Week 6 Assignment [4 Points]

Generative AI

Saarland University – Winter Semester 2024/25

Submission due: 5 December 2024 by 6pm CET

genai-w24-tutors
genai-w24-tutors@mpi-sws.org

1 Reading Assignment

This week's reading assignment comprises of the following:

(R.1) [DALL-E 1] Zero-Shot Text-to-Image Generation [1].
Paper link: `http://proceedings.mlr.press/v139/ramesh21a/ramesh21a.pdf`.

(R.2) [CLIP] Learning Transferable Visual Models From Natural Language Supervision [2].
Paper link: `https://proceedings.mlr.press/v139/radford21a/radford21a.pdf`.

(R.3) [LLaVA] Visual Instruction Tuning [3].
Paper link: `https://proceedings.neurips.cc/paper_files/paper/2023/file/6dcf277ea32ce3288914faf369fe6de0-Paper-Conference.pdf`.

[OPTIONAL] Below, we are providing additional material covering content relevant to the lecture:

- Image Transformer [4].
  Paper link: `http://proceedings.mlr.press/v80/parmar18a/parmar18a.pdf`.
- [Vision Transformer ViT] An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale [5]. Paper link: `https://openreview.net/pdf?id=YicbFdNTTy`.
- Denoising Diffusion Probabilistic Models [6]. Paper link: `https://proceedings.neurips.cc/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf`.
- High-Resolution Image Synthesis with Latent Diffusion Models [7]. Paper link: `https://openaccess.thecvf.com/content/CVPR2022/papers/Rombach_High-Resolution_Image_Synthesis_With_Latent_Diffusion_Models_CVPR_2022_paper.pdf`.
- Diffusion-LM Improves Controllable Text Generation [8].
  Paper link: `https://openreview.net/pdf?id=3s9IrEsjLyk`.
- [DALL-E 2] Hierarchical Text-Conditional Image Generation with CLIP Latents [9].
  Paper link: `https://arxiv.org/pdf/2204.06125`.
- [DALL-E 3] Improving Image Generation with Better Captions [10].
  Paper link: `https://cdn.openai.com/papers/dall-e-3.pdf`.

2 Exercise Assignment

This week's exercise assignment comprises of a few conceptual questions on how multimodal generative models work and the role of different model architectures. The questions should be answered in the PDF submission file – see instructions in Section 4.

(E.1) Briefly explain the similarities and differences between the Image Transformer and DALL-E 1 models.

(E.2) Write down the mathematical equations for the forward process and reverse process in diffusion models.

(E.3) Briefly explain the role of the CLIP model in LLaVA model's architecture.

(E.4) Briefly explain the role of the CLIP model in DALL-E 2 model's architecture.

(E.5) Briefly explain the similarities and differences between DALL-E 1 and DALL-E 2 models.

3 Implementation Based Assignment

In this week's implementation assignment, you will evaluate the capabilities of multimodal models, such as GPT-4o and DALL-E 3.

3.1 Demo Scripts

In these questions, the objective is to understand the capabilities of multimodal foundation models for visual understanding and image generation in a general setting. We will use GPT-4o to output keywords that describe given images and DALL-E 3 to generate images for given keywords. We have provided the following files in `week6_implementation.zip`:

- `demo_image2text.py`: A script that makes use of GPT-4o[1] to understand an image and provide a list of keywords that describe the image. GPT-4o is queried via the OpenAI API[2].
- `input_demo_image2text.png`: This file contains the image to be analyzed.
- `demo_text2image.py`: A script that uses DALL-E 3[3] to generate images based on keywords provided as input. DALL-E 3 is queried via the OpenAI API.
- `input_demo_text2image.txt`: This file contains the list of keywords that will be used to generate images.

We begin with the visual understanding part. Note that first you will need to upload the file containing your OpenAI API key to Colab, then change the `«OPENAI_API_KEY_FILE»` placeholder with its actual name, as described in Section 5.2. As part of the assignment, you will execute the provided code without implementing any new functionality. The `demo_image2text.py` script will read the input image file `input_demo_image2text.png`, generate keywords describing the image using GPT-4o, and save them to the `output_demo_image2text/` directory. This process will be done 3 times, resulting in 3 output files. Once the code is executed, you can review the generated output.

(I.1) Run the `demo_image2text.py` script. Then provide a qualitative analysis of the generated keywords (found in `output_demo_image2text/I1_{a,b,c}.txt`), focusing on how they match the content of the input image. You can answer this question in a few sentences and no detailed analysis is required. This should be answered in the PDF submission file – see instructions in Section 4.

We continue with the image generation part. Again, change the `«OPENAI_API_KEY_FILE»` placeholder with its actual name. Next, you will execute the script `demo_text2image.py`, which will read the input text file `input_demo_text2image.txt`, generate images using DALL-E 3, and save them to the `output_demo_text2image/` directory. This process will be done 3 times, resulting in 3 output files. Once the code is executed, you can review the generated output.

(I.2) Run the `demo_text2image.py` script. Then provide a qualitative analysis of the generated images (found in `output_demo_text2image/I2_{a,b,c}.png`), focusing on how they match the input keywords. You can answer this question in a few sentences and no detailed analysis is required. This should be answered in the PDF submission file – see instructions in Section 4.

[1]You can read more about GPT-4o here: https://openai.com/index/hello-gpt-4o/.

[2]You can check the documentation for the OpenAI API: https://platform.openai.com/docs/api-reference/introduction.

[3]You can read more about DALL-E 3 here: https://openai.com/index/dall-e-3/.

3.2 Visual Programming

Next, we will explore visual programming within the "Hour of Code" (HoC) domain[4]. The focus will be on comparing how models handle ASCII and image-based representations of grids. Tasks include extracting grid elements, generating action sequences to navigate the grid, and generating grids from given input elements. This comparison highlights the strengths and limitations of state-of-the-art models in processing structured visual data.

3.2.1 Background on the Domain

This assignment revolves around a grid navigation system inspired by "Hour of Code" (HoC). To get more familiar with HoC, we recommend that you solve the following three mazes:

- Maze 4: `https://studio.code.org/hoc/4`
- Maze 8: `https://studio.code.org/hoc/8`
- Maze 18: `https://studio.code.org/hoc/18`

Each task is visually specified through a single grid that represents the environment, including the starting position and orientation of the avatar, the location of the goal, and any obstacles (walls) present in the grid. To solve the task, one must execute a sequence of commands (code blocks) that guide the avatar from its starting position to the goal. In this assignment, the commands will be limited to basic actions such as movement and turning, without involving programming constructs like conditionals or loops. Below, we provide further details on the grid elements and the commands available.

More concretely, the grid consists of the following elements:

- AVATAR: The avatar is represented by a blue dart, indicating its current location and orientation. The orientation can be one of NORTH, EAST, SOUTH, WEST. The avatar moves around the grid following given commands.
- GOAL: Represented by a red star, this marks the target location that the avatar must reach.
- WALLS: These are gray grid cells that block movement. If the avatar attempts to enter a wall, the task fails.
- EMPTY GRID-CELL: These are free white cells that the avatar can move through freely.

As mentioned earlier, this assignment will only consider commands involving for basic actions, without any programming constructs. More concretely, the avatar can execute the following basic commands to navigate the grid:

- `move_forward`: Moves the avatar one cell forward in its current direction. If the move causes the avatar to collide with a wall or leave the grid boundary, the task fails.
- `turn_left`: Rotates the avatar 90 degrees counterclockwise without changing its position.
- `turn_right`: Rotates the avatar 90 degrees clockwise without changing its position.

3.2.2 Grid to Elements of the Grid

In these questions, the objective is to extract grid elements such as the avatar's position, orientation, the goal, and walls from grid representations. We will compare how well GPT-4o performs when processing ASCII-based representation versus image-based representation of grids. We have provided the following files in `week6_implementation.zip`:

- `visualprog_grid_to_elements.py`: A script that uses GPT-4o to extract grid elements (avatar, goal, walls) from ASCII or image input. GPT-4o is queried via the OpenAI API.
- `input_visualprog_task1_grid.txt`: This contains the ASCII representation of a 4×4 grid.
- `input_visualprog_task1_grid.png`: This contains the image representation of a 4×4 grid.
- `system_prompt_ascii.txt` and `system_prompt_vis.txt`: These files contain the system prompts that provide background information about the domain and specific representations.

[4]You can learn more about Hour of Code here: `https://studio.code.org/hoc/8`.

First, change the «OPENAI_API_KEY_FILE» placeholder with its actual name, as described in Section 5.2. Next, you will execute the provided code by setting different variables in the `__main__` function without implementing any new functionality. More concretely, in the `__main__` function of `visualprog_grid_to_elements.py`, the script will process the input file, extract grid elements, and write the results to the `output_grid_to_elements/` directory. This process will be done 3 times, resulting in 3 output files. Once the code is executed, you can review the generated outputs.

(I.3) In the `__main__` function of `visualprog_grid_to_elements.py`, set the `input_type` to `"ascii"` to process the ASCII-based representation of the grid in `input_visualprog_task1_grid.txt`. Run the script to extract grid elements. The outputs will be written to `output_grid_to_elements/I3_{a,b,c}.txt`. These `.txt` files should be included as part of the ZIP submission file – see instructions in Section 4.

(I.4) In the `__main__` function of `visualprog_grid_to_elements.py`, set the `input_type` to `"vis"` to process the image-based represenation of the grid in `input_visualprog_task1_grid.png`. Run the script to extract grid elements. The outputs will be written to `output_grid_to_elements/I4_{a,b,c}.txt`. These `.txt` files should be included as part of the ZIP submission file – see instructions in Section 4.

(I.5) Provide a qualitative comparison of the extracted grid elements from the ASCII and image representations of the grid, focusing on whether the provided output is correct with respect to the provided input. This should be answered in the PDF submission file – see instructions in Section 4.

- For each representation format (ASCII and image), analyze the three outputs generated by the script. How often are the grid elements (avatar, goal, walls) extracted correctly for each format?
- Provide an overall qualitative comparison between the two input types. Which input type appears easier for the model to process accurately? You can answer this question in a few sentences, and no detailed analysis is required.

3.2.3 Grid to Code

In these questions, the objective is to generate sequences of basic actions that correctly guide the avatar to the goal. We will compare how well GPT-4o performs when generating these action sequences when processing ASCII-based representation versus image-based representation of grids. We have provided the following files in `week6_implementation.zip`:

- `visualprog_grid_to_seq.py`: A script that uses GPT-4o to generate sequences of actions (e.g., `move_forward`, `turn_left`, `turn_right`) for navigating a grid. GPT-4o is queried via the OpenAI API.
- `input_visualprog_task1_grid.txt`: This contains the ASCII representation of a 4×4 grid.
- `input_visualprog_task1_grid.png`: This contains the image representation of a 4×4 grid.
- `system_prompt_ascii.txt` and `system_prompt_vis.txt`: These files contain the system prompts that provide background information about the domain and specific representations.

First, change the «OPENAI_API_KEY_FILE» placeholder with its actual name, as described in Section 5.2. Next, you will execute the provided code by setting different variables in the `__main__` function without implementing any new functionality. More concretely, in the `__main__` function of `visualprog_grid_to_seq.py`, the script will process the input file, generate a sequence of actions, and write the results to the `output_grid_to_seq/` directory. This process will be done 3 times, resulting in 3 output files. Once the code is executed, you can review the generated outputs.

(I.6) In the `__main__` function of `visualprog_grid_to_seq.py`, set the `input_type` to `"ascii"` to process the ASCII-based representation of the grid in `input_visualprog_task1_grid.txt`. Run the script to generate the sequence of commands. The outputs will be written to `output_grid_to_seq/I6_{a,b,c}.txt`. These `.txt` files should be included as part of the ZIP submission file – see instructions in Section 4.

(I.7) In the `__main__` function of `visualprog_grid_to_seq.py`, set the `input_type` to `"vis"` to process the image-based represenation of the grid in `input_visualprog_task1_grid.png`. Run the script to generate the sequence of commands. The outputs will be written to `output_grid_to_seq/I7_{a,b,c}.txt`. These `.txt` files should be included as part of the ZIP submission file – see instructions in Section 4.

(I.8) Provide a qualitative comparison of the generated sequences of commands for the ASCII and image representations of the grid, focusing on whether the provided output is correct with respect to the provided input. This should be answered in the PDF submission file – see instructions in Section 4.

- For each representation format (ASCII and image), analyze the three outputs generated by the script. How often does the generated sequence of commands successfully guide the avatar to the goal without collisions?
- Provide an overall qualitative comparison between the two input types. Which input type appears easier for the model to process accurately? You can answer this question in a few sentences, and no detailed analysis is required.

3.2.4 Elements of the Grid to Grid

In these questions, the objective is to generate visual grids from given input elements, such as the avatar's position, orientation, the goal, and the walls. We will compare how well GPT-4o and DALL-E 3 generate grids based on the input. Note that GPT-4o will generate an ASCII-based representation and DALL-E 3 will generate an image-based representation of the grid. We have provided the following files in `week6_implementation.zip`:

- `visualprog_elements_to_grid.py`: A script that uses GPT-4o to generate a grid in ASCII-based representation and DALL-E 3 to generate a grid in image-based representation based on the given input elements. GPT-4o and DALL-E 3 are queried via the OpenAI API.
- `input_visualprog_task1_gridelements.txt`: This file contains the input elements describing a grid (e.g., avatar position, goal location, and walls).
- `system_prompt_ascii.txt` and `system_prompt_vis.txt`: These files contain the system prompts that provide background information about the domain and specific representations.

First, change the `«OPENAI_API_KEY_FILE»` placeholder with its actual name, as described in Section 5.2. Next, you will execute the provided code by setting different variables in the `__main__` function without implementing any new functionality. More concretely, in the `__main__` function of `visualprog_elements_to_grid.py`, the script will process the input file, generate the grid, and write the results to the `output_elements_to_grid/` directory. This process will be done 3 times, resulting in 3 output files. Once the code is executed, you can review the generated outputs.

(I.9) In the `__main__` function of `visualprog_elements_to_grid.py`, set the `input_type` to `"ascii"` to generate grid in ASCII-based representation using GPT-4o from the elements in `input_visualprog_task1_gridelements.txt`. Run the script to generate the grid. The outputs will be written to `output_elements_to_grid/I9_{a,b,c}.txt`. These `.txt` files should be included as part of the ZIP submission file – see instructions in Section 4.

(I.10) In the `__main__` function of `visualprog_elements_to_grid.py`, set the `input_type` to `"vis"` to generate grid in image-based representation using DALL-E 3 from the elements in `input_visualprog_task1_gridelements.txt`. Run the script to generate the grid. The outputs will be written to `output_elements_to_grid/I10_{a,b,c}.png`. These `.png` files should be included as part of the ZIP submission file – see instructions in Section 4.

(I.11) Provide a qualitative comparison of the generated grids from GPT-4o (as ASCII text) and DALL-E 3 (as image). When evaluating, consider the accuracy in terms of whether the generated grid correctly captures the input elements. This should be answered in the PDF submission file – see instructions in Section 4.

- For each representation format (ASCII and image), analyze the three outputs generated by the script. How often are the generated grids accurate and consistent with input elements?
- Provide an overall qualitative comparison between generated grids from GPT-4o (as ASCII text) and DALL-E 3 (as image). Which model and representation type led to better results? You can answer this question in a few sentences, and no detailed analysis is required.

4 Submission Instructions

Please submit the following two files:

- **<Lastname>_<Matriculation>_week6.pdf**. This PDF file will report answers to the following questions: E.1, E.2, E.3. E.4, E.5, I.1, I.2, I.5, I.8, and I.11. The PDF file should be generated in LaTeX using NeurIPS 2024 style files; see further formatting instructions in Section 5.1. The PDF file size should **not exceed** 1**mb**. Please use the following naming convention: Replace <Lastname> and <Matriculation> with your respective Lastname and Matriculation number (e.g., Singla_1234567_week6.pdf).
- **<Lastname>_<Matriculation>_week6.zip**. This ZIP file will report answers to the following questions: I.3, I.4, I.6, I.7, I.9, and I.10. Importantly, the ZIP file should unzip to a folder named **<Lastname>_<Matriculation>_week6**. Inside this folder, include the generated `.txt` files and `.png` files. The ZIP file size should **not exceed** 10**mb**. Replace <Lastname> and <Matriculation> with your respective Lastname and Matriculation number (e.g., Singla_1234567_week6.zip).

5 Technical Instructions

5.1 Instructions for Preparing PDFs

Refer to Week 1 assignment.

5.2 Instructions for Implementation

We have provided you with a key through which you can access the OpenAI API. The key is located in the file `{id}_{name}_openai.txt` (e.g., `0_Singla_openai.txt`). Please refer to the lecture slide #43 on how to locate this file with the key.

In order to use this key with the provided scripts, upload this file to Colab as usual (refer to Week 1 assignment). Next, the scripts provided in Week 6 assignment automatically read the content of the file and set the necessary values for the OpenAI API, however, you need to replace the `«OPENAI_API_KEY_FILE»` placeholder with the name of the `{id}_{name}_openai.txt` file (e.g., `0_Singla_openai.txt`).

References

[1] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-Shot Text-to-Image Generation. In *ICML*, 2021.

[2] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In *ICML*, 2021.

[3] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual Instruction Tuning. In *NeurIPS*, 2023.

[4] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image Transformer. In *ICML*, 2018.

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*, 2021.

[6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *NeurIPS*, 2020.

[7] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-Resolution Image Synthesis with Latent Diffusion Models. In *CVPR*, 2022.

[8] Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. Diffusion-LM Improves Controllable Text Generation. In *NeurIPS*, 2022.

[9] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical Text-Conditional Image Generation with CLIP Latents. *CoRR*, abs/2204.06125, 2022.

[10] James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntang Zhuang, Joyce Lee, Yufei Guo, Wesam Manassra, Prafulla Dhariwal, Casey Chu, and Yunxin Jiao. Improving Image Generation with Better Captions. https://cdn.openai.com/papers/dall-e-3.pdf, 2023.

Based on template from https://media.neurips.cc/Conferences/NeurIPS2024/Styles.zip