Pelican: Correcting Hallucination in Vision-LLMs via Claim Decomposition and Program of Thought Verification

Pritish Sahu[*], Karan Sikka[*], Ajay Divakaran

SRI International, Princeton, NJ

`{pritish.sahu, karan.sikka, ajay.divakaran}@sri.com`

Abstract

Large Visual Language Models (LVLMs) struggle with hallucinations in visual instruction following task(s), limiting their trustworthiness and real-world applicability. We propose **`Pelican`** – a novel framework designed to detect and mitigate hallucinations through claim verification. **`Pelican`** first decomposes the visual claim into a chain of sub-claims based on first-order predicates. These sub-claims consist of (predicate, question) pairs and can be conceptualized as nodes of a computational graph. We then use Program-of-Thought prompting to generate Python code for answering these questions through flexible composition of external tools. **`Pelican`** improves over prior work by introducing (1) intermediate variables for precise grounding of object instances, and (2) shared computation for answering the sub-question to enable adaptive corrections and inconsistency identification. We finally use reasoning abilities of LLMs to verify the correctness of the claim by considering the consistency and confidence of the (question, answer) pairs from each sub-claim. Our experiments reveal a drop in hallucination rate by $\sim 8\% - 32\%$ across various baseline LVLMs and a 27% drop compared to approaches proposed for hallucination mitigation on MMHal-Bench. Results on two other benchmarks further corroborate our results.

1 Introduction

Large Vision Language Models (LVLM) have seen significant advancements in recent years (Liu et al., 2023b; Wu et al., 2023). They typically integrate visual tokens into the embedding space of a Large Language Model (LLM), leveraging the linguistic capabilities of LLMs while incorporating visual information for multimodal understanding. Despite substantial performance gains, LVLMs suffer from hallucinations due to limited training data, lack of precise grounding, and over-reliance on language priors (Liu et al., 2024a).

Prior works have focused on scaling training data for reducing hallucinations, as demonstrated by the improved performance of LLaVA-1.5 (Liu et al., 2023b) that used many academic datasets during instruction tuning. Other works have created high-quality visual instruction tuning datasets. For example, LRV (Liu et al., 2023a) included diverse examples as well as adversarial questions referencing non-existent objects. Another promising direction has been to improve the model by using variants of reinforcement learning with human feedback (RLHF) to align the model with human preferences (Chen et al., 2023; Yu et al., 2023) or by training the model to correct itself through self-feedback during inference (Lee et al., 2023). Woodpecker (Yin et al., 2023), inspired by the fact-checking task in NLP (Guo et al., 2022), recently proposed to correct hallucinations by modeling the problem as verifying and correcting visual claims generated by LVLMs. The method involved extracting key concepts from the outputs, formulating questions, answering them using visual tools such as VQA, and collating the outputs into a visual claim, which is then used to refine the original output with an LLM. Our work builds upon and extends this claim verification paradigm.

We propose **`Pelican`** (Figure 1), a novel and structured pipeline for verifying visual claims to detect and correct hallucinations. **`Pelican`** addresses the weaknesses of prior methods for claim verification such as lack of precise grounding, weak contextual integration and visual referencing, and ineffective reasoning over the claim and the visual context. **`Pelican`** first breaks down a complex claim into more manageable sub-claims by using a set of predefined first-order predicates tailored to the visual question answering (VQA) task. These sub-claims are represented as a chain of (predicate, question) pairs, with each question stemming

[*] Equal contribution

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 8228–8248
November 12-16, 2024 ©2024 Association for Computational Linguistics

from its corresponding predicate. The resulting chain can be interpreted as a computational graph (Figure 2), where each node corresponds to a predicate/question. The verification of the overall claim is then accomplished by answering the questions in a sequential manner. **Pelican** uses Program-of-Thought prompting to synthesize Python code that seamlessly integrates external tools with Python operators, offering greater flexibility compared to previous methods like Woodpecker. The introduction of intermediate variables to reference specific object instances is another innovation, which is crucial for precise grounding, particularly in claims involving multiple objects. Furthermore, **Pelican** shares computations from previous nodes in the chain while answering questions, enabling adaptive corrections and the identification of inconsistencies in the reasoning process, distinguishing it from earlier works. **Pelican** creates a visual table representation, that includes key visual entities, stored as a Pandas dataframe to simplify manipulation of key entities in the code generation step. The information from the sub-claims is then combined, and the reasoning capabilities of LLM are used to assess the correctness of the claim and generate a refinement in case of hallucinations. Robustness in this step is ensured by using in-context examples and promoting CoT-style reasoning that considers the correctness, confidence, and relevance of the generated answers for each sub-claim. By integrating the flexibility of reasoning in language with the precision of computational methods, **Pelican** is able to achieve strong improvements over SOTA methods. We show significant drop in hallucinations on standard benchmarks (MMHal-Bench, GAVIE) and improve visual understanding accuracy on the MME dataset. Our ablation study reveals the contribution of the key innovations on the final performance. Through qualitative examples, we demonstrate how the model identifies and corrects hallucinated locations.

Our contributions are as follows, we:

1. Propose a robust pipeline for identifying hallucination in LVLMs by decomposing the visual claim into sub-claims that consist of questions grounded in first-order predicates.

2. Enable precise grounding through intermediate variables for referencing object instances.

3. Generate Python code to answer sub-questions with external tools, enabling flexible tool composition with Python operators.

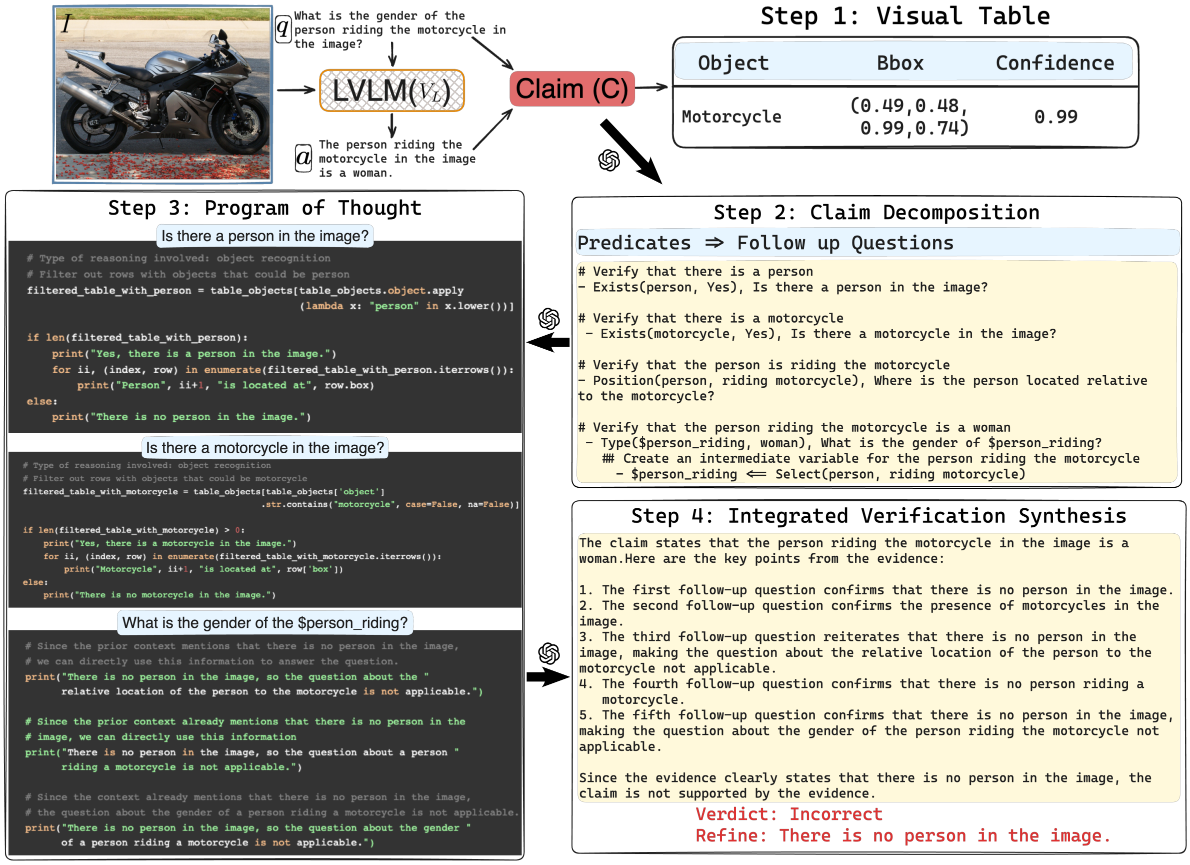
4. Enhance reasoning by sharing computations between questions, allowing for adaptive corrections and identify inconsistencies.

5. Demonstrate consistent performance improvement over different baseline LVLMs on several benchmarks, as well as improvements relative to existing approaches for mitigating hallucinations. For MMHal-Bench, we reduce by $\sim 8\% - 32\%$ on LVLMs and 27% over best hallucination mitigation baseline. We also show similar improvements on GAVIE and MME.

2 Related Work

Large Visual Language Models and Hallucinations Recent LVLMs (Alayrac et al., 2022; Zhu et al., 2023; Dai et al., 2024; Huang et al., 2024; Peng et al., 2023; Liu et al., 2024c, 2023b) demonstrate superior performance on established benchmarks with strong instruction-following and zero-shot reasoning capabilities. However, these models still suffer from hallucinations and provide incorrect answers or fabricate visual details. This issue arises from several sources: the lack of diverse training data leading to insufficient representation of varied visual contexts (Zhu et al., 2023; Dai et al., 2024), over-reliance on natural language cues (Hu et al., 2023; Liu et al., 2024c, 2023b), a yes-bias tendency to affirmatively answer regardless of the visual content (Liu et al., 2024c), short text descriptions that do not fully cover the image (Dai et al., 2024), and synthetic data generation that can introduce non-factual statements (Liu et al., 2024c). We propose a post-hoc procedure to address hallucinations in LVLMs by integrating visual tools with LLMs to ground and reason about their outputs by analyzing a chain of simpler sub-claims. We refer readers to Appendix D for other works.

Claim Decomposition The use of Language Models (LMs) for fact-checking has gained significant attention. (Lee et al., 2020) leveraged LLMs to verify simple facts, while (Atanasova et al., 2022) addressed the challenge of insufficient evidence. (Li et al., 2023a) enhanced fact-checking by retrieving current information from the web, and (Cao, 2023) incorporated Chain of Thought (CoT) prompting. Subsequent works (Kotonya and Toni, 2020; Atanasova, 2024) focused on generating explanations by summarizing the comments provided



What is the gender of the person riding the motorcycle in the image?

The person riding the motorcycle in the image is a woman.

Step 1: Visual Table

Object	Bbox	Confidence
Motorcycle	(0.49,0.48, 0.99,0.74)	0.99

Step 2: Claim Decomposition

Predicates ⇒ Follow up Questions

```
# Verify that there is a person
- Exists(person, Yes), Is there a person in the image?

# Verify that there is a motorcycle
 - Exists(motorcycle, Yes), Is there a motorcycle in the image?

# Verify that the person is riding the motorcycle
- Position(person, riding motorcycle), Where is the person located relative
to the motorcycle?

# Verify that the person riding the motorcycle is a woman
 - Type($person_riding, woman), What is the gender of $person_riding?
   ## Create an intermediate variable for the person riding the motorcycle
     - $person_riding ⇐ Select(person, riding motorcycle)
```

Step 3: Program of Thought

Is there a person in the image?

```python
# Type of reasoning involved: object recognition
# Filter out rows with objects that could be person
filtered_table_with_person = table_objects[table_objects.object.apply
                                          (lambda x: "person" in x.lower())]

if len(filtered_table_with_person):
    print("Yes, there is a person in the image.")
    for ii, (index, row) in enumerate(filtered_table_with_person.iterrows()):
        print("Person", ii+1, "is located at", row.box)
else:
    print("There is no person in the image.")
```

Is there a motorcycle in the image?

```python
# Type of reasoning involved: object recognition
# Filter out rows with objects that could be motorcycle
filtered_table_with_motorcycle = table_objects[table_objects['object']
                                               .str.contains("motorcycle", case=False, na=False)]

if len(filtered_table_with_motorcycle) > 0:
    print("Yes, there is a motorcycle in the image.")
    for ii, (index, row) in enumerate(filtered_table_with_motorcycle.iterrows()):
        print("Motorcycle", ii+1, "is located at", row['box'])
else:
    print("There is no motorcycle in the image.")
```

What is the gender of the $person_riding?

```python
# Since the prior context mentions that there is no person in the image,
# we can directly use this information to answer the question.
print("There is no person in the image, so the question about the "
      relative location of the person to the motorcycle is not applicable.")

# Since the prior context already mentions that there is no person in the
# image, we can directly use this information
print("There is no person in the image, so the question about a person "
      riding a motorcycle is not applicable.")

# Since the context already mentions that there is no person in the image,
# the question about the gender of a person riding a motorcycle is not applicable.
print("There is no person in the image, so the question about the gender "
      of a person riding a motorcycle is not applicable.")
```

Step 4: Integrated Verification Synthesis

```
The claim states that the person riding the motorcycle in the image is a
woman.Here are the key points from the evidence:

1. The first follow-up question confirms that there is no person in the image.
2. The second follow-up question confirms the presence of motorcycles in the
image.
3. The third follow-up question reiterates that there is no person in the
image, making the question about the relative location of the person to the
motorcycle not applicable.
4. The fourth follow-up question confirms that there is no person riding a
   motorcycle.
5. The fifth follow-up question confirms that there is no person in the image,
making the question about the gender of the person riding the motorcycle not
applicable.

Since the evidence clearly states that there is no person in the image, the
claim is not supported by the evidence.
```

Verdict: Incorrect
Refine: There is no person in the image.

Figure 1: Overview of **Pelican** . Given an image (I) and a question ($\hat{q}$), LVLM (V_L) outputs an answer ($\hat{a}$). We transform the pair ($\hat{q}$, $\hat{a}$) to our claim (c). Our pipeline: Step 1: Visual Table constructs a tabular representation of the image, identifying the locations of visual objects using detection tools. For example, a row is created for the detected "motorcycle" with its bounding box. Step 2: Claim Decomposition generates a list of granular sub-claims and follow-up questions. This example also shows an intermediate variable, $person_riding, that is used for referencing a specific object. Step 3: Program of Thought translates these questions into Python code using the POT approach. Final Step: Integrated Verification Synthesis performs comprehensive reasoning assessments to validate the original claims using the answers from Step 3.

by professional fact-checkers. In the domain of fact-checking, claim verification focuses on validating claims by retrieving information from external sources. Recent works, such as (Li et al., 2023a; Wang and Shu, 2023), involve decomposing claims into granular sub-claims for verification with an LLM. Woodpecker (Yin et al., 2023) proposed a similar framework for visual claim verification to correct hallucination. Our work advances these approaches by tightly integrating the chain of sub-claims through (1) synthesizing Python code for answering sub-claims, and (2) using shared variables and shared computations between sub-claims to improve efficiency and consistency.

Tool Calling Recent works, such as VisProg, HuggingGPT, and ViperGPT (Gupta and Kembhavi, 2023; Shen et al., 2023; Schick et al., 2023; Surís et al., 2023) fall under the umbrella of Program-of-Thought prompting (Chen et al., 2022) that leverage the strong coding capabilities of LLMs to compose external tools for complex multi-modal tasks. VisProg uses in-context learning to generate modular programs for visual tasks, while HuggingGPT employs ChatGPT to plan and manage AI models from Hugging Face. The proposed work is inspired by these ideas and extends this ability to answer questions via code execution and further integrate responses from the chain of sub-claims to verify the overall claim.

3 Pelican

Problem Formulation: We are given outputs from a Large Vision Language Model (LVLM) V_L, which provides an answer $a = \mathcal{V}(I, q)$ for a given image $\mathcal{I} \in \mathcal{R}^{M \times N \times 3}$ (where M and N represent the height and width of the image) and question q. We combine the question-answer pair (q, a) into a claim $\mathcal{C}$ about the image using a Large Language Model (LLM) for the remaining steps. We formulate the problem of detecting and correcting hallucinations in a as a visual claim verification task (Yin et al., 2023), which inputs $(\mathcal{I}, \mathcal{C})$ and output (d, r), where $d \in \{correct, incorrect\}$ and r refer to the decision regarding the correctness of the claim and a rewrite (if needed) respectively.

We propose **Pelican** (Figure 1) for verifying visual claims to detect and correct hallucinations. **Pelican** first parses and decomposes the claim into granular sub-claims, consisting of follow-up questions, by leveraging reasoning capabilities of LLMs. Compared to prior works (Yin et al., 2023), **Pelican** introduces intermediate variables to reference specific instances of objects, enabling precise grounding and visual referencing throughout the verification process. We then harness the coding capabilities of LLMs to answer the follow-up questions by translating them into Python code using a Program-of-Thought approach (Chen et al., 2022) This allows for the flexible composition of visual tools using native Python operators. We share the results from previous computations when answering the next question in the chain to facilitate adaptive corrections and catch inconsistencies in the reasoning process often arising from brittleness of visual tools. We also handle the limitations of visual tools (e.g., miss-detections) by mapping the image into a tabular representation that consists of visual objects identified in the image. We argue that the proposed decomposition implicitly converts the claim into a computational graph with dependencies between sub-questions, while the PoT framework enables **Pelican** to answer questions by combining the flexibility of reasoning in natural language with the precision of computational methods, resulting in a more effective and robust claim verification process. We next discuss these components in detail.

3.1 Visual Table

Our approach relies on tools such as object detectors to ground visual entities in the image. During our initial experiments, we identified several limitations with off-the-shelf object detectors. For instance, closed-vocabulary object detectors are more powerful but cannot detect novel objects, while open-vocabulary detectors may incorrectly identify objects that are not present, leading to false-positives. To address these limitations, we carefully crafted a pipeline (refer to Appendix B in the appendix) that utilizes YOLO and Grounding-DINO to determine the presence and location of an object in the image. Given the image and the claim, we produce a visual table representation by first parsing the claim using an LLM, with in-context examples, to identify key entities that are tangible and can be visually grounded. This is done to reduce the likelihood of false positives. For example, the claim *"The disposable coffee cups are upside down on the nightstand"* will be parsed into $\{cups, nightstand\}$. These entities are then passed to the detectors to create a table $\mathcal{T}$.

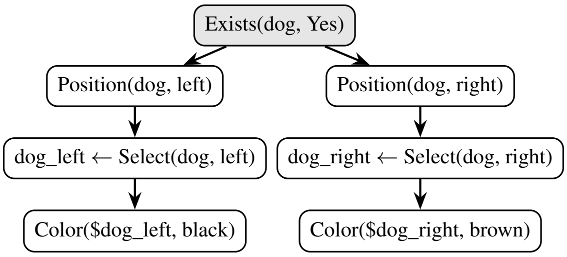


Figure 2: Computational graph representation of the generated sub-claims with predicates as the node and edges defined by their dependencies.

Although this step could be offloaded to the code generation step, we chose not to do so for two primary reasons: (1) to avoid complex Python code and allow it to focus on the high-level task of composing tools and reasoning, and (2) because the visual table is stored as a Pandas dataframe, it provides a flexible data structure that can be easily manipulated in Python. To further reduce the false positives from Grounding-DINO, we also utilize the Visual Question Answering (VQA) tool to verify the existence of the detected objects.

3.2 Claim Decomposition

Since answering simpler questions about images using external tools (e.g., determining if an object exists) is more reliable, we first propose a novel way to decompose the visual claim $\mathcal{C}$ into atomic sub-claims. Each sub-claim is represented as a pair consisting of a predicate and its corresponding question (p_i, q_i). Each (q_i) can be conceptualized as a node in a computational graph (Figure 2), where the edges represent the logical connections between the questions. This graph structure will be employed to reason about the claim and determine its veracity by systematically answering the sub-claims.

To achieve this decomposition, we define a set of predicates such as $Exists$ and $Position$, which are loosely structured based on a taxonomy of questions posed in VQA tasks (Table 4). These predicates serve as the foundation for deriving specific questions. For instance, a claim involving an object's properties and interactions may be broken down into predicates concerning the presence, attributes, and relations of that object. We transform the claim $\mathcal{C}$ into predicates $\mathcal{P} = \{p_i\}_{i=1}^{L}$ by prompting an LLM with in-context examples. We use a set of ~ 10 examples covering all the predicates and scenarios such as claims with negations. Additionally, we generate a chain of questions $\mathcal{Q} = \{q_i\}_{i=1}^{K}$ grounded in each of the predicates.

We introduce intermediate variables v to reference specific object instances, which is critical in verifying claims about specific object instances. Moreover, this approach allows us to create dependencies between nodes of the computational graph. This step not only reduces computational redundancy but also improves the reliability of the verification process.

3.3 Program of Thought (PoT)-based Sub-Claim Verification

We now wish to verify each sub-claim, generated in the decomposition step, by generating a visually grounded answer to each sub-question q_i. While an off-the-shelf instruction-tuned LVLM can be used to answer questions, this approach is constrained by issues in LVLMs, such as hallucinations and lack of interpretability. Instead, we employ a PoT-based strategy to synthesize programmatic instructions that composes different visual tools to infer the answers to these sub-claims. For example, a question about the color of an object might be translated into a code snippet that first uses a detector to crop the object and then applies a VQA tool to determine the object's color in the image. A key advantage of using code to answer the question is the ability to combine different visual tools flexibly with Python operators.

We denote this step by the function λ which inputs the image, current question and the context from prior questions and answer to generate Python code c_k as $\lambda(\mathcal{T}, q_k, \{q_i, a_i\}_{i=1}^{k-1}) = c_k$, which is then used to derive the answer as $a_k = \text{exec}(c_k)$, where exec is a Python interpreter. We introduce several innovations compared to prior claim verification works to perform robust verification:

1. Sharing computations between sub-claims: We provide the answers derived in the previous sub-claims as context when answering the next question in the chain. We found this process to reduce duplicate computations, adaptively correct follow-up questions, and also catch inconsistencies in the reasoning process resulting from errors in the visual tools. For example, a follow-up question about *the color of a car* might not realize that multiple cars are present in the image and may thus end up generating code without loops over the object.

2. Intermediate variable: The intermediate variable created in the decomposition step help the framework to reference specific object instances, e.g., *object on the right*. We found this to improve verification of complex questions requiring contextual reasoning around multiple objects.

3. Reasoning in Language + Computations: We created our in-context examples to perform reasoning in both language, which provides flexibility, and via computations in the Python code (e.g., through creation of variables and composition with tools).

4. Visual table: Finally, this module integrates seamlessly with the table prepared in the initial step (subsection 3.1) of our pipeline. This table, which catalogs the visual entities detected and their attributes, serves as a reference point for validating the existence and properties of objects within the image. By cross-referencing the outputs of the vision tools with the entries in this table, the module can confirm or refute the predicates with a high degree of confidence.

3.4 Integrated Verification Synthesis

In this step, we use the answers obtained from the PoT-based verification step to perform a comprehensive reasoning assessment and validate the original claim. This step is crucial for detecting hallucinations and for drawing accurate and reliable conclusions. By aggregating the responses to each sub-claim, the system can make an informed decision about the overall validity of the claim.

We denote this step $d, r = \mathcal{V}(\mathcal{C}, P, \{q_i, a_i\}_{i=1}^{K})$, where the function $\mathcal{V}$ take in the original claim, predicate, and questions and answers from the sub-claims and outputs d, r which denote the decision and a re-write for claims that contain hallucinations. We realize this with an LLM with a detailed instruction and a few in-context examples that encourages the LLM to verify the claim by considering the evidence along with the consistency and coherence of the generated answers using CoT-style reasoning.

Any discrepancies or inconsistencies are flagged as potential hallucinations, prompting a closer inspection of the questionable elements. This rigorous cross-examination helps in identifying errors that may have arisen due to incorrect or ambiguous interpretations of the visual data. To mitigate hallucinations, we use the same LLM to rewrite the original claim based on the verified answers. This involves rephrasing the claim to eliminate elements that were identified as hallucinated or incorrect.

Through this integrated verification synthesis, our pipeline not only verifies the accuracy of claims but also actively improves the clarity and correctness of the information presented. This approach significantly reduces the risk of hallucinations, ensuring trustworthiness and precision.

4 Experiment

4.1 Experimental Setup

Tasks and Benchmarks. We use the following LVLM benchmarks for evaluation.

1. **MMHal-Bench** (Sun et al., 2023) evaluates informativeness and hallucinations in responses by comparing model outputs with human responses and object labels using GPT-4. The benchmark includes 96 image-question pairs across 8 categories and 12 topics, focusing on open-ended questions. We report the informativeness score and hallucination rate.

2. **GAVIE** (Liu et al., 2023a) evaluates hallucination by measuring the accuracy and relevancy (instruction-following ability) of responses in an open-ended manner without needing human-annotated ground truth answers We selected a random subset of 250 questions and use GPT-4o for evaluation.

3. **MME** (Fu et al., 2023) evaluates LVLMs' perception and cognition through "Yes" or "No" questions for each test image. It has 14 subtasks: 10 for perception and 4 for cognition. Following (Yin et al., 2023), we use the existence, count, position, and color subsets to assess hallucination at object and attribute levels. We report the sum of accuracy and accuracy plus (when the model correctly answers both "Yes" and "No" questions per image).

We discuss evaluation on hallucination benchmarks (MMHal-Bench, GAVIE) in subsection 4.2, and on the visual understanding benchmark (MME) in subsection 4.3.

Baselines. We use InstructBlip (Dai et al., 2024), LLaVA-v1.5-7B (Liu et al., 2023b), LLaVA-v1.6-7B (Liu et al., 2024b), mPlug-OWL (Ye et al., 2023a), and mPlug-OWL2 (Ye et al., 2023b) as the baseline models whose responses will be evaluated with **Pelican**. We also compare with approaches proposed to address hallucinations– Woodpecker (Yin et al., 2023) and Volcano (Lee et al., 2023). Refer to Appendix E for implementation details.

4.2 Hallucination Detection and Mitigation

Table 1 and Table 2 show the performance evaluation of **Pelican** on the hallucination benchmarks– MMHal-Bench and GAVIE. The results highlight the superiority of **Pelican** in both reducing hallucinations and improving performance on the given question-answering task. Table 1 demonstrates improvements by using **Pelican** to refine the responses of several LVLMs. For each row, ✓ and × denote performance with and without applying **Pelican** respectively. Across all three hallucination benchmarks, **Pelican** improves the relevancy and accuracy scores and reduces the hallucination rate. This gain is particular larger for earlier models such as InstructBlip, mPlug-OWL, LLaVA-v1.5 and mPlug-OWL2. We observe the hallucination rate to drop by $\sim 15 - 32\%$ since **Pelican** is able to address issues such as weak visual grounding (by using external tools) and the bias towards "Yes" answers. LLaVA-v1.6 is a recent LVLM that leverages dynamic high-resolution and high-quality data to achieve SOTA performance. We achieve a reduction in hallucination score on MMHal-Bench of 0.3 with **Pelican**. Since we use LLaVA-v1.6 as the VQA tool in **Pelican**, this result shows that our approach does not completely rely on this tool and composes different tools to mitigate hallucinations (refer to Table 2 for results with a different VQA tool). Our algorithm obtains a lower relevancy score on GAVIE for both LLaVA-v1.6 and mPlug-OWL2 since relevancy focuses on instruction following performance. This occurs since the refinement generated by **Pelican** is conditioned only on the original claim and may not directly respond to the original question due to lost context.

Table 2 compares **Pelican** against two methods designed to reduce hallucinations: Woodpecker, which uses a claim verification pipeline, and Volcano, which employs a self-feedback guided refine-

Model	Pelican	MMHal-Bench		GAVIE			MME				
		Score ↑	Hal-Rate ↓	Acc ↑	Rel ↑	Avg ↑	Existence	Count	Position	Color	Total
InstructBlip-7B	×	1.71	0.66	5.61	7.1	6.36	**185**	58	58	143	444
	✓	**2.26**	**0.51**	**6.66**	**7.6**	**7.13**	175	**153**	**147**	**152**	**627**
mPlug-OWL	×	1.34	0.74	3.88	7.1	5.49	95	48	50	55	248
	✓	**2.35**	**0.50**	**6.55**	**7.5**	**7.03**	**175**	**150**	**133**	**153**	**611**
LLaVA-v1.5-7B	×	2.02	0.61	5.64	7.4	6.52	**175**	88	103	105	471
	✓	**2.27**	**0.52**	**6.51**	**7.6**	**7.06**	**175**	**153**	**122**	**140**	**590**
mPlug-OWL2	×	1.88	0.65	5.82	**7.9**	6.86	150	83	58	118	409
	✓	**2.26**	**0.51**	**6.42**	7.6	**7.01**	**175**	**142**	**121**	**140**	**578**
LLaVA-v1.6-7B	×	**3.24**	0.41	6.13	**7.8**	**6.97**	**195**	155	138	190	678
	✓	3.04	**0.38**	**6.33**	7.5	6.92	185	**172**	**178**	180	**715**

Table 1: Results on hallucination (MMHal-Bench, GAVIE) and visual understanding (MME) benchmarks. MMHal-Benchscores range from 0-6, with hallucination rate (Hal-Rate) indicating the proportion of scores below 3. GAVIE measures accuracy (Acc) and relevancy (Rel) on a 0-10 scale, with Avg representing their average. MME reports the sum of accuracy and accuracy plus for each category (object-level and attribute-level), with Total representing the sum across all categories. ✓ denotes results corrected by **Pelican** . The best result for each setting is highlighted in bold.

Model	MMHal-Bench		GAVIE			MME				
	Score ↑	Hal-Rate ↓	Acc ↑	Rel ↑	Avg ↑	Existence	Count	Position	Color	Total
Woodpecker (InstructBlip)	1.71	0.67	5.48	7.4	6.44	160	78	90	100	428
Woodpecker (LLaVA-v1.6-7B)	1.73	0.66	5.38	7.4	6.39	165	78	90	100	433
Volcano-7B	2.21	0.57	5.32	7.5	6.41	195	152	107	160	614
Volcano-13B	2.44	0.52	5.97	**8.1**	**7.04**	**195**	158	118	**185**	656
Pelican	**3.04**	**0.38**	**6.33**	7.5	6.92	185	**172**	**178**	180	**715**

Table 2: Results compared against Woodpecker (Yin et al., 2023) and Volcano (Lee et al., 2023), two methods previously proposed for correcting hallucination. The best scores are highlighted in bold, and the second-best scores are underlined.

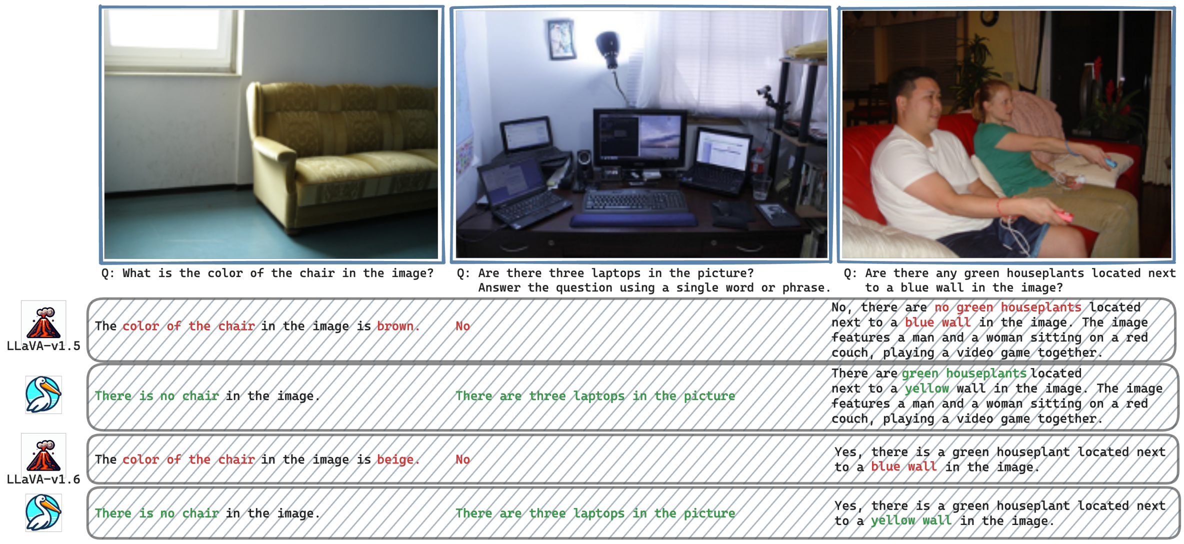
Model	MMHal-Bench		MME				
	Score ↑	Hal-Rate ↓	Existence	Count	Position	Color	Total
Pelican	**2.27**	**0.52**	**175**	**153**	**122**	140	**590**
Pelican w/o sh_var (v)	2.23	0.55	170	148	98	**160**	576
Pelican w/o sh_var(v), sh_comp	2.24	0.54	170	148	97	70	485
Pelican w VQA(LLaVA-v1.5)	2.20	0.55	170	143	123	145	581

Table 3: Ablation study to highlight the performance contribution of different innovations proposed in **Pelican** . “sh_var" refers to shared variable and “sh_comp" refers to shared computations.

ment model. For a fair comparison with Woodpecker, we ran the author’s implementation with both InstructBlip and LLaVA-v1.6 as the VQA tool. Our model shows a 27% drop in hallucination from the best-performing baseline (Volcano-13B), highlighting the robustness and reliability of **Pelican** in handling multimodal hallucinations. Moreover, the 38% reduction in hallucination rate compared to Woodpecker emphasizes our algorithmic innovations in visual claim verification (Section 1).

4.3 Visual Understanding

We evaluate **Pelican** on visual understanding using the MME benchmark, as shown in the last column of Table 1 and Table 2. We focus on four key categories that significantly contribute to hallucination: object-level (existence and count) and attribute-level (position and color). Our results demonstrate that **Pelican** significantly improves visual understanding when integrated with these models. For instance, mPlug-OWL initially underperforms, but when integrated with our approach, we observe a remarkable 146% improvement. Moreover, compared to LLaVA-v1.6, which exhibits superior performance, we achieve an overall 5.4% improvement, particularly over $\sim$ 40% improvement in the count and position categories, which are considered the most challenging. To accurately answer questions about count and position, the model must localize, and reference regions based on size, considering that objects can be in the foreground or background and may blend with



	Q: What is the color of the chair in the image?	Q: Are there three laptops in the picture? Answer the question using a single word or phrase.	Q: Are there any green houseplants located next to a blue wall in the image?
LLaVA-v1.5	The color of the chair in the image is brown.	No	No, there are no green houseplants located next to a blue wall in the image. The image features a man and a woman sitting on a red couch, playing a video game together.
	There is no chair in the image.	There are three laptops in the picture	There are green houseplants located next to a yellow wall in the image. The image features a man and a woman sitting on a red couch, playing a video game together.
LLaVA-v1.6	The color of the chair in the image is beige.	No	Yes, there is a green houseplant located next to a blue wall in the image.
	There is no chair in the image.	There are three laptops in the picture	Yes, there is a green houseplant located next to a yellow wall in the image.

Figure 3: An illustration of hallucination in LVLMs. Three examples showcasing different types of question-answering styles, where both LLaVA-v1.5 and LLaVA-v1.6 hallucinates to the question. **Pelican** refines the answer from these models exhibiting significantly reduced hallucinations.

similar colors from the surroundings. Furthermore, the model needs to contextualize the relative locations of objects within the image. Overcoming these challenges requires the model to perform detailed reasoning about the key elements and their attributes and relationships. Similar patterns are observed in Table 2, where we outperform all baseline models, especially in count and position. However, in some cases, Volcano-13B achieves higher scores (10% and 5% higher accuracy in existence and color) compared to **Pelican**. This can be attributed to Volcano-13B being trained specifically to rewrite its response based on self-critique and using a larger LLM backbone.

4.4 Ablation Studies

Table 3 shows our ablation study conducted on **Pelican** to to assess the impact of different components on performance. The study includes four configurations: original (**Pelican**), without shared variables between sub-claims (**Pelican** sh_var(v)), without shared variables and shared computations (**Pelican** w/o sh_var(v), sh_comp), and lastly we replace LLaVA-v1.6 with LLaVA-v1.5 for VQA tasks ("**Pelican** w VQA"). We evaluated these configurations on MMHal-Bench and MME benchmarks. The results demonstrate that removing shared variables and shared computations between sub-claims leads to a noticeable fall in performance, particularly on the MME benchmark, where the total score drops from 590 to 485. In particular, the drop on position is higher (122 → 98) as the shared variables are responsible for referencing specific object instances which is important for such question (e.g., *Is the motorcycle on the right side of the bus?*). Additionally, replacing LLaVA-v1.6 with LLaVA-v1.5 for VQA tasks results in reduction in the score and a small increase in hallucination rate on MMHal-Bench and a 9 point decrease in Total on MME. This shows both that **Pelican** is robust to the underlying VQA tools, but overall performance will improve with better tools. Figure 3 presents a qualitative comparison with LLaVA-v1.5 and LLaVA-v1.6, illustrating that these models struggle to accurately detect object existence, color, location, and count, leading to overall failure. In contrast, our proposed approach successfully addresses these challenges.

Our performance improvement primarily stems from two key innovations: shared computations and shared variables. Additionally, the claim decomposition process does not involve visual information, making it agnostic to whether LVLMs or LLMs are used, as both models operate solely on textual content. Initial experiments with a REACT agent, using a single LLM for sequential tasks, failed due to brittleness in long-range reasoning and an inability to adapt to the varied nature of code generation. Overall, these findings highlight the effectiveness of our proposed approach and the critical role of each component in reducing hallucinations and enhancing visual understanding.

5 Conclusion

We propose **Pelican** , a novel solution for detecting and mitigating hallucinations via visual claim decomposition and program of thought. **Pelican** is a structured pipeline that breaks down claims into granular sub-claims and uses a robust verification mechanism to ensure thorough checking and validation of each aspect of the visual information. Our approach improves upon the state-of-the-art benchmarks results on hallucination and visual understanding. We also show improved performance on visual tasks where models often struggle and tend to hallucinate, highlighting **Pelican**'s strength in providing comprehensive visual understanding. Our unique principled approach involves decomposing claims into sub-claims, using shared variables, and leveraging in-context examples to guide Python code execution for precise and deterministic answers. Moreover, we employ an LLM to reason over these answers, mimicking human-like comprehension and argumentation. The process systematically addresses and corrects hallucinations, ensuring higher accuracy and reliability in both visual and textual understanding, making it a valuable addition to any LVLM.

6 Limitations

We discuss the limitations of **Pelican** in this section.

1. **Brittleness of Visual Tools:** We used visual tools such as VQA and object detection models to verify individual sub-claims, but these tools often fail. For instance, YOLO and DETR object detectors frequently struggle with (1) very small objects, (2) objects out of their normal context, and (3) visual entities outside the detector's vocabulary. Although Grounding-DINO can detect novel objects, it tends to produce several false positives. Similar failures were observed with the LLaVA-v1.6 VQA tool, which often failed to correctly answer questions regarding object attributes. Despite our efforts to mitigate these limitations through the visual table representation and object class verification, some failures persist.

2. **Randomness and Lack of Consistency in LLM outputs**: Even with the temperature set to 0, we observed randomness in the outputs of our pipeline. Consistently extracting visual entities in the key concept extraction step was particularly challenging. For example, the LLM often broke compound nouns such as “sports ball" into “ball" or “bath towel" to “towel" causing the object detector to fail. When the LLM reduces “bath towel" to “towel," YOLO/DETR fail to detect “towel" because it is not in their list of classes. Even Grounding-DINO, which can detect “bath towel" fails to detect “towel." The sensitivity of outputs to the prompt was manageable in claim decomposition and code generation, but the code generation step sometimes produced code that would fail in the Python interpreter. We addressed this issue by generating the code three times until it did not produce any errors.

3. **Issues with using a claim for verification:** Current pipeline transforms the (question, answer) pair into a claim, which is used for verification. The refined output produced by our pipeline is directly used for evaluation, instead of transforming it back into an answer conditioned on the context. This often results in lower relevancy scores, which measure instruction-following performance.

4. **Ability to handle conflicting evidence:** In specific cases, we observe our pipeline unable to handle conflicting evidence, usually due to tool failures. In such cases, the pipeline may incorrectly declare the claim as true or false, leading to inaccuracies.

7 Ethical Considerations

Detecting and mitigating hallucinations inherently enhances ethical soundness. Our claim-decomposition fosters transparency and empowering enhanced control over ethical considerations. It also fosters broader sets of checks and balances that render ethical abuses more challenging, thus reducing the potential for misuse.

References

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. 2022. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. 2015. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433.

Pepa Atanasova. 2024. Generating fact checking explanations. In *Accountable and Explainable Methods for Complex Reasoning over Text*, pages 83–103. Springer.

Pepa Atanasova, Jakob Grue Simonsen, Christina Lioma, and Isabelle Augenstein. 2022. Fact checking with insufficient evidence. *Transactions of the Association for Computational Linguistics*, 10:746–763.

Zechen Bai, Pichao Wang, Tianjun Xiao, Tong He, Zongbo Han, Zheng Zhang, and Mike Zheng Shou. 2024. Hallucination of multimodal large language models: A survey. *arXiv preprint arXiv:2404.18930*.

Lang Cao. 2023. Enhancing reasoning capabilities of large language models: A graph-based verification approach. *arXiv preprint arXiv:2308.09267*.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Yangyi Chen, Karan Sikka, Michael Cogswell, Heng Ji, and Ajay Divakaran. 2023. Dress: Instructing large vision-language models to align and interact with humans via natural language feedback. *arXiv preprint arXiv:2311.10081*.

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale N Fung, and Steven Hoi. 2024. Instructblip: Towards general-purpose vision-language models with instruction tuning. *Advances in Neural Information Processing Systems*, 36.

Chaoyou Fu, Peixian Chen, Yunhang Shen, Yulei Qin, Mengdan Zhang, Xu Lin, Jinrui Yang, Xiawu Zheng, Ke Li, Xing Sun, Yunsheng Wu, and Rongrong Ji. 2023. Mme: A comprehensive evaluation benchmark for multimodal large language models. *arXiv preprint arXiv:2306.13394*.

Zhijiang Guo, Michael Schlichtkrull, and Andreas Vlachos. 2022. A survey on automated fact-checking. *Transactions of the Association for Computational Linguistics*, 10:178–206.

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

Hongyu Hu, Jiyuan Zhang, Minyi Zhao, and Zhenbang Sun. 2023. Ciem: Contrastive instruction evaluation method for better instruction tuning. *arXiv preprint arXiv:2309.02301*.

Shaohan Huang, Li Dong, Wenhui Wang, Yaru Hao, Saksham Singhal, Shuming Ma, Tengchao Lv, Lei Cui, Owais Khan Mohammed, Barun Patra, et al. 2024. Language is not all you need: Aligning perception with language models. *Advances in Neural Information Processing Systems*, 36.

Liqiang Jing and Xinya Du. 2024. Fgaif: Aligning large vision-language models with fine-grained ai feedback. *arXiv preprint arXiv:2404.05046*.

Neema Kotonya and Francesca Toni. 2020. Explainable automated fact-checking: A survey. *arXiv preprint arXiv:2011.03870*.

Nayeon Lee, Belinda Z Li, Sinong Wang, Wen-tau Yih, Hao Ma, and Madian Khabsa. 2020. Language models as fact checkers? *arXiv preprint arXiv:2006.04102*.

Seongyun Lee, Sue Hyun Park, Yongrae Jo, and Minjoon Seo. 2023. Volcano: mitigating multimodal hallucination through self-feedback guided revision. *arXiv preprint arXiv:2311.07362*.

Miaoran Li, Baolin Peng, Michel Galley, Jianfeng Gao, and Zhu Zhang. 2023a. Self-checker: Plug-and-play modules for fact-checking with large language models. *arXiv preprint arXiv:2305.14623*.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Wayne Xin Zhao, and Ji-Rong Wen. 2023b. Evaluating object hallucination in large vision-language models. *arXiv preprint arXiv:2305.10355*.

Fuxiao Liu, Kevin Lin, Linjie Li, Jianfeng Wang, Yaser Yacoob, and Lijuan Wang. 2023a. Mitigating hallucination in large multi-modal models via robust instruction tuning. In *The Twelfth International Conference on Learning Representations*.

Hanchao Liu, Wenyuan Xue, Yifei Chen, Dapeng Chen, Xiutian Zhao, Ke Wang, Liping Hou, Rongjun Li, and Wei Peng. 2024a. A survey on hallucination in large vision-language models. *arXiv preprint arXiv:2402.00253*.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2023b. Improved baselines with visual instruction tuning. *arXiv preprint arXiv:2310.03744*.

Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. 2024b. Llava-next: Improved reasoning, ocr, and world knowledge.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2024c. Visual instruction tuning. *Advances in neural information processing systems*, 36.

Zhiliang Peng, Wenhui Wang, Li Dong, Yaru Hao, Shaohan Huang, Shuming Ma, and Furu Wei. 2023. Kosmos-2: Grounding multimodal large language models to the world. *arXiv preprint arXiv:2306.14824*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: language models can teach themselves to use tools. 2023. *arXiv preprint arXiv:2302.04761*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. arxiv. *arXiv preprint arXiv:2303.17580*.

Zhiqing Sun, Sheng Shen, Shengcao Cao, Haotian Liu, Chunyuan Li, Yikang Shen, Chuang Gan, Liang-Yan Gui, Yu-Xiong Wang, Yiming Yang, et al. 2023. Aligning large multimodal models with factually augmented rlhf. *arXiv preprint arXiv:2309.14525*.

Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11888–11898.

Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. 2024. Yolov9: Learning what you want to learn using programmable gradient information. *arXiv preprint arXiv:2402.13616*.

Haoran Wang and Kai Shu. 2023. Explainable claim verification via knowledge-grounded reasoning with large language models. *arXiv preprint arXiv:2310.05253*.

Jiayang Wu, Wensheng Gan, Zefeng Chen, Shicheng Wan, and S Yu Philip. 2023. Multimodal large language models: A survey. In *2023 IEEE International Conference on Big Data (BigData)*, pages 2247–2256. IEEE.

Qinghao Ye, Haiyang Xu, Guohai Xu, Jiabo Ye, Ming Yan, Yiyang Zhou, Junyang Wang, Anwen Hu, Pengcheng Shi, Yaya Shi, et al. 2023a. mplug-owl: Modularization empowers large language models with multimodality. *arXiv preprint arXiv:2304.14178*.

Qinghao Ye, Haiyang Xu, Jiabo Ye, Ming Yan, Haowei Liu, Qi Qian, Ji Zhang, Fei Huang, and Jingren Zhou. 2023b. mplug-owl2: Revolutionizing multi-modal large language model with modality collaboration. *arXiv preprint arXiv:2311.04257*.

Shukang Yin, Chaoyou Fu, Sirui Zhao, Tong Xu, Hao Wang, Dianbo Sui, Yunhang Shen, Ke Li, Xing Sun, and Enhong Chen. 2023. Woodpecker: Hallucination correction for multimodal large language models. *arXiv preprint arXiv:2310.16045*.

Tianyu Yu, Yuan Yao, Haoye Zhang, Taiwen He, Yifeng Han, Ganqu Cui, Jinyi Hu, Zhiyuan Liu, Hai-Tao Zheng, Maosong Sun, et al. 2023. Rlhf-v: Towards trustworthy mllms via behavior alignment from fine-grained correctional human feedback. *arXiv preprint arXiv:2312.00849*.

Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. 2023. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*.

A Predicates and Tools

We list out the predicates and tools invoked for each predicates in Table 4. We defined these predicates by using a taxonomy of question-types from prior works on VQA (Antol et al., 2015) and multimodal hallucinations (Bai et al., 2024).

B Visual Table

In subsection 3.1, we propose a pipeline to detect objects referenced in a claim by integrating both YOLOv9 and Grounding DINO. Here, we provide a detailed overview of the pipeline that takes the best of both worlds, YOLOv9 which is a closed-vocabulary set and Grounding DINO which offers an open-vocabulary set, allowing us to balance the strengths and weaknesses of both tools. Closed-vocabulary models like YOLOv9 are reliable for the classes they are trained on but fail to detect novel classes. Conversely, open-vocabulary models like Grounding DINO can detect novel objects but may produce false positives, such as mistakenly identifying a pillow that isn't present.

To address these issues, our pipeline first checks if the object class is present in YOLOv9's closed set. If the class exists and YOLOv9 does not detect the object, we conclude the object is not found. If the class is not present, we use Grounding DINO to verify the object's presence. Additionally, for the GAVIE benchmark, we incorporate a bounding box verification tool. This tool uses the coordinates of detected objects and queries a VQA module (LLaVA-v1.6) to confirm if the objects are indeed present in the image, enhancing the accuracy of our detection system.

C Instruction & Prompts Templates

In this section, we provide the list of prompt templates used in various steps of **Pelican**.

We put the instructions and prompts we use for extracting visual elements, claim decomposition, program of thought verification and integrating verification.

D Additional Related Works

Apart from the works presented in section 2 we discuss a few other relevant works.

Large Visual Language Models and Hallucinations Several approaches have been proposed to mitigate hallucinations in Large Visual Language Models (LVLMs). One line of work employs variants of Reinforcement Learning from Human Feedback (RLHF) to enhance LVLMs (Yu et al., 2023; Chen et al., 2023; Jing and Du, 2024). DRESS (Chen et al., 2023), for instance, utilizes fine-grained feedback in the form of critique and refinement obtained from Language Models (LLMs) to align the LVLM with human preferences. On the other hand, RLHF-V acquires segment-level feedback directly from humans and applies dense direct preference optimization to achieve alignment. A recent work, Volcano (Lee et al., 2023), proposes distilling the capability of self-critique within the LVLM to improve performance during inference. This is accomplished by generating instruction-following data containing critique and improved responses from LLMs, which is then used to fine-tune the LLaVa-1.5 model.

E Implementation Details:

Pelican Our pipeline uses specific prompts, listed in Appendix C, for different components of our pipeline. These prompts were fed to GPT-4o for producing responses. For accurate understanding of visual content, we use several tools as described in Appendix A. Specifically, we realize these tools using:

- **Object Detection:** We used Yolo-v9 (Wang et al., 2024) or DETR (Carion et al., 2020) based on benchmark to localize image regions specific to entities mentioned in the claim. Grounding Dino was used to detect objects not present in the vocabularly of these detectors.
- **VQA:** We used LLaVA-v1.6 for visual question answering the attribute related questions related to the image.
- `Relative_location`: We implemented this function in python to determine the relative location (left, right, top, bottom) of object1 with respect to object2 by comparing their bounding boxes.

We also use Pandas library for efficient data manipulation and analysis of the visual table. We would like to note that we guided the LLM generating Python code to use these tools with a few in-context examples. In this case, we often found the LLM to compose different combinations than what we had provided in these examples.

Claim Generation

System Message

Given a question answer pair about an image convert it into a claim. Avoid removing any information from the question or the answer.

Examples:
{in-context examples}

Human

Question: **{claim}**

Answer: **{predicates}**

Claim:

Figure 4: Prompt template to generate a claim using both the question and answer.

Claim To Visual Element

System Message

Given a question answer pair about an image convert it into a claim. Avoid removing any information from the question or the answer.
You are an expert in analyzing a given caption about an image and extracting key elements such as objects, nouns, words/text present in the image.
Run the extraction process in two steps.

Use the following steps for extraction:
Step 1:
Extract and list all physical objects mentioned in the given caption and phrases. Only include objects that have a tangible presence and can be perceived visually. Exclude abstract concepts, ideas, or entities that do not have a concrete physical existence. This includes:
- Objects (or stuff classes) that occupy space like car, ball, couch, tree, sky, water.
- Compound nouns or phrases that refer to a single entity, such as "Samsung Galaxy phone", "tennis ball", "toilet seat", "carriage seat", "stop sign". Keep these together and do not split them into separate parts.
- Text that can be seen or read written on boards, signs, or in any region of the image using Optical Character Recognition (OCR).

Exclude abstract concepts (quality, action, feeling, state) and descriptors/adjectives (appearance, personality, emotion, trait) preceding the nouns.
Ensure to capture compound nouns as a single entity without splitting them into separate parts.

Avoid including elements that refer to the image itself (e.g., image, photo, picture, scene) they do not provide meaningful information about the image content.

Step 2:
Remove the adjectives (appearance, personality, emotion, trait) from the elements identified in Step 1 without removing the object name completely.
Convert the elements into their singular form, except for uncountable nouns (e.g., water, sand) or compound nouns that refer to a single entity (e.g., "Red Bull").

For example:
- Change 'blue sky' to 'sky'
- Change 'black phone' to 'phone'
- Change 'cars' to 'car'
- Change 'people' to 'person'
- Keep 'sunglasses' as 'sunglasses'
- Keep 'Samsung Galaxy phone' as 'Samsung Galaxy phone'
- Keep 'tennis ball' as 'tennis ball'

Examples:
{in-context examples}

Human

Caption: **{caption}**

Figure 5: Template for prompting LLM to perform key concept extraction.

Verify Boxes

System Message

You are given a text response from an expert AI model about whether an image contains a specific object. The AI model's response will be in natural language.
- If the response mentions a different object than the one originally specified, then extract only the most relevant object(s) to replace the original object.
- If the response clearly states that the specified object is not present in the image, then respond with "NA".
- If the AI model's response confirms the presence of the original object, then simply return the name of that object.

Examples:
{in-context examples}

Human

Caption: **{caption}**

Figure 6: Prompt template to detect the existence of an object in a boxed area.

Claim Decomposition

System Message

You are an expert at first order logic. The task is to analyze claims about an image:

1. Define all the predicates in the claim.
   - Focus on key objects, scene, relationships, and attributes mentioned in the claim.
   - Avoid including elements that refer to the image itself (e.g., image, photo, picture, scene) in the predicates, as they do not provide meaningful information about the content of the image.
   - Use the following predicates:
     - Exists: Verify the presence or absence of an object/ocr-token
     - Scene: Verify the presence or absence of a scene
     - Count: Verify the number of objects
     - Attribute_name: Verify the attribute of an object e.g. Color, Type, Wearing
     - Position: Verify the position of an object relative to another object
   - When dealing with complex claims involving multiple objects or relationships, CREATE intermediate variables using 'Select' to focus on specific objects based on their attributes or relationships. You can then use $variable_name to refer to these intermediate variables in subsequent predicates.
2. Parse the predicates into a relevant chain of questions.

Examples:
{in-context examples}

Human

Claim: **{claim}**

Figure 7: Template for prompting LLM to decompose the claim into a list of predicates.

Evidence

System Message

You are an expert at writing python code for answering questions based on prior context, given pandas table, and given tools.The answers are being used to collect evidence to verify questions about a claim in the next step.

You are provided with a table named `table_objects` with the following columns:
- `object`: name of the object
- `box`: Bounding box in [center_x, center_y, width, height] format. Coordinates are normalized based on the size of the image.

This table was generated by computer vision algorithms such as object detectors to obtain an initial list of detections. Multiple rows might contain the same object (e.g. "truck" and "trucks" or "stop sign" and "sign).

You can use the following tools:

TOOLS:

1. iou(box1, box2):
   Return the IoU (Intersection over Union) score between the two bounding boxes. `box1` and `box2` should lists of coordinates. Use an iou >= 0.3 to determine two different objects have a considerable overlap.

2. grounded_vqa(image, question, bbox=None):
   Use this tool to answer questions about the image. The `image` argument should be a file path, numpy array, or PIL image. The `bbox` argument is optional and should be a tuple or list of coordinates in the format (center_x, center_y, width, height). Use the `bbox` argument to crop the specific region of interest for questions focusing on attributes of objects. For questions requiring contextual reasoning (such as spatial reasoning), omit the `bbox` argument. The tool returns a tuple containing the answer and its confidence score (low, medium, high).
   DO NOT use this tool for counting objects as it can return incorrect answers.
   Examples:
   - Non-contextual question: "What color is the car?" (use `bbox` to focus on the car)
   - Contextual question: "Where is the car located?" (omit `bbox` to consider the entire image)

3. relative_location(box1, box2, obj1, obj2):
   Determine the relative location of obj1 with respect to obj2 based on their bounding boxes. The `box1` and `box2` arguments should be tuples or lists of coordinates in the format (center_x, center_y, width, height). The `obj1` and `obj2` arguments are the names of the objects being compared. The function prints the relative location (in terms of left, right, top, bottom) of obj1 with respect to obj2 based on their bounding boxes.

4. get_combined_box(box1, box2):
   Combine two bounding boxes to create a larger box that encompasses both objects. The `box1` and `box2` arguments should be tuples or lists of coordinates in the format (center_x, center_y, width, height). The function returns a new bounding box that includes both objects. Given this table, the `image_url`, user's question, and prior context, write Python code to answer the question as best as you can. If multiple objects of the same type are found, consider their IoUs and confidences to determine the most relevant one. If the `grounded_vqa` tool returns a low confidence score, indicate that the answer may not be reliable. It is also okay to invoke multiple tools to gain more evidence. Use the prior context to accomplish the task efficiently by reusing the information already provided.The Python code should be returned within markdown quotes ```python ```. Also, make sure to explicitely print the answers in the code.

Examples:
{in-context examples}

Human

Prior Context: **{context}**

Question: **{question}**

Figure 8: Prompt Template for function calling based on the task type for, e.g., to answer about visual attributes it calls grounded_vqa.

Integrate Verification

System Message

You are an AI assistant trained to verify and correct claims made by other AI algorithms about an image. The claims could be incorrect due to logical inconsistencies, visual hallucinations, or faulty reasoning. Your task is to:

1. Assess the validity of the claim based on the evidence provided in the form of predicates and follow-up questions with answers.
2. Rewrite the claim if it is incorrect by providing a corrected version that aligns with the evidence.

Input Format:
- Claim: A statement about the image that needs to be verified.
- Predicates: A list of conditions that need to be checked to verify the claim.
- Follow-up Questions: A series of questions and answers that provide evidence for the predicates.

For each input, follow these steps:
1. Analyze the relevance of each predicate and follow-up question to the claim.
2. Consider the confidence scores provided with the answers to the follow-up questions. If the confidence scores are low or not provided, treat the evidence as less reliable.
3. If the follow-up questions provide conflicting information, prioritize the evidence with higher confidence scores or mention the contradiction in your reasoning.
4. Identify any logical inconsistencies or faulty reasoning in the claim, such as statements that contradict the evidence or make unsupported assumptions.
5. Provide a step-by-step reasoning for your decision, citing the key evidence that led to your conclusion.
6. Give a **verdict** as "Verdict:\nCorrect" if the claim is supported by the evidence or "Verdict:\nIncorrect" if the claim is not supported or contradicted by the evidence.
7. If the claim is incorrect, rewrite the claim based on the evidence provided, ensuring that the rewritten claim is consistent with the available information. If the claim is correct then write "Rewrite: NA" to help with parsing.

Note: Focus only on verifying the specific claim provided, without considering any additional information that may be present in the image but not mentioned in the claim.

Examples:
{in-context examples}

Human

Claim: **{claim}**

Predicates: **{predicates}**

Followup Questions: **{evidence}**

Figure 9: Prompt template to summarize the sub-claim responses and verify whether the claim was correct. Finally, rewrite that claim (if it has hallucinations) based on the responses.

Predicates	Tools	Reasoning
Exist	Object Detector	Trained specifically to detect and locate objects
OCR	VQA	Trained on OCR-based datasets
Count	Object Detector	Detecting every instance of object, assists in counting.
Attribute	VQA	Trained on paired image, caption datasets such as COCO, Visual Genome
Location (rel)	VQA, `Relative_location`	Bbox from object detectors provide relative location
Scene	VQA	Trained on caption datasets

Table 4: Different predicates along with tools and the type of reasoning involved. `Relative_location`is a function to determine the relative location of an object relative to another object based on their bounding boxes.

Baselines: For our experiments, we ran the inference for all the models locally with temperature set to 0.

- InstructBlip (Dai et al., 2024): We used the model “instructblip-vicuna-7b"[1] provided by authors on Huggingface.
- LLaVA-v1.5-7B (Liu et al., 2023b) We used the code and the model provided[2] by the authors.
- LLaVA-v1.6-7B (Liu et al., 2024b): We used the model “llava-v1.6-mistral-7b-hf"[3] provided by the authors on Huggingface.
- mPlug-OWL (Ye et al., 2023a): We use the model and the code provided[4] by the authors.
- mPlug-OWl2 (Ye et al., 2023b):We use the model and the code provided[5] by the authors.
- Volcano (Lee et al., 2023): We used the code[6] provided by the authors.
- Woodpecker (Yin et al., 2023): We used the code[7] provided by the authors. Woodpecker originally uses BLIP2 as the VQA tool, which can limit its performance since BLIP2 is not instruction-tuned. We thus replaced BLIP2 with Instruct-blip and LLaVA-v1.7-7B for a fair comparion with our work.

Evaluation with LLMs: We used gpt-4-0613 for evaluating MMHal-Bench. GPT-4o was used for evaluation with GAVIE due to lower API costs and claims of similar quality.

Model	Benchmark	Time	Cost
LLaVA-1.5	MMHal-Bench	1.185s	N/A
Our Model	MMHal-Bench	5m58s	\$0.234
LLaVA-1.5	MME-existence	0.184s	N/A
Our Model	MME-existence	2.275s	\$0.011

Table 5: Inference time and cost comparison of LLaVA-1.5 and our model across MMHal-Bench and MME-existence benchmarks.

Comparative Analysis of Inference Time and API Costs: We provide the inference time and costs (averaged over 10 examples) for the MME and MMHal-Bench benchmarks using LLaVA-1.5 and the proposed model (Pelican). MME has lower computational costs than MMHal-Bench due to its simpler claims involving single objects or attributes. For Pelican, all tools except the LLM were hosted locally. Pelican's slower performance is due to its stagewise approach; we did not optimize the VQA and detector models for batched requests, which could yield a 2-3x speedup. Additionally, parallelizing tool calls could improve efficiency. The time is also impacted by the number of tool calls, which is smaller for MME (simpler claims) compared to MMHal-Bench. The per-sample API cost varies by dataset, but remains relatively low.

Additional Evaluation on POPE: Additionally, we add results on POPE (Li et al., 2023b) by selecting 300 examples, following Woodpecker, with 100 samples selected from each of the three settings. We observe consistent improvements over baseline LVLMs, except for a minor accuracy drop in LLaVA-v1.6-7B (Liu et al., 2024b), which was offset by higher precision. We notice the accuracy remains at 91% across all three models with **`Pelican`**. This consistency is attributed to the Yes/No nature of the questions in the POPE dataset, allowing us to achieve 91% accuracy with LLaVA-v1.6 as the VQA model, regardless of the baseline models' errors. The few failure cases are primarily due to tool

[1] https://huggingface.co/Salesforce/instructblip-vicuna-7b
[2] https://github.com/haotian-liu/LLaVA
[3] https://huggingface.co/llava-hf/llava-v1.6-mistral-7b-hf
[4] https://github.com/X-PLUG/mPLUG-Owl
[5] https://github.com/X-PLUG/mPLUG-Owl
[6] https://github.com/kaistAI/Volcano
[7] https://github.com/BradyFU/Woodpecker

Model	Acc.	Precision	Recall
llava-v1.6-mistral-7b	91.67	90.85	92.67
+Ours	91.00	93.62	88.00
instructblip-vicuna-7b	86.00	85.06	87.33
+Ours	91.00	93.62	88.00
mplug-owl2-llama2-7b	80.66	75.00	92.00
+Ours	91.00	93.62	88.00

Table 6: Performance comparison of baseline models and “+Ours" on POPE.

failures.

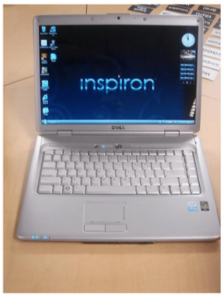
F Qualitative Results

Refer to Figure 10 and Figure 12 to understand the pipeline depicted for `Pelican`. Figure 10 illustrates a simple example from MME, where mPlug-OWL incorrectly detects a potted plant that doesn't exist. We create a predicate to verify the presence of the “potted plant" followed by its corresponding question and answer. The verification output reasons, leading us to predict the claim as incorrect. We then refine the output by stating, “There is no potted plant in this image." This example also highlights the brittleness of the tools (refer to limitations in section 6), as “potted plant" is not a class in YOLO/DETR, so the code defaults to Grounding-DINO (refer to implementation in Appendix E), which detects the plant with low confidence.

Figure 12 demonstrates a more complex process, where LLaVA-v1.5 misidentifies a face painting as glasses. The detection tools find a “man" with high confidence via YOLO, and “glasses" are detected again via Grounding-DINO with low confidence. Next, we prepare a list of predicates and follow-up questions. Next, we show the Python code for each follow-up question. Finally, the evidence gathered is used for reasoning to predict the claim as incorrect.

G Licensing Information

The images used in our paper are sourced from MSCOCO, Visual Genome and OpenImages that are under different Creative Commons licenses. Images Figure 1, left in Figure 3, Figure 12 are licensed under CC BY 2.0, which allows for sharing and adaptation with appropriate attribution. center in Figure 3, Figure 10 are under CC BY-NC 2.0, permitting non-commercial use with attribution. Image on the right in Figure 3 is taken from Visual Genome dataset which is under CC BY 4.0 which allows for sharing and adaptation with appropriate attribution. Users should refer to the specific license links provided for each image to ensure compliance with usage terms.



Claim: There is a potted plant in this image

Visual Table

Object	Count	Confidence
Plant	1	31%

Predicates

-- Exists(potted_plant, Yes) # Verify that there is a potted plant

Follow up Questions

-- Is there a potted plant in the image?

Program Of Thought

Q1.

```
# Type of reasoning involved: object recognition
# Filter out rows with objects that could be potted plant
filtered_table_with_potted_plant = table_objects
                                       [table_objects.object.apply(lambda x: "potted plant" in x.lower())]

if len(filtered_table_with_potted_plant):
    print("Yes, there is a potted plant in the image.")
    for ii, (index, row) in enumerate(filtered_table_with_potted_plant.iterrows()):
        print("Potted plant", ii+1, "is located at", row.box)
else:
    print("There is no potted plant in the image.")
```

Evidence

-- Is there a potted plant in the image? There is no potted plant in the image.

Verification Output

Reasoning:
Reasoning:

1. The claim states that there is a potted plant in the image. The predicate to verify this is "Exists(potted_plant, Yes), " which requires confirmation of the presence of a potted plant. The follow-up question directly addresses this by asking if there is a potted plant in the image, and the answer is "There is no potted plant in the image."
2. Since the follow-up question clearly states that there is no potted plant in the image, the claim is not supported by the evidence.
3. Since the follow-up question clearly states that there is no potted plant in the image, the claim is not supported by the evidence.>)
4. Since the evidence clearly indicates that the man is not wearing glasses, the claim is not supported by the evidence.

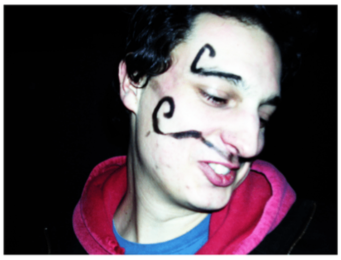
Verdict

Incorrect

Rewrite

There is no potted plant in this image.

Figure 10: From MME benchmark category existence, the claim is generated from the incorrect answer given by mPlug-OWL. Below show the entire detection and mitigation pipeline of **Pelican**.



Claim: The man in the image is wearing black glasses.

Visual Table

Object	Count	Confidence	bbox
Man	1	79.7%	[0.57, 0.49, 0.76, 0.99]
Glasses	1	28.5%	[0.57, 0.49, 0.76, 0.99]

Predicates

-- Exists(man, Yes) # Verify that there is a man
-- Exists(glasses, Yes) # Verify that there are glasses
-- Wearing(man, glasses) # Verify that the man is wearing glasses
-- $man_wearing_glasses <- Select(man, glasses) # Create an intermediate variable for the man wearing glasses
-- Color(glasses, black) # Verify that the glasses are black

Follow up Questions

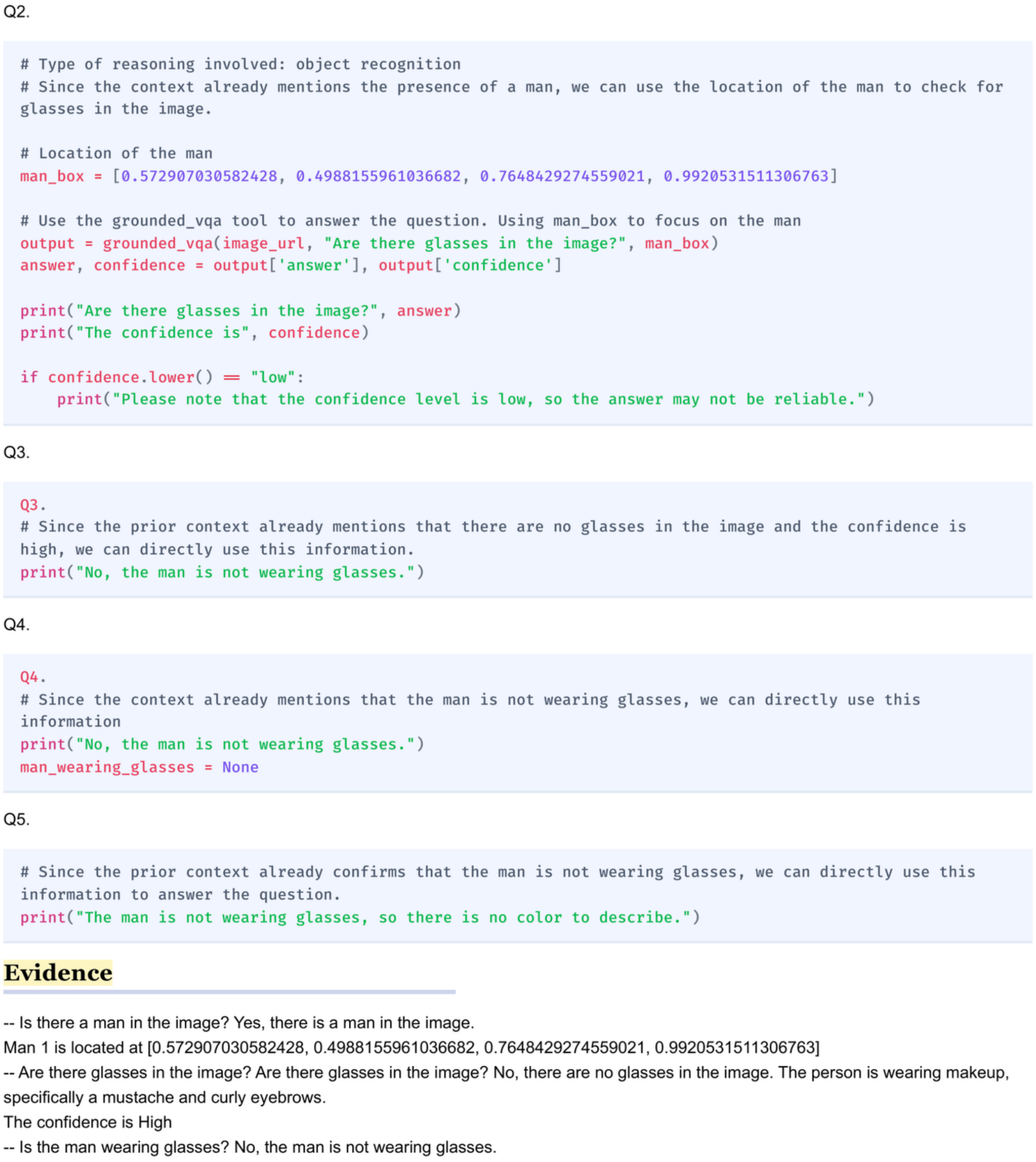
-- Is there a man in the image?
-- Are there glasses in the image?
-- Is the man wearing glasses?
-- man_wearing_glasses <- Select(man, glasses)
-- What color are the glasses that $man_wearing_glasses is wearing?

Program Of Thought

Q1.

```
# Type of reasoning involved: object recognition
# Filter out rows with objects that could be a man
filtered_table_with_man = table_objects[table_objects.object.apply(lambda x: "man" in x.lower())]

if len(filtered_table_with_man):
    print("Yes, there is a man in the image.")
    for ii, (index, row) in enumerate(filtered_table_with_man.iterrows()):
        print("Man", ii+1, "is located at", row.box)
else:
    print("There is no man in the image.")
```



Q2.

```
# Type of reasoning involved: object recognition
# Since the context already mentions the presence of a man, we can use the location of the man to check for
glasses in the image.

# Location of the man
man_box = [0.572907030582428, 0.4988155961036682, 0.7648429274559021, 0.9920531511306763]

# Use the grounded_vqa tool to answer the question. Using man_box to focus on the man
output = grounded_vqa(image_url, "Are there glasses in the image?", man_box)
answer, confidence = output['answer'], output['confidence']

print("Are there glasses in the image?", answer)
print("The confidence is", confidence)

if confidence.lower() == "low":
    print("Please note that the confidence level is low, so the answer may not be reliable.")
```

Q3.

```
Q3.
# Since the prior context already mentions that there are no glasses in the image and the confidence is
high, we can directly use this information.
print("No, the man is not wearing glasses.")
```

Q4.

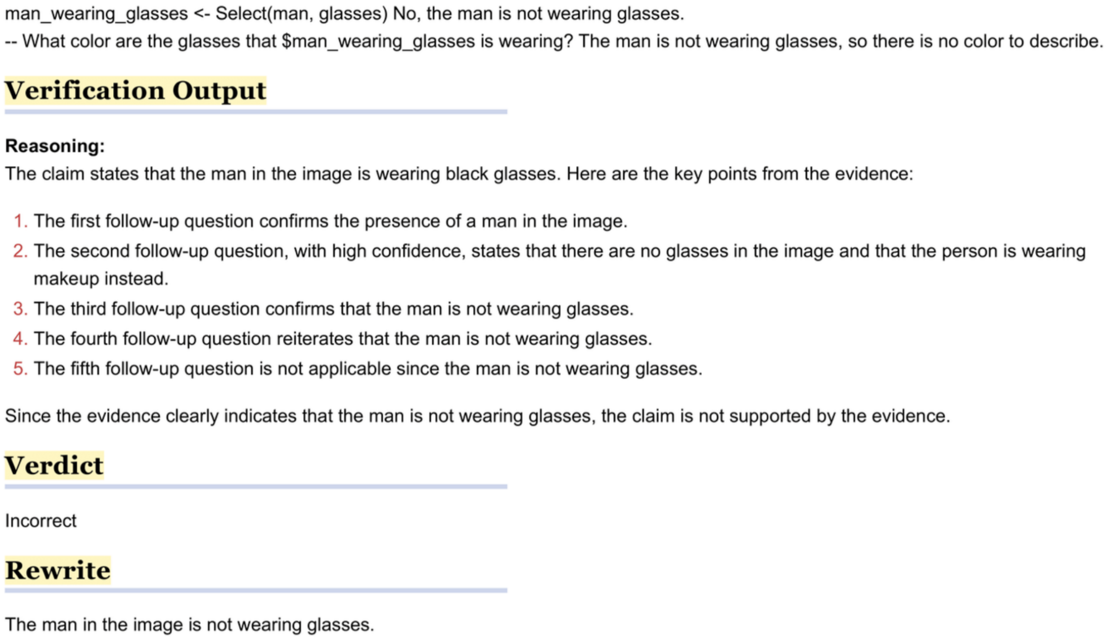
```
Q4.
# Since the context already mentions that the man is not wearing glasses, we can directly use this
information
print("No, the man is not wearing glasses.")
man_wearing_glasses = None
```

Q5.

```
# Since the prior context already confirms that the man is not wearing glasses, we can directly use this
information to answer the question.
print("The man is not wearing glasses, so there is no color to describe.")
```

Evidence

-- Is there a man in the image? Yes, there is a man in the image.
Man 1 is located at [0.572907030582428, 0.4988155961036682, 0.7648429274559021, 0.9920531511306763]
-- Are there glasses in the image? Are there glasses in the image? No, there are no glasses in the image. The person is wearing makeup, specifically a mustache and curly eyebrows.
The confidence is High
-- Is the man wearing glasses? No, the man is not wearing glasses.



man_wearing_glasses <- Select(man, glasses) No, the man is not wearing glasses.
-- What color are the glasses that $man_wearing_glasses is wearing? The man is not wearing glasses, so there is no color to describe.

Verification Output

Reasoning:
The claim states that the man in the image is wearing black glasses. Here are the key points from the evidence:

1. The first follow-up question confirms the presence of a man in the image.
2. The second follow-up question, with high confidence, states that there are no glasses in the image and that the person is wearing makeup instead.
3. The third follow-up question confirms that the man is not wearing glasses.
4. The fourth follow-up question reiterates that the man is not wearing glasses.
5. The fifth follow-up question is not applicable since the man is not wearing glasses.

Since the evidence clearly indicates that the man is not wearing glasses, the claim is not supported by the evidence.

Verdict

Incorrect

Rewrite

The man in the image is not wearing glasses.

Figure 12: From MMHal-Bench, the claim is generated from the incorrect answer given by LLaVA-v1.5. Below show the entire detection and mitigation pipeline of `Pelican`. Part 2