A Multimodal Automated Interpretability Agent

Tamar Rott Shaham [1 *] **Sarah Schwettmann** [1 *]
Franklin Wang [1] **Achyuta Rajaram** [1] **Evan Hernandez** [1] **Jacob Andreas** [1] **Antonio Torralba** [1]

Abstract

This paper describes MAIA, a **M**ultimodal **A**utomated **I**nterpretability **A**gent. MAIA is a system that uses neural models to automate neural model understanding tasks like feature interpretation and failure mode discovery. It equips a pre-trained vision-language model with a set of tools that support iterative experimentation on subcomponents of other models to explain their behavior. These include tools commonly used by human interpretability researchers: for synthesizing and editing inputs, computing maximally activating exemplars from real-world datasets, and summarizing and describing experimental results. *Interpretability experiments* proposed by MAIA compose these tools to describe and explain system behavior. We evaluate applications of MAIA to computer vision models. We first characterize MAIA's ability to describe (neuron-level) features in learned representations of images. Across several trained models and a novel dataset of synthetic vision neurons with paired ground-truth descriptions, MAIA produces descriptions comparable to those generated by expert human experimenters. We then show that MAIA can aid in two additional interpretability tasks: reducing sensitivity to spurious features, and automatically identifying inputs likely to be mis-classified.[†]

1. Introduction

Understanding of a neural model can take many forms. Given an image classifier, for example, we may wish to recognize when and how it relies on sensitive features like race or gender, identify systematic errors in its predictions, or learn how to modify the training data and model architecture to improve accuracy and robustness. Today, this kind of understanding requires significant effort on the part of researchers—involving exploratory data analysis, formulation of hypotheses, and controlled experimentation (Nushi et al., 2018; Zhang et al., 2018). As a consequence, this kind of understanding is slow and expensive to obtain even about the most widely used models.

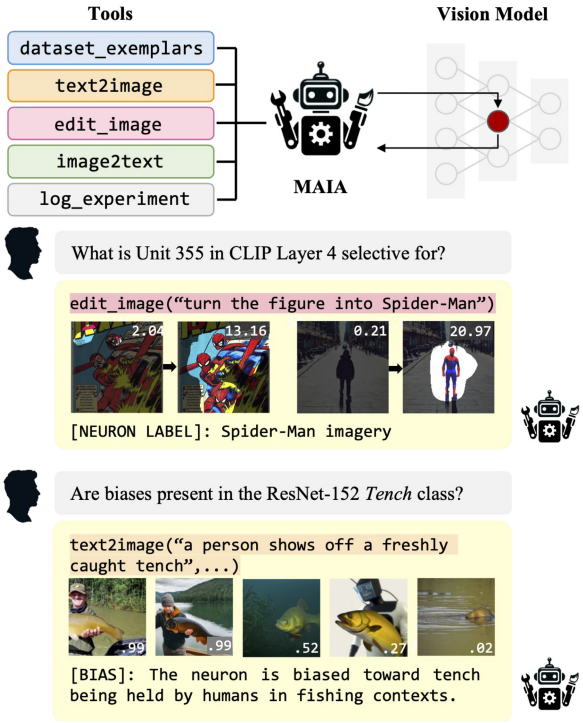

Figure 1. **MAIA framework.** MAIA autonomously conducts experiments on other systems to explain their behavior.

Recent work on **automated interpretability** (*e.g.* Hernandez et al., 2022; Bills et al., 2023; Schwettmann et al., 2023) has begun to address some of these limitations by using *learned models themselves* to assist with model understanding tasks—for example, by assigning natural language descriptions to learned representations, which may then be used to surface features of concern. But current methods are useful almost exclusively as tools for hypothesis generation; they characterize model behavior on a limited set of inputs, and are often low-precision (Huang et al., 2023). How can we build tools that help users understand models, while combining the flexibility of human experimentation with the scalability of automated techniques?

[*]Equal contribution [1]MIT CSAIL. Correspondence to: <tamarott@mit.edu, schwett@mit.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[†]Website: https://multimodal-interpretability.csail.mit.edu/maia

Figure 2. **MAIA experiments for labeling neurons.** MAIA iteratively writes programs that compose common interpretability tools to conduct experiments on other systems. At each step, MAIA autonomously makes and updates hypotheses in light of experimental outcomes, showing sophisticated scientific reasoning capabilities. Generated code is executed with a Python interpreter and the outputs (shown above, neuron activation values overlaid in white, masks thresholded at 0.95 percentile of activation maps) are returned to MAIA.

This paper introduces a prototype system we call the **M**ultimodal **A**utomated **I**nterpretability **A**gent (MAIA), which combines a pretrained vision-language model backbone with an API containing tools designed for conducting experiments on deep networks. MAIA is prompted with an explanation task (*e.g.* "describe the behavior of unit 487 in layer 4 of CLIP" or "in which contexts does the model fail to classify `labradors`?") and designs an *interpretability experiment* that composes experimental modules to answer the query. MAIA's modular design (Figure 1) enables flexible evaluation of arbitrary systems and straightforward incorporation of new experimental tools. Section 3 describes the current tools in MAIA's API, including modules for synthesizing and editing novel test images, which enable direct hypothesis testing during the interpretation process.

We evaluate MAIA's ability to produce predictive explanations of vision system components using the neuron description paradigm (Bau et al., 2017; 2020; Oikarinen & Weng, 2022; Bills et al., 2023; Singh et al., 2023; Schwettmann et al., 2023) which appears as a subroutine of many interpretability procedures. We additionally introduce a novel dataset of *synthetic vision neurons* built from an open-set concept detector with ground-truth selectivity specified via text guidance. In Section 4, we show that MAIA desriptions of both synthetic neurons and neurons in the wild are more predictive of behavior than baseline description methods, and in many cases on par with human labels.

MAIA also automates model-level interpretation tasks where descriptions of learned representations produce actionable insights about model behavior. We show in a series of experiments that MAIA's iterative experimental approach can be applied to downstream model auditing and editing tasks including spurious feature removal and bias identification in a trained classifier. Both applications demonstrate the adaptability of the MAIA framework across experimental settings: novel end-use cases are described in the user prompt to the agent, which can then use its API to compose programs that conduct task-specific experiments. While these applications show preliminary evidence that procedures like MAIA which automate both experimentation and description have high potential utility to the interpretability workflow, we find that MAIA still requires human steering to avoid common pitfalls including confirmation bias and drawing conclusions from small sample sizes. Fully automating end-to-end interpretation of other systems will not only require more advanced tools, but agents with more advanced capabilities to reason about how to use them.

2. Related work

Interpreting deep features. Investigating individual neurons inside deep networks reveals a range of human-interpretable features. Approaches to describing these neurons use exemplars of their behavior as explanation, either by visualizing features they select for (Zeiler & Fergus, 2014; Girshick et al., 2014; Karpathy et al., 2015; Mahendran & Vedaldi, 2015; Olah et al., 2017) or automatically categorizing maximally-activating inputs from real-world datasets (Bau et al., 2017; 2020; Oikarinen & Weng, 2022; Dalvi et al., 2019). Early approaches to translating visual exemplars into language descriptions drew labels from fixed vocabularies (Bau et al., 2017), or produced descriptions in the form of programs (Mu & Andreas, 2021).

Automated interpretability. Later work on automated interpretability produced open-ended descriptions of learned features in the form of natural language text, either curated from human labelers (Schwettmann et al., 2021) or generated directly by learned models (Hernandez et al., 2022; Bills et al., 2023; Gandelsman et al., 2024). However, these labels are often unreliable as *causal* descriptions of model behavior without further experimentation (Huang et al., 2023). Schwettmann et al. (2023) introduced the Automated Interpretability Agent protocol for experimentation on black-box systems using a language model agent, though this agent operated purely on language-based exploration of inputs, which limited its action space. MAIA similarly performs iterative experimentation rather than labeling features in a single pass, but has access to a library of interpretability tools as well as built-in vision capabilities. MAIA's modular design also supports experiments at different levels of granularity, ranging from analysis of individual features to sweeps over entire networks, or identification of more complex network subcomponents (Conmy et al., 2023).

Language model agents. Modern language models are promising foundation models for interpreting other networks due to their strong reasoning capabilities (OpenAI, 2023a). These capabilities can be expanded by using the LM as an *agent*, where it is prompted with a high-level goal and has the ability to call external tools such as calculators, search engines, or other models in order to achieve it (Schick et al., 2023; Qin et al., 2023). When additionally prompted to perform chain-of-thought style reasoning between actions, agentic LMs excel at multi-step reasoning tasks in complex environments (Yao et al., 2023). MAIA leverages an agent architecture to generate and test hypotheses about neural networks trained on vision tasks. While ordinary LM agents are generally restricted to tools with textual interfaces, previous work has supported interfacing with the images through code generation (Surís et al., 2023; Wu et al., 2023). More recently, large multimodal LMs like GPT-4V have enabled the use of image-based tools directly (Zheng et al., 2024; Chen et al., 2023). MAIA follows this design and is, to our knowledge, the first multimodal agent equipped with tools for interpreting deep networks.

3. MAIA Framework

MAIA is an agent that autonomously conducts experiments on other systems to explain their behavior, by composing interpretability subroutines into Python programs. Motivated by the promise of using language-only models to complete one-shot visual reasoning tasks by calling external tools (Surís et al., 2023; Gupta & Kembhavi, 2023), and the need to perform *iterative* experiments with both visual and numeric results, we build MAIA from a pretrained multimodal model with the ability to process images directly. MAIA is implemented with a GPT-4V vision-language model (VLM) backbone (OpenAI, 2023b) . Given an interpretability query (*e.g. Which neurons in Layer 4 are selective for forested backgrounds?*), MAIA runs experiments that test specific hypotheses (*e.g.* computing neuron outputs on images with edited backgrounds), observes experimental outcomes, and updates hypotheses until it can answer the user query.

We enable the VLM to design and run interpretability experiments using the MAIA API, which defines two classes: the `System` class and the `Tools` class, described below. The API is provided to the VLM in its system prompt. We include a complete API specification in Appendix A. The full input to the VLM is the API specification followed by a "user prompt" describing a particular interpretability task, such as explaining the behavior of an individual neuron inside a vision model with natural language (see Section 4). To complete the task, MAIA uses components of its API to write a series of Python programs that run experiments on the system it is interpreting. MAIA outputs function definitions as strings, which we then execute internally using the Python interpreter. The Pythonic implementation enables flexible incorporation of built-in functions and existing packages, *e.g.* the MAIA API uses the PyTorch library (Paszke et al., 2019) to load common pretrained vision models.

3.1. System API

The `System` class inside the MAIA API instruments the system to be interpreted and makes subcomponents of that system individually callable. For example, to probe single neurons inside ResNet-152 (He et al., 2016), MAIA can use the `System` class to initialize a neuron object by specifying its number and layer location, and the model that the neuron belongs to: `system = System(unit_id, layer_id, model_name)`. MAIA can then design experiments that test the neuron's activation value on different image inputs by running `system.neuron(image_list)`, to return activation values and masked versions of the images in the list that highlight maximally activating regions (See Figure 2 for examples). While existing approaches to common interpretability tasks such as neuron labeling require training specialized models on task-specific datasets (Hernandez et al., 2022), the MAIA system class supports querying arbitrary vision systems without retraining.

3.2. Tool API

The `Tools` class consists of a suite of functions enabling MAIA to write modular programs that test hypotheses about system behavior. MAIA tools are built from common interpretability procedures such as characterizing neuron behavior using real-world images (Bau et al., 2017) and performing causal interventions on image inputs (Hernandez et al., 2022; Casper et al., 2022), which MAIA then composes into more complex experiments (see Figure 2). When programs written by MAIA are compiled internally as Python code, these functions can leverage calls to other pretrained models to compute outputs. For example, `tools.text2image(prompt_list)` returns synthetic images generated by a text-guided diffusion model, using prompts written by MAIA to test a neuron's response to specific visual concepts. The modular design of the tool library enables straightforward incorporation of new tools as interpretability methods grow in sophistication. For the experiments in this paper we use the following set:

Dataset exemplar generation. Previous studies have shown that it is possible to characterize the prototypical behavior of a neuron by recording its activation values over a large dataset of images (Bau et al., 2017; 2020). We give MAIA the ability to run such an experiment on the validation set of ImageNet (Deng et al., 2009) and construct the set of 15 images that maximally activate the system it is interpreting. Interestingly, MAIA often chooses to begin experiments by calling this tool (Figure 2). We analyze the importance of the `dataset_exemplars` tool in our ablation study (4.3).

Image generation and editing tools. MAIA is equipped with a `text2image(prompts)` tool that synthesizes images by calling Stable Diffusion v1.5 (Rombach et al., 2022a) on text prompts. Generating inputs enables MAIA to test system sensitivity to fine-grained differences in visual concepts, or test selectivity for the same visual concept across contexts (*e.g.* the *bowtie* on a pet and on a child in Figure 2). We analyze the effect of using different text-to-image models in Section 4.3. In addition to synthesizing new images, MAIA can also edit images using Instruct-Pix2Pix (Brooks et al., 2022) by calling `edit_images(image, edit_instructions)`. Generating and editing synthetic images enables hypothesis tests involving images lying outside real-world data distributions, *e.g.* the addition of antelope horns to a horse (Figure 2, see *Causal intervention on image input*).

Image description and summarization tools. To limit confirmation bias in MAIA's interpretation of experimental results, we use a multi-agent framework in which MAIA can ask a new instance of GPT-4V with no knowledge of experimental history to describe highlighted image regions in individual images, `describe_images(image_list)`, or summarize what they have in common across a group of

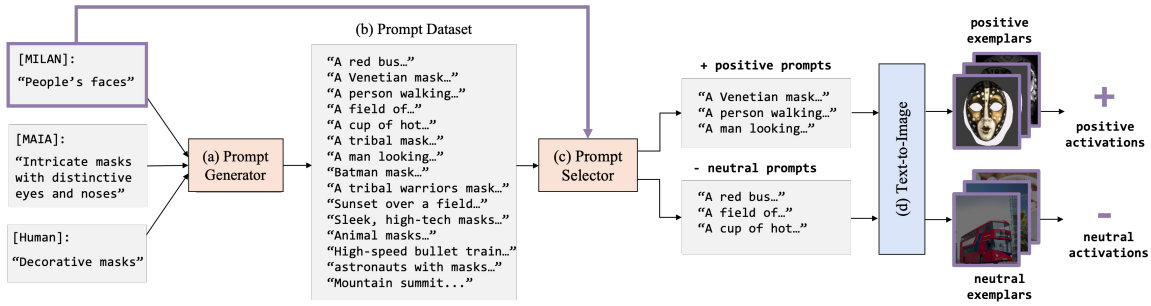

Figure 3. **Predictive evaluation protocol**. We compare neuron labeling methods by assessing how well their labels predict neuron activation values on unseen data. For each neuron we perform the following steps: (a) An LM uses candidate neuron labels to generate a set of image prompts that should maximally/neutrally activate the neuron. (b) All prompts (positive and neutral) from all methods are combined into one dataset. (c) For each labeling method, a new LM selects prompts from the Prompt Dataset that are likely to produce maximal and neutral neuron activations, if that label were accurate. (d) A text-to-image model generates all corresponding images, and the average activation values for positive and neutral images are recorded. A predictive neuron label will produce exemplars with maximally positive activations relative to the neutral baseline.

images, `summarize_images(image_list)`. We observe that MAIA uses this tool in situations where previous hypotheses failed or when observing complex combinations of visual content.

Experiment log. MAIA can document the results of each experiment (*e.g.* images, activations) using the `log_experiment` tool, to make them accessible during subsequent experiments. We prompt MAIA to finish experiments by logging results, and let it choose what to log (*e.g.* data that clearly supports or refutes a particular hypothesis).

4. Evaluation

The MAIA framework is task-agnostic and can be adapted to new applications by specifying an interpretability task in the user prompt to the VLM. Before tackling model-level interpretability problems (Section 5), we evaluate MAIA's performance on the black-box neuron description task, a widely studied interpretability subroutine that serves a variety of downstream model auditing and editing applications (Gandelsman et al., 2024; Yang et al., 2023; Hernandez et al., 2022). For these experiments, the user prompt to MAIA specifies the task and output format (a longer-form `[DESCRIPTION]` of neuron behavior, followed by a short `[LABEL]`), and MAIA's `System` class instruments a particular vision model (*e.g.* `ResNet-152`) and an individual unit indexed inside that model (*e.g.* `Layer 4 Unit 122`). Task specifications for these experiments may be found in Appendix B. We find MAIA correctly predicts behaviors of individual vision neurons in three trained architectures (Section 4.1), and in a synthetic setting where ground-truth neuron selectivities are known (Section 4.2). We also find descriptions produced by MAIA's interactive procedure to be more predictive of neuron behavior than descriptions of a fixed set of dataset exemplars, using the MILAN baseline from Hernandez et al. (2022). In many cases, MAIA descriptions are on par with those by human experts using the MAIA API. In Section 4.3, we perform ablation studies to test how components of the MAIA API differentially affect description quality.

4.1. Neurons in vision models

We use MAIA to produce natural language descriptions of a subset of neurons across three vision architectures trained under different objectives: ResNet-152, a CNN for supervised image classification (He et al., 2016), DINO (Caron et al., 2021), a Vision Transformer trained for unsupervised representation learning (Grill et al., 2020; Chen & He, 2021), and the CLIP visual encoder (Radford et al., 2021), a ResNet-50 model trained to align image-text pairs. For each model, we evaluate descriptions of 100 units randomly sampled from a range of layers that capture features at different levels of granularity (ResNet-152 *conv*.1, *res*.1-4, DINO MLP 1-11, CLIP *res*.1-4). Figure 2 shows examples of MAIA experiments on neurons from all three models, and final MAIA labels. We also evaluate a baseline non-interactive approach that only labels dataset exemplars of each neuron's behavior using the MILAN model from Hernandez et al. (2022). Finally, we collect human annotations of a random subset (25%) of neurons labeled by MAIA and MILAN, in an experimental setting where human experts write programs to perform interactive analyses of neurons using the MAIA API. Human experts receive the MAIA user prompt, write programs that run experiments on the neurons, and return neuron descriptions in the same format. See Appendix C3 for details on the human labeling experiments.

We evaluate the accuracy of neuron descriptions produced by MAIA, MILAN, and human experts by measuring how well they *predict* neuron behavior on unseen test images (Figure 3). Similar to evaluation approaches that produce contrastive or counterfactual exemplars to reveal model decision boundaries (Gardner et al., 2020; Kaushik et al., 2020),

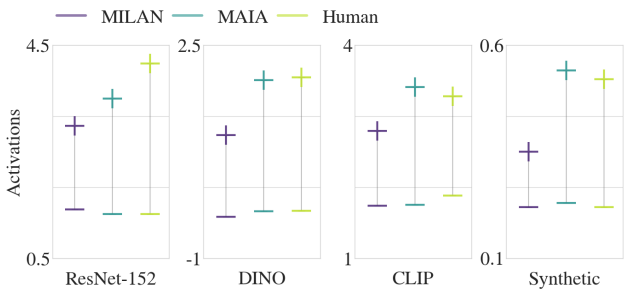

Figure 4. **Predictive evaluation results.** The average positive activation values ("+") for MAIA labels outperform MILAN and are comparable to human descriptions for both real and synthetic neurons. Neutral activations ("-") are comparable across methods.

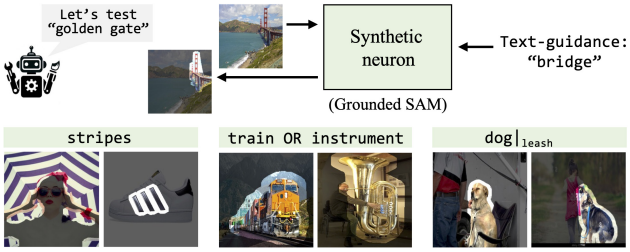

Figure 5. **Synthetic neuron implementation.** Segmentation of input images is performed by an open-set concept detector with text guidance specifying ground-truth neuron selectivity. Synthetic neurons return masked images and synthetic activation values corresponding to the probability a concept is present in the image.

we use candidate neuron labels to generate new images that should elicit maximally positive activations relative to a neutral baseline. For a given neuron, we generate a pool of image candidates by providing MAIA, MILAN, and human labels to a Prompt Generator model (implemented with a new instance of GPT-4). For each candidate label (*e.g. intricate masks*), the Prompt Generator is instructed to write 7 image prompts that should generate maximally activating images (*e.g. A Venetian mask, A tribal mask,...*), and 7 prompts for neutral images (unrelated to the label) that should elicit baseline activations (*e.g. A red bus, A field of flowers,...*). All positive and neutral prompts from all labeling methods (MAIA, MILAN, and human experts) form a Prompt Dataset of 42 prompts per neuron. Next, we evaluate the accuracy of each candidate label by using a Prompt Selector LM (implemented with another GPT-4 instance) to match that label with the 7 prompts it is most and least likely to entail. We then generate the corresponding images using a text-to-image model (DALL-E3) and measure neuron activation values on those images. If a neuron label is predictive of activations, it will be matched with positive exemplars that maximally activate the neuron relative to the neutral baseline. Combining prompts from all methods into one test set (vs. evaluating each model separately) more rigorously evaluates the completeness of each candidate label: an incomplete description produced by one labeling method (e.g. *trains* for a neuron selective for *trains* OR *dogs*) could be matched with a "neutral" image prompt describing *dogs*, which would in fact elicit high activation. This method primarily discriminates between labeling procedures: whether it is informative depends on the labeling methods themselves producing relevant exemplar prompts.

We report the average activation values of positive and neutral exemplars for MAIA, MILAN, and human labels across all tested models in Figure 4. **MAIA outperforms MILAN across all models and is often on par with expert predictions.** This trend persists across different averaging techniques (such as normalizing by activation percentile, see Appendix C1). While MILAN is a relevant neuron labeling baseline, we note that comparisons to task-specific procedures that use learned models to label a fixed set of exemplars only evaluate part of MAIA's full functionality. MAIA is easily adaptable to downstream auditing applications that require additional experimentation, where one-shot neuron labeling procedures are insufficient (see Section 5.1). Table A3 provides additional comparisons of MAIA to neuron labeling baselines, and shows evaluation results by layer.

4.2. Synthetic neurons

Following the procedure in Schwettmann et al. (2023) for validating the performance of automated interpretability methods on synthetic test systems mimicking real-world behaviors, we construct a set of *synthetic vision neurons* with known ground-truth selectivity. We simulate concept detection performed by neurons inside vision models using semantic segmentation. Synthetic neurons are built using an open-set concept detector that combines Grounded DINO (Liu et al., 2023) with SAM (Kirillov et al., 2023) to perform text-guided image segmentation. The ground-truth behavior of each neuron is determined by a text description of the concept(s) the neuron is selective for (Figure 5). To capture real-world behaviors, we derive neuron labels from MILAN-NOTATIONS, a dataset of 60K human annotations of neurons across seven trained vision models (Hernandez et al., 2022). Neurons in the wild display a diversity of behaviors: some respond to individual concepts, while others respond to complex combinations of concepts (Bau et al., 2017; Fong & Vedaldi, 2018; Olah et al., 2020; Mu & Andreas, 2021; Gurnee et al., 2023). We construct three types of synthetic neurons with increasing levels of complexity: monosemantic neurons that recognize single concepts (*e.g. stripes*), polysemantic neurons selective for logical disjunctions of concepts (*e.g. trains* OR *instruments*), and *conditional* neurons that only recognize a concept in presence of another concept (*e.g. dog*$|_{leash}$). Following the instrumentation of real neurons in the MAIA API, synthetic vision neurons accept image input and return a masked image highlighting the concept they are selective for (if present), and an activation

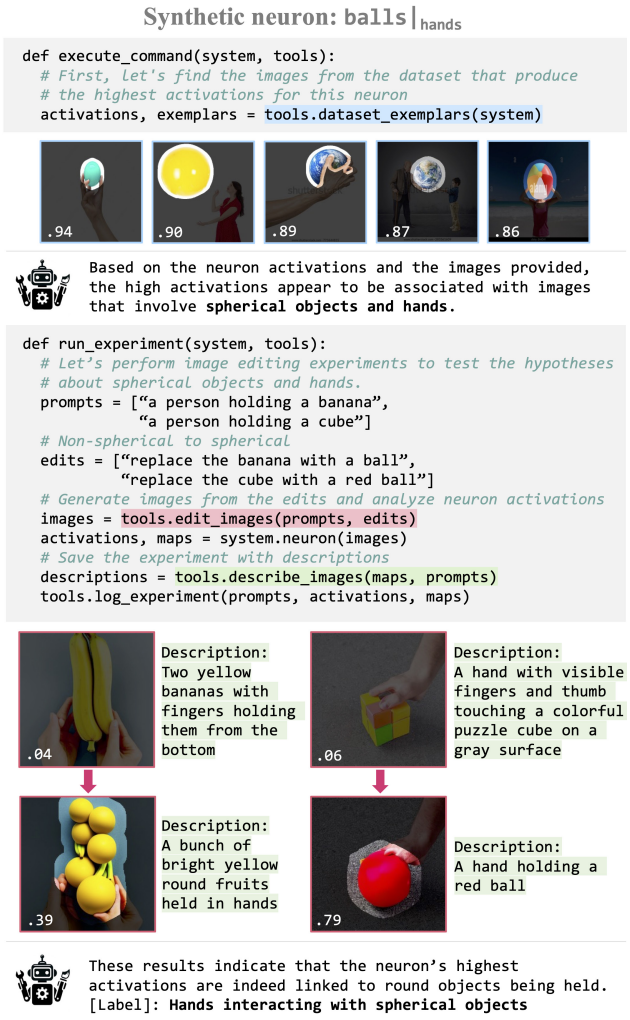

Figure 6. **MAIA synthetic neuron interpretation.**

value (corresponding to the confidence of Grounded DINO in the presence of the concept). Dataset exemplars for synthetic neurons are calculated by computing 15 top-activating images per neuron from the CC3M dataset (Sharma et al., 2018). Figure 5 shows examples of each type of neuron; the full list of 85 synthetic neurons is provided in Appendix C4. The set of concepts that can be represented by synthetic neurons is limited to simple concepts by the fidelity of open-set concept detection using current text-guided segmentation methods. We verify that all concepts in the synthetic neuron dataset can be segmented by Grounded DINO in combination with SAM, and provide further discussion of the limitations of synthetic neurons in Appendix C4.

MAIA interprets synthetic neurons using the same API and procedure used to interpret neurons in trained vision models (Section 4.1). In contrast to neurons in the wild, we can evaluate descriptions of synthetic neurons directly against ground-truth neuron labels. We collect comparative annotations of synthetic neurons from MILAN, as well as expert annotators (using the procedure from Section 4.1 where human experts manually label a subset of 25% of neurons using the MAIA API). We recruit human judges from Amazon Mechanical Turk to evaluate the agreement between synthetic neuron descriptions and ground-truth labels in pairwise two-alternative forced choice (2AFC) tasks. For each task, human judges are shown the ground-truth neuron label (*e.g. tail*) and descriptions produced by two labeling procedures (*e.g.* "fluffy and textured animal tails" and "circular objects and animals"), and asked to select which description better matches the ground-truth label. Further details are provided in Appendix C4. Table 1 shows the results of the 2AFC study (the proportion of trials in which procedure A was favored over B, and 95% confidence intervals). According to human judges, **MAIA labels better agree with ground-truth labels when compared to MILAN, and are even slightly preferred over expert labels** on the subset of neurons they described (while human labels are largely preferred over MILAN labels). We also use the predictive evaluation framework described in Section 4.1 to generate positive and neutral sets of exemplar images for all synthetic neurons. Figure 4 shows MAIA descriptions are better predictors of synthetic neuron activations than MILAN descriptions, on par with labels produced by human experts.

Table 1. **2AFC test.** Human subjects selected which method best agrees with the ground truth synthetic neuron label.

MAIA vs. MILAN	MAIA vs. Human	Human vs. MILAN
$0.73 \pm 4e^{-4}$	$0.53 \pm 1e^{-3}$	$0.83 \pm 5e^{-4}$

4.3. Tool ablation study

MAIA's modular design enables straightforward addition and removal of tools from its API. We test three different settings to quantify sensitivity to different tools: (i) labeling neurons using only the `dataset_exemplar` function without the ability to synthesize images, (ii) relying only on generated inputs without the option to compute maximally activating dataset exemplars, and (iii) replacing the Stable Diffusion `text2image` backbone with DALL-E 3. While the first two settings do not fully compromise performance, neither ablated API achieves the same average accuracy as the full MAIA system (Figure 7). These results emphasize the combined utility of tools for experimenting with real-world and synthetic inputs: MAIA performs best when initializing experiments with dataset exemplars and running additional tests with synthetic images. Methods like MILAN that label precomputed exemplars could thus be incorporated into the MAIA API as tools, and used to initialize experimentation. We also find that using DALL-E as the `text2image` backbone improves performance (Figure 7). This suggests that the agent is bounded by the performance of its tools rather than its ability to use them—and as interpretability tools grow in sophistication, so will MAIA.

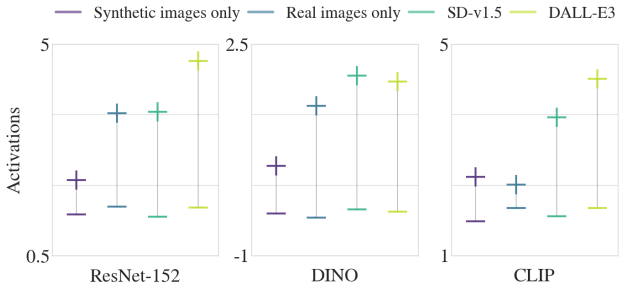

Figure 7. **Ablation study.** We use the predictive evaluation protocol to quantify MAIA's sensitivity to different tools. Top performance is achieved when experimenting with both real and synthetic data, and when using DALL-E 3 for image generation. More details in Appendix C2.

4.4. MAIA failure modes

Consistent with the result in Section 4.3 that MAIA performance improves with DALL-E 3, we additionally observe that SD-v1.5 and InstructPix2Pix sometimes fail to faithfully generate and edit images according to MAIA's instructions. To mitigate these failures, we instruct MAIA to prompt positive image-edits (*e.g. replace* the bowtie with a plain shirt) rather than negative edits (*e.g. remove* the bowtie), but occasional failures still occur (see Figure 8 and Appendix D). While proprietary versions of tools may be of higher quality, they also introduce prohibitive rate limits and costs associated with API access. As similar limitations apply to the GPT-4V backbone itself, we tested the performance of free and non-proprietary VLMs as alternative MAIA backbones. Currently, off-the-shelf alternatives still significantly lag behind GPT-4V performance (consisitent with evaluation of open-source models' ability to interpret functions in Schwettmann et al. (2023)), but our initial experiments suggest their performance may improve with fine-tuning (see Appendix D3). The MAIA system is designed modularly so that open-source alternatives can be incorporated in the future as their performance improves.

5. Applications

MAIA is a flexible system that automates model understanding tasks at different levels of granularity: from labeling individual features to diagnosing model-level failure modes. To demonstrate the utility of MAIA for producing actionable insights for human users (Vaughan & Wallach, 2020), we conduct experiments that apply MAIA to two model-level tasks: (i) spurious feature removal and (ii) bias identification in a downstream classification task. In both cases MAIA uses the API as described in Section 3. In an additional experiment, we evaluate the downstream utility of MAIA descriptions by measuring the extent to which they equip humans to make predictions about system behavior (see details in Appendix E).

Figure 8. **MAIA tool failures.** MAIA is limited by the reliability of its tools. Common image editing failure modes (using InstructPix2Pix) include failing to remove objects, misinterpreting the instructions (e.g. removing the incorrect object), and changing too much or too little of the image. MAIA's image generation tool (SD-v1.5) is sometimes unreliable for negative instructions (e.g. a flagpole *without* a flag), and sometimes deviates from the text prompt by adding or excluding image components.

5.1. Removing spurious features

Learned spurious features impose a challenge when machine learning models are applied in real-world scenarios, where test distributions differ from training set statistics (Storkey et al., 2009; Beery et al., 2018; Bissoto et al., 2020; Xiao et al., 2020; Singla et al., 2021). We use MAIA to remove learned spurious features inside a classification network, finding that *with no access to unbiased examples nor grouping annotations*, MAIA can identify and remove such features, improving model robustness under distribution shift by a wide margin, with an accuracy approaching that of fine-tuning on balanced data.

We run experiments on ResNet-18 trained on the Spawrious dataset (Lynch et al., 2023), a synthetically generated dataset involving four dog breeds with different backgrounds. In the train set, each breed is spuriously correlated with a certain background type, while in the test set, the breed-background pairings are changed (see Figure 9). We use MAIA to find a subset of final layer neurons that robustly predict a single dog breed independently of spurious features (see Appendix F3). While other methods like Kirichenko et al. (2023) remove spurious correlations by retraining the last layer on *balanced* datasets, we only provide MAIA access to top-activating images from the *unbalanced* validation set and prompt MAIA to run experiments to determine robustness. We then use the features MAIA selects to train an unregularized logistic regression model on the unbalanced data.

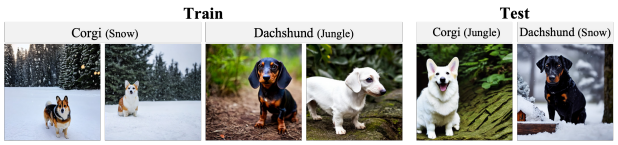

Figure 9. **Spawrious dataset examples.** Train data contains spurious correlations between dog breeds and their backgrounds.

Table 2. Final layer spurious feature removal results.

Subset	Selection Method	# Units	Balanced	Test Acc.
All	Original Model	512	✗	0.731
ℓ_1 Top 50	All	50	✗	0.779
	Random	22	✗	0.705 ± 0.05
	ℓ_1 Top 22	22	✗	0.757
	MILAN	23	✗	0.786
	MILAN (GPT-4V)	23	✗	0.690
	MAIA	22	✗	**0.837**
All	ℓ_1 Hyper. Tuning	147	✓	0.830
	ℓ_1 Top 22	22	✓	**0.865**

As a demonstration, we select 50 of the most informative neurons using ℓ_1 regularization on the *unbalanced* dataset and have MAIA run experiments on each one. MAIA selects 22 neurons it deems to be robust. Traning an unregularized model on this subset significantly improves accuracy, as reported in Table 2. For comparison, we repeat the same task using interpretability procedures like MILAN that rely on precomputed exemplars (both with the original model of (Hernandez et al., 2022) and with GPT-4V, see Appendix F2 for experimental details). Both achieved significantly lower accuracy. To further show that the sparsity of MAIA's neuron selection is not the only reason for its performance improvements, we also benchmark MAIA's performance against ℓ_1 regularized fitting on both unbalanced and balanced versions of the dataset. On the unbalanced dataset, ℓ_1 drops in performance when subset size reduces from 50 to 22 neurons. Using a small *balanced* dataset to hyperparameter tune the ℓ_1 parameter and train the logistic regression model on all neurons achieves performance comparable to MAIA's chosen subset, although MAIA did not have access to any balanced data. For a fair comparison, we test the performance of an ℓ_1 model which matches the sparsity of MAIA, but trained on the balanced dataset. See Appendix F2 for more details.

5.2. Revealing biases

MAIA can be used to automatically surface model-level biases. Specifically, we apply MAIA to investigate biases in the outputs of a CNN (ResNet-152) trained on a supervised ImageNet classification task. The MAIA system is easily adaptable to this experiment: the output logit corresponding to a specific class is instrumented using the `system` class, and returns class probability for input images. MAIA is provided with the class label and instructed (see Appendix G) to find settings in which the classifier ranks images related to that class with relatively lower probability values, or shows a clear preference for a subset of the class. Figure 10 presents results for a subset of ImageNet classes. This simple paradigm suggests that MAIA's generation of synthetic data could be widely useful for identifying regions of the input distribution where a model exhibits poor performance. While this exploratory experiment surfaces only broad failure categories, MAIA enables other experiments targeted at end-use cases identifying specific biases.

Figure 10. **MAIA bias detection.** MAIA iteratively conducts experiments and generates synthetic inputs to surface biases in ResNet-152 output classes. In some cases, MAIA discovers uniform behavior over the inputs (*e.g. flagpole*).

6. Conclusion

We introduce MAIA, an agent that automates interpretability tasks including feature interpretation and bias discovery. By composing pretrained modules, MAIA conducts experiments to make and test hypotheses about the behavior of other systems. While human supervision is needed to maximize its effectiveness and catch common mistakes, initial experiments with MAIA show promise, and we anticipate that interpretability agents will be increasingly useful as they grow in sophistication.

Impact statement

As AI systems take on higher-stakes roles and become more deeply integrated into research and society, scalable approaches to auditing for reliability will be vital. MAIA is a protoype for a tool that can help human users ensure AI systems are transparent, reliable, and equitable.

We think MAIA augments, but does not replace, human oversight of AI systems. MAIA still requires human supervision to catch mistakes such as confirmation bias and image generation/editing failures. Absence of evidence (from MAIA) is not evidence of absence: though MAIA's toolkit enables causal interventions on inputs in order to evaluate system behavior, MAIA's explanations do not provide formal verification of system performance.

Acknowlegements

We are grateful for the support of the MIT-IBM Watson AI Lab, the Open Philanthropy foundation, Hyundai Motor Company, ARL grant W911NF-18-2-021, Intel, the National Science Foundation under grant CCF-2217064, the Zuckerman STEM Leadership Program, and the Viterbi Fellowship. The funders had no role in experimental design or analysis, decision to publish, or preparation of the manuscript. The authors have no competing interests to report.

References

Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Bau, D., Zhou, B., Khosla, A., Oliva, A., and Torralba, A. Network dissection: Quantifying interpretability of deep visual representations. In *Computer Vision and Pattern Recognition*, 2017.

Bau, D., Zhu, J.-Y., Strobelt, H., Lapedriza, A., Zhou, B., and Torralba, A. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, 2020. ISSN 0027-8424. doi: 10.1073/pnas.1907375117. URL https://www.pnas.org/content/early/2020/08/31/1907375117.

Beery, S., Van Horn, G., and Perona, P. Recognition in terra incognita. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 456–473, 2018.

Bills, S., Cammarata, N., Mossing, D., Tillman, H., Gao, L., Goh, G., Sutskever, I., Leike, J., Wu, J., and Saunders, W. Language models can explain neurons in language models. https://openaipublic.blob.core.windows.net/neuron-explainer/paper/index.html, 2023.

Bissoto, A., Valle, E., and Avila, S. Debiasing skin lesion datasets and models? not so fast. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 740–741, 2020.

Brooks, T., Holynski, A., and Efros, A. A. Instructpix2pix: Learning to follow image editing instructions. *arXiv preprint arXiv:2211.09800*, 2022.

Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., and Joulin, A. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9650–9660, 2021.

Casper, S., Hariharan, K., and Hadfield-Menell, D. Diagnostics for deep neural networks with automated copy/paste attacks. *arXiv preprint arXiv:2211.10024*, 2022.

Chen, L., Zhang, Y., Ren, S., Zhao, H., Cai, Z., Wang, Y., Wang, P., Liu, T., and Chang, B. Towards end-to-end embodied decision making via multi-modal large language model: Explorations with gpt4-vision and beyond, 2023.

Chen, X. and He, K. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15750–15758, 2021.

Conmy, A., Mavor-Parker, A. N., Lynch, A., Heimersheim, S., and Garriga-Alonso, A. Towards automated circuit discovery for mechanistic interpretability. *arXiv preprint arXiv:2304.14997*, 2023.

Dalvi, F., Durrani, N., Sajjad, H., Belinkov, Y., Bau, A., and Glass, J. What is one grain of sand in the desert? analyzing individual neurons in deep nlp models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 6309–6317, 2019.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Fong, R. and Vedaldi, A. Net2vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8730–8738, 2018.

Gandelsman, Y., Efros, A. A., and Steinhardt, J. Interpreting clip's image representation via text-based decomposition, 2024.

Gardner, M., Artzi, Y., Basmova, V., Berant, J., Bogin, B., Chen, S., Dasigi, P., Dua, D., Elazar, Y., Gottumukkala, A., Gupta, N., Hajishirzi, H., Ilharco, G., Khashabi, D., Lin, K., Liu, J., Liu, N. F., Mulcaire, P., Ning, Q., Singh, S., Smith, N. A., Subramanian, S., Tsarfaty, R., Wallace, E., Zhang, A., and Zhou, B. Evaluating models' local decision boundaries via contrast sets, 2020.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Avila Pires, B., Guo, Z., Gheshlaghi Azar, M., et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

Gupta, T. and Kembhavi, A. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14953–14962, 2023.

Gurnee, W., Nanda, N., Pauly, M., Harvey, K., Troitskii, D., and Bertsimas, D. Finding neurons in a haystack: Case studies with sparse probing. *arXiv preprint arXiv:2305.01610*, 2023.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hernandez, E., Schwettmann, S., Bau, D., Bagashvili, T., Torralba, A., and Andreas, J. Natural language descriptions of deep visual features. In *International Conference on Learning Representations*, 2022.

Huang, J., Geiger, A., D'Oosterlinck, K., Wu, Z., and Potts, C. Rigorously assessing natural language explanations of neurons. *arXiv preprint arXiv:2309.10312*, 2023.

Karpathy, A., Johnson, J., and Fei-Fei, L. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Kaushik, D., Hovy, E., and Lipton, Z. C. Learning the difference that makes a difference with counterfactually-augmented data, 2020.

Kirichenko, P., Izmailov, P., and Wilson, A. G. Last layer re-training is sufficient for robustness to spurious correlations, 2023.

Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollár, P., and Girshick, R. Segment anything. *arXiv:2304.02643*, 2023.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., and Willing, C. Jupyter notebooks – a publishing format for reproducible computational workflows. In Loizides, F. and Schmidt, B. (eds.), *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pp. 87 – 90. IOS Press, 2016.

Liu, H., Li, C., Li, Y., Li, B., Zhang, Y., Shen, S., and Lee, Y. J. Llava-next: Improved reasoning, ocr, and world knowledge, January 2024. URL https://llava-vl.github.io/blog/2024-01-30-llava-next/.

Liu, S., Zeng, Z., Ren, T., Li, F., Zhang, H., Yang, J., Li, C., Yang, J., Su, H., Zhu, J., et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild, 2015.

Lynch, A., Dovonon, G. J.-S., Kaddour, J., and Silva, R. Spawrious: A benchmark for fine control of spurious correlation biases, 2023.

Mahendran, A. and Vedaldi, A. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5188–5196, 2015.

Mu, J. and Andreas, J. Compositional explanations of neurons, 2021.

Nushi, B., Kamar, E., and Horvitz, E. Towards accountable ai: Hybrid human-machine analyses for characterizing system failure. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, volume 6, pp. 126–135, 2018.

Oikarinen, T. and Weng, T.-W. Clip-dissect: Automatic description of neuron representations in deep vision networks. *arXiv preprint arXiv:2204.10965*, 2022.

Olah, C., Mordvintsev, A., and Schubert, L. Feature visualization. *Distill*, 2(11):e7, 2017.

Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., and Carter, S. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

OpenAI. Gpt-4 technical report, 2023a.

OpenAI. Gpt-4v(ision) technical work and authors. https://openai.com/contributions/gpt-4v, 2023b. Accessed: [insert date of access].

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., Zhao, S., Hong, L., Tian, R., Xie, R., Zhou, J., Gerstein, M., Li, D., Liu, Z., and Sun, M. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision, 2021.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695, June 2022a.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models, 2022b.

Sagawa, S., Koh, P. W., Hashimoto, T. B., and Liang, P. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization, 2020.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools, 2023.

Schwettmann, S., Hernandez, E., Bau, D., Klein, S., Andreas, J., and Torralba, A. Toward a visual concept vocabulary for gan latent space. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6804–6812, 2021.

Schwettmann, S., Shaham, T. R., Materzynska, J., Chowdhury, N., Li, S., Andreas, J., Bau, D., and Torralba, A. Find: A function description benchmark for evaluating interpretability methods, 2023.

Sharma, P., Ding, N., Goodman, S., and Soricut, R. Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2556–2565, 2018.

Singh, C., Hsu, A. R., Antonello, R., Jain, S., Huth, A. G., Yu, B., and Gao, J. Explaining black box text modules in natural language with language models, 2023.

Singla, S., Nushi, B., Shah, S., Kamar, E., and Horvitz, E. Understanding failures of deep networks via robust feature extraction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12853–12862, 2021.

Storkey, A. et al. When training and test sets are different: characterizing learning transfer. *Dataset shift in machine learning*, 30(3-28):6, 2009.

Surís, D., Menon, S., and Vondrick, C. Vipergpt: Visual inference via python execution for reasoning, 2023.

Vaughan, J. W. and Wallach, H. A human-centered agenda for intelligible machine learning. *Machines We Trust: Getting Along with Artificial Intelligence*, 2020.

Wah, C., Branson, S., Welinder, P., Perona, P., and Belongie, S. *The Caltech-UCSD Birds-200-2011 Dataset*. Caltech Vision Lab, Jul 2011.

Wu, C., Yin, S., Qi, W., Wang, X., Tang, Z., and Duan, N. Visual chatgpt: Talking, drawing and editing with visual foundation models, 2023.

Xiao, K., Engstrom, L., Ilyas, A., and Madry, A. Noise or signal: The role of image backgrounds in object recognition. *arXiv preprint arXiv:2006.09994*, 2020.

Yang, Y., Panagopoulou, A., Zhou, S., Jin, D., Callison-Burch, C., and Yatskar, M. Language in a bottle: Language model guided concept bottlenecks for interpretable image classification, 2023.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models, 2023.

Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pp. 818–833. Springer, 2014.

Zhang, J., Wang, Y., Molino, P., Li, L., and Ebert, D. S. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE*

transactions on visualization and computer graphics, 25 (1):364–373, 2018.

Zheng, B., Gou, B., Kil, J., Sun, H., and Su, Y. Gpt-4v(ision) is a generalist web agent, if grounded, 2024.

Zou, X., Yang, J., Zhang, H., Li, F., Li, L., Wang, J., Wang, L., Gao, J., and Lee, Y. J. Segment everything everywhere all at once, 2023.

Appendix

A. MAIA Library

The full MAIA API provided in the system prompt is reproduced below.

```
import torch
from typing import List, Tuple

class System:
    """
    A Python class containing the vision model and the specific neuron to interact with.

    Attributes
    ----------
    neuron_num : int
        The unit number of the neuron.
    layer : string
        The name of the layer where the neuron is located.
    model_name : string
        The name of the vision model.
    model : nn.Module
        The loaded PyTorch model.

    Methods
    -------
    load_model(model_name: str) -> nn.Module
        Gets the model name and returns the vision model from PyTorch library.
    neuron(image_list: List[torch.Tensor]) -> Tuple[List[int], List[str]]
        returns the neuron activation for each image in the input image_list as well as the activation map
        of the neuron over that image, that highlights the regions of the image where the activations
        are higher (encoded into a Base64 string).
    """
    def __init__(self, neuron_num: int, layer: str, model_name: str, device: str):
        """
        Initializes a neuron object by specifying its number and layer location and the vision model that the neuron
              belongs to.
        Parameters
        -------
        neuron_num : int
            The unit number of the neuron.
        layer : str
            The name of the layer where the neuron is located.
        model_name : str
            The name of the vision model that the neuron is part of.
        device : str
            The computational device ('cpu' or 'cuda').
        """
        self.neuron_num = neuron_num
        self.layer = layer
        self.device = torch.device(f"cuda:{device}" if torch.cuda.is_available() else "cpu")
        self.model = self.load_model(model_name)


    def load_model(self, model_name: str) -> torch.nn.Module:
        """
        Gets the model name and returns the vision model from pythorch library.
        Parameters
        ----------
        model_name : str
            The name of the model to load.

        Returns
        -------
        nn.Module
            The loaded PyTorch vision model.

        Examples
        --------
        >>> # load "resnet152"
        >>> def run_experiment(model_name) -> nn.Module:
        >>>   model = load_model(model_name: str)
        >>>   return model
        """
        return load_model(model_name)

    def neuron(self, image_list: List[torch.Tensor]) -> Tuple[List[int], List[str]]:
        """
```

```
        The function returns the neuron's maximum activation value (in int format) for each of the images in the
             list as well as the activation map of the neuron over each of the images that highlights the regions of
              the image where the activations are higher (encoded into a Base64 string).

        Parameters
        ----------
        image_list : List[torch.Tensor]
            The input image

        Returns
        -------
        Tuple[List[int], List[str]]
            For each image in image_list returns the activation value of the neuron on that image, and a masked
                 image,
            with the region of the image that caused the high activation values highlighted (and the rest of the
                 image is darkened). Each image is encoded into a Base64 string.


        Examples
        --------
        >>> # test the activation value of the neuron for the prompt "a dog standing on the grass"
        >>> def run_experiment(system, tools) -> Tuple[int, str]:
        >>>     prompt = ["a dog standing on the grass"]
        >>>     image = tools.text2image(prompt)
        >>>     activation_list, activation_map_list = system.neuron(image)
        >>>     return activation_list, activation_map_list
        >>> # test the activation value of the neuron for the prompt "a dog standing on the grass" and the neuron
             activation value for the same image but with a lion instead of a dog
        >>> def run_experiment(system, tools) -> Tuple[int, str]:
        >>>     prompt = ["a dog standing on the grass"]
        >>>     edits = ["replace the dog with a lion"]
        >>>     all_image, all_prompts = tools.edit_images(prompt, edits)
        >>>     activation_list, activation_map_list = system.neuron(all_images)
        >>>     return activation_list, activation_map_list

        """
        return neuron(image_list)

class Tools:
    """
    A Python class containing tools to interact with the neuron implemented in the system class,
    in order to run experiments on it.

    Attributes
    ----------
    experiment_log: str
        A log of all the experiments, including the code and the output from the neuron.


    Methods
    -------
    dataset_exemplars(system: object) -> Tuple(List[int],List[str])
        This experiment provides good coverage of the behavior observed on a very large dataset of images and
             therefore represents the typical behavior of the neuron on real images.
        This function characterizes the prototipycal behavior of the neuron by computing its activation on all
             images in the ImageNet dataset and returning the 15 highest activation values and the images that
             produced them.
        The images are masked to highlight the specific regions that produce the maximal activation. The images are
             overlaid with a semi-opaque mask, such that the maximally activating regions remain unmasked.
    edit_images(prompt_list_org_image : List[str], editing_instructions_list : List[str]) -> Tuple[List[Image.Image
         ], List[str]]
        This function enables loclized testing of specific hypotheses about how variations on the content of a
             single image affect neuron activations.
        Gets a list of input prompt and a list of corresponding editing instructions, then generate images according
              to the input prompts and edits each image based on the instructions given in the prompt using a text-
             based image editing model.
        This function is very useful for testing the causality of the neuron in a controlled way, for example by
             testing how the neuron activation is affected by changing one aspect of the image.
        IMPORTANT: Do not use negative terminology such as "remove ...", try to use terminology like "replace ...
             with ..." or "change the color of ... to ...".
    text2image(prompt_list: str) -> Tuple[torcu.Tensor]
        Gets a list of text prompt as an input and generates an image for each prompt in the list using a text to
             image model.
        The function returns a list of images.
    summarize_images(self, image_list: List[str]) -> str:
        This function is useful to summarize the mutual visual concept that appears in a set of images.
        It gets a list of images at input and describes what is common to all of them, focusing specifically on
             unmasked regions.
    describe_images(synthetic_image_list: List[str], synthetic_image_title:List[str]) -> str
        Provides impartial descriptions of images. Do not use this function on dataset exemplars.
```

```
    Gets a list of images and generat a textual description of the semantic content of the unmasked regions
        within each of them.
    The function is blind to the current hypotheses list and therefore provides an unbiased description of the
        visual content.
log_experiment(activation_list: List[int], image_list: List[str], image_titles: List[str],
    image_textual_information: Union[str, List[str]]) -> None
    documents the current experiment results as an entry in the experiment log list. if self.
        activation_threshold was updated by the dataset_exemplars function,
    the experiment log will contains instruction to continue with experiments if activations are lower than
        activation_threshold.
    Results that are loged will be available for future experiment (unlogged results will be unavailable).
    The function also update the attribure "result_list", such that each element in the result_list is a
        dictionary of the format: {"<prompt>": {"activation": act, "image": image}}
    so the list contains all the resilts that were logged so far.
"""

def __init__(self):
    """
    Initializes the Tools object.

    Parameters
    ----------
    experiment_log: store all the experimental results
    """
    self.experiment_log = []
    self.results_list = []


def dataset_exemplars(self, system: object) -> Tuple(List[int],List[str])
    """
    This method finds images from the ImageNet dataset that produce the highest activation values for a specific
         neuron.
    It returns both the activation values and the corresponding exemplar images that were used
    to generate these activations (with the highly activating region highlighted and the rest of the image
        darkened).
    The neuron and layer are specified through a 'system' object.
    This experiment is performed on real images and will provide a good approximation of the neuron behavior.

    Parameters
    ----------
    system : object
        An object representing the specific neuron and layer within the neural network.
        The 'system' object should have 'layer' and 'neuron_num' attributes, so the dataset_exemplars function
        can return the exemplar activations and masked images for that specific neuron.

    Returns
    -------
    tuple
        A tuple containing two elements:
        - The first element is a list of activation values for the specified neuron.
        - The second element is a list of exemplar images (as Base64 encoded strings) corresponding to these
            activations.

    Example
    -------
    >>> def run_experiment(system, tools)
    >>>     activation_list, image_list = self.dataset_exemplars(system)
    >>>     return activation_list, image_list
    """

    return dataset_exemplars(system)

def edit_images(self, prompt_list_org_image : List[str], editing_instructions_list : List[str]) -> Tuple[List[
    Image.Image], List[str]]:
    """
    This function enables localized testing of specific hypotheses about how variations in the content of a
        single image affect neuron activations.
    Gets a list of prompts to generate images, and a list of corresponding editing instructions as inputs. Then
        generates images based on the image prompts and edits each image based on the instructions given in the
         prompt using a text-based image editing model (so there is no need to generate the images outside of
        this function).
    This function is very useful for testing the causality of the neuron in a controlled way, for example by
        testing how the neuron activation is affected by changing one aspect of the image.
    IMPORTANT: for the editing instructions, do not use negative terminology such as "remove ...", try to use
        terminology like "replace ... with ..." or "change the color of ... to"
    The function returns a list of images, constructed in pairs of original images and their edited versions,
        and a list of all the corresponding image prompts and editing prompts in the same order as the images.

    Parameters
    ----------
    prompt_list_org_image : List[str]
```

```
        A list of input prompts for image generation. These prompts are used to generate images which are to be
            edited by the prompts in editing_instructions_list.
    editing_instructions_list : List[str]
        A list of instructions for how to edit the images in image_list. Should be the same length as
            prompt_list_org_image.
        Edits should be relatively simple and describe replacements to make in the image, not deletions.

    Returns
    -------
    Tuple[List[Image.Image], List[str]]
        A list of all images where each unedited image is followed by its edited version.
        And a list of all the prompts corresponding to each image (e.g. the input prompt followed by the editing
            instruction).

    Examples
    --------
    >>> # test the activation value of the neuron for the prompt "a dog standing on the grass" and the neuron
         activation value for the same image but with a cat instead of a dog
    >>> def run_experiment(system, tools) -> Tuple[int, str]:
    >>>     prompt = ["a dog standing on the grass"]
    >>>     edits = ["replace the dog with a cat"]
    >>>     all_image, all_prompts = tools.edit_images(prompt, edits)
    >>>     activation_list, activation_map_list = system.neuron(all_images)
    >>>     return activation_list, activation_map_list
    >>> # test the activation value of the neuron for the prompt "a dog standing on the grass" and the neuron
         activation values for the same image but with a different action instead of "standing":
    >>> def run_experiment(system, tools) -> Tuple[int, str]:
    >>>     prompts = ["a dog standing on the grass"]*3
    >>>     edits = ["make the dog sit","make the dog run","make the dog eat"]
    >>>     all_images, all_prompts = tools.edit_images(prompts, edits)
    >>>     activation_list, activation_map_list = system.neuron(all_images)
    >>>     return activation_list, activation_map_list
    """

    return edit_images(image, edits)


def text2image(self, prompt_list: List[str]) -> List[Image.Image]:
    """Gets a list of text prompts as input, generates an image for each prompt in the list using a text to
         image model.
    The function returns a list of images.

    Parameters
    ----------
    prompt_list : List[str]
        A list of text prompts for image generation.

    Returns
    -------
    List[Image.Image]
        A list of images, corresponding to each of the input prompts.

    Examples
    --------
    >>> # test the activation value of the neuron for the prompt "a dog standing on the grass"
    >>> def run_experiment(system, tools) -> Tuple[int, str]:
    >>>     prompt = ["a dog standing on the grass"]
    >>>     image = tools.text2image(prompt)
    >>>     activation_list, activation_map_list = system.neuron(image)
    >>>     return activation_list, activation_map_list
    >>> # test the activation value of the neuron for the prompt "a fox and a rabbit watch a movie under a
         starry night sky" "a fox and a bear watch a movie under a starry night sky" "a fox and a rabbit watch a
          movie at sunrise"
    >>> def run_experiment(system, tools) -> Tuple[int, str]:
    >>>     prompt_list = ["a fox and a rabbit watch a movie under a starry night sky", "a fox and a bear watch
         a movie under a starry night sky","a fox and a rabbit watch a movie at sunrise"]
    >>>     images = tools.text2image(prompt_list)
    >>>     activation_list, activation_map_list = system.neuron(images)
    >>>     return activation_list, activation_map_list
    """

    return text2image(prompt_list)

def summarize_images(self, image_list: List[str]) -> str:
    """
    This function is useful to summarize the mutual visual concept that appears in a set of images.
    It gets a list of images at input and describes what is common to all of them, focusing specifically on
         unmasked regions.

    Parameters
    ----------
```

```
        image_list : list
            A list of images in Base64 encoded string format.

        Returns
        -------
        str
            A string with a descriptions of what is common to all the images.

        Example
        -------
        >>> # tests dataset exemplars and return textual summarization of what is common for all the maximally
             activating images
        >>> def run_experiment(system, tools):
        >>>     activation_list, image_list = self.dataset_exemplars(system)
        >>>     prompt_list = []
        >>>     for i in range(len(activation_list)):
        >>>          prompt_list.append(f'dataset exemplar {i}') # for the dataset exemplars we don't have prompts,
             therefore need to provide text titles
        >>>     summarization = tools.summarize_images(image_list)
        >>>     return summarization
        """

        return summarize_images(image_list)

    def describe_images(self, image_list: List[str], image_title:List[str]) -> str:
        """
        Provides impartial description of the highlighted image regions within an image.
        Generates textual descriptions for a list of images, focusing specifically on highlighted regions.
        This function translates the visual content of the highlited region in the image to a text description.
        The function operates independently of the current hypothesis list and thus offers an impartial description
             of the visual content.
        It iterates through a list of images, requesting a description for the
        highlighted (unmasked) regions in each synthetic image. The final descriptions are concatenated
        and returned as a single string, with each description associated with the corresponding
        image title.

        Parameters
        ----------
        image_list : list
            A list of images in Base64 encoded string format.
        image_title : callable
            A list of strings with the image titles that will be use to list the different images. Should be the
                 same length as image_list.

        Returns
        -------
        str
            A concatenated string of descriptions for each image, where each description
            is associated with the image's title and focuses on the highlighted regions
            in the image.

        Example
        -------
        >>> def run_experiment(system, tools):
        >>>     prompt_list = ["a fox and a rabbit watch a movie under a starry night sky", "a fox and a bear watch
             a movie under a starry night sky","a fox and a rabbit watch a movie at sunrise"]
        >>>     images = tools.text2image(prompt_list)
        >>>     activation_list, image_list = system.neuron(images)
        >>>     descriptions = tools.describe_images(image_list, prompt_list)
        >>>     return descriptions
        """

        return describe_images(image_list, image_title)


    def log_experiment(self, activation_list: List[int], image_list: List[str], image_titles: List[str],
         image_textual_information: Union[str, List[str]]):
        """documents the current experiment results as an entry in the experiment log list. if self.
             activation_threshold was updated by the dataset_exemplars function,
        the experiment log will contain instruction to continue with experiments if activations are lower than
             activation_threshold.
        Results that are logged will be available for future experiments (unlogged results will be unavailable).
        The function also updates the attribute "result_list", such that each element in the result_list is a
             dictionary of the format: {"<prompt>": {"activation": act, "image": image}}
        so the list contains all the results that were logged so far.

        Parameters
        ----------
        activation_list : List[int]
            A list of the activation values that were achived for each of the images in "image_list".
        image_list : List[str]
```

```
    A list of the images that were generated using the text2image model and were tested. Should be the same
        length as activation_list.
image_titles : List[str]
    A list of the text lables for the images. Should be the same length as activation_list.
image_textual_information: Union[str, List[str]]
    A string or a list of strings with additional information to log such as the image summarization and/or
        the image textual descriptions.

Returns
-------
    None


Examples
--------
>>> # tests the activation value of the neuron for the prompts "a fox and a rabbit watch a movie under a
     starry night sky", "a fox and a bear watch a movie under a starry night sky", "a fox and a rabbit watch
      a movie at sunrise", describes the images and logs the results and the image descriptions
>>> def run_experiment(system, tools):
>>>     prompt_list = ["a fox and a rabbit watch a movie under a starry night sky", "a fox and a bear watch
     a movie under a starry night sky","a fox and a rabbit watch a movie at sunrise"]
>>>     images = tools.text2image(prompt_list)
>>>     activation_list, activation_map_list = system.neuron(images)
>>>     descriptions = tools.describe_images(images, prompt_list)
>>>     tools.log_experiment(activation_list, activation_map_list, prompt_list, descriptions)
>>>     return
>>> # tests dataset exemplars, use umage summarizer and logs the results
>>> def run_experiment(system, tools):
>>>     activation_list, image_list = self.dataset_exemplars(system)
>>>     prompt_list = []
>>>     for i in range(len(activation_list)):
>>>         prompt_list.append(f'dataset_exemplars {i}') # for the dataset exemplars we don't have prompts,
      therefore need to provide text titles
>>>     summarization = tools.summarize_images(image_list)
>>>     log_experiment(activation_list, activation_map_list, prompt_list, summarization)
>>>     return
>>> # test the effect of changing a dog into a cat. Describes the images and logs the results.
>>> def run_experiment(system, tools) -> Tuple[int, str]:
>>>     prompt = ["a dog standing on the grass"]
>>>     edits = ["replace the dog with a cat"]
>>>     all_images, all_prompts = tools.edit_images(prompt, edits)
>>>     activation_list, activation_map_list = system.neuron(all_images)
>>>     descriptions = tools.describe_images(activation_map_list, all_prompts)
>>>     tools.log_experiment(activation_list, activation_map_list, all_prompts, descriptions)
>>>     return
>>> # test the effect of changing the dog's action on the activation values. Describes the images and logs
     the results.
>>> def run_experiment(system, prompt_list) -> Tuple[int, str]:
>>>     prompts = ["a dog standing on the grass"]*3
>>>     edits = ["make the dog sit","make the dog run","make the dog eat"]
>>>     all_images, all_prompts = tools.edit_images(prompts, edits)
>>>     activation_list, activation_map_list = system.neuron(all_images)
>>>     descriptions = tools.describe_images(activation_map_list, all_prompts)
>>>     tools.log_experiment(activation_list, activation_map_list, all_prompts, descriptions)
>>>     return
"""

return log_experiment(activation_list, image_list, prompt_list, description)
```

B. MAIA user prompt: neuron description

````
Your overall task is to describe the visual concepts that maximally activate a neuron inside a deep network
for computer vision.
To do that you are provided with a library of Python functions to run experiments on the specific neuron
(inside the "System" class) given the functions provided in the "Tools" class. Make sure to use a variety of
tools from the library to maximize your experimentation power.
Some neurons might be selective for very specific concepts, a group of unrelated concepts, or a general
concept, so try to be creative in your experiment and try to test both general and specific concepts. If a
neuron is selective for multiple concepts, you should describe each of those concepts in your final
description.
At each experiment step, write Python code that will conduct your experiment on the tested neuron, using the
following format:
[CODE]:
```python
def run_experiment(system, tools)
   # gets an object of the system class, an object of the tool class, and performs experiments on the neuron
   with the tools
   ...
   tools.log_experiment(...)
```
Finish each experiment by documenting it by calling the "log_experiment" function. Do not include any
additional implementation other than this function. Do not call "execute_command" after defining it. Include
only a single instance of experiment implementation at each step.

Each time you get the output of the neuron, try to summarize what the inputs that activate the neuron have in
common (where that description is not influenced by previous hypotheses). Then, write multiple hypotheses that
could explain the visual concept(s) that activate the neuron. Note that the neuron can be selective for more
than one concept.
For example, these hypotheses could list multiple concepts that the neuron is selective for (e.g. dogs OR cars
OR birds), provide different explanations for the same concept, describe the same concept at different levels
of abstraction, etc. Some of the concepts can be quite specific, test hypotheses that are both general and
very specific.
Then write a list of initial hypotheses about the neuron selectivity in the format:
[HYPOTHESIS LIST]:
Hypothesis_1: <hypothesis_1>
...
Hypothesis_n: <hypothesis_n>.

After each experiment, wait to observe the outputs of the neuron. Then your goal is to draw conclusions from
the data, update your list of hypotheses, and write additional experiments to test them. Test the effects of
both local and global differences in images using the different tools in the library.
If you are unsure about the results of the previous experiment you can also rerun it, or rerun a modified
version of it with additional tools. Use the following format:
[HYPOTHESIS LIST]: ## update your hypothesis list according to the image content and related activation values.

Only update your hypotheses if image activation values are higher than previous experiments.
[CODE]: ## conduct additional experiments using the provided python library to test *ALL* the hypotheses. Test
different and specific aspects of each hypothesis using all of the tools in the library. Write code to run the
 experiment in the same format provided above. Include only a single instance of experiment implementation.

Continue running experiments until you prove or disprove all of your hypotheses. Only when you are confident
in your hypothesis after proving it in multiple experiments, output your final description of the neuron in
the following format:

[DESCRIPTION]: <final description> ## Your description should be selective (e.g. very specific: "dogs running
on the grass" and not just "dog") and complete (e.g. include all relevant aspects the neuron is selective for).

In cases where the neuron is selective for more than one concept, include in your description a list of all
the concepts separated by logical "OR".

[LABEL]: <final label drived from the hypothesis or hypotheses> ## a label for the neuron generated from the
hypothesis (or hypotheses) you are most confident in after running all experiments. They should be concise and
complete, for example, "grass surrounding animals", "curved rims of cylindrical objects", "text displayed on
computer screens", "the blue sky background behind a bridge", and "wheels on cars" are all appropriate. You
should capture the concept(s) the neuron is selective for. Only list multiple hypotheses if the neuron is
selective for multiple distinct concepts. List your hypotheses in the format:
[LABEL 1]: <label 1>
[LABEL 2]: <label 2>
````

C. Evaluation experiment details

In Table A3 we provide full evaluation results by layer, as well as the number of units evaluated in each layer. Units were sampled uniformly at random, for larger numbers of units in later layers with more interpretable features.

Table A3. Evaluation results by layer

			MILAN		CLIP-Dissect		MAIA		Human	
Arch.	Layer	# Units	+	−	+	−	+	−	+	−
ResNet-152	conv. 1	10	7.23	3.38	6.7	2.71	7.28	3.53	7.83	3.16
	res. 1	15	0.82	0.73	0.83	0.55	0.78	0.69	0.46	0.64
	res. 2	20	0.98	0.92	1.02	0.66	1.02	0.90	0.83	0.95
	res. 3	25	1.28	0.72	0.98	0.68	1.28	0.70	2.59	0.58
	res. 4	30	5.41	2.04	7.1	1.61	7.10	1.74	7.89	1.99
	Avg.		**2.99**	**1.42**	**3.37**	**1.14**	**3.50**	**1.33**	**4.15**	**1.34**
DINO-ViT	MLP 1	5	1.10	0.94	0.76	0.94	1.19	0.74	0.63	0.34
	MLP 3	5	0.63	0.96	0.96	1.06	0.81	0.87	0.55	0.89
	MLP 5	20	0.85	1.01	1.11	0.94	1.33	0.97	0.84	0.84
	MLP 7	20	1.42	0.77	1.16	0.66	1.67	0.82	2.58	0.54
	MLP 9	25	3.50	-1.15	0.81	-0.83	6.31	-0.81	8.64	-1.06
	MLP 11	25	-1.56	-1.94	-1.12	-1.65	-1.41	-1.84	-0.61	-2.49
	Avg.		**1.03**	**-0.32**	**0.44**	**-0.2**	**1.93**	**0.54**	**1.97**	**-0.23**
CLIP-RN50	res. 1	10	1.92	2.16	2.34	1.82	2.10	2.07	1.65	2.15
	res. 2	20	2.54	2.46	2.61	1.91	2.78	2.39	2.22	2.81
	res. 3	30	2.24	1.70	2.33	1.45	2.27	1.70	2.41	1.96
	res. 4	40	3.56	1.30	4.75	1.36	4.90	1.39	4.92	1.29
	Avg.		**2.79**	**1.74**	**3.36**	**1.55**	**3.41**	**1.75**	**3.29**	**1.89**

C1. Predictive evaluation with activation normalizing.

Averaging the raw activation values might be largely affected by neurons with a relatively high activation range. Optimally activation values should be reported in percentile, relative to the activation range of each neuron individually, however neuron activation ranges are difficult to be precisely estimated. Instead, we perform the following estimation: we normalized the activation value of each neuron by its 95% percentile activation value (assuming 0% corresponds to an activation value of 0, which is a valid assumption for convolutional networks with ReLU activation like ResNet and CLIP). Results are reported in Table C1, showing the same trends as pre-normalization.

Table A4. Predictive evaluation results normalizing by 95% percentile

	ResNet-152				CLIP-RN5			
exemplars	MILAN	CLIP-Dissect	MAIA	Human	MILAN	CLIP-Dissect	MAIA	Human
+	1.51	1.55	1.63	2.34	1.53	1.89	1.9	1.4
-	0.83	0.75	0.78	0.63	0.77	0.77	0.76	0.75

C2. Ablation studies

We use the subset of 25% neurons labeled by human experts to perform the ablation studies. Results of the predictive evaluation procedure described in Section 4 are shown below. Using DALL-E 3 improves performance over SD-v1.5.

Table A5. Numerical data for the ablations in Figure 7.

		ImageNet	SD-v1.5	DALL-E 3
ResNet-152	+	3.53	3.56	4.64
	−	1.54	1.33	1.53
DINO-ViT	+	1.48	1.98	1.88
	−	-0.37	-0.23	-0.27
CLIP-RN50	+	2.34	3.62	4.34
	−	1.90	1.75	1.90

C3. Human expert neuron description using the MAIA tool library

We recruited 8 human interpretability researchers to use the MAIA API to run experiments on neurons in order to describe their behavior. This data collection effort was approved by MIT's Committee on the Use of Humans as Experimental Subjects. Humans received task specification via the MAIA user prompt, wrote programs using the functions inside the MAIA API, and produced neuron descriptions in the same format as MAIA. All human subjects had knowledge of Python. Humans labeled 25% of the units in each layer labeled by MAIA (one human label per neuron). Testing was administered via JupyterLab (Kluyver et al., 2016), as displayed in Figure A11. Humans also labeled 25% of the synthetic neurons using the same workflow. The median number of interactions per neuron for humans was 7. However, for more difficult neurons the number of interactions were as high as 39.

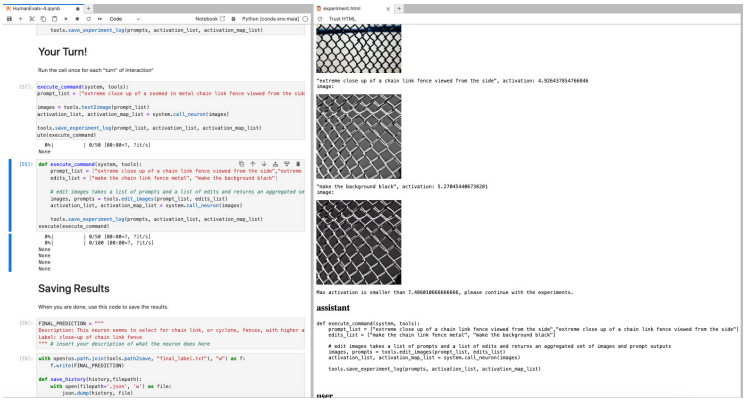

Figure A11. Example interface for humans interpreting neurons with the same tool library used by MAIA.

C4. Synthetic neurons

To provide a ground truth against which to test MAIA, we constructed a set of synthetic neurons that reflect the diverse response profiles of neurons in the wild. We used three categories of synthetic neurons with varying levels of complexity: *monosemantic* neurons that respond to single concepts, *polysemantic* neurons that respond to logical disjunctions of concepts, and *conditional* neurons that respond to one concept conditional on the presence of another. The full set of synthetic neurons across all categories is described in Table A6. To capture real-world neuron behaviors, concepts are drawn from MILANNOTATIONS, a dataset of 60K human annotations of prototypical neuron behaviors (Hernandez et al., 2022).

Synthetic neurons are constructed using Grounded DINO (Liu et al., 2023) in combination with SAM (Kirillov et al., 2023). Specifically, Grounded-DINO implements open-vocabulary object detection by generating image bounding boxes corresponding to an input text prompt. These bounding boxes are then fed into SAM as a soft prompt, indicating which part of the image to segment. To ensure the textual relevance of the bounding box, we set a threshold to filter out bounding boxes that do not correspond to the input prompt, using similarity scores which are also returned as synthetic neuron activation values. We use the default thresholds of 0.3 for bounding box accuracy and 0.25 for text similarity matching, as recommended in (Liu et al., 2023). After the final segmentation maps are generated, per-object masks are combined and dilated to resemble outputs of neurons inside trained vision models, instrumented via MAIA's `System` class.

We also implement compound synthetic neurons that mimic polysemantic neurons found in the wild (via logical disjunction), and neurons that respond to complex combinations of concepts (via logical conjunction). To implement polysemantic neurons (*e.g.* selective for *A* `OR` *B*), we check if at least one concept is present in the input image (if both are present, we merge segmentation maps across concepts and return the mean of the two activation values). To implement conditional neurons (*e.g.* selective for $A|_B$), we check if A is present, and if the condition is met (B is present) we return the mask and activation value corresponding to concept A.

The set of concepts that can be represented by synthetic neurons is limited by the fidelity of open-set concept detection using current text-guided segmentation methods. We manually verify that all concepts in the synthetic neuron dataset can be consistently segmented by Grounded DINO in combination with SAM. There are some types of neuron behavior, however, that cannot be captured using current text-guided segmentation methods. Some neurons inside trained vision models implement low-level procedures (*e.g.* edge-detection), or higher level perceptual similarity detection (*e.g.* sensitivity to radial wheel-and-spoke patterns common to flower petals and bicycle tires) that Grounded DINO + SAM cannot detect. Future implementations could explore whether an end-to-end single model open-vocabulary segmentation system, such as Segment Everything Everywhere All at Once (Zou et al., 2023), could perform segmentation for richer neuron labels.

Table A6. **Synthetic neurons.** Concepts are drawn from MILANNOTATIONS.

Monosemantic	*Polysemantic* (A OR B)	*Conditional* ($A\|_B$)
arch	animal, door	ball, hand
bird	animal, ship	beach, people
blue	baby, dog	bird, tree
boat	bird, dog	bridge, sky
brick	blue, yellow	building, sky
bridge	bookshelf, building	cup, handle
bug	cup, road	dog, leash
building	dog, car	fence, animal
button	dog, horse	fish, water
car window	dog, instrument	grass, dog
circle	fire, fur	horse, grass
dog	firework, whisker	instrument, hand
eyes	hand, ear	skyline, water
feathers	necklace, flower	sky, bird
flame	people, building	snow, road
frog	people, wood	suit, tie
grass	red, purple	tent, mountain
hair	shoe, boat	water, blue
hands	sink, pool	wheel, racecar
handle	skirt, water	
hat	stairs, fruit	
jeans	temple, playground	
jigsaw	truck, train	
legs	window, wheel	
light		
neck		
orange		
paws		
pencil		
pizza		
roof		
shirt		
shoes		
sky		
snake		
spiral		
stripes		
sunglasses		
tail		
text		
tires		
tractor		
vehicle		
wing		
yarn		

Evaluation of synthetic neuron labels using human judges. This data collection effort was approved by MIT's Committee on the Use of Humans as Experimental Subjects. To control for quality, workers were required to have a HIT acceptance rate of at least 99%, be based in the U.S., and have at least 10,000 approved HITs. Workers were paid $0.10 per annotation. 10 human judges performed each comparison task.

D. Failure modes

D1. Tool Failures

MAIA is often constrained by the capabilities of its tools. As shown in Figure 8, the Instruct-Pix2Pix (Brooks et al., 2022) and Stable Diffusion (Rombach et al., 2022b) models sometimes fail to follow the precise instructions in the caption. Instruct-Pix2Pix typically has trouble making changes which are relative to objects within the image and also fails to make changes that are unusual (such as the example of replacing a person with a vase). Stable Diffusion typically has difficulty assigning attributes in the caption to the correct parts of the image. These errors in image editing and generation sometimes confuse MAIA and cause it to make the wrong prediction.

D2. Confirmation Bias

In some scenarios, when MAIA generates an image that has a higher activation than the dataset exemplars, it will assume that the neuron behaves according to that single exemplar. Instead of conducting additional experiments to see if there may be a more general label, MAIA sometimes stops experimenting and outputs a final label that is specific to that one image. For instance, in Figure A12 MAIA generates one underwater image that attains a higher activation and outputs an overly specific description without doing any additional testing.

D3. Failure modes of non-proprietary VLMs as alternative MAIA backbones.

We tested the performance of two models as the MAIA backbone VLM:

1. LLaVA-Next (Liu et al., 2024), a top-performing (improved over LLaVA-1.5) open source VLM built from LLaMA.
2. Gemini 1.0 Pro (Anil et al., 2023), which Google currently provides for free with a low rate limit (60 RPM).

When used as the MAIA backbone for the neuron interpretation task, both models show initial potential: they understand the task, use the System class and the Tools class from MAIA's API, and write code specifying interpretation experiments. However, while both models show promise, we found significant shortcomings compared to GPT-4V which limit their current usefulness as backbones of MAIA. Some of these shortcomings we observed in both models:

Weaker hypothesis generation. Key to MAIA is the

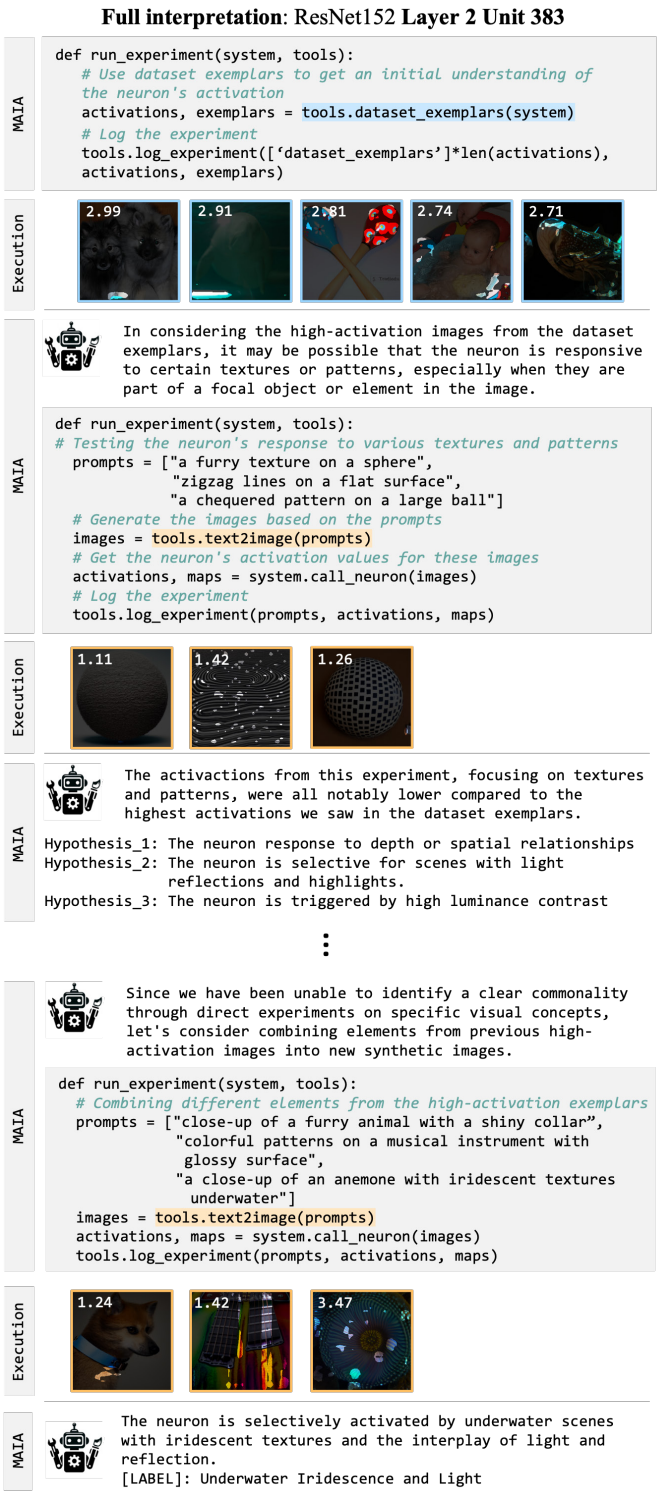

Figure A12. Example of MAIA having confirmation bias towards a single generated example, instead of generating further experiments to test other possibilities.

ability of the backbone VLM to make and update hypotheses in light of experimental findings. After receiving image outputs of experiments, both LLaVA-Next and Gemini are biased toward describing the images rather than updating their hypothesis about system behavior (suggesting both were trained for the task of image captioning, which might be solvable with fine-tuning).

Hallucination. We observed that both models often output hypotheses unrelated to experimental outcomes, and sometimes hallucinate experimental results. For example, when Gemini got the activation value and masked image for the prompt *"a dog standing on the grass"* it replied with: `a cat laying on the grass, activation: 14.21`, which is an image never generated, and a hallucinated activation value.

Overfitting to the examples of the tool usage presented in the MAIA API. Rather than designing new experiments, both models often reproduce code that appears in the MAIA API usage examples. For example, both Llava and Gemini will generate images of *"a dog standing on the grass"* which is provided as an example for an input prompt to the text2image tool.

Gemini did show some advantages over LLaVA: while Gemini occasionally produced some syntax errors (*e.g.* calling `system.dataset_exemplars` instead of `tools.dataset_exemplars`), Llava code was often not executable. Furthermore, LLaVa is restricted to getting only one input image at each interaction, which severely restricts its experimentation ability. Even when inputting several images in a row, the model is biased toward analyzing the last one.

E. Utility of MAIA descriptions for human users

We run additional crowd-sourcing experiments to quantify the extent to which neuron descriptions equip humans to predict system behavior. Given a text description of a neuron (produced either by MILAN, MAIA or human experts), human participants predicted the neuron's activations on images from the ImageNet validation set. In this setting, a correct description would inform more accurate predictions of behavior.

Specifically, participants saw a language description of a neuron (e.g. "this neuron is selective for human hands interacting with weightlifting equipment") and four sets of images. One of those sets contains images that strongly activate the neuron described in text, and the other 3 contain randomly sampled distractors (capturing "baseline activity"). Humans are asked to use the text description to select the strongly activating set of images (chance is 25%).

Two notes on experimental design: (i) We used sets of images (4 images each) in the multiple choice task instead of single images to provide better coverage of concept space (some text labels describe more concepts than a single image could show). (ii) In this task, humans distinguish strongly activating images from weakly activating images (like the automated evaluation in the main paper), instead of ordering images along a continuum by how strongly they activate the neuron. This is because the explanations produced by the interpretability methods only describe concepts that strongly activate the neuron, and do not provide enough information for human observers to characterize the full distribution of the neuron's activity.

The table below shows results from this experiment and 95% confidence intervals across interpretation procedures for a given model. For each model we evaluated the same subset of 25 neurons as in Section 4. 10 human participants performed each task (using the same crowdworker selection criteria as in Section 4.2). MAIA descriptions are more useful for predicting neuron behavior than MILAN descriptions in all three models studied.

Table A7. Subject predictions of maximally activating images based on neuron descriptions.

	MILAN	MAIA	Human
ResNet-152	66.0 ± 0.06	78.5 ± 0.05	85.45 ± 0.04
CLIP-RN50	57.19 ± 0.06	65.21 ± 0.06	69.6 ± 0.06
DINO-ViT	48.0 ± 0.06	67.39 ± 0.06	81.2 ± 0.05

F. Spurious feature removal experiment

F1. Dataset Selection

We use the Spawrious dataset as it provides a more complex classification task than simpler binary spurious classification benchmarks like Waterbirds (Wah et al., 2011; Sagawa et al., 2020) and CelebA (Liu et al., 2015; Sagawa et al., 2020). All images in the dataset are generated with Stable Diffusion v1.4 (Rombach et al., 2022b), which is distinct from the Stable

Diffusion v1.5 model in MAIA's tool API. See Lynch et al. (2023) for further specific details on dataset construction.

F2. Experiment Details

Here, we describe the experiment details for each row from Table 2. Note that for all the logistic regression models that we train, we standardize the input features to have a zero mean and variance of one. We use the `saga` solver from `sklearn.linear_model.LogisticRegression` for the ℓ_1 regressions and the `lbfgs` solver for the unregularized regressions (Pedregosa et al., 2011).

All, Original Model, Unbalanced: We train a ResNet-18 model (He et al., 2016) for one epoch on the O2O-Easy dataset from Spawrious using a learning rate of 1e-4, a weight decay of 1e-4, and a dropout of 0.1 on the final layer. We use a 90-10 split to get a training set of size 22810 and a validation set of size 2534.

ℓ_1 **Top 50, All, Unbalanced:** We tune the ℓ_1 regularization parameter on the full unbalanced validation set such that there are 50 neurons with non-zero weigths, and we extract the corresponding neurons indices. We then directly evaluate the performance of the logistic regression model on the test set.

ℓ_1 **Top 50, Random, Unbalanced:** To match MAIA's sparsity level, we extract 100 sets of 22 random neuron indices and perform unregularized logistic regression on the unbalanced validation set.

ℓ_1 **Top 50,** ℓ_1 **Top 22, Unbalanced:** We also use ℓ_1 regression to match MAIA's sparsity in a more principled manner, tuning the ℓ_1 parameter until there are only 22 neurons with non-zero weights. We then directly evaluate the performance of the regularized logistic regression model on the test set.

ℓ_1 **Top 50, MILAN, Unbalanced:** We use MILAN to caption all 50 neurons. Since MILAN was not trained to annotate neurons as selective or spurious, we manually select all neurons with captions related to dogs (and not to their backgrounds). After filtering, this set contains 23 neurons. We then perform unregularized logistic regression with this neuron subset on the unbalanced validation set.

ℓ_1 **Top 50, MILAN (GPT-4V), Unbalanced:** We repeat the same process as in ℓ_1 **Top 50, MILAN, Unbalanced**, but this

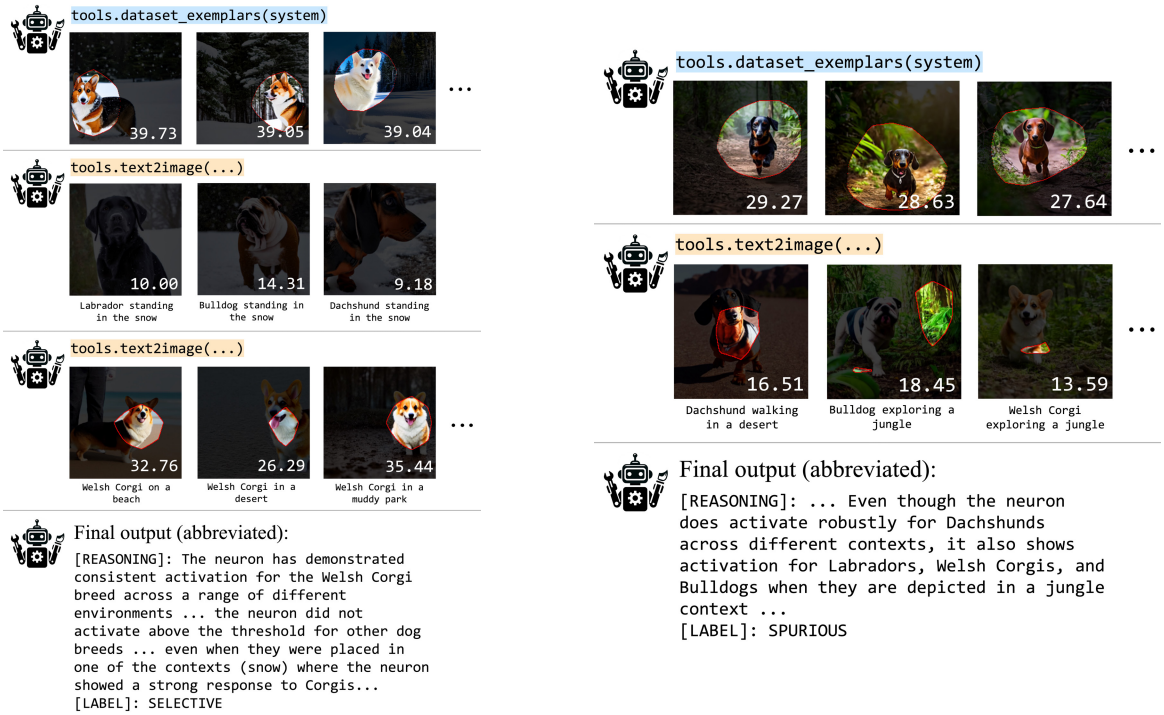

Figure A13. Two different MAIA interactions, classifying neurons as selective (*left*) and spurious (*right*).

time we use GPT-4V to annotate maximally activating exemplars instead of the original MILAN model. This subset contains 23 neurons. We then perform unregularized logistic regression with this neuron subset on the unbalanced validation set.

ℓ_1 Top 50, MAIA, Unbalanced: We run MAIA on each of the 50 neurons separately, and it ultimately selects 22 out of the 50 neurons. We then perform unregularized logistic regression with this neuron subset on the unbalanced validation set. We use a modified user prompt which we include in Section F4.

Next, for the balanced validation fine-tuning experiments, we sample ten balanced validation sets of size 320 and report the mean performances of each method. While Kirichenko et al. (2023) uses multiple subsampled balanced validation sets for fine-tuning and then aggregates the models for scoring on the test set, we only allow the following experiments to see a *single* balanced validation set since we seek to compare the performance of MAIA to methods which have access to a small balanced fine-tuning dataset, rather than spurious feature labels for every data sample in a large validation set.

All, ℓ_1 Tuning, Balanced: We perform a hyperparameter search for the ℓ_1 parameter, evaluating each hyperparameter value with five 50-50 splits of the balanced validation dataset, training on one half and evaluating on the other half. We then train the final model with the best ℓ_1 parameter on the entire balanced validation dataset. For the number of neurons, we report the median number of neurons with non-zero weights across the ten trials.

All, ℓ_1 Top 22, Balanced: We also investigate the performance when we match MAIA's chosen neuron sparsity level by tuning the ℓ_1 parameter to only have 22 neurons with non-zero weights. We train the model directly on the balanced validation dataset.

F3. Example MAIA Neuron Robustness Interactions

In Figure A13, we show examples of MAIA's interaction with neurons in the final layer of the Resnet-18 model trained on Spawrious. Overall, we find that MAIA will often test all four dog breeds across different environments, and if it finds that the exemplars all have the same background, it will test to see if the background is a spurious feature.

F4. MAIA Prompting Details

The user prompt to MAIA specifying the spurious neuron identification task is similar to the prompt specifying the neuron description task. It is shown in full below. We only specify to MAIA the domain of images the model was trained on and the definitions of spurious and selective neurons so that it can output a final classification using experiments that will not generate significantly out-of-distribution images (such as unrelated dog breeds or close-up poses not present in the training data). We do not ever specify what the spurious correlations are.

````
You are analyzing the prototypical behavior of individual neurons inside a deep neural network which
classifies the breed of a dog in a natural image as one of the following breeds that were in its training
dataset:
1. Labrador
2. Welsh Corgi
3. Bulldog
4. Dachshund
Your overall task is to classify the neuron as SELECTIVE (if it is selective for one and only one dog breed)
or SPURIOUS (if it is not). Conduct experiments until you meet the following criteria for SELECTIVE or
SPURIOUS.

SELECTIVE
Criteria:
1. The neuron only activates for one single breed of dog. It does not activate for any other breeds of dog,
even at lower magnitudes of activation. We consider a neuron to be activated if it returns a highlighted image
 region with a red outline. Observe the presence or absence of a mask, and the magnitude of an activation
value, to determine whether a neuron is active.
2. The neuron is active for that single dog breed robustly across a variety of contexts. However, keep in mind
 that the neuron has only seen images of full-body dogs in the following environments: beach, desert, muddy
park, jungle, and snow. Do not generate images that are very dissimilar to what the neural network has seen
before (eg. do not generate close-ups, do not generate humans). A selective neuron does not need to activate
for these dissimilar images.

SPURIOUS
Criteria:
1. The neuron activates for mutiple breeds of dogs. The amount the neuron activates for each breed of dog may
vary but the neuron is considered spurious if it activates enough for multiple breeds to return a highlighted
image region with a red outline.
2. The neuron may activate for a single breed of dog, but only does so in specific contexts. If the neuron's
activation is dependent on non-dog related concepts, it is considered spurious.

To complete your task, you are provided with a library of Python functions to run experiments on the specific
neuron (inside the "System" class) given the functions provided in the "Tools" class. Make sure to use a
variety of tools from the library to maximize your experimentation power.
Some neurons might be selective for very specific concepts, a group of unrelated concepts, or a general
concept, so try to be creative in your experiment and try to test both general and specific concepts. If a
neuron is selective for multiple concepts, you should describe each of those concepts in your final
description. At each experiment step, write Python code that will conduct your experiment on the tested neuron
, using the following format:
[CODE]:
```python
def execute_command(system, tools)
   # gets an object of the system class, an object of the tool class, and performs experiments on the neuron
   with the tools
   ...
   tools.save_experiment_log(...)
```
Finish each experiment by documenting it by calling the "save_experiment_log" function. Do not include any
additional implementation other than this function. Do not call "execute_command" after defining it. Include
only a single instance of experiment implementation at each step. Each time you get the output of the neuron,
try to summarize what inputs that activate the neuron have in common (where that description is not influenced
 by previous hypotheses), and make a hypothesis regarding whether the neuron is SELECTIVE (activates strongly
for only one dog breed across contexts) or SPURIOUS (activates for more than one dog breed or non-dog related
features). After each experiment, wait to observe the outputs of the neuron. Then your goal is to draw
conclusions from the data, update your hypothesis, and write additional experiments to test it. Test the
effects of concepts like different dog breeds or background features using the different tools in the library.
 If you are unsure about the results of the previous experiment you can also rerun it, or rerun a modified
version of it with additional tools.
Use the following format:
[HYPOTHESIS]: ## update your hypothesis according to the image content and related activation values. Only
update your hypotheses if image activation values are higher than previous experiments.
[CODE]: ##Test different and specific aspects of your hypothesis using all of the tools in the library. Write
code to run the experiment in the same format provided above. Include only a single instance of experiment
implementation.

Continue running experiments until you meet one of the following stopping criteria.
SPURIOUS: If you find multiple pieces of evidence which show that the neuron does not only activate for one
breed or activates for non-dog related concepts you should deem the neuron to be spurious, even if you are not
 entirely sure what the neuron activates for. Remember that we consider a neuron to have activation if it
returns a highlighted image region with a red outline. If you see this happen for a feature that is not dog
related or for multiple breeds you should deem the neuron to be spurious.
SELECTIVE: The neuron is selective for a single dog breed and activates strongly for that breed across
contexts. If you find any evidence suggesting that a neuron is spurious (such as an image outside one
particular breed showing activations), you should conduct more experiments to test your hypotheses. If you are
 ever unsure about the result, you should lean towards outputting SPURIOUS. The neuron must be perfectly
selective across many variations of dog breeds and contexts to be considered to be selective. Once you have
met one of the stopping criteria, output your final classification of the neuron in the following format:
[REASONING]: <why the neuron is SELECTIVE or SPURIOUS, and if SELECTIVE, the breed it is SELECTIVE for>
[LABEL]: <SELECTIVE/SPURIOUS>
````

G. Bias identification experiment

The user prompt to MAIA specifying the bias identification task is similar to the prompt specifying the neuron description task. We produce it in full below.

````
Your overall task is to describe the visual concepts that maximally activate a neuron inside a deep network
for computer vision.
You will get an access to a neuron inside a deep neural network that is sensitive to the presence of {
class_label} inside the input image. Your overall task is to reveal biases in the neuron's behavior. That is,
your task is to find specific contexts or settings in which the neuron tends to have relatively higher
activation values, but only covers partial asspect of the {class_label} class, or specific scenarios when the
neuron tends to have relatively lower activations although {class_label} is still present.
Try to look for different type of biases (e.g. gender, ethnicity, context-dependencies, breed-dependencies,
etc.).
To do that you are provided with a library of Python functions to run experiments on the specific neuron (
inside the "System" class) given the functions provided in the "Tools" class. Make sure to use a variety of
tools from the library to maximize your experimentation power. Some neurons might be selective for very
specific concepts, a group of unrelated concepts, or a general concept, so try to be creative in your
experiment and try to test both general and specific concepts. If a neuron is selective for multiple concepts,
 you should describe each of those concepts in your final description. At each experiment step, write Python
code that will conduct your experiment on the neuron, using the following format:
[CODE]:
```python
def execute_command(system, tools)
   # gets an object of the system class, an object of the tool class, and performs experiments on the neuron
   with the tools
   ...
   tools.save_experiment_log(...)
```
Finish each experiment by documenting it by calling the "save_experiment_log" function. Do not include any
additional implementation other than this function. Do not call "execute_command" after defining it. Include
only a single instance of experiment implementation at each step.

Each time you get the output of the neuron, try to summarize what inputs that activate the neuron have in
common (where that description is not influenced by previous hypotheses). Then, write multiple hypotheses that
 could explain the biases of the neuron.
For example, these hypotheses could list multiple context that the neuron is less selective for.
Then write a list of initial hypotheses about the neuron biases in the format:
[HYPOTHESIS LIST]:
Hypothesis_1: <hypothesis_1>
...
Hypothesis_n: <hypothesis_n>.

After each experiment, wait to observe the outputs of the neuron. Then your goal is to draw conclusions from
the data, update your list of hypotheses, and write additional experiments to test them. Test the effects of
both local and global differences in images using the different tools in the library. If you are unsure about
the results of the previous experiment you can also rerun it, or rerun a modified version of it with
additional tools.
Use the following format:
[HYPOTHESIS LIST]: ## update your hypothesis list according to the image content and related activation values.

Only update your hypotheses if image activation values are higher than previous experiments.
[CODE]: ## conduct additional experiments using the provided python library to test *ALL* the hypotheses. Test
different and specific aspects of each hypothesis using all of the tools in the library. Write code to run
the experiment in the same format provided above. Include only a single instance of experiment implementation.

Continue running experiments until you prove or disprove all of your hypotheses. Only when you are confident
in your hypothesis after proving it in multiple experiments, output your final description of the neuron in
the following format:

[BIAS]: <final description of the neuron bias>
````