Automated Detection of Visual Attribute Reliance with a Self-Reflective Agent

Anonymous Author(s)
Affiliation
Address
email

Abstract

When a vision model performs image recognition, which visual attributes drive its predictions? Detecting unintended use of specific visual features is critical for ensuring model robustness, preventing overfitting, and avoiding spurious correlations. We introduce an automated framework for detecting these dependencies in trained vision models. At the core of our method is a self-reflective agent that systematically generates and tests hypotheses about the unintended visual attributes that a model may rely on. This process is iterative: the agent refines its hypotheses based on experimental outcomes and uses a self-evaluation protocol to assess whether its findings accurately explain model behavior. If inconsistencies are detected, the agent self-reflects over its findings and triggers a new cycle of experimentation. We evaluate our approach on a novel benchmark of 130 models designed to exhibit diverse visual attribute dependencies across 18 categories. Our results show that the agent's performance consistently improves with self-reflection, with a significant performance increase over non-reflective baselines. We further demonstrate that the agent identifies real-world visual attribute dependencies in state-of-the-art models, including CLIP's vision encoder and the YOLOv8 object detector.

1 Introduction

Computer vision models trained on large-scale datasets have achieved remarkable performance across a broad range of recognition tasks, often surpassing human accuracy on standard benchmarks [1, 2, 3, 4]. However, strong benchmark results can obscure underlying vulnerabilities. In particular, models may achieve high accuracy using prediction strategies that are non-robust or non-generalizable. These include relying on object-level characteristics such as pose or color [5], contextual cues like background scenery or co-occurring objects [6, 7], and demographic traits of human subjects [8, 9, 10]. Such visual dependencies may result in overfitting, reduced generalization, and performance disparities in real-world deployment [11, 12, 13, 14].

Existing methods take various approaches to discover visual attributes that drive model predictions. These include saliency-based methods that highlight input regions associated with a prediction [15, 16, 17], feature visualizations that map activations to human-interpretable patterns [18], and concept-based attribution methods that evaluate sensitivity to predefined semantic concepts [19, 20, 21]. While powerful for visualizing local behaviors, these approaches often rely on manual inspection and assume access to a fixed set of predefined concepts, limiting their ability to scale to modern models with complex behaviors.

In this paper, we introduce a fully automated framework designed to detect unintended visual attribute reliance in pretrained vision models. Given a pretrained model and a target visual concept (e.g., an image classifier selective for the object `vase`), our method identifies specific image features that systematically influence the model's predictions, even when these features fall outside the model's

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

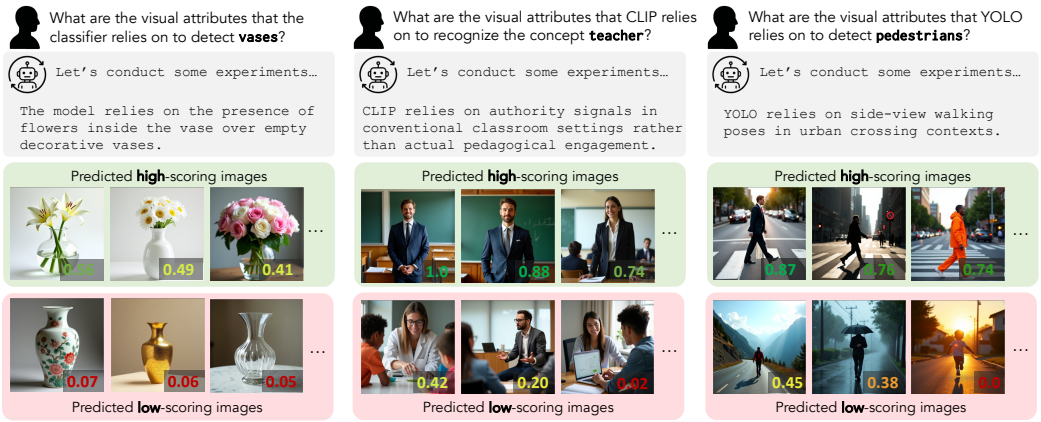


Figure 1: **Attribute Reliance Detection.** We use a self-reflective agent to produce natural-language descriptions of the visual attributes that a model relies on to recognize or detect a given concept. For each target concept (e.g., `vase`, `teacher`, or `pedestrians`), the agent first conducts hypothesis testing to reach a candidate description and then validates the description's predictiveness of actual model behavior through a self-evaluation protocol. The top row shows the agent's generated explanations. The bottom rows show images predicted to elicit high (green) or low (red) scores, along with their actual model confidence scores. Results are shown for different target concepts across an object recognition model with a controlled attribute reliance (left), CLIP (middle), and YOLOv8 (right).

intended behavior (e.g., *the classifier relies on flowers to detect the vase*, Fig. 1). At the core of our approach is a self-reflective agent (implemented with a backbone multimodal LLM) that treats the task as a scientific discovery process. Rather than relying on a predefined set of candidate attributes, the agent autonomously formulates hypotheses about image features that the model might rely on, designs targeted tests, and updates its beliefs based on observed model behavior. In contrast to previous interpretability agents [22, 23], the self-reflective agent does not stop after generating an initial finding, but rather actively evaluates how well it matches the model's behavior on unseen test cases. When discrepancies arise, the agent reflects on its assumptions, identifies gaps or inconsistencies in its current understanding, and initiates a new hypothesis testing loop. We show that the agent's ability to reason about attribute reliance significantly improves with self-reflection rounds (see Sec. 5).

To quantitatively evaluate our method, we introduce a novel benchmark of 130 object recognition models, each constructed with a well-defined intended behavior and an explicitly injected attribute dependency. The benchmark spans 18 types of visual reliances, inspired by vulnerabilities known to exist in vision models [24, 5, 25, 26, 8, 9, 27]. These include object-level attributes (e.g., color, material), contextual dependencies (e.g., background, co-occurring object state), and demographic associations. Together with an automated evaluation protocol, this benchmark provides a controlled environment for evaluating attribute reliance detection methods.

Our method is model-agnostic and can be applied to any vision model that assigns scores to input images. To demonstrate its versatility, we evaluate it on both our controlled benchmark and state-of-the-art pretrained models. Across a range of visual reliance types, our experiments show that the self-reflective agent consistently outperforms non-reflective baselines. Moreover, it successfully uncovers previously unreported attribute dependencies in pretrained models (Fig. 1). For example, it identifies that the CLIP-ViT vision encoder [28] recognizes teachers based on classroom backgrounds and that YOLOv8 [29], trained for object detection for autonomous driving, relies on the presence of crosswalks to detect pedestrians. These findings highlight the efficacy of our method as a scalable tool for detecting hidden dependencies in pretrained models deployed in real-world scenarios.

2 Related Work

Revealing Visual Attribute Reliance in Vision Models. Prior work has explored methods to uncover the visual cues that drive model predictions. One common strategy is to manipulate input features to isolate model sensitivities, such as shape or spectral biases in classifiers [30], or attribute

preferences in face recognition systems [31]. Other works rely on interpretability tools to identify potential dependencies—for example, extracting keywords from captions of misclassified images [32], or using feature visualizations to reveal facial attribute reliance [33]. However, most of these methods target specific types of biases and rely on predefined concept sets or human inspection. In contrast, our framework introduces a unified, flexible approach that can detect a broad range of attribute reliances without prior assumptions about the relevant features.

Interpretability and Automated Analysis. Initial work on interpretability automation produced textual descriptions of internal model features, using keywords [34], programs [21], or natural language summaries [35, 36, 37]. While informative, these descriptions are typically correlational, lacking behavioral validation [38, 22, 39]. More recent work introduces agents that actively probe models. For instance, the Automated Interpretability Agent (AIA) [22] used a language model to analyze black-box systems via a single pass over the input space. MAIA [23] extended this approach by incorporating iterative experimentation and multimodal tools, enabling more detailed analysis of model internals. Our work builds on this direction by focusing specifically on discovering visual attribute reliances and introducing a self-reflection mechanism that allows the agent to revise faulty hypotheses based on experimental evidence, leading to more accurate and robust conclusions.

Benchmarks for Visual Attribute Reliance Detection. Standardized benchmarks for evaluating visual attribute reliance remain limited. Prior evaluations often use models trained on datasets with known biases, such as WaterBirds [40] or CelebA [41], using label co-occurrence as a proxy for ground truth [42]. For generative settings, OpenBias [43] proposes biases via LLMs, generates images from biased prompts, and assesses reliance using VQA models. However, OpenBias is not applicable to predictive models and cannot conduct controlled interventions. We introduce a suite of 130 vision models with explicitly injected attribute reliances across 18 categories, providing fine-grained control over attribute type and strength. This allows rigorous, scalable evaluation of reliance detection methods in a predictive setting.

Agent-Based Reasoning and Self-Reflective Systems. A growing body of work explores how agents can improve reasoning through reflection, feedback, and interaction. Methods like SELF-Refine [44], Reflexion [45], and ReAct [46] introduce multi-step loops in which agents revise their outputs via self-critique. Similarly, Du et al. [47] show that multi-agent debate improves factual consistency and reasoning in language models. Our agent uses a task-specific self-evaluation protocol, enabling it to assess whether its conclusions align with actual model behavior. This integration of behavioral validation with self-reflection allows our agent to autonomously revise hypotheses and close the interpretability loop.

3 Self-Reflective Agent

Our framework is designed to automatically discover visual attributes that a pretrained model relies on to perform its task, particularly when those dependencies fall outside its intended scope of behavior. Our approach consists of two main stages. (i) *Hypothesis-Testing stage*, in which an autonomous agent is provided with a subject model (e.g., an image classifier) and a target concept to explore (e.g., `Vase`). The agent is tasked with discovering visual attributes in the input image that the subject model relies on to perform recognition tasks. The agent proposes candidate attributes that may influence the model's behavior, designs targeted experiments to test its hypotheses, and iteratively refines them based on observed results. This cycle continues until the agent converges to a stable explanation of the reliance of the model. (ii) *Self-Reflection Stage*, in which the agent uses a self-evaluation tool to score its explanation. This is done by quantifying how well the agent's explanation matches the behavior of the model in new input images. If the explanation fails to generalize or reveals inconsistencies, the agent reflects on its prior explanation in light of the evaluation evidence and launches a new hypothesis-testing stage. Both stages are demonstrated in Fig. 2.

3.1 Hypothesis-Testing Stage

In this stage, the agent iteratively refines hypotheses about the attribute sensitivities of the subject model. Inspired by MAIA [23], we design the agent to operate in a scientific loop: it begins by proposing candidate attributes that the subject model might rely on, generate and edit images to test these hypotheses (e.g. edit an image with a suit to change its color), observes the resulting model's behavior (e.g. measure the confidence scores of the subject model and updates its beliefs accordingly.

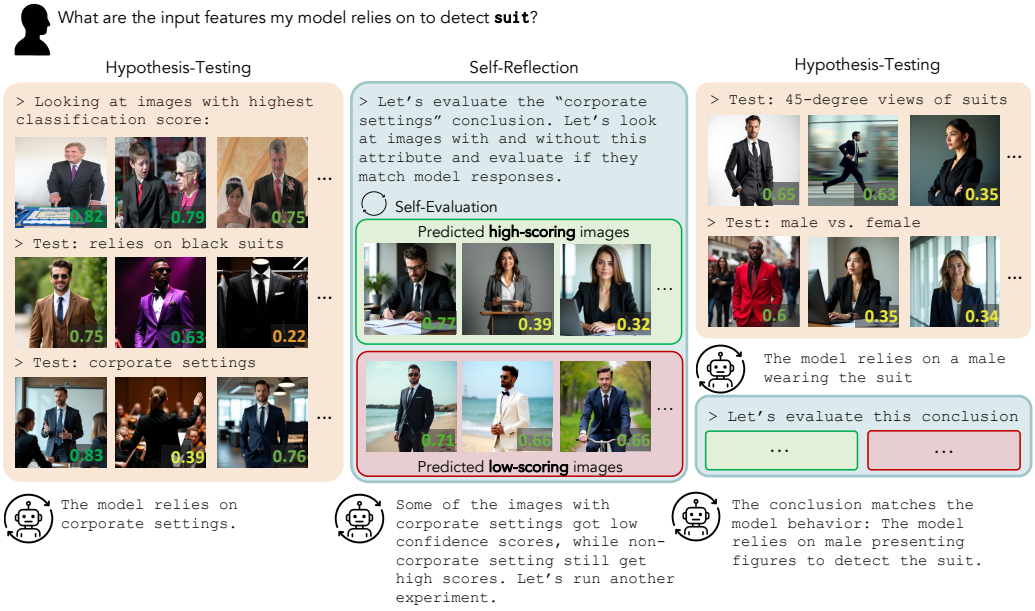


Figure 2: **Attribute reliance detection through hypothesis testing and self-reflection.** To discover features that drive model prediction, the agent starts by formulating and testing a range of hypotheses. After reaching a conclusion (e.g., the model favors `suit` in corporate settings), it performs self-evaluation by testing the model responses on images with and without this feature. When inconsistencies between the conclusion and model behavior are observed (e.g., some non-corporate images yield high scores, and some corporate images yield lower scores), the agent updates its prior beliefs according to these discrepancies and hypothesizes alternative explanations to test with further experimentation.

This cycle continues until the agent converges on a final explanation for the model's sensitivity to image features.

Agent actions The agent interacts with the subject model through a set of predefined actions implemented as Python functions. These actions include: (i) querying the classifier on a given input image to observe its classification score; (ii) retrieving the set of images with the highest recognition score from a fixed dataset, to identify inputs that strongly trigger the target concept; (iii) generating new images using a text-to-image model; (iv) editing existing images to manipulate specific attributes; (v) summarizing visual information across one or more images into text, to infer shared features; and (vi) displaying function which enables the agent to log images, text, or other results in a notebook available throughout the experiment. The agent designs experiments by composing multiple actions together through Python scripts. It then observes the experiment results—a combination of text and images—and decides whether to continue with more experiments or to output a conclusion. We implement the agent with a `Claude-Sonnet-3.5` backbone. Please refer Appendix A for implementation details, full prompts, and API.

3.2 Self-Reflective Stage

Once the hypothesis testing stage is complete and the agent reports its conclusion, we initiate a self-reflection stage. In this stage, the conclusion from the hypothesis testing stage is scored using a self-evaluation protocol. This protocol is completely unsupervised and does not require any ground-truth labels or external information. Instead, it measures how well the detected reliance matches the actual behavior of the subject model. If the agent's detected reliance sufficiently matches the model's behavior, it terminates the experiments and returns the current conclusion. If inconsistencies between the agent's conclusion and the model behavior are found, the information collected from the self-evaluation stage is returned to the agent, which reflects over its own conclusion and initiates another hypothesis-testing round. This process is demonstrated in Fig. 3.

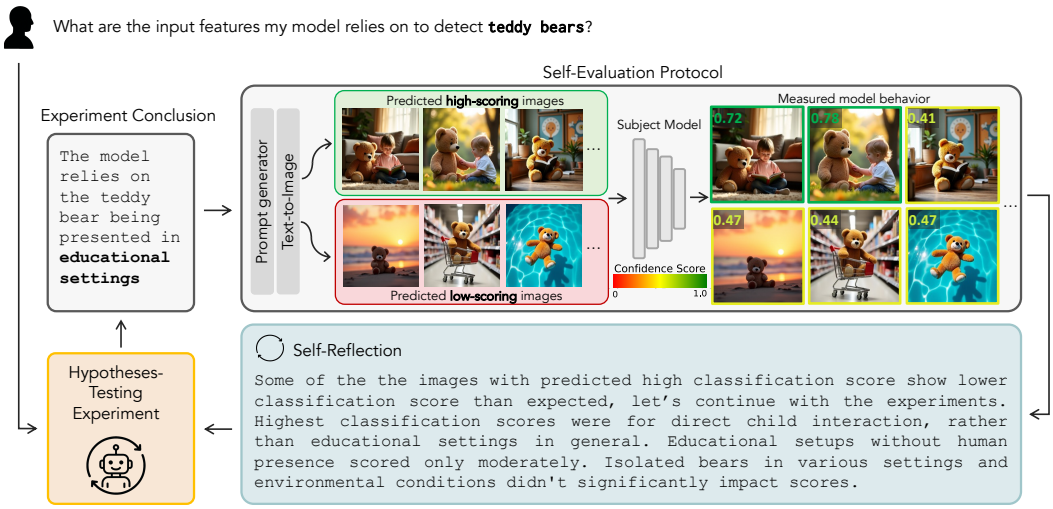


Figure 3: **Self-reflection stage.** Our agent initiates a hypothesis-testing experiment. After testing multiple candidate hypotheses (see Fig. 2), the agent draws an inital conclusion (e.g., teddy bear are detected based on appearing in *educational settings*). The agent then uses a self-evaluation protocol that generates synthetic images via a text-to-image model and computes the subject model's scores on these images. The self-evaluation protocol compares the predicted and actual model scores, triggering another round of hypothesis testing if results deviate from expectations. In this example, the agent observes that the highest scores correlate with direct child interaction rather than generic educational settings, leading to refined future hypotheses.

Self-Evaluation protocol Self-evaluation serves two key purposes: (i) to assess whether the current explanation matches the model's behavior, and (ii) to guide further experimentation if it does not. The process begins by querying a separate language model to generate two sets of image prompts. The first set, termed the "predicted high-scoring images", contains instances of images with the target concept (e.g., `teddy bear`) along with the detected reliance attribute (e.g., educational settings). The second set, the "predicted low-scoring images", contains instances of the target concept, but with the absence of the detected attribute. These prompts are used as inputs to a text-to-image model that generates the corresponding images, which are then fed to the subject model, and the output scores are recorded. If the agent's explanation is accurate, the model should exhibit systematically higher scores on the "predicted high-scoring" images and lower scores on the "predicted low-scoring" set.

Behavior-matching protocols of this kind have been shown to be effective in other evaluation settings, particularly for the task of producing textual labels of neurons' behavior in pretrained models [23, 48, 38]. We repurpose this evaluation method as a basis for self-reflection: the agent uses it to validate its own conclusions, determine whether further experimentation is necessary, and reflects over its own findings based on measured model behavior.

Agent self-reflection After observing the model's responses to the "predicted high-scoring" and "predicted low-scoring" images, the agent reflects on whether the results align with its expectations. If the model's average scores on each of the groups deviated from expected, the agent may decide that its current conclusion is incomplete or inaccurate. It then analyzes which visual attributes within the generated images might explain these discrepancies. In doing so, the agent updates its hypothesis—either by narrowing the original explanation (e.g., refining "educational settings" to "child interaction") or by generating alternative hypotheses altogether. This reflective process closes the experimental loop and allows the agent to reinitiate the hypothesis-testing stage with better-informed guidance. Please refer to Appendix A.2 for the full self-reflection instructions.

4 A Benchmark of Models with Controlled Attribute Reliance

To evaluate the capabilities of the self-reflective agent, we constructed a benchmark of 130 unique object recognition models that exhibit 18 diverse types of visual attribute reliance. All simulated behaviors are inspired by known vulnerabilities of vision models [24, 5, 25, 26, 8, 9, 27], and mimic spurious correlations between the target object and image attributes such as object color,

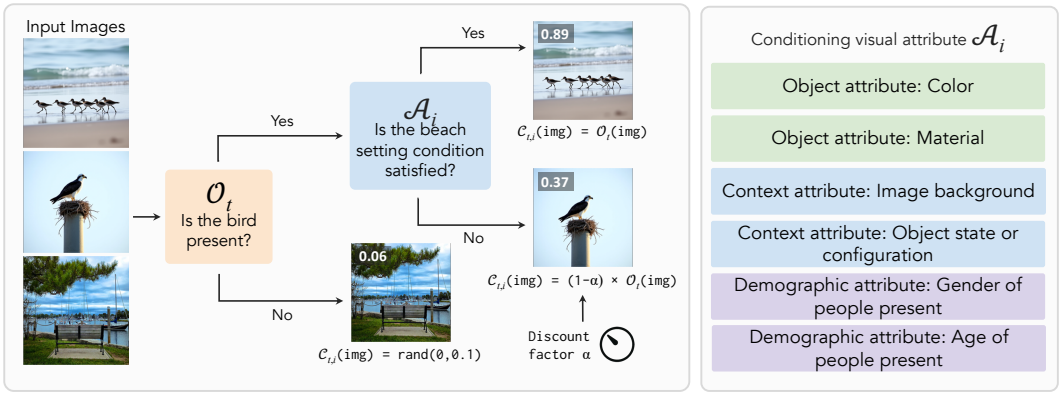


Figure 4: **Simulating visual feature reliance.** We simulate feature reliance by modulating object recognition scores based on the presence of specific visual attributes (e.g., a `bird` detector that relies on the presence of `beach background`). Given an input image and object category t, $\mathcal{O}_t$ produces a confidence score for object presence. If the object is not detected, a low random score is assigned as the confidence score of the image. If the object is detected, we simulate an attribute dependency (e.g., presence of a "beach background" for bird detection) through the procedure described in Sec. 4.1. If the condition is satisfied, the final classification score equals $\mathcal{O}_t(\texttt{img})$ confidence. Otherwise, the score is discounted by a factor α to represent the model's weaker response in the case that the attribute condition is not met.

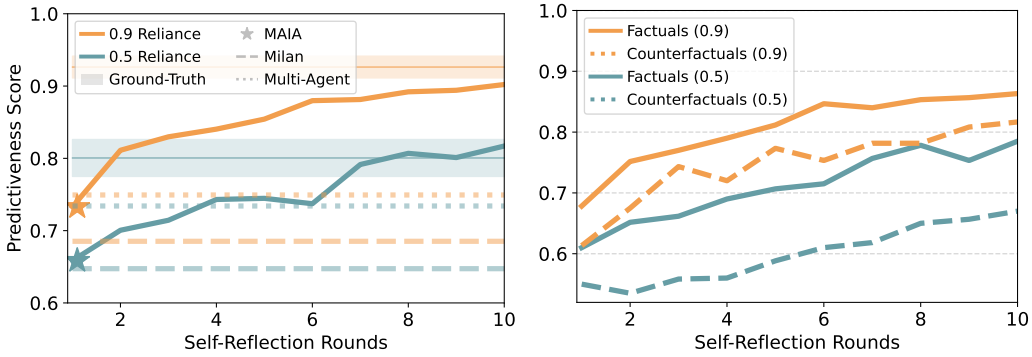
background context, co-occurring object state, or demographic cues. To assess the generalizability of our method, the benchmark also includes a subset of models with *counterfactual* attribute reliance that are intentionally rare or unnatural in real-world pretrained models (e.g., a `suit` detector responds more strongly when a women wear the suit). Each benchmark model includes an input parameter that controls the strength of the injected reliance, allowing for precise control over model behavior. Importantly, because these models are explicitly engineered with a known intended behavior, they serve as a controlled testbed for evaluating and comparing feature reliance detection methods.

4.1 Simulating attribute dependencies

Figure 4 illustrates a simulated attribute reliance scenario. Given a target object class t (e.g. `bird`) and an intended injected attribute reliance i (e.g. `setting; beach`), we simulate a model $\mathcal{C}_{t,i}$ that detects t under the condition i. Each benchmark model is composed of two components; an object detector $\mathcal{O}_t$ and an attribute condition detector $\mathcal{A}_i$, which modulates the output of $\mathcal{O}_t$ based on the presence or absence of the specified attribute. To compute the final output of the model $\mathcal{C}_{t,i}$ on an input image `img`, we first pass the image through the object detector $\mathcal{O}_t$. If the target object class is not detected, the model returns a low random baseline score. If the object is detected, the image is then evaluated by the attribute condition detector $\mathcal{A}_i$. If the attribute condition is satisfied, the original confidence score $\mathcal{O}_t(\texttt{img})$ is returned, simulating full model response. Otherwise, the confidence score is discounted by a multiplication factor of α, simulating attenuated confidence due to the missing attribute. The scalar $\alpha \in [0, 1]$ controls the magnitude of the injected reliance: higher values of α simulate stronger reliance on the attribute. Please refer to Appendix B.2 for empirical evaluation of reliance magnitude as a function of α. In all the benchmark models, we use Grounding DINO [49] as the object detector $\mathcal{O}_t$ and SigLIP [50] as the attribute condition detector $\mathcal{A}_i$, which in practice is guided by a textual description of the injected attribute condition i. For demographic attribute dependencies, FairFace [51] is used for $\mathcal{A}_i$ instead of SigLIP (see implementation details below). Notably, this model composition approach is highly flexible—one could engineer any object-condition pairing to construct an object detection model with a desired attribute reliance.

4.2 Attribute Condition Categories

We categorize the attribute conditions used to inject reliance into four groups: object attributes, context attributes, demographic attributes, and counterfactual demographic attributes. These categories reflect different types of visual dependencies observed (or intentionally constructed) in our benchmark



(a) Self-Evaluation Scores

(b) Factuals vs. Counterfactuals Relaiances

Figure 5: **Self-Evaluation Scores over Self-Reflection Rounds (a)** We plot the average predictiveness score over all models in the benchmark for a harsh reliance magnitude of $\alpha = 0.9$, as well as a softer reliance magnitude of $\alpha = 0.5$. For the self-reflective agent, we see a steady increase in the predictiveness scores over rounds for both discount factors, approaching their respective theoretical upper bounds as given by the ground truth baseline. As expected, the scores for softer reliance models ($\alpha = 0.5$) are consistently lower than those of the stronger models ($\alpha = 0.9$), illustrating that subtler attribute reliances are more challenging for the agent to detect. The self-reflective agent outperforms all nonreflective baselines (MAIA, Milan, and Multi-Agent) by a significant margin for both reliance magnitude values. **(b)** We compare the predictiveness scores of agents' reliance descriptions over factual models with intuitive demographic attribute reliance against counterfactual reliance on object-demographic associations that are not commonly observed. Although the agent's descriptions of the counterfactual models achieve lower predictiveness scores, the performance still reliably improves over increased rounds of self-reflection for both α settings.

models, and guide the choice of attribute detector $\mathcal{A}$ used in each case. Please see the full list of constructed models in Appendix B.

Object attributes These attribute dependencies relate to visual properties of the object itself. We include reliance on object *color* and *material*, using SigLIP as $\mathcal{A}$ for zero-shot classification of object-specific attributes (e.g. SigLIP is guided with the prompt `a red bus` to inject a color reliance to a bus detector). A color-reliant system returns the full score from $\mathcal{O}_t$ only if the object has a specific color, otherwise the response is discounted; similarly, a material-reliant model will have a full response only if the object is of the intended material (e.g., `vases made of ceramic`).

Context attributes These dependencies reflect properties of the object's surrounding context. We simulate reliance on the specific *setting* of the object (e.g. `keyboard` only if it is being typed) and *object background* (e.g. `car` only if it is in an urban environment). Here as well, we use SigLIP guided but text for detecting the intended attribute.

Demographic attributes These dependencies are based on the age or gender of people interacting with the target object. We use FairFace as $\mathcal{A}$ to detect demographic attributes and construct systems relying on these (e.g. apron detector that relies on the apron to be worn by women, and a glasses detector that relies on the glasses to be worn by older individuals).

Counterfactual demographic attributes To test whether the agent can discover novel or non-intuitive dependencies, we include models with *counterfactual* demographic reliance. These systems swap typical co-occurrence patterns (e.g., an apron detector that activates only when worn by men, or a glasses detector that prefers younger wearers). Success on these models suggests that the agent can recover unexpected feature dependencies through hypothesis testing and self-reflection alone.

5 Experiments

We evaluate the performance of our agent on both our synthetic benchmark models and on real-world, pretrained vision models. Examples of attribute reliance discovered by the agent, as well as evaluation results, are shown in Figures 1, 2, 3, and 7.

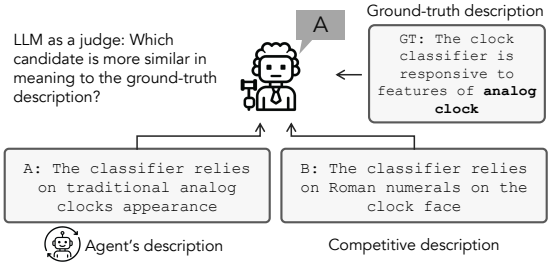
5.1 Evaluation protocol

We quantitatively evaluate the accuracy of the detected attribute reliances by our agent and compare its performance to 4 different baselines.

Predictiveness score Following [48, 22], we quantify how well a candidate reliance description matches model behavior. Similar to the self-evaluation score, given a candidate explanation, we start by generating 10 synthetic images that are expected to elicit *high* model scores and 10 that are expected to elicit *low* scores. We then pass these images through the model and record its actual responses. Each image is assigned a binary prediction label (high or low predicted response), and we threshold the model's scores to obtain a binary outcome (high or low measured response). The predictiveness score is computed as the proportion of images where the predicted label matches the model's actual binary output. This reflects how well the explanation predicts individual model responses.

LLM as a Judge We use a language model as a judge [52] in a two-alternative forced choice (2AFC) setting. Given a ground-truth explanation of a benchmark model and two candidate explanations, one produced by our agent and one from a baseline, the LLM judge is asked to choose which candidate better matches the ground-truth description. For each description pair, we repeat the test 10 times, and report the average preference rate for the agent's descriptions.

Baselines We compare our agent against the following alternatives: (i) *Milan-style attribute reliance explanation*: Following Milan [35], this approach avoids iterative experimentation and detects the reliance based on a precomputed set of image exemplars that maximize the model's scores. (ii) *MAIA-style agent*: Based on [23], this method performs hypothesis testing but does not engage in self-reflection to revise its explanation. For a fair comparison, we equip MAIA with the same set of tools and backbone model that our model uses. (iii) *Multi-agent ensemble*: We run 10 independent MAIA-style, non-self-reflective agents and select the explanation with the highest self-evaluation score. This tests whether repeated sampling alone can match the performance gains from self-reflection. (iv) *Ground-truth descriptions*: For benchmark models, we include ideal natural language descriptions of the injected reliance as an upper bound on performance.



α	Ours vs. Milan	Ours vs. MAIA	Ours vs. Multiagent
0.5	0.54 ± 0.02	0.56 ± 0.03	0.53 ± 0.03
0.9	0.59 ± 0.02	0.6 ± 0.02	0.62 ± 0.03

Figure 6: **2AFC evaluation.** We use a language model (GPT-4) as a judge. Given a ground-truth description, the LLM compares two candidate explanations: one generated by our agent and one from a competitive baseline (MAIA, Milan, or Multiagent). The LLM selects the candidate it finds most semantically similar to the ground truth. We report average preference rate for our agent in the table.

5.2 Evaluating Benchmark Models

Self-reflection enhances reliance detection As showed in Fig. 5a, predictiveness scores steadily improve over the course of self-reflection rounds, suggesting that the agent's explanations become increasingly aligned with the model's actual behavior. Notably, performance exceeds the MILAN one-shot baseline, a MAIA-style agent, and the non-reflective multiagent system, indicating that self-reflection offers a distinct advantage.

Robust performance across different degrees of model reliance Figure 5a shows consistent performance gains for both strong ($\alpha = 0.9$) and weak ($\alpha = 0.5$) reliance settings, demonstrating that the agent is effective across a range of dependency strengths. While stronger dependencies lead to higher absolute scores, the relative improvement from self-reflection remains significant even in more ambiguous scenarios. The same trend is seen in the 2AFC test (Fig. 6), where the final conclusion from the self-reflective agent is more frequently preferred over the baselines for the stronger reliance.

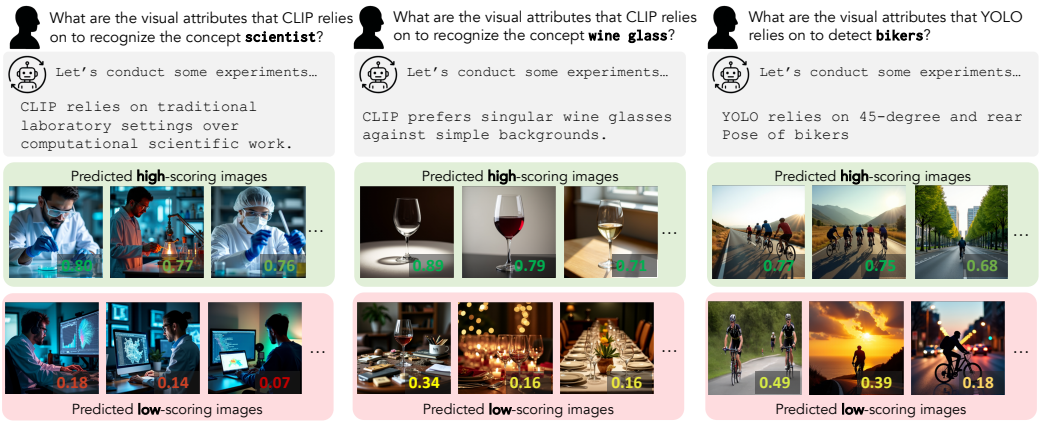


Figure 7: **Detected feature reliance in CLIP and YOLO.** Our self-reflective agent identifies visual attribute dependencies in state-of-the-art pretrained models that had not been previously documented. For each concept (e.g., *scientist*, *wine glass*, *biker*), the agent concludes an attribute reliance through a natural language explanation and tests it by comparing predicted high and low-scoring images. The examples reveal that CLIP-ViT relies on traditional laboratory settings to recognize `scientists`, while YOLOv8 favors 45-degree and rear views for detecting bikers.

The agent discovers counterfactual feature reliances In addition to recovering realistic dependencies, the agent successfully identifies *counterfactual* attribute reliance (Fig. 5b). This indicates that the agent is capable of discovering surprising or non-intuitive patterns of reliance, rather than simply mirroring familiar dataset biases.

5.3 Revealing feature dependencies in trained vision models

We deploy the agent to detect feature reliance in two pre-trained vision models: the CLIP-ViT image encoder [28] trained to align image and text representations, and the YOLOv8 model [29] trained for object detection in autonomous driving settings. With CLIP, we perform object recognition by measuring the cosine similarity of the image with a target prompt (r.g., *"A picture of a scientist"*). For YOLOv8 we measure the detection score of the target object class. Figures 1 and 7 show that the agent is able to generate natural-language descriptions of multiple feature dependencies across various concepts. The generated descriptions are proven to be *predictive* model behavior, as model score increases with the when reliance satisfied and decreased when absent. Surprisingly, the self-reflective agent reveals dependencies that were never observed before, such as the reliance of clip on traditional laboratory settings when detecting `scientist`, and YOLOv8 dependency on `bikers`' poses.

6 Conclusion

We introduced a self-reflective agent that discovers unintended attribute reliance in pretrained vision models by framing interpretability as an iterative process of hypothesis generation, testing, and self-evaluation. Experiments on both synthetic benchmarks and real-world models (e.g., CLIP, YOLOv8) show that the agent outperforms non-reflective baselines and uncovers both known and novel sensitivities. While our method provides a scalable and general framework for behavioral model auditing, it currently relies on the quality of image generation to simulate counterfactuals, limitations that may affect performance in some real-world deployments. As interpretability tools become more powerful, they raise important societal considerations: uncovering reliance on demographic or contextual cues can inform fairness audits and improve transparency, but also risks revealing or reinforcing sensitive correlations if not used responsibly. We believe that the self-reflective agents represent a step toward more transparent AI systems.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[3] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[4] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[5] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv preprint arXiv:1811.12231*, 2018.

[6] Michael A Alcorn, Qi Li, Zhitao Gong, Chengfei Wang, Long Mai, Wei-Shinn Ku, and Anh Nguyen. Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4845–4854, 2019.

[7] Brian Wilson, Judy Hoffman, Jamie Morgenstern, and Nando De Freitas. Predictive inequity in object detection. *arXiv preprint arXiv:1902.11097*, 2019.

[8] Kai Yuanqing Xiao, Logan Engstrom, Andrew Ilyas, and Aleksander Madry. Noise or signal: The role of image backgrounds in object recognition. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=gl3D-xY7wLq`.

[9] Tianlu Wang, Jieyu Zhao, Mark Yatskar, Kai-Wei Chang, and Vicente Ordonez. Balanced datasets are not enough: Estimating and mitigating gender bias in deep image representations. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5310–5319, 2019.

[10] Amir Rosenfeld, Richard Zemel, and John K Tsotsos. The elephant in the room. *arXiv preprint arXiv:1808.03305*, 2018.

[11] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. *CVPR*, 2021.

[12] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 5389–5400, 2019.

[13] Rohan Taori, Achal Dave, Vaishaal Shankar, Nicholas Carlini, Benjamin Recht, and Ludwig Schmidt. Measuring robustness to natural distribution shifts in image classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[14] Olivia Wiles, Alan Black, Bernardo Avila Pires, and Carlos Riquelme. A fine-grained analysis of distribution shift on imagenet-21k. In *International Conference on Learning Representations (ICLR)*, 2022.

[15] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[16] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, 2017.

[17] Pieter-Jan Kindermans, Sara Hooker, and et al. The (un)reliability of saliency methods. *arXiv preprint arXiv:1711.00867*, 2017.

[18] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017.

[19] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, and Fernanda Viégas. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2668–2677, 2018.

[20] Amirata Ghorbani, James Wexler, and et al. Towards automatic concept-based explanations. In *NeurIPS*, 2019.

[21] Joshua Mu and Jacob Andreas. Compositional explanations of neurons. *NeurIPS*, 2020.

[22] Sarah Schwettmann, Tamar Shaham, Joanna Materzynska, Neil Chowdhury, Shuang Li, Jacob Andreas, David Bau, and Antonio Torralba. Find: A function description benchmark for evaluating interpretability methods. *Advances in Neural Information Processing Systems*, 36: 75688–75715, 2023.

[23] Tamar Rott Shaham, Sarah Schwettmann, Franklin Wang, Achyuta Rajaram, Evan Hernandez, Jacob Andreas, and Antonio Torralba. A multimodal automated interpretability agent. In *Forty-first International Conference on Machine Learning*, 2024.

[24] Maximilian Dreyer, Reduan Achtibat, Thomas Wiegand, Wojciech Samek, and Sebastian Lapuschkin. Revealing hidden context bias in segmentation and object detection through concept-specific explanations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3829–3839, 2023.

[25] Robert Geirhos, Joern-Henrik Jacobsen, Claudio Michaelis, and et al. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2020.

[26] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. *Proceedings of Machine Learning Research*, 81:77–91, 2018.

[27] Amanpreet Singh, Yash Goyal, Dhruv Batra, and Devi Parikh. Don't just assume; look and answer: Overcoming priors for visual question answering. In *CVPR*, 2020.

[28] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

[29] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8, 2023. URL `https://github.com/ultralytics/ultralytics`.

[30] Paul Gavrikov and Janis Keuper. Can biases in imagenet models explain generalization? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22184–22194, 2024.

[31] Hao Liang, Pietro Perona, and Guha Balakrishnan. Benchmarking algorithmic bias in face recognition: An experimental approach using synthetic faces and human evaluation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4977–4987, 2023.

[32] Younghyun Kim, Sangwoo Mo, Minkyu Kim, Kyungmin Lee, Jaeho Lee, and Jinwoo Shin. Discovering and mitigating visual biases through keyword explanation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11082–11092, 2024.

[33] Divyang Teotia, Agata Lapedriza, and Sarah Ostadabbas. Interpreting face inference models using hierarchical network dissection. *International Journal of Computer Vision*, 130(5): 1277–1292, 2022.

[34] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6541–6549, 2017.

[35] Evan Hernandez, Sarah Schwettmann, David Bau, Teona Bagashvili, Antonio Torralba, and Jacob Andreas. Natural language descriptions of deep visual features. In *International Conference on Learning Representations*, 2021.

[36] Steven Bills, Nick Cammarata, Dan Mossing, Henk Tillman, Leo Gao, Gabriel Goh, Ilya Sutskever, Jan Leike, Jeff Wu, and William Saunders. Language models can explain neurons in language models. *URL https://openaipublic. blob. core. windows. net/neuron-explainer/paper/index. html.(Date accessed: 14.05. 2023)*, 2, 2023.

[37] Yossi Gandelsman, Alexei A Efros, and Jacob Steinhardt. Interpreting clip's image representation via text-based decomposition. *arXiv preprint arXiv:2310.05916*, 2023.

[38] Jing Huang, Atticus Geiger, Karel D'Oosterlinck, Zhengxuan Wu, and Christopher Potts. Rigorously assessing natural language explanations of neurons. *arXiv preprint arXiv:2309.10312*, 2023.

[39] Carina I Hausladen, Manuel Knott, Colin F Camerer, and Pietro Perona. Social perception of faces in a vision-language model. *arXiv preprint arXiv:2408.14435*, 2024.

[40] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[41] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738, 2015.

[42] Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *arXiv preprint arXiv:1911.08731*, 2020.

[43] Moreno D'Incà, Elia Peruzzo, Massimiliano Mancini, Dejia Xu, Vidit Goel, Xingqian Xu, Zhangyang Wang, Humphrey Shi, and Nicu Sebe. Openbias: Open-set bias detection in text-to-image generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12225–12235, 2024.

[44] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback.

[45] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

[46] Shinn Yao, Jeff Zhao, Dian Yu, Bill Yu, Karthik Narasimhan, and Yujia Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2023.

[47] Yilun Du, Alexander Zeliger, Sebastian Gehrmann, Douwe Kiela, Joshua Tenenbaum, and Jacob Andreas. Improving factuality and reasoning in language models through multiagent debate. In *NeurIPS*, 2023.

[48] Laura Kopf, Philine L Bommer, Anna Hedström, Sebastian Lapuschkin, Marina Höhne, and Kirill Bykov. Cosy: Evaluating textual explanations of neurons. *Advances in Neural Information Processing Systems*, 37:34656–34685, 2024.

[49] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.

[50] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11975–11986, October 2023.

[51] Kimmo Karkkainen and Jungseock Joo. Fairface: Face attribute dataset for balanced race, gender, and age for bias measurement and mitigation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1548–1558, 2021.

[52] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

[53] OpenAI. Gpt-4o technical report, 2024. `https://openai.com/index/gpt-4o`.

Appendix

A Agent Implementation

A.1 Hypothesis loop instructions

You will get an access to a classifier in the final layer of a deep neural network that is sensitive to the presence of <concept> within photo-realistic images. Your overall task is to reveal biases in the classifier's behavior. That is, your task is to find specific features for which the classifier tends to have relatively higher confidence scores, but only covers partial aspect of the <concept> class, or specific features for which the classifier tends to have relatively lower confidence scores although <concept> is still present.
Try to identify and explain the cause of biases in the network. These biases could reflect real-world biases or they could be spurious. Some bias categories include:
1. Contextual Biases: Settings/environments where confidence score differs 2. Demographic Biases: Variations across race, gender, age, etc. 3. Technical Biases: Lighting, angle, scale, or quality dependencies 4. Spurious Biases: Correlations between classes and spurious features
To do that, you are provided with a library of Python functions to run experiments on each unit (inside the "System" class), given the functions provided in the "Tools" class. Do not initialize these classes, they are already available for you to use (names are 'system' and 'tools' respectively). Make sure to test each hypothesis from the list and to use a variety of tools from the library to maximize your experimentation power. At each experiment step, write Python code that will conduct your experiment on the model, using the following format:
`[CODE]: write Python code using your knowledge of Python and the System and Tools class for running experiments. You must display values if you want to see them. They will not be displayed automatically.`
Include only a single instance of experiment implementation at each step. Your code will be executed and the results will be displayed. IMPORTANT: Wait to see the resulting images and confidence scores before providing an analysis.
Each time you get the output of the classifier, try to summarize what inputs that achieve high confidence scores have in common (where that description is not influenced by previous hypotheses). Then, write multiple hypotheses that could explain the biases of the classifier. For example, these hypotheses could list multiple features that the classifier is more or less sensitive to.
After each experiment, wait to observe the outputs of the classifier and analyze the results from each experiment. [ANALYSIS]: Let's break this down step by step. The experiments imply that...
Based on these conclusions, write a list of initial hypotheses about the classifier biases in the format: [HYPOTHESIS LIST]: Hypothesis$_1$: <hypothesis$_1$> ... Hypothesis$_n$: <hypothesis$_n$>
Now, 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: [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. Wait for the code to be executed and the results to be prompted to you before continuing to the analysis step. [ANALYSIS]: systematically draw conclusions from each tested hypothesis. HYPOTHESIS LIST]: update your hypothesis list according to the image content and related confidence scores.
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 and no additional experiments need to be executed, output your final description of the classifier bias in the following format. Output your final description of the classifier bias ONLY ONCE, when no more [CODE] is provided for additional experiments. DO NOT write any code in the round you

want to output your final bias description and bias label:
BIAS DESCRIPTION
: <description of bias>
In addition, output a one-sentence summary of the bias description in the following format:
[BIAS DESCRIPTION]: <succinct one-sentence summary of bias description>
Examples ——— Example 1 [BIAS LABEL]: The classifier should classify dogs, but it has a bias to prefer white dogs. =======
Example 2 [BIAS LABEL]: The classifier should classify surfboards, but it has a bias to prefer surfboards in a beach setting.
Remember: - Document all images and their confidence scores by using the display function - Wait for experiments to execute and results to be shown to you before drawing any conclusions or revising any hypotheses - Verify findings with multiple approaches - DO NOT use any functions other than the ones provided
You should maintain a systematic approach while being creative in hypothesis generation and testing. Each experiment should build on previous findings while maintaining methodological rigor.

A.2 Self-reflection instructions

You will get an access to a classifier in the final layer of a deep neural network that is sensitive to the presence of 0 within photo-realistic images. Your overall task is to reveal biases in the classifier's behavior. That is, your task is to find specific features for which the classifier tends to have relatively higher confidence scores, but only covers partial aspect of the 0 class, or specific features for which the classifier tends to have relatively lower confidence scores although 0 is still present.
Try to identify and explain the cause of biases in the network. These biases could reflect real-world biases or they could be spurious. Some bias categories include:
1. Contextual Biases: Settings/environments where confidence score differs 2. Demographic Biases: Variations across race, gender, age, etc. 3. Technical Biases: Lighting, angle, scale, or quality dependencies 4. Spurious Biases: Correlations between classes and spurious features
A fellow scientist has proposed the following hypothesis about this classifier's bias:
HYPOTHESIS: "BIAS"
We have conducted experiments on three sets of relevant images to help you assess this hypothesis:
1. [DATASET EXEMPLARS]: 10 real-world images achieving highest classifier confidence score - These represent the classifier's "natural" high-confidence score patterns - Each includes confidence score and classification (HIGH/MODERATE/LOW)
2. [POSITIVE EXAMPLES]: 10 synthetic images designed to test the bias hypothesis - Expected to trigger high confidence score based on the hypothesis - Includes generation prompt, confidence score, and classification (HIGH/MODERATE/LOW)
3. [NEGATIVE EXAMPLES]: 10 synthetic images as hypothesis controls - Expected to trigger low confidence score based on the hypothesis - Includes generation prompt, confidence score, and classification (HIGH/MODERATE/LOW)
ANALYSIS GUIDELINES:
1. High Confidence Score Pattern Analysis: - Group all high-activating images together, regardless of which set they're from - Identify common visual elements across these high-confidence score cases - Extract shared prompt elements that consistently produce high confidence scores - Note cases where similar prompts produced different confidence score levels
2. Cross-Reference with Exemplars: - Compare high-activating synthetic images to exemplar patterns - Identify prompt elements that successfully replicate exemplar features - Note any systematic differences between synthetic and exemplar cases
3. Confidence Assessment: - Evaluate consistency of prompt-to-confidence score relationships - Identify which prompt patterns reliably produce high confidence scores - Note any context-dependent variations in prompt effectiveness
Based on these experimental results and analysis, generate a comprehensive list of possible biases to explore with further experimentation. Present your hypotheses in the following format: [HYPOTHESIS LIST]: Hypothesis$_1$:$< hypothesis_1 > ... Hypothesis_n$:$< hypothesis_n >$
Strong hypotheses should explain both intended and unintended high-confidence score patterns while identifying reliable prompt elements that trigger the classifier.

A.3 Image generation for self-evaluation

You will get access to a classifier in the final layer of a deep neural network that detects the presence of 0 within photo-realistic images for autonomous driving. Your task is to validate a hypothesis about the classifier's biases. A researcher has concluded:
HYPOTHESIS: <BIAS>
Generate two sets of image prompts: 1. 10 prompts for images containing 0 that should trigger HIGH classifier confidence score based on the hypothesis 2. 10 prompts for images containing 0 that should trigger LOW classifier confidence score based on the hypothesis
If the hypothesis suggests multiple bias scenarios, distribute the prompts to cover all cases. Make sure each image only tests ONE bias scenario at a time so different possible sources of bias do not confound.
Return your results in this exact format:
<POSITIVE PROMPTS> 1. [First high-confidence score prompt] 2. [Second high-confidence score prompt] ... 10. [Tenth high-confidence score prompt] </POSITIVE PROMPTS>
<NEGATIVE PROMPTS> 1. [First low-confidence score prompt] 2. [Second low-confidence score prompt] ... 10. [Tenth low-confidence score prompt] </NEGATIVE PROMPTS>
Remember to start the positive examples with the opening tag <POSITIVE PROMPTS> and end the positive examples with closing tag </POSITIVE PROMPTS>. Remember to start the negative examples with opening tag <NEGATIVE PROMPTS> and end the negative examples with closing tag </NEGATIVE PROMPTS>.

A.4 API and implementation details

The API of our agent is based on that of MAIA [23] with a few important modifications: (i) our agent is able to define experiments in free-form code blocks, where initially MAIA had to define all its code within a function (execute_command) that was then executed by the codebase. This allows our agent to both write multiple blocks of code per experiment and to access variables defined in previous experiments. (ii) the agent is now able to log and display any text/images generated during its experiments in the format of its choosing with a single, flexible function (display), whereas MAIA had to rely on individual tools to display their own results in predetermined formats. (iii) The agent can use more recent VLLMs, particularly Claude-3.5-sonnet, where the original codebase used gpt-4-vision-preview.

A.5 Agent tools

To support experimentation and interpretability analysis, we provide a `Tools` class with utilities for:

- **Text-to-image generation** from prompts via pretrained diffusion models.
- **Prompt-based image editing** to create controlled counterfactuals.
- **Image summarization** that identifies common semantic or non-semantic features across a set of images.
- **Region-based image descriptions** that generate textual descriptions of highlighted activation regions.
- **Exemplar retrieval** for a given classifier unit, returning representative images and their scores.

A.6 Supported backbone models.

The agent ships with a small self-contained toolkit that lets us run end-to-end vision experiments (image generation, editing, and logging) through a single, uniform interface.

- **Text-to-Image Generation**:
  - Flux Image Generator- use for all agent experiments in the paper
  - DALL·E 3 (OpenAI API) (use for the evaluation)
- **Image Editing**:
  - InstructDiffusion (Stable Diffusion variant with instruction tuning)
- **Image Description and Summarization**:
  - GPT-4o [53], used via API to describe image regions and summarize visual commonalities across images.

A.7 Interface and Logging.

All generated or edited images are stored in Base64 format for transmission and display. The framework logs each experiment (prompt, image, activation, description) and supports export as an interactive HTML report for reproducibility.

Overall, the toolkit enables the agent to generate or edit images for hypothesis testing, score them different models, analyse the outcomes, and package the entire run into a report, making each experiment swift, scalable, and fully reproducible. For a comprehensive overview of hardware requirements, see Table 1.

A.8 Resources

All our experiments were conducted on a single NVIDIA RTX 3090 (24 GB) GPU. The agent backbone (Claude-3.5-sonnet) was used through Anthropic API, and the prompt generator for self-reflection (GPT4o) and the evaluator modern in the 2AFC experiment (GPT4) were accessed through Open-AI API.

Model (Inference)	**Peak VRAM ↓ (GB)**	**#Params ↓ (M)**
SAM ViT–H	∼7.0	632
Grounding DINO Swin–T	∼0.45	174
SigLIP So400m (P14/384)	∼2.1	878
CLIP ViT–L/14 (OpenAI)	∼2.04	428
FairFace ResNet-34	∼0.06	21.8
YOLOv8-m (Ultralytics)	∼0.07	25.9
Stable Diffusion 3.5 Medium (FP16)	∼9–10	2 500
FLUX.dev (12B, 4-bit + offload)	∼10-11	12 000
InstructPix2Pix (SD-1.5 base)	∼6-7	890
Instruction Diffusion	∼10	1 000
RetinaFace MobileNetV3	∼0.02	1.7
Average Experiment	∼19.5-20.5	–

Table 1: Peak GPU memory and parameter scale of all models used at inference time.

A.9 API prompt

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

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

    Methods
    -------
    call_classifier(image_list: List[torch.Tensor]) -> Tuple[List[int], List[str]]
        Returns the classifier confidence score for each image in the input
            image_list as well as the original image (encoded into a Base64 string)
            .
    """

    def __init__(self, classifier_num: int, layer: str, model_name: str, device:
        str):
        """
        Initializes a classifier object by specifying its number and layer location
             and the vision model that the classifier belongs to.
        Parameters
        -------
        classifier_num : int
            The unit number of the classifier.
        layer : str
            The name of the layer where the classifier is located.
        model_name : str
            The name of the vision model that the classifier is part of.
        device : str
            The computational device ('cpu' or 'cuda').
        """
        self.classifier_num = classifier_num
        self.layer = layer
        self.device = torch.device(f"cuda:{device}" if torch.cuda.is_available()
            else "cpu")

    def call_classifier(self, image_list: List[torch.Tensor]) -> Tuple[List[float],
         List[str]]:
        """
        The function returns the classifiers maximum confidence score (in float
            format) for each of the images in the list as well as the original
            image (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 confidence score of the
                classifier on that image, and the original image encoded into a
                Base64 string.
```

```
        Examples
        --------
        >>> # test the confidence score of the classifier for the prompt "a dog
            standing on the grass"
        >>> prompt = ["a dog standing on the grass"]
        >>> image = tools.text2image(prompt)
        >>> score_list, image_list = system.call_classifier(image)
        >>> for score, image in zip(score_list, image_list):
        >>>     tools.display(image, f"Confidence Score: {score}")
        >>>
        >>> # test the confidence score of the classifier for the prompt "a dog
            standing on the grass" and maintain robustness to noise
        >>> prompts = ["a dog standing on the grass"]*5
        >>> images = tools.text2image(prompts)
        >>> score_list, image_list = system.call_classifier(images)
        >>> tools.display(image_list[0], f"Confidence Score: {statistics.mean(
            score_list)}")
        >>>
        >>> # test the confidence score of the classifier for the prompt "a
            landscape with a tree and river"
        >>> # for the same image but with different seasons:
        >>> prompts = ["a landscape with a tree and a river"]*3
        >>> original_images = tools.text2image(prompts)
        >>> edits = ["make it autumn","make it spring","make it winter"]
        >>> all_images, all_prompts = tools.edit_images(original_images, edits)
        >>> score_list, image_list = system.call_classifier(all_images)
        >>> for score, image, prompt in zip(score_list, image_list, all_prompts):
        >>>     tools.display(image, f"Prompt: {prompt}\nConfidence Score: {score}"
            )
        """

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

    Attributes
    ----------
    text2image_model_name : str
        The name of the text-to-image model.
    text2image_model : any
        The loaded text-to-image model.
    images_per_prompt : int
        Number of images to generate per prompt.
    path2save : str
        Path for saving output images.
    threshold : any
        Confidence score threshold for classifier analysis.
    device : torch.device
        The device (CPU/GPU) used for computations.
    experiment_log: str
        A log of all the experiments, including the code and the output from the
            classifier
        analysis.
    exemplars : Dict
        A dictionary containing the exemplar images for each unit.
    exemplars_scores : Dict
        A dictionary containing the confidence scores for each exemplar image.
    exemplars_thresholds : Dict
        A dictionary containing the threshold values for each unit.
    results_list : List
        A list of the results from the classifier analysis.
```

```
    Methods
    -------
    dataset_exemplars(system: System)->List[Tuple[int, 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 classifier on real images. This function characterizes the
        prototypical behavior of the classifier by computing its confidence score
            on
        all images in the ImageNet dataset and returning the 15 highest confidence
        scores and the images that produced them in Base64 encoded string format.
    edit_images(self, base_images: List[str], editing_prompts: List[str]) -> Tuple[
        List[List[str]], List[str]]
        This function enables localized testing of specific hypotheses about how
        variations on the content of a single image affect classifier confidence
            scores.
        Gets a list of input images in Base64 encoded string format and a list of
        corresponding editing instructions, then edits each provided image based on
             the
        instructions given in the prompt using a text-based image editing model.
            The
        function returns a list of images in Base64 encoded string format and list
            of the
        relevant prompts. This function is very useful for testing the causality of
             the
        classifier in a controlled way, or example by testing how the classifier
            confidence
        score 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) -> List[str]
        Gets a list of text prompts as an input and generates an image for each
        prompt using a text to image model. The function returns a
        list of images in Base64 encoded string format.
    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.
    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 generates a textual
        description of the semantic content of each of them.
        The function is blind to the current hypotheses list and
        therefore provides an unbiased description of the visual content.
    display(self, *args: Union[str, Image.Image]):
        This function is your way of displaying experiment data. You must call
        this on results/variables that you wish to view in order to view them.
    """

    def __init__(self, path2save: str, device: str, DatasetExemplars:
        DatasetExemplars = None, images_per_prompt=1, text2image_model_name='sd'):
        """
        Initializes the Tools object.

        Parameters
        ----------
        path2save : str
            Path for saving output images.
        device : str
            The computational device ('cpu' or 'cuda').
        DatasetExemplars : object
            an object from the class DatasetExemplars
```

```
        images_per_prompt : int
            Number of images to generate per prompt.
        text2image_model_name : str
            The name of the text-to-image model.
        """
    def dataset_exemplars(self, system: System) -> List[Tuple[float, str]]
        """
        This method finds images from the ImageNet dataset that produce the highest
             confidence scores for a specific classifier.
        It returns both the confidence scores and the corresponding exemplar images
             that were used to generate these confidence scores.
        This experiment is performed on real images and will provide a good
            approximation of the classifier behavior.

        Parameters
        ----------
        system : System
            The system representing the specific classifier and layer within the
                neural network.
            The system should have 'layer' and 'classifier_num' attributes, so the
                dataset_exemplars function
            can return the exemplar confidence scores and images for that specific
                classifier.

        Returns
        -------
        List
            For each exemplar image, stores a tuple containing two elements:
            - The first element is the confidence score for the specified
                classifier.
            - The second element is the exemplar images (as Base64 encoded strings)
                 corresponding to the confidence score.

        Example
        -------
        >>> exemplar_data = tools.dataset_exemplars(system)
        >>> for score, image in exemplar_data:
        >>>     tools.display(image, f"Confidence Score: {score}")
        """

    def edit_images(self,
                    base_images: List[str],
                    editing_prompts: List[str]) -> Tuple[List[List[str]], List[str
                        ]]:
        """
        Generates or uses provided base images, then edits each base image with a
        corresponding editing prompt. Accepts either text prompts or Base64
        encoded strings as sources for the base images.

        The function returns a list containing lists of images (original and edited
             ,
        interleaved) in Base64 encoded string format, and a list of the relevant
        prompts (original source string and editing prompt, interleaved).

        Parameters
        ----------
        base_images : List[str]
            A list of images as Base64 encoded strings. These images are to be
            edited by the prompts in editing_prompts.
        editing_prompts : List[str]
            A list of instructions for how to edit the base images derived from
            `base_images`. Must be the same length as `base_images`.
```

```
        Returns
    -------
    Tuple[List[List[str]], List[str]]
        - all_images: A list where elements alternate between:
            - A list of Base64 strings for the original image(s) from a source.
            - A list of Base64 strings for the edited image(s) from that source
                .
          Example: [[orig1_img1, orig1_img2], [edit1_img1, edit1_img2], [
              orig2_img1], [edit2_img1], ...]
        - all_prompts: A list where elements alternate between:
            - The original source string (text prompt or Base64) used.
            - The editing prompt used.
          Example: [source1, edit1, source2, edit2, ...]
        The order in `all_images` corresponds to the order in `all_prompts`.

    Raises
    ------
    ValueError
        If the lengths of `base_images` and `editing_prompts` are not equal.

    Examples
    --------
    >>> # test the confidence score of the classifier for the prompt "a
         landscape with a tree and river"
    >>> # for the same image but with different seasons:
    >>> prompts = ["a landscape with a tree and a river"]*3
    >>> original_images = tools.text2image(prompts)
    >>> edits = ["make it autumn","make it spring","make it winter"]
    >>> all_images, all_prompts = tools.edit_images(original_images, edits)
    >>> score_list, image_list = system.call_classifier(all_images)
    >>> for score, image, prompt in zip(score_list, image_list, all_prompts):
    >>>     tools.display(image, f"Prompt: {prompt}\nConfidence Score: {score}"
         )
    >>>
    >>> # test the confidence score of the classifier on the highest scoring
         dataset exemplar
    >>> # under different conditions
    >>> exemplar_data = tools.dataset_exemplars(system)
    >>> highest_scoring_exemplar = exemplar_data[0][1]
    >>> edits = ["make it night","make it daytime","make it snowing"]
    >>> all_images, all_prompts = tools.edit_images([highest_scoring_exemplar]*
         len(edits), edits)
    >>> score_list, image_list = system.call_classifier(all_images)
    >>> for score, image, prompt in zip(score_list, image_list, all_prompts):
    >>>     tools.display(image, f"Prompt: {prompt}\nConfidence Score: {score}"
         )
    """

def text2image(self, prompt_list: List[str]) -> List[List[str]]:
    """
    Takes a list of text prompts and generates images_per_prompt images for
         each using a
    text to image model. The function returns a list of a list of
         images_per_prompt images
    for each prompt.

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

    Returns
    -------
    List[List[str]]
        A list of a list of images_per_prompt images in Base64 encoded string
             format for
        each input prompts.
```

```
    Examples
    --------
    >>> # Generate images from a list of prompts
    >>> prompt_list = [ a   dog standing on the  g r a s s ,
    >>>                  a   dog sitting on a  c o u c h ,
    >>>                  a   dog running through a  f i e l d ]
    >>> images = tools.text2image(prompt_list)
    >>> score_list, image_list = system.call_classifier(images)
    >>> for score, image in zip(score_list, image_list):
    >>>     tools.display(image, f"Confidence Score: {score}")
    """

def display(self, *args: Union[str, Image.Image]):
    """
    Displays a series of images and/or text in the chat, similar to a Jupyter
        notebook.

    Parameters
    ----------
    *args : Union[str, Image.Image]
        The content to be displayed in the chat. Can be multiple strings or
            Image objects.

    Notes
    -------
    Displays directly to chat interface.

    Example
    -------
    >>> # Display a single image
    >>> prompt = ["a dog standing on the grass"]
    >>> images = tools.text2image(prompt)
    >>> score_list, image_list = system.call_classifier(images)
    >>> for score, image in zip(score_list, image_list):
    >>>     tools.display(image, f"Confidence Score: {score}")
    >>>
    >>> # Display a single image from a list
    >>> prompts = ["a dog standing on the grass"]*5
    >>> images = tools.text2image(prompts)
    >>> score_list, image_list = system.call_classifier(images)
    >>> tools.display(image_list[0], f"Confidence Score: {statistics.mean(
        score_list)}")
    >>>
    >>> # Display a list of images
    >>> prompt_list = [ a   dog standing on the  g r a s s ,
    >>>                  a   dog sitting on a  c o u c h ,
    >>>                  a   dog running through a  f i e l d ]
    >>> images = tools.text2image(prompt_list)
    >>> score_list, image_list = system.call_classifier(images)
    >>> for score, image in zip(score_list, image_list):
    >>>     tools.display(image, f"Confidence Score: {score}")
    """

def summarize_images(self, image_list: List[str]) -> str:
    """
    Gets a list of images and describes what is common to all of them.


    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
        -------
        >>> # Summarize a classifier's dataset exemplars
        >>> exemplars = [exemplar for _, exemplar in tools.dataset_exemplars(system
            )] # Get exemplars
        >>> summarization = tools.summarize_images(exemplars)
        >>> tools.display(summarization)
        """

    def describe_images(self, image_list: List[str], image_title:List[str]) -> str:
        """
        Generates textual descriptions for a list of images, focusing
        specifically on highlighted regions. The final descriptions are
        concatenated and returned as a single string, with each description
        associated with the corresponding image title.

        Parameters
        ----------
        image_list : List[str]
            A list of images in Base64 encoded string format.
        image_title : List[str]
            A list of titles for each image in the image_list.

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

        Example
        -------
        >>> prompt_list = ["a dog standing on the grass",
        >>>                "a dog sitting on a couch",
        >>>                "a dog running through a field"]
        >>> images = tools.text2image(prompt_list)
        >>> score_list, image_list = system.call_classifier(images)
        >>> descriptions = tools.describe_images(image_list, prompt_list)
        >>> tools.display(descriptions)
```

B Benchmark models

B.1 Systems specification

We provide below the full list of objects and categories used for our benchmark.

	Gender		Age	
	Female	Male	Young	Old
1	Apron (“kitchen")	Tie	Laptop	Glasses
2	Umbrella	Beer	Cell phone	Book
3	Scarf	Skateboard	Skateboard	Hat
4	Cat	Suit	Bicycle	Tie
5	Book	Laptop	Teddy bear	Wine glass
6	Handbag	motorcycle		
7	Wine glass	surfboard		
8	Hair drier	Frisbee		
9	Teddy bear	Guitar		
10	Dress	Cap		

Table 2: Feature categories and corresponding objects associated with gender and age stereotypes.

	Color					Material	
	Red	Green	Blue	Black	White	Wooden	Ceramic
1	Bus	Bus	Bus	Bus	Bus	Table	Vase
2	Umbrella	Umbrella	Umbrella	Umbrella	Umbrella	Chair	Bowl
3	Tie	Tie	Tie	Tie	Tie	Bench	Cup
4	Kite	Kite	Kite	Kite	Kite		
5	Frisbee	Frisbee	Frisbee	Frisbee	Frisbee		

Table 3: Feature categories and corresponding objects associated with color and material properties.

	Setting						State
	Kitchen	Living Room	Office	Wilderness	City	Beach	Misc.
1	Table	Table	Table	Bird	Bird	Bird	Airplane (Flying)
2	Chair	Chair	Chair	Car	Car	Car	Bicycle (Ridden)
3	Cat	Cat	Cat	Dog	Dog	Dog	Clock (Analog)
4	Dog	Dog	Dog	Horse	Horse	Horse	Keyboard (Typing)
5	Vase	Vase	Vase	Bench	Bench	Bench	Kite (Flying)
6	Wine glass	Wine glass	Wine glass				Umbrella (Open)
7							Vase (With flowers)

Table 4: Feature categories and corresponding objects associated with different settings and states.

	Gender		Age	
	Female	Male	Young	Old
1	Tie	Apron	Glasses	Laptop
2	Beer	Umbrella	Book	Cell phone
3	Skateboard	Scarf	Hat	Skateboard
4	Suit	Cat	Tie	Bicycle
5	Laptop	Book	Wine glass	Teddy bear
6	motorcycle	Handbag		
7	surfboard	Wine glass		
8	Frisbee	Hair drier		
9	Guitar	Teddy bear		
10	Cap	Dress		

Table 5: Feature categories and corresponding objects with flipped gender and age associations.

B.2 Dependency strength versus discount factor

We investigate how a *discount factor* $\alpha \in [0, 1]$ of our synthetic model attenuates the score of the synthetic model henever a predefined attribute condition is *not* satisfied. Figure 8 shows the mean classification accuracy for six attribute groups: AGE, COLOR, GENDER, SETTING, SIZE, and STATE.

- **No discount (α=1.0).** Baseline accuracies for all groups remain high (≥ 0.73).
- **Small discount ($0 < \alpha \leq 0.3$).** Accuracy drops slightly (under five percentage points), indicating that mild penalties leave decisions largely intact.
- **Medium discount ($0.3 < \alpha \leq 0.5$).** Accuracy decreases almost linearly—GENDER is most robust, while SIZE and SETTING degrade faster.
- **High–extreme discount ($\alpha > 0.5$).** A sharp collapse occurs; COLOR and SIZE fall below 0.20 at $\alpha \approx 0.7$, and all groups eventually saturate between 0.05 and 0.25.

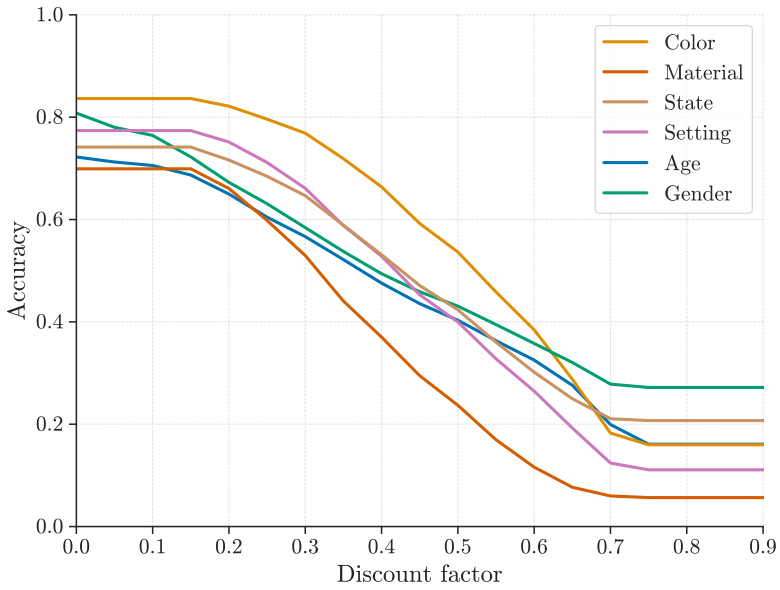


Figure 8: Mean accuracy versus discount factor α for six attribute groups.

C Results

C.1 Additional Analysis

Figures 9 and 10 report the average self-evaluation scores across ten rounds of self-reflection, under discount factors of α=0.5 and α=0.9, respectively. We observe a consistent upward trend in performance across all five attribute categories—GENDER, AGE, COLOR, STATE, and SETTING. This trend holds across both mild and severe reliance scenarios, suggesting that the iterative refinement process is effective in improving the quality of the agent's hypotheses and explanations. While early rounds exhibit fluctuations (especially at α=0.5), later rounds show stabilization and convergence toward higher evaluation scores. The improvement is more pronounced under the lighter discounting condition (α=0.5), where the agent starts from lower scores but achieves a comparable gain. It also noticable that for a disambiguate attribute dependency (α=0.5) more self-reflection rounds are necessary, whereas with (α=0.9) a saturation is achieved earlier. This demonstrates that self-reflection enables the agent to recover explanatory accuracy even in challenging scenarios.

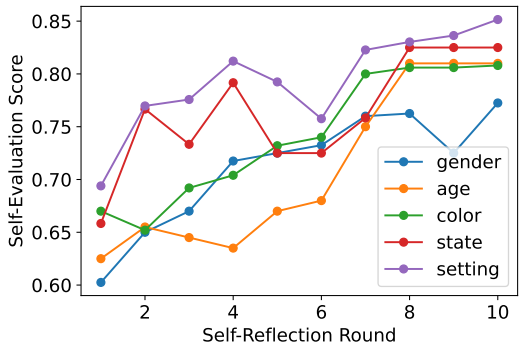


Figure 9: $\alpha = 0.5$

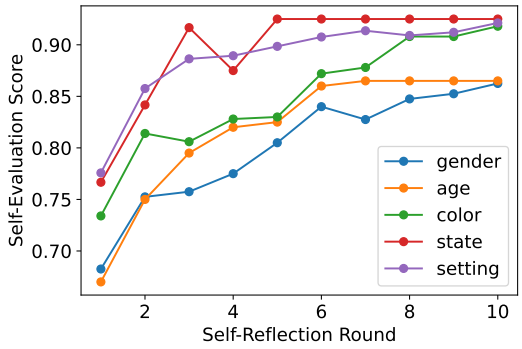


Figure 10: $\alpha = 0.9$