USENIX Security '24 Artifact Appendix: Moderating Illicit Online Image Promotion for Unsafe User Generated Content Games Using Large Vision-Language Models

Keyan Guo*, Ayush Utkarsh*, Wenbo Ding*, Isabelle Ondracek*,
Ziming Zhao◇, Guo Freeman†, Nishant Vishwamitra‡, Hongxin Hu*
*University at Buffalo, ◇Northeastern University
†Clemson University, ‡The University of Texas at San Antonio,

A Artifact Appendix

This artifact appendix is meant to be a self-contained document that describes a roadmap for evaluating our artifact. It should include a

A.1 Abstract

In this work, we aim to address the issue of illicit image-based promotions of unsafe user generated content games (UGCGs) on social media. In our study, we collect a real-world dataset comprising 2,924 images that display diverse sexually explicit and violent content used to promote UGCGs by their game creators. We additionally create a cutting-edge system, UGCG-Guard, designed to aid social media platforms in effectively identifying images used for illicit UGCG promotions. This system leverages recently introduced large vision-language models (VLMs) and employs a novel conditional prompting strategy for zero-shot domain adaptation, along with chain-of-thought (CoT) reasoning for contextual identification. UGCG-Guardachieves outstanding results, with an accuracy rate of 94% in detecting these images used for the illicit promotion of such games in real-world scenarios.

A.2 Description & Requirements

We have prepared a CSV file that stores the image path of each UGCG image and its annotation label. Meanwhile, we prepared Python 3 scripts to reproduce the results of UGCG-Guard. The evaluator should have a Python 3 environment ready to run the prepared scripts.

A.2.1 Security, privacy, and ethical concerns

Our artifact contains an unsafe image dataset of inappropriate content, such as sexually explicit and violent images from Roblox games. Please avoid viewing the images directly if you are not comfortable with them.

A.2.2 How to access

The artifact can be accessed via the GitHub link: https://github.com/UBSec/UGCG-Guard/tree/1072d5c51a0e7bae2290da08e957e5b1d86cd7b6.

A.2.3 Hardware dependencies

UGCG-Guard is a framework for integrating large VLMs to detect insecure UGCG images. Our framework can utilize both open-source and closed large VLMs. In this work, we have prepared two scripts to ensure evaluators can successfully run UGCG-Guard with different environments.

The open-source large LVLM-based UGCG-Guard requires a runtime environment with over 50 GB of total GPU memory.

The closed LVLM-based UGCG-Guard requires only API requests and has no hardware dependencies.

A.2.4 Software dependencies

None.

A.2.5 Benchmarks

None.

A.3 Set-up

The Python 3 environment is necessary for our artifacts. In addition, please make sure that "base64", "requests", and "pandas" are installed in your Python 3 environment.

A.3.1 Installation

For open-source large LVLM-based UGCG-Guard, we have prepared the "requirements.txt". Run the following code in your Python environment to install the necessary dependencies.

```
pip install -r requirements.txt
```

For the closed large UGCG-Guard based on LVLM, make sure you have the OpenAI Python library. If not, you can install it using the following code.

```
pip install --upgrade openai
```

A.3.2 Basic Test

The evaluators can run scripts directly for artifact evaluation. They will output an error message if any dependencies are missing.

A.4 Evaluation workflow

A.4.1 Closed large LVLM-based UGCG-Guard

1. Open the script "gpt.py" and insert your OpenAI API key by changing the code in line 11:
   ```
   api_key = "YOUR_API_KEY"
   ```
2. Run "gpt.py", and the results will be stored in a new CSV file named "ugcg_gpt.csv". Testing the whole dataset will cost around 10 dollars. You can randomly select a subset with a number of images to test by adding a line of code under line 14, such as:
   ```
   df = df.sample(1000)
   ```
3. After the "ugcg_gpt.csv" result file is generated, you can run "view_result.py" to calculate and output the accuracy, precision, recall, and F1-score.

A.4.2 Open-source large LVLM-based UGCG-Guard

1. Run "requirements.txt" to set up your Python environment.
2. The model we used in this artifact is InstructBLIP from HuggingFace. Note that you may modify the code below to distribute the running task properly to different GPUs.
   ```
   device_map = infer_auto_device_map(
       model, max_memory={0: "28GiB", 1: "
       28GiB"},no_split_module_classes=['
       InstructBlipVisionModel', '
       InstructBlipQFormerModel', '
       LlamaDecoderLayer'])
   ```
3. Run the script "blip.py". The results will be automatically stored in "ugcg_blip.csv".
4. Open "ugcg_blip.csv" and look into the column "blip_output", if any of the answers for Q2-Q7 is "Yes", the label should be "unsafe".
5. Compare the predictions with the ground truth in the "label" column. "1" indicts unsafe images and "0" indicts safe images.

A.4.3 Major Claims

(C1): UGCG-Guard achieves a satisfactory performance for detecting unsafe UGCG images. This is proven by the experiment described in Section 6.3 in our paper, whose results are reported in Table 2.

A.4.4 Experiments

Either of the following experiments can be implemented to reproduce our results.

(E1): Closed large LVLM-based UGCG-Guard [20 human-minutes + 1.5 compute-hour + 280 MB disk]:
Preparation: Please set up your Python 3 environment and ensure the required dependencies are installed. Prepare an OpenAI API key to send requests that use GPT models.
Execution: Run the "gpt.py" file in the Python 3 environment and wait patiently for the CSV file to be generated.
Results: *The results are automatically calculated and printed to the terminal. After you execute the "view_result.py" file. The expected output should be around 94% accurate.*

(E2): Open-source large LVLM-based UGCG-Guard [1.5 human-hour + 2 compute-hour + 3 GB disk]:
Preparation: Please set up your Python 3 environment and make sure you have the required dependencies installed. The experiment requires a sufficient GPU, otherwise the execution will fail due to lack of memory.
Execution: Run the "blip.py" file in the Python 3 environment and wait patiently for the CSV file to be generated.
Results: Open "ugcg_blip.csv" with tools such as Microsoft Excel and observe the output stored in the "blip_output" column. If any of the answers for Q2-Q7 is "Yes", please mark the predicted label as "1". Otherwise, label it as "0". Compare the predicted label with the records in the "label" column. Calculate the accuracy with the following equation:

$$accuracy = \frac{\text{The number of correct predictions}}{\text{The number of total predictions}}$$

The expected output should be around 94% accurate.

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/usenixsec2024/.