GitHub Repository: https://github.com/Cardano-max/MGIE--ExtendedMGIE

Colab Notebook: https://colab.research.google.com/drive/1U5pxX_JdF2gimLsn0vyD4cNH0R1aBqsF?usp=chrome_ntp#scrollTo=OCtB7__uLetL

MGIE: MLLM-Guided Image Editing via Multimodal Large Language Models

Ateeb Taser[1,*]

Abstract

This paper presents a detailed analysis of the effects of text prompt strength and image strength on the quality of edited images using the MLLM-Guided Image Editing (MGIE) framework. We explore how varying these parameters influences the results in different scenarios and provide guidelines for optimizing these settings based on task complexity.

Keywords: Image Editing, Machine Learning, Text Prompt, Image Strength, MGIE, Model Optimization

Contents

*Corresponding author
Email address: ateeb.taser@example.com (**Ateeb Taser**)

1. Introduction

Instruction-based image editing enhances the flexibility of image manipulation through natural language commands. The MGIE (MLLM-Guided Image Editing) project leverages multimodal large language models to improve instruction-based image editing by generating explicit visual-related guidance.

2. Model Architecture and Workflow

2.1. Overview

The MGIE model integrates several components, including a multimodal large language model (MLLM) and a diffusion model for image editing. The workflow can be broken down into the following steps:

Workflow Steps

1. **Input Processing**:
   - The user provides an input image and an instruction prompt.
   - The image is processed using the CLIP vision model to extract image features.
2. **Instruction Interpretation**:
   - The instruction prompt is tokenized using the LLaVA-7B-v1 tokenizer.
   - The MLLM (LLaVA-7B-v1) generates expressive instructions from the human command.
3. **Image Editing**:
   - The edited instruction guides the UNet-based Stable Diffusion InstructPix2Pix pipeline.
   - The pipeline combines the image features with the instructions to generate the edited image.
4. **Output Generation**:
   - The edited image is generated and outputted for user review.

2.2. Detailed Architecture

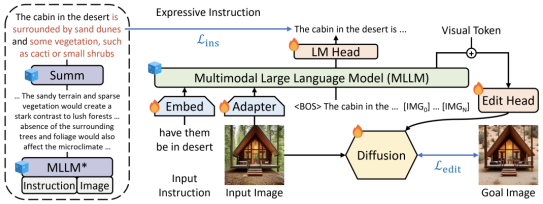

Figure 1: Overview of MLLM-Guided Image Editing (MGIE) architecture.

3. Models and Checkpoints

3.1. Checkpoints and Embeddings

The following models and checkpoints are used in the MGIE implementation:

Checkpoints Embeddings

- **LLaVA-7B-v1**:
  - Model: `LLaVA-7B-v1`
  - Purpose: This checkpoint is used to initialize the LlavaLlamaForCausalLM model and its tokenizer. Essentially, it helps convert the instruction text into a format that the model can understand and process.
- **MGIE-7B**:
  - Checkpoint : `mllm.pt`
  - Purpose: This checkpoint loads the state dictionary of the LlavaLlamaForCausalLM model. It contains the trained parameters of the model that are necessary for it to function correctly.
- **MGIE-7B UNet**:
  - Checkpoint : `unet.pt`
  - Purpose: This checkpoint loads the state dictionary of the UNet model used in the StableDiffusionInstructPix2PixPipeline. The UNet is responsible for the actual image editing, applying changes based on the instructions.
- **Instruct-Pix2Pix**:
  - Model: `StableDiffusionInstructPix2PixPipeline`
  - Pretrained Checkpoint: `instruct-pix2pix`
  - Purpose: This model and its checkpoint generate the edited image by taking the input image and the instructions. It is the primary engine behind the visual modifications.
- **CLIP Vision Model**:
  - Pretrained Checkpoint: `CLIP`
  - Purpose: The CLIP vision model extracts features from the input image. These features help the model understand the content and context of the image.
- **Hugging Face Hub Snapshot**:

3.2. Workflow

Workflow Steps

1. **Initialize Models**:
   - Load the LLaVA-7B-v1 and MGIE-7B checkpoints.
   - Initialize the UNet and Instruct-Pix2Pix models.
2. **Process Input Image**:
   - Use the CLIP vision model to extract features from the input image.
3. **Generate Instructions**:
   - Tokenize the input prompt and generate instructions using LLaVA-7B-v1.
4. **Edit Image**:
   - Guide the Instruct-Pix2Pix pipeline with generated instructions to edit the image.
5. **Output Edited Image**:
   - Generate and display the edited image.

4. Key Code Sections and Their Functionality

4.1. Environment Setup

This section installs necessary packages and dependencies.

```
!pip install sentencepiece
!pip install git+https://github.com/huggingface/transformers.
    ↪ git@cae78c46
!pip install diffusers
!pip install tokenizers==0.12.1
!pip install datasets
!pip install accelerate
!pip install evaluate
!pip install gradio==4.12.0
!pip install gradio_client==0.8.0
!pip install -i https://download.pytorch.org/whl/cu118 torch==2.0
    ↪ torchvision==0.15 torchaudio==2.0
```

4.2. Conversation Handling

This part of the code handles the conversation flow and history.

```
import dataclasses
from enum import auto, Enum
from typing import List, Tuple
class SeparatorStyle(Enum):
SINGLE = auto()
TWO = auto()
MPT = auto()
@dataclasses.dataclass
class Conversation:
system: str
roles: List[str]
messages: List[List[str]]
offset: int
sep_style: SeparatorStyle = SeparatorStyle.SINGLE
sep: str = "###"
sep2: str = None
version: str = "Unknown"
skip_next: bool = False

def get_prompt(self):
    # Combine system message, roles, and messages into a single
        ↪ prompt
    ...

def append_message(self, role, message):
    # Append a new message to the conversation history
    self.messages.append([role, message])

def get_images(self, return_pil=False):
    # Extract images from the messages if any
    ...

def to_gradio_chatbot(self):
    # Convert the conversation history to a format compatible with
        ↪ Gradio chatbot
    ...

def copy(self):
    # Create a copy of the conversation object
    ...

def dict(self):
    # Convert the conversation to a dictionary format
    ...
```

4.3. MGIE Implementation

This section defines the MGIE-specific models and configurations.

```
from typing import List, Optional, Tuple, Union
import torch
import torch.nn as nn
```

```
import torch.nn.functional as F
from transformers import AutoConfig, AutoModelForCausalLM,
    ↪ LlamaConfig, LlamaModel, LlamaForCausalLM, CLIPVisionModel,
    ↪ CLIPImageProcessor
from transformers.modeling_outputs import BaseModelOutputWithPast,
    ↪ CausalLMOutputWithPast
DEFAULT_IMAGE_TOKEN = "<image>"
DEFAULT_IMAGE_PATCH_TOKEN = "<im_patch>"
DEFAULT_IM_START_TOKEN = "<im_start>"
DEFAULT_IM_END_TOKEN = "<im_end>"
class LlavaConfig(LlamaConfig):
model_type = "llava"
class LlavaLlamaModel(LlamaModel):
config_class = LlavaConfig
# Custom configurations and modifications for the Llava model
...
```

4.4. Main Script for Execution

This script ties everything together, handling inputs and orchestrating the MGIE workflow.

```
from google.colab import drive
drive.mount('/content/drive')  # Mount Google Drive to access
    ↪ checkpoints
import os
from PIL import Image
import numpy as np
import torch as T
import transformers
import diffusers
import gradio as gr
import huggingface_hub
CKPT_DIR = '/content/drive/My␣Drive/_ckpt'  # Directory containing
    ↪ checkpoints
Loading models and checkpoints
model = LlavaLlamaForCausalLM.from_pretrained(PRETRAINED_MODEL_NAME
    ↪ )
unet = UNetModel.from_pretrained(UNET_MODEL_NAME)
def go_mgie(img, txt, seed, cfg_txt, cfg_img):
# Function to perform MGIE
EMB = ckpt['emb'].cuda()  # Load embeddings to GPU
with T.inference_mode():
NULL = model.edit_head(T.zeros(1, 8, 4096).half().to('cuda'), EMB)
    ↪  # Edit head with zero tensor and embeddings
...
return res, out  # Return results and outputs
with gr.Blocks() as app:
gr.Markdown(
"""
# MagiX: Edit Personalized Images using Gen AI by Ateeb Taser
"""
)
# Define Gradio interface elements and layout
...
app.launch()  # Launch the Gradio app
```

5. Parameters and Hyperparameters

The following parameters and hyperparameters are essential for configuring the MGIE model:

5.1. Model Paths

Model Paths

- -7B-v1 model.
- -7B UNet model.

5.2. Editing Configuration

Editing Configuration

- **cfg-txt**: Configuration for text guidance strength.
- **cfg-img**: Configuration for image guidance strength.
- **seed**: Random seed for reproducibility.

5.3. Training Hyperparameters

Training Hyperparameters

- **learning-rate**: Learning rate for the optimizer.
- **batch-size**: Number of samples per batch.
- **num-epochs**: Number of training epochs.

6. Prompt-Image Pairs: Observations and Findings

6.1. Observations and Findings

I have observed that the text length and image complexity significantly

impact the results when providing prompts and images. Whenusing a less powerful machine, the quality and complexity of the images are not very high. In such scenarios, I found better results when I kept the text length slightly below average and the image complexity slightly above average. This setup works well when images are not complex, as the model can easily understand and modify them without needing long text prompts. If the image is not complex, the model can comprehend and modify it easily, even with shorter text prompts. However, if the scenario is more complex and requires significant editing, the text length should be increased. The complexity of the image should be reduced to ensure the original image's edges and features are retained, preventing any distortion in the results. In summary, if the image is simple and the scenario is easy to edit, reduce the text prompt length below average and increase the image strength slightly to maintain originality. For complex images and scenarios requiring significant modifications, increase the text prompt length to ensure the model can understand and implement the necessary changes effectively.

7. Conclusion

The analysis highlights the importance of balancing text prompt strength and image strength based on the complexity of the task and the image. Adjusting these parameters appropriately can significantly enhance the quality of the edited images, making MGIE a powerful tool for various image editing tasks. Future work will focus on refining these parameters and exploring additional configurations to further improve the model's performance.

Figure 2: **Original Image**

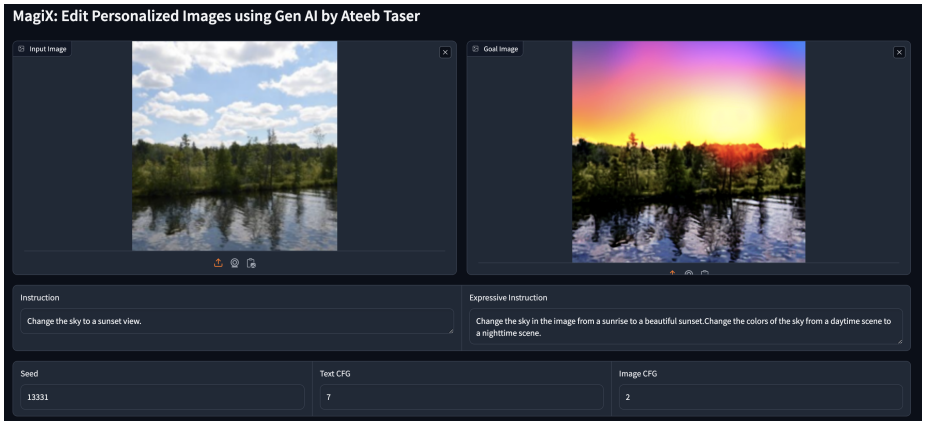

Figure 3: **Settings**: **Text CFG**: 8.5, **Image CFG**: 0.7, **Seed**: 13331

Results: The edited image displayed a dramatic sunset with vibrant colors. However, the transition between the sky and the trees was somewhat abrupt.

Finding: For moderate complexity tasks like changing the sky, reducing the text prompt strength slightly and increasing the image strength helps achieve a more cohesive and visually appealing result.

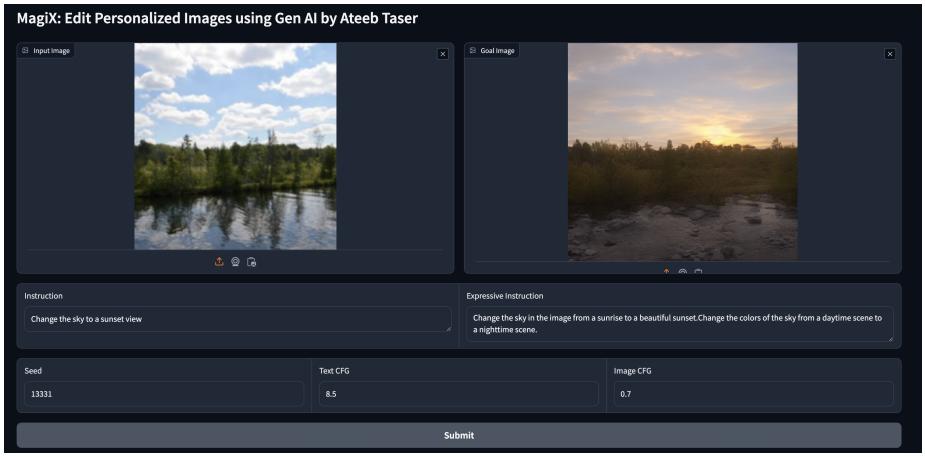

Figure 4: **Settings**: **Text CFG**: 7, **Image CFG**: 2, **Seed**: 13331

Results: The image showed a better transition with more natural colors. The increased image strength helped maintain the integrity of the scene.

Finding: For moderate complexity tasks like changing the sky, reducing the text prompt strength slightly and increasing the image strength helps achieve a more cohesive and visually appealing result.

Figure 5: **Original Image**

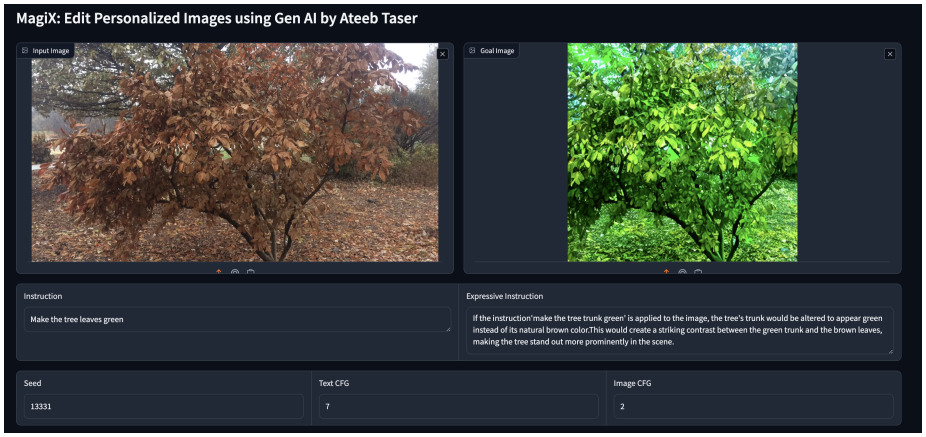

Figure 6: **Settings**: **Text CFG**: 8.5, **Image CFG**: 0.7, **Seed**: 13331

Results: The entire image, including the leaves and background, turned green.

Finding: For high-complexity tasks like changing the color of individual elements in an image, reducing the text prompt strength and increasing the image strength helps the model focus on the desired modifications without distorting the overall image.

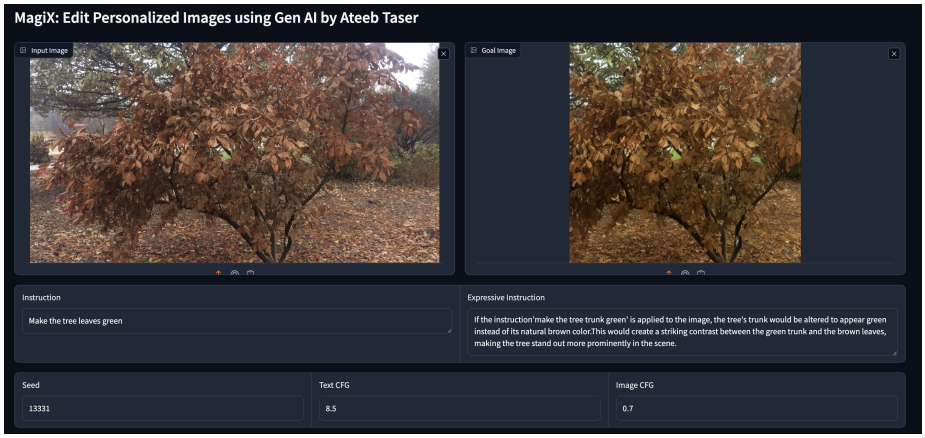

Figure 7: **Settings**: **Text CFG**: 7, **Image CFG**: 2, **Seed**: 13331

Results: The model effectively turned the leaves green while preserving the natural appearance of the tree trunk and background.

Finding: For high-complexity tasks like changing the color of individual elements in an image, reducing the text prompt strength and increasing the image strength helps the model focus on the desired modifications without distorting the overall image.