USENIX Security ’25 Artifact Appendix: Self-interpreting Adversarial Images

Tingwei Zhang[†] Collin Zhang[†] John X. Morris[†] Eugene Bagdasarian[§] Vitaly Shmatikov[†]
[†]Cornell Tech [§]University of Massachusetts Amherst
{tingwei, collinzhang, jxm3}@cs.cornell.edu eugene@cs.umass.edu shmat@cs.cornell.edu

A Artifact Appendix

A.1 Abstract

This artifact provides the implementation and evaluation framework for the paper *Self-Interpreting Adversarial Images*, which introduces a novel class of indirect, cross-modal prompt injection attacks on Visual Language Models (VLMs). These attacks embed hidden meta-instructions in natural-looking images to covertly influence model outputs toward adversary-specified meat-objectives (e.g., sentiment, political bias, language, or style), while maintaining plausible and coherent responses.

The artifact includes code, data, and instructions to reproduce the paper’s core results on MiniGPT-4, LLaVA, and InstructBLIP. It supports generation of adversarial image soft prompts, evaluation under diverse meta-objectives, preservation of semantics, and robustness testing across perturbation norms and defense strategies.

This artifact ensures reproducibility, facilitates research on cross-modal attacks, and provides a practical framework for evaluating VLM vulnerabilities and defenses.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The goal of this work is to support the safe deployment of VLMs and to encourage research on defenses against cross-modal prompt injection attacks. All datasets and models used in this artifact are publicly available, and the artifact itself is non-destructive and intended solely for research and evaluation purposes.

A.2.2 How to access

The code is available at this URL: `https://figshare.com/articles/software/Self-interpreting_Adversarial_Images/29219051?file=56929889`

A.2.3 Hardware dependencies

The artifact evaluation requires the following hardware configuration:

- **CPU**: We used dual Intel(R) Xeon(R) Gold 6448Y processors with 64 cores in total, but comparable hardware will likely suffice.
- **Memory**: Our machine had 256 GiB of system RAM, however much less will suffice.
- **GPU**: Our experiments were run on either two NVIDIA A40 GPUs (each with 48 GB of memory) or a single NVIDIA A100 GPU with 80 GB of memory. Smaller GPUs may still work, but will likely require a reduced batch size.
- **Storage**: At least 40 GiB to store datasets, model checkpoints, and experiment results.

A.2.4 Software dependencies

The artifact requires the following operating system and essential software packages:

- **Operating System**: The code has been tested on Ubuntu 20.04 LTS. It is recommended to use this OS for compatibility and to ensure reproducibility of the results. Other Linux distributions may work but are not guaranteed.
- **Python**: Python 3.9 or higher is required. It is recommended to use a virtual environment (such as Conda) to manage dependencies. For better compatibility and to avoid package conflicts, install the dependencies for each model independently.
- **CUDA and cuDNN**: To utilize GPU acceleration, ensure that CUDA 11.0 or higher and cuDNN are installed.

A.3 Set-up

A.3.1 Installation

Follow the **Setup** instructions in the `README.md` file of our GitHub repository to download the model checkpoints for each model. Additionally, set up the environment according to the instructions provided in each model's repository.

A.3.2 Basic test

Activate your conda environment, then run the following commands:

1. Generate an image soft prompt:

```
python minigpt_visual_attack.py \
    --data_path instruction_data/coco_1/
    Sentiment/dataset.csv \
    --instruction negative \
    --image_file clean_images/coco_1.jpg \
    --save_dir output/example
```

2. Perform inference on the generated image:

```
python -u minigpt_inference.py \
    --gpu_id 0 \
    --data_path instruction_data/coco_1/
    Sentiment/dataset.csv \
    --image_file output/example/bad_prompt.bmp \
    --output_file output/example/result.jsonl
```

You can customize the attack by replacing the image, dataset, model, and instruction parameters with your own choices. The model's response will be saved as `result.jsonl` in the specified output directory.

A.4 Evaluation Workflow

A.4.1 Major claims

(C1): We evaluate our method on the available open-source VLMs with meta-instructions corresponding to different meta-objectives and show that **image perturbations encoding meta instructions are as effective as steering models' outputs as explicit instructions**. This corresponds to Table 2 in the paper. This is proven by experiment (E1).

(C2): We demonstrate that **meta-instructions preserve image semantics**. We use several metrics, including embedding and structural similarity(SSIM) and oracle LLM evaluation, to show that target VLMs' responses are based on the visual content of input images. This corresponds to Table 3 and Table 4 in the paper. This is proven by experiments (E2) and (E3).

(C3): We analyze stealthiness of our method and performance under different perturbation norms, demonstrating that **our method remains effective across various perturbation constraints**. This corresponds to Table 5 in the paper. This is proven by experiment (E4).

(C4): We analyze **transferability of image soft prompts**, showing that the attack can be effective even when the adversary does not know which specific VLM the victim will be applying to the adversary's images. This corresponds to Table 6 in the paper. This is proven by experiment (E5).

(C5): We survey several **countermeasures** against our attack based on feature distillation and anomaly detection based on the embedding consistency of augmentations. This corresponds to Table 7 and Table 8 in the paper. This is proven by experiments (E6) and (E7).

A.4.2 Experiments

(E1): *[Image soft prompt generation and evaluation] [2 human-hours + 200 compute-hours (can be reduced with more or larger GPUs, or run part of the experiments)]: This experiment generates the results of Table 2 in the paper.*
Execution: Read `script/README.md` for generating and evaluating single images. To reproduce the entire Table 2, run all shell files in `script/` with the naming format `script_{instruction}_{model}`, which include the code for generating attacks, inferencing models with image soft prompts, and baselines.
Results: The model output will be saved to JSON files. Run the Jupyter notebook `eval_instruction_following.ipynb` with the correct paths to evaluate the attack success.

(E2): *[Image semantics preservation - embedding and SSIM similarity] [1 human-hour + 2 compute-hours]: This experiment generates the results of Table 3 in the paper.*
Execution: For embedding and SSIM similarity analysis, use the `eval_content_preserving.ipynb` notebook. Update the "model_name" and "sim_metric" parameters in the first code block, and ensure all file paths are correctly configured.
Results: The similarity metrics will be computed and presented in the notebook. Embedding similarity results fill columns 1, 3, and 5 of Table 3, while SSIM similarity results fill columns 2, 4, and 6 of Table 3.

(E3): *[Image semantics preservation - oracle LLM evaluation] [2 human-hours]: This experiment generates the results of Table 4 in the paper.*
Execution: For columns 1, 4, and 7: Query the target VLM whether the label accurately represents the content of the perturbed image. The script can be found in the same script as (E1) in the inference stage, with the commented header "Content preservation evaluation". The result will be saved in the same directory as the image soft prompt. Run the `eval_content_preserving.ipynb` notebook; the second block will generate the result.
For columns 2, 5, and 8: These depict the metric of output relevant to clean image. For columns 3, 6, and 9: These depict the metric of output relevant to perturbed image. We take the clean image/perturbed image and the target model's response to clean/perturbed image and use the

following query: "With yes or no, determine if the output is relevant to the image and answers the question?". Look for the percentage of yes in the result.

Results: For columns 1, 4, and 7: The results will be displayed in the notebook. For the result columns: The result needs to be manually calculated.

(E4): *[Stealthiness analysis under different perturbation norms] [1 human-hour + 600 compute-hours (can be reduced with more or larger GPUs, or run part of the experiments)]: This experiment generates the results of Table 5 in the paper.*

Execution: Navigate to `script/minigpt4/minigpt4_coco_script/` and execute the constraint-specific scripts: `script_sentiment_minigpt4_16.sh` (L_∞ norm, $\varepsilon = 16$), `script_sentiment_minigpt4_l2_12.sh` (L_2 norm, $\varepsilon = 12$), `script_sentiment_minigpt4_l2_24.sh` (L_2 norm, $\varepsilon = 24$), and `script_sentiment_minigpt4_l2_6.sh` (L_2 norm, $\varepsilon = 6$).

Results: Follow the same evaluation method as (E1) using the evaluation notebook, updating the file paths to point to the corresponding constraint-specific result directories. The notebook evaluation will display attack success rates across different perturbation constraints.

(E5): *[Transferability analysis] [1 human-hour + 2 compute-hours]: This experiment generates the results of Table 6 in the paper.*

Execution: Execute `script/minigpt4/minigpt4_coco_script/script_sentiment_minigpt4_transfer.sh` to perform inference on different models using image soft prompts generated by MiniGPT-4. Query GPT-4o with the same image soft prompts to obtain additional results.

Results: Evaluate the results using `eval_instruction_following.ipynb` with correct file paths. For GPT-4o results, perform manual calculation.

(E6): *[Feature Distillation defense] [1 human-hour + 2 compute-hours]: This experiment generates the results of Table 7 in the paper.*

Execution: Execute `script/minigpt4/minigpt4_coco_script/script_sentiment_minigpt4_defense.sh` to apply Gaussian and JPEG defense during inference.

Results: Evaluate the results using `eval_instruction_following.ipynb` with correct file paths.

(E7): *[Anomaly detection defense] [1 human-hour + 1.5 compute-hours]: This experiment generates the results of Table 8 in the paper.*

Execution: Run the notebook `eval_anomaly_detection.ipynb` to generate augmentations and compare similarity between image soft prompt embeddings and their augmentation embeddings with unperturbed image embeddings and their augmentation embeddings.

Results: The anomaly detection results will be displayed in the notebook.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.