vocabulary projections. For simplicity, we only consider the top token at each logit lens position.

The model's generated answer appears in the logit-lens over visual tokens in 95 of 100 images. Among these, **the model's generated count is the most frequently surfacing number in 79 cases (83.2%)**, with a median rank of 1 and a mean rank of 1.24. This extends Neo et al. (2025)'s single qualitative observation into a systematic finding: when Qwen3-VL solves a counting task, the correct count has typically already surfaced inside the residual stream of the visual tokens.

Register token overlap. Recent work has identified "register tokens" (visual tokens whose hidden-state norms are outliers) as playing a special role in vision transformers (Darcet et al., 2023). We test whether these tokens coincide with token positions in which number tokens appear in any layer at that position. We define register tokens as those whose L2 norm at the post-adapter language model input exceeds two standard deviations above the per-image mean.

Register tokens account for a mean of 11.13 positions per image, while count-number positions average 30.81. On average, **only 2.3% of register tokens overlap with count-number positions**, suggesting that the count signal is not from these register tokens.

Zero ablation. We test whether the number signal comes from the language model input by ablating the positions where count numbers appear and measuring whether count tokens show up in other token positions in the logit lens. For each image, we replace hidden states at count-number positions with zero vectors at the language model input using `run_with_hooks` and recompute the logit lens over remaining image tokens.

We find that the average number of count-number positions **drops from 30.8 to 11.8**, suggesting that these numbers may be "scratchpad tokens" for the model to process the count.

Attention blocking. We test whether count-bearing positions causally influence the model's prediction by zeroing out attention weights from the final sequence position to all count-token positions, across all layers and heads. **The generated answer changes in only 4 of 100 images**. This negative result highlights how difficult it can be to isolate mechanisms in VLMs, possibly due to the redundancy in information across tokens.

Activation Patching. We select an image pair from CountBench sharing the same object class but differing in count (in this case, 2 vs. 7 children), designating the smaller count as source. We cache all 36 layers of residual-stream activations for both images and perform three patching conditions at each layer ℓ independently: (1) *image-only*, replacing all image-token activations at layer ℓ in the target run with source activations; (2) *text-only*, replacing all text-token activations; and (3) *last-k visual tokens*, replacing only the final 16 or 32 image tokens. We report the logit difference $\Delta\ell = \text{logit}(\text{source count}) - \text{logit}(\text{target count})$ at the final sequence position, normalized by dividing by the clean logit difference (target run with no patching) so that 0 indicates no effect and 1 indicates full recovery of the source answer.

We report the results in Figure 3. Image-only patching achieves near-full recovery in early layers but declines from layer 15 onward, while text-only patching shows the inverse pattern, consistent with the cross-modal information transfer observed by

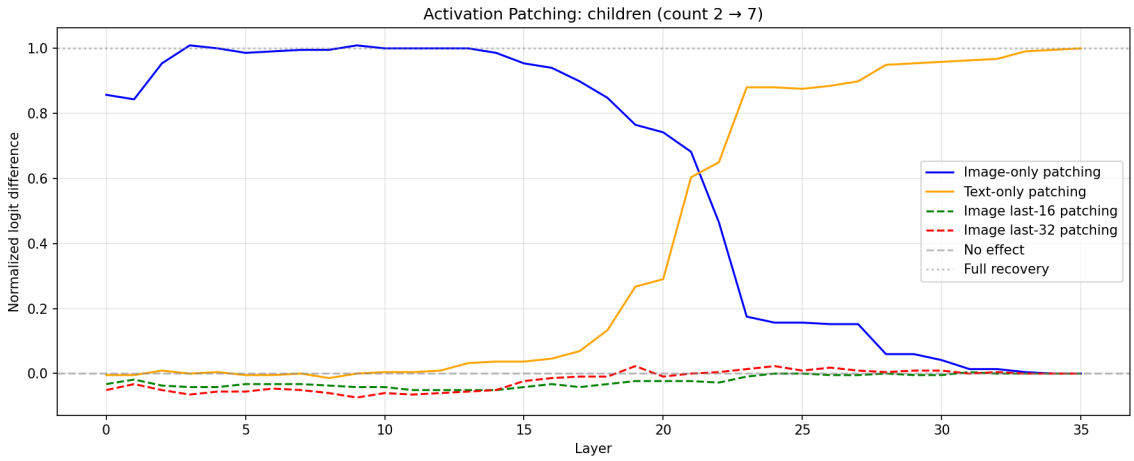

Figure 3: Layer-by-layer activation patching on a CountBench image pair (2 vs. 7 children). Normalized logit difference of 0 indicates no effect; 1 indicates full recovery of the source count. Image-only and text-only patching show a crossover around layers 15–25, while patching only the last 16 or 32 image tokens has negligible effect.

Zhang et al. (2025). Notably, patching only the last 16 or 32 image tokens produces near-zero effect across all layers, suggesting the count signal is distributed broadly across image tokens rather than concentrated in a small subset.

5 System Evaluations

5.1 Testing

Testing Paradigm. Testing interpretability libraries presents a practical challenge as loading full-sized VLMs requires significant GPU memory. We follow the testing approach by TransformerLens, with different categories of testing:

Unit tests test isolated primitives like activation cache operations, hook name parsing and validation, adapter registry lookups, and other utilities.

Integration tests load the smallest supported model (SmolVLM-256M) to verify preprocessor behavior, forward and generate paths, and batch input behavior.

Acceptance tests comprehensively test each supported model via a capability registry that tracks per-model properties, so that model-specific behaviors are only tested when applicable. These tests verify core API contracts, activation shapes for all hook types, cache correctness, hook composition, and others.

These categories help us ensure that the testing during development can be done in a tractable, meaningful manner.

We also want to highlight the following validations made in our tests:

Validating hidden states and attention patterns. We validate cache correctness by comparing activations obtained via `run_with_cache` against those from HuggingFace's native `output_attentions=True` and `output_hidden_states=True` flags. For residual stream activations, we verify element-wise equality within floating-point tolerance. For attention patterns, we additionally verify that softmax of cached attention scores plus the causal mask matches the native attention outputs.

Intervention Sanity Checks. We verify that hook-based interventions produce measurable causal effects: patching a constant activation into a hook point changes the forward-pass output, while returning None from a hook function leaves the output unchanged. We also verify cleanup invariants: we ensure that hooks are fully removed after both normal exits and exceptions from `run_with_hooks`, ensuring no residual state leaks between experiments.

5.2 Compute Time Benchmarking

We measure the runtime overhead of our hook-based `run_with_cache` against HuggingFace's native `output_hidden_states` and `output_attentions` flags. We run six forward-pass conditions on 300 CountBench images using LLaVA-1.5-7B, timing each with `time.perf_counter()` after 3 warmup iterations. For hook conditions, we report the total time including cache finalization, where hooks are removed and any deferred computation (e.g.

attention pattern recomputation from cached Q/K inputs) is performed. We also include two activation patching scenarios using `run_with_hooks`: one patching `hook_resid_post` and one patching attention head outputs via `attn.hook_z`, each targeting the last 10 image tokens at every layer.

Results are in Table 1. For caching and patching hidden states, Spectra's hooks add negligible overhead. For caching attention patterns, Spectra's hook-based approach takes 15% longer due to having to recompute the attention pattern during cache finalization. Head patching incurs 72% overhead due to the reshape and writeback cost of modifying attention heads across all layers.

Table 1: Hook overhead benchmark on 300 CountBench images (LLaVA-1.5-7B).

Method	Total (s)	vs Baseline
Plain forward	24.0	—
Caching hidden states		
HuggingFace	24.3	+1.3%
Spectra	24.6	+2.1%
Caching attention patterns		
HuggingFace	26.7	+11.1%
Spectra	30.9	+28.4%
Activation intervention (`run_with_hooks`)		
Residual-stream patch	25.6	+6.4%
Head patch	41.4	+72.3%

6 Conclusion

We hope that this library provides good abstractions for interpretability researchers to write experiments more easily, especially given the nuances in interpretability code that state-of-the-art coding agents currently struggle with.

We note that Spectra currently supports primarily the language model component in VLMs. Recent work by Li et al. (2025) takes mid-layer hidden states from the image encoder to analyze via logit lens in the LLM. We hope to support similar operations for image encoders in the near future, and have written the codebase with this in mind (e.g. with the hook names starting with `lm`).

Acknowledgments

This research is supported by the National Research Foundation, Singapore and Infocomm Media Development Authority under its Trust Tech Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Infocomm Media Development Authority. This research is also partially supported by the National Research Foundation, Singapore's RSS Scheme (NRF-RSS2022-009).

References

Estelle Aflalo, Meng Du, Shao-Yen Tseng, Yongfei Liu, Chenfei Wu, Nan Duan, and Vasudev Lal. 2022. Vl-interpret: An interactive visualization tool for interpreting vision-language transformers. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 21406–21415.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, and 45 others. 2025a. Qwen3-vl technical report. *Preprint*, arXiv:2511.21631.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, and 8 others. 2025b. Qwen2.5-vl technical report. *Preprint*, arXiv:2502.13923.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, and 6 others. 2023. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Yukun Li, Han Zhang, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. 2026. Conditional memory via scalable lookup: A new axis of sparsity for large language models. *Preprint*, arXiv:2601.07372.

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

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. 2023. Vision transformers need registers. In *The Twelfth International Conference on Learning Representations*.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, and 1 others. 2022. Toy models of superposition. *arXiv preprint arXiv:2209.10652*.

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.

Michal Golovanevsky, William Rudman, Vedant Palit, Carsten Eickhoff, and Ritambhara Singh. 2025. What do vlms notice? a mechanistic interpretability pipeline for gaussian-noise-free text-image corruption and evaluation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11462–11482.

Nicholas Jiang, Anish Kachinthaya, Suzanne Petryk, and Yossi Gandelsman. 2025. Interpreting and editing vision-language representations to mitigate hallucinations. In *The Thirteenth International Conference on Learning Representations*.

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

Yueyan Li, Chenggong Zhao, Zeyuan Zang, Caixia Yuan, and Xiaojie Wang. 2025. Reading images like texts: Sequential image understanding in vision-language models. *arXiv preprint arXiv:2509.19191*.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning.

Andrés Marafioti, Orr Zohar, Miquel Farré, Merve Noyan, Elie Bakouch, Pedro Cuenca, Cyril Zakka, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Vaibhav Srivastav, Joshua Lochner, Hugo Larcher, Mathieu Morlon, Lewis Tunstall, Leandro von Werra, and Thomas Wolf. 2025. Smolvlm: Redefining small and efficient multimodal models. *Preprint*, arXiv:2504.05299.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372.

Jack Merullo, Louis Castricato, Carsten Eickhoff, and Ellie Pavlick. 2023. Linearly mapping from image to text space. In *The Eleventh International Conference on Learning Representations*.

Neel Nanda and Joseph Bloom. 2022. Transformerlens. https://github.com/TransformerLensOrg/TransformerLens.

Clement Neo, Luke Ong, Philip Torr, Mor Geva, David Krueger, and Fazl Barez. 2025. Towards interpreting visual information processing in vision-language models. In *The Thirteenth International Conference on Learning Representations*.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, and 1 others. 2022. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*.

Roni Paiss, Ariel Ephrat, Omer Tov, Shiran Zada, Inbar Mosseri, Michal Irani, and Tali Dekel. 2023. Teaching clip to count to ten. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3170–3180.

Vedant Palit, Rohan Pandey, Aryaman Arora, and Paul Pu Liang. 2023. Towards vision-language mechanistic interpretability: A causal tracing tool for blip. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2856–2861.

Hala Sheta, Eric Haoran Huang, Shuyu Wu, Ilia Alenabi, Jiajun Hong, Ryker Lin, Ruoxi Ning, Daniel Wei, Jialin Yang, Jiawei Zhou, and 1 others. 2025. From behavioral performance to internal competence: Interpreting vision-language models with vlm-lens. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 886–895.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. In *The Eleventh International Conference on Learning Representations*.

Zhi Zhang, Srishti Yadav, Fengze Han, and Ekaterina Shutova. 2025. Cross-modal information flow in multimodal large language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 19781–19791.

A HuggingFace vs Spectra: A Code Comparison

Refer to Figure 4 for a comparison between HuggingFace and using Spectra.

B Web App Example

Refer to Figure 5 for an example on how the web app enables quick visualizations.

Raw HuggingFace Transformers (50 lines)

```python
from transformers import (
    Qwen2_5_VLForConditionalGeneration,
    AutoProcessor)
from qwen_vl_utils import process_vision_info
import torch

model = Qwen2_5_VLForConditionalGeneration \
    .from_pretrained(
    "Qwen/Qwen2.5-VL-7B-Instruct",
    torch_dtype=torch.float16,
    device_map="auto")
processor = AutoProcessor.from_pretrained(
    "Qwen/Qwen2.5-VL-7B-Instruct")
# Must be looked up per model
IMAGE_TOKEN_ID = 151655  # <|image_pad|>

def make_inputs(image):
    messages = [{"role": "user", "content": [
        {"type": "image", "image": image},
        {"type": "text",
         "text": "What is in the image?"}]}]
    text = processor.apply_chat_template(
        messages, tokenize=False,
        add_generation_prompt=True)
    # Manually append after assistant marker
    text += "This is an image of a"
    image_inputs, _ = process_vision_info(
        messages)
    return processor(text=[text],
        images=image_inputs,
        return_tensors="pt").to(model.device)

src_inputs = make_inputs(source_image)
tgt_inputs = make_inputs(target_image)

# Capture source residual via pre-hook
source_cache = {}
def capture_hook(module, args):
    source_cache["resid"] = \
        args[0].detach().clone()

handle = model.model.layers[0] \
    .register_forward_pre_hook(capture_hook)
with torch.no_grad():
    model(**src_inputs)
handle.remove()

# Compute spatial grid with merge factor
t, h, w = \
    src_inputs["image_grid_thw"][0].tolist()
merge = model.config \
    .vision_config.spatial_merge_size
grid_h, grid_w = h // merge, w // merge

# Find image tokens and select last row
src_img_pos = (src_inputs["input_ids"][0]
    == IMAGE_TOKEN_ID) \
    .nonzero(as_tuple=True)[0]
tgt_img_pos = (tgt_inputs["input_ids"][0]
    == IMAGE_TOKEN_ID) \
    .nonzero(as_tuple=True)[0]
src_last_row = src_img_pos[
    (grid_h-1)*grid_w : grid_h*grid_w]
tgt_last_row = tgt_img_pos[
    (grid_h-1)*grid_w : grid_h*grid_w]

# Patch last row into target run
def patch_hook(module, args):
    patched = args[0].clone()
    patched[0, tgt_last_row] = \
        source_cache["resid"][0, src_last_row]
    return (patched,) + args[1:]

handle = model.model.layers[0] \
    .register_forward_pre_hook(patch_hook)
with torch.no_grad():
    patched_output = model(**tgt_inputs)
handle.remove()
```

Spectra (22 lines)

```python
from vlm_spectra import HookedVLM

model = HookedVLM.from_pretrained(
    "Qwen/Qwen2.5-VL-7B-Instruct")
hook = "lm.blocks.0.hook_resid_pre"

src_inputs = model.prepare_messages(
    "What is in the image?", source_image,
    assistant_prefill="This is an image of a")
tgt_inputs = model.prepare_messages(
    "What is in the image?", target_image,
    assistant_prefill="This is an image of a")

# Cache source residual
with model.run_with_cache([hook]) as cache:
    model.forward(src_inputs)
src_resid = cache[hook]

# Spatial layout: last row of image patches
info = model.process_image(source_image)
row_start = (info.grid_h - 1) * info.grid_w
row_end = info.grid_h * info.grid_w

src_img, _ = model.get_image_token_range(
    src_inputs)
tgt_img, _ = model.get_image_token_range(
    tgt_inputs)

# Patch last row into target
def patch_fn(module, args, kwargs, output):
    output[0,
        tgt_img+row_start : tgt_img+row_end
    ] = src_resid[0,
        src_img+row_start : src_img+row_end]
    return output

with model.run_with_hooks([(hook, patch_fn)]):
    patched_output = model.forward(tgt_inputs)
```

Figure 4: Activation patching: replacing the last row of image-token residual-stream activations from a source image into a target forward pass at layer 0. Left: raw HuggingFace requires model-specific token IDs, manual hook registration and cleanup, and explicit spatial grid computation accounting for the adapter's merge factor. Right: Spectra abstracts architecture details including assistant prefill handling, reducing the code from 50 to 22 lines.

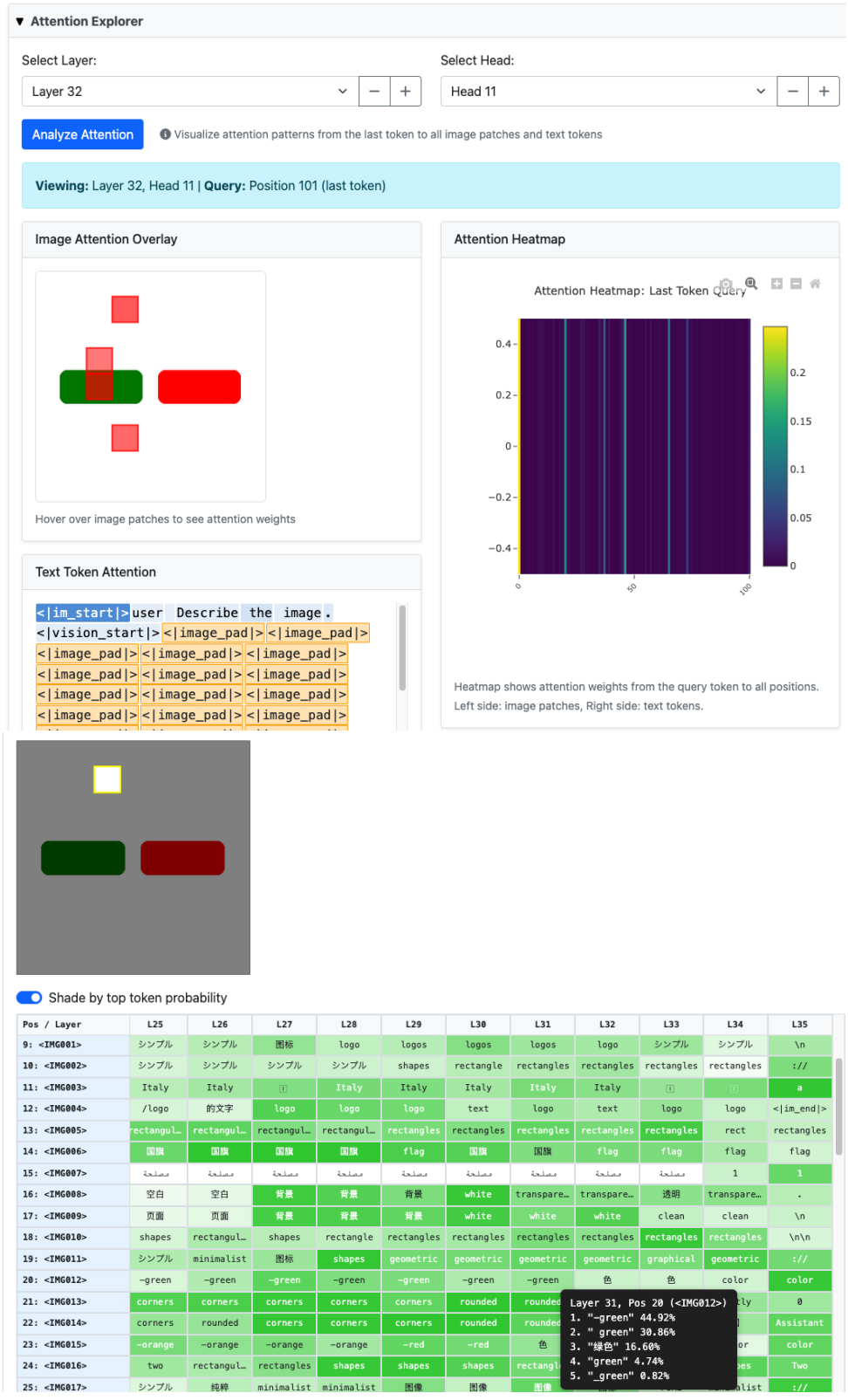

Figure 5: Example on how the web app can help with formulating quick hypotheses. In this example, an attention head helping to predict the "green" token (identified through direct logit attribution, not pictured) is attending to the green rectangle spot, but also other random background tokens. The logit lens then subsequently show that these background tokens correspond to decoding to "green" in the logit lens!