PicassoGen: Picture Assembler System using Stepwise Object Arrangement

Abstract

Despite significant advancements in text-to-image generation models, these models often struggle with accurately generating images when specific details, such as positional information and object counts, are provided. To address this, we applied LLM agents with an inpainting module to enhance the generation process under detailed constraints. Our contribution includes integrating LLM agents with advanced techniques like subgoal decomposition, planning, reflection, and tool usage tailored for text-to-image tasks. We also developed an iterative refinement process using inpainting techniques instead of traditional layout generation. These novel approaches significantly improve the accuracy and coherence of generated images in response to detailed prompts. Our code is available at: https://github.com/chaksseu/PicassoGen

1 Introduction

Text-to-image generation models have made remarkable strides in recent years, producing increasingly realistic and coherent images from textual descriptions. However, these models often struggle when tasked with generating images that adhere to specific details such as positional information and object counts. This limitation poses a significant challenge, especially in applications requiring precise visual outputs from detailed textual inputs, such as in design, virtual reality, and automated content creation.

The difficulty lies in the inherent complexity of translating detailed and structured textual information into corresponding visual representations. While current models can generate images from high-level descriptions, they often fail to accurately interpret and implement specific constraints like object placement and quantity. This gap underscores the need for more sophisticated approaches that can handle the intricacies of detailed text-to-image generation tasks.

Existing methods have seen partial success but often fall short in maintaining the fidelity of positional and numerical details in generated images. Traditional layout generation techniques, for example, lack the flexibility and precision needed for such tasks. Moreover, the integration of these models with advanced planning and reasoning capabilities remains underexplored.

In this work, we propose PicassoGen, a novel framework that leverages large language model (LLM) agents equipped with advanced techniques such as subgoal decomposition, planning, reflection, and tool usage, specifically tailored for text-to-image tasks. Our approach employs an inpainting module for iterative refinement, moving away from traditional layout generation methods. This innovative combination allows for a more accurate and coherent translation of detailed textual prompts into images.

Recent advancements in LLM-based autonomous agents have shown great potential in various domains by exhibiting human-like intelligence and decision-making capabilities [1, 2, 3, 4, 5]. These agents perform complex tasks by leveraging the generative and interpretative power of LLMs, enhancing performance and reliability in tasks such as software development [6], human behavior simulation [7], and resource optimization [8]. Inspired by these advancements, our method utilizes

LLM agents to process and refine text prompts, ensuring that the generated images accurately reflect the specified details.

Moreover, integrating LLMs with visual modalities has shown promising results in text-to-image generation. Techniques such as aligning image features with linguistic space and decomposing complex prompts into simpler sub-tasks have improved the accuracy of generated images [9, 10]. Our approach builds on these foundations by introducing an iterative refinement process using inpainting techniques, which allows for precise placement and detailing of objects within the generated images.

We employ state-of-the-art models such as GPT-4o [11, 12], Stable Diffusion[13], and DALL-E [14, 15] in our framework. These models are utilized without additional training, facilitating easy updates and improvements as newer models become available. By leveraging the AutoGen [16] framework, we implement a robust and flexible system capable of handling a variety of text-to-image generation tasks.

Our contributions are twofold. First, we demonstrate the effectiveness of LLM agents in enhancing text-to-image generation through advanced reasoning and planning techniques. Second, we introduce an iterative refinement process utilizing inpainting techniques, which significantly improves the adherence of generated images to specified details. These contributions not only advance the state-of-the-art in text-to-image generation but also open new avenues for future research in the field.

The results of our experiments show that our approach significantly outperforms existing methods in generating images that accurately reflect the detailed constraints of the input prompts. This work sets a new benchmark for text-to-image generation, offering a robust framework for future developments and applications in this domain.

2 Related Work

2.1 Large Language Model Based Autonomous Agents

Large language model (LLM) based autonomous agents are designed to perform complex tasks by leveraging the generative and interpretative power of LLMs, exhibiting human-like intelligence and decision-making capabilities. An autonomous agent is defined as a system situated within and part of an environment that senses and acts on it over time to pursue its own agenda and influence future perceptions [1]. The integration of LLMs into autonomous agents has significantly advanced their capabilities, enabling them to perform complex tasks. Various frameworks [2, 3, 4, 5] highlight the potential and versatility of LLM-based autonomous agents in different domains.

For instance, frameworks enhancing structured communication and collaboration among agents in software development have shown improvements in performance and reliability by mimicking interactions within a software development team, facilitating efficient task delegation and collaborative problem-solving [6]. Systems that simulate human behavior provide insights into language interaction and social dynamics within a multi-agent framework, enabling realistic and contextually appropriate responses [7]. Other systems optimize resource allocation, context switching, and concurrent execution of agents, addressing challenges like scheduling and context management to improve the efficiency and reliability of LLM-based agents [8]. Additionally, some approaches incorporate Standardized Operating Procedures (SOPs) into multi-agent collaborations, streamlining workflows and reducing errors through structured interactions [17]. These ensure that each agent operates within a well-defined framework, enhancing the coherence and effectiveness of multi-agent operations. Combined, these studies inform our approach by illustrating the effectiveness of structured communication in multi-agent systems and the advantages of leveraging LLMs for autonomous operations.

A notable framework facilitates next-generation LLM applications through a multi-agent conversational approach, emphasizing structured communication and collaboration to improve task execution accuracy and performance [16]. By leveraging these techniques, our method improves upon existing text-to-image generation methods, offering enhanced precision and reliability in generating images that accurately reflect the input text prompts.

Collectively, these frameworks represent significant advancements in LLM-based autonomous agents, showcasing various approaches to improving precision, reliability, and efficiency in complex, multi-agent environments. Our research builds upon these advancements to further enhance the capabilities of LLM-based agents, particularly in generating accurate visual content from textual descriptions.

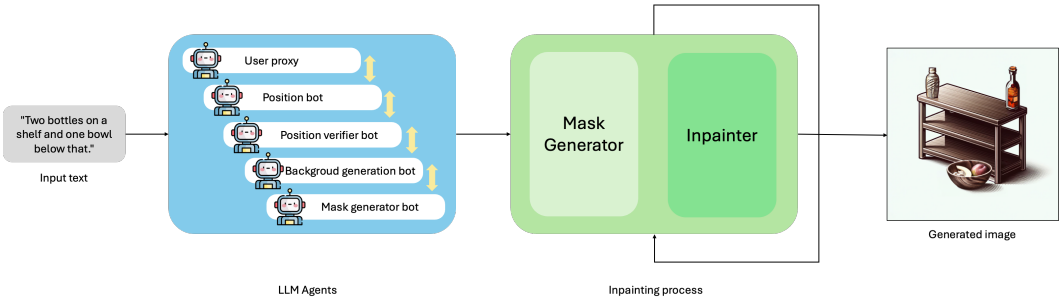


Figure 1: Overview of our PicassoGen framework for text-to-image generation.

2.2 Accurate Text-to-Image Generation with LLMs

Recent advancements have focused on leveraging large language models (LLMs) to enhance their ability to handle text-to-image generation tasks with high accuracy. Several methods incorporate additional training data to align image features with linguistic space, enabling LLMs to better understand and generate detailed visual content [9]. Other strategies dynamically integrate off-the-shelf vision models, extracting image information into text form to improve comprehension and generation [10]. This scheduling ability of LLMs allows for the seamless integration of visual data, ensuring that the generated images adhere closely to the specified details in the text prompts.

A notable framework utilizes LLMs to progressively generate images by decomposing complex prompts into simpler sub-tasks, each focusing on generating a single object [9]. This method emphasizes the importance of planning and feedback control, enabling the generation of images with multiple objects that maintain precise spatial relations and attribute bindings. Another innovative framework leverages large language models to handle lengthy and intricate text prompts, extracting critical components such as bounding box coordinates and detailed object descriptions. This method generates an initial scene layout and iteratively refines the representation of each object to ensure consistency with the textual descriptions, significantly improving the recall of complex prompts [10].

These advancements collectively represent significant progress in using LLMs for text-to-image generation, showcasing various approaches to integrating visual modalities into language models. By building on these foundational technologies, our work aims to further enhance the capabilities of LLM-based agents in generating accurate and detailed visual content from textual descriptions. Our approach introduces the concept of LLM agents and utilizes inpainting techniques during the refinement process, which were not previously employed in this field.

3 Approach

3.1 Overview

Our approach leverages large language model (LLM) agents to significantly enhance the accuracy of text-to-image generation tasks. This method ensures that the final generated image precisely reflects the detailed specifications of the text prompt, particularly in terms of object count, size, and position. Our novel contribution lies in the integration of LLM agents equipped with advanced techniques such as subgoal decomposition, planning, reflection, and tool usage specifically tailored for text-to-image tasks. Furthermore, we introduce an developed iterative refinement process utilizing inpainting techniques instead of traditional layout generation. To our knowledge, these innovative approaches have not been explored in prior works.

Initially, a text prompt is provided as input. This prompt is analyzed by a series of LLM agents, each responsible for extracting and processing different aspects of the required information. The agents work collaboratively to generate background prompts, object names and descriptions, layout information, and mask images. By leveraging these techniques, our method improves upon existing text-to-image generation methods, offering enhanced precision and reliability in generating images that accurately reflect the input text prompts.

Once the necessary information is extracted by the LLM agents, the image generation process begins with the creation of the background. Subsequently, an iterative refinement process is employed to accurately place and detail each object within the image. This process ensures that the objects are accurately represented in terms of their descriptions, positions, sizes, and quantities as specified in the initial text prompt.

The overall architecture relies on advanced LLMs and text-to-image models, such as GPT-4o[11, 12], Stable Diffusion[13], and DALL-E[14, 15]. We used GPT-4o as a baseline for text generation tasks due to its advanced language understanding capabilities. DALL-E and Stable Diffusion[13] were utilized for inpainting tasks. These models are utilized without additional training, which allows for easy updates and improvements as newer models become available. While we utilized the AutoGen [16] framework (`https://github.com/microsoft/autogen`) for implementing our LLM agents, the most part of the our system was coded directly by us. This integration facilitates a robust and flexible framework capable of handling a variety of text-to-image generation tasks.

3.2 LLM Agent Process

The LLM agent process involves several key agents and steps, utilizing a framework inspired by AutoGen [16], ChatDev [6], and MetaGPT [17]. These agents include:

- **User Proxy**: Receives the initial text prompt and forwards it to the other agents.
- **Position Bot**: Generates layout information, including the position and size of each object.
- **Position Verifier Bot**: Validates the layout generated by the Position Bot, requesting re-generation if necessary.
- **Background Generation Bot**: Creates a background prompt based on the extracted information.
- **Mask Generation Bot**: Generates mask images for the objects based on the layout information.

The pseudo-code for this process is detailed below:

Algorithm 1 LLM Agent Process

Input: Text prompt p
Extract object names and descriptions
Generate initial layout (positions and sizes) for objects
Verify layout based on p
if layout is invalid **then**
    Regenerate layout with feedback
end if
Create background prompt (optional)
Generate mask images based on layout
Output: Background prompt(optional), object names, layout, and mask images

The LLM agents employ advanced techniques such as planning, tools, reflection, self-critics, subgoal decomposition, and a code interpreter to enhance their functionality. By incorporating concepts of collaborative agents and tool-augmented LLM agents, the process ensures efficient processing and high accuracy in task execution. These methods allow the agents to handle complex instructions and generate detailed and accurate outputs. The integration of collaborative and tool-augmented LLM agents significantly enhances the overall performance, making the approach robust and adaptable to various text-to-image generation challenges [1].

3.3 Iterative Refinement Process

The iterative refinement process ensures that each object is accurately represented in the final image. The steps are as follows:

1. **Masking**: Apply the generated mask images to the background.

2. **Inpainting Prompt Creation**: Create prompts for inpainting each object using their names and descriptions.
3. **Inpainting**: Use the inpainting prompt and masked image to generate the final image.

This process is repeated for the total number of objects, ensuring precise placement and detailing. Given a complex prompt p, the LLM decomposes it into N object-wise sub-prompts $p_{1:N}$. The decomposition follows a predefined order, generating each object conditioned on the previous objects. This iterative refinement allows the model to maintain accurate spatial relationships and attribute bindings throughout the image generation process.

Algorithm 2 Iterative Refinement Process

Input: Background prompt(optional), object names, layout, and mask images
for each object i in the list of objects **do**
    Apply mask to the background
    Create inpainting prompt for object i
    Inpaint object i on the masked image
end for
Output: Final generated image

By leveraging these techniques, our method offers enhanced precision and reliability in generating images that accurately reflect the input text prompts. This integration of collaborative and tool-augmented LLM agents significantly enhances the overall performance, making the approach robust and adaptable to various text-to-image generation challenges.

4 Experiments

4.1 Data

In our study, we adopted the benchmark method proposed by (Lian et al., 2023)[18]. to generate a diverse set of object positioning prompts. We utilized a set of 10 object classes: ['backpack', 'book', 'bottle', 'bowl', 'car', 'cat', 'chair', 'cup', 'dog', 'laptop']. Each prompt specifies a number of objects ranging from a minimum of 1 to a maximum of 5.

To ensure a balanced distribution of objects, we randomly assigned the positions of each object to one of four possible locations: left, right, top, or down. Additionally, to maintain spatial diversity within each prompt, when one object class was assigned to a specific position, the other object class was assigned to the opposite position. For instance, if 'backpack' was placed on the left, 'book' would be placed on the right, and if 'bottle' was positioned at the top, 'bowl' would be placed down.

Following this methodology, we generated a total of 100 unique prompts. The generated prompts are designed to test the robustness and accuracy of these algorithms in different spatial configurations, providing a reliable benchmark for performance assessment.

4.2 Evaluation method

We employed two distinct evaluation metrics to assess the performance of our model: generative numeracy and spatial reasoning.

4.2.1 Generative Numeracy

Generative numeracy measures the accuracy of the number of objects generated in comparison to the number of objects specified in the prompt. For each prompt, we calculate the absolute difference between the specified and generated numbers of each object class, k_1 and $\hat{k}_1$ for object 1, and k_2 and $\hat{k}_2$ for object 2. This difference is then normalized by dividing it by the sum of the specified and generated counts for each object class. The average of these normalized differences is used as the metric. The formula used is as follows:

$$\text{Generative Numeracy} = \frac{1}{2}\left(\frac{|k_1 - \hat{k}_1|}{k_1 + \hat{k}_1} + \frac{|k_2 - \hat{k}_2|}{k_2 + \hat{k}_2}\right) \tag{1}$$

This metric provides an average measure of discrepancy between the intended and generated number of objects across all prompts. A lower value indicates higher accuracy in generating the correct number of objects as specified by the prompts.

4.2.2 Spatial Reasoning

Spatial reasoning evaluates the correctness of the relative positions of objects generated in response to the prompts. Each object's position is classified into one of four possible locations: left, right, top, or down. For each prompt, the relative positioning of the generated objects is compared to the specified positions. This comparison is done in a binary fashion, where a match is counted as correct and a mismatch as incorrect. The accuracy is then calculated as the percentage of correct positional matches out of the total possible matches.

$$\text{Spatial Reasoning Accuracy} = \frac{\text{Number of Correct Positional Matches}}{\text{Total Possible Matches}} \tag{2}$$

This metric provides a percentage score indicating how well the generated objects adhere to the specified spatial configurations. Higher percentages reflect better spatial reasoning capabilities of the model.

4.3 Experimental details

The experiment was conducted on 30 randomly generated prompts. Each prompt specifies the type and location of objects. No information other than the text prompt is given to the model.

The same text prompts were fed to both the baseline model and PicassoGen, and the results were evaluated according to two metrics (See Tab. 1).

Model Name	Generative numeracy ↓	Spatial reasoning ↑
Stable Diffusion v1.4[13]	0.33	0.20
Ours	**0.31**	**0.72**

Table 1: Evaluation results.

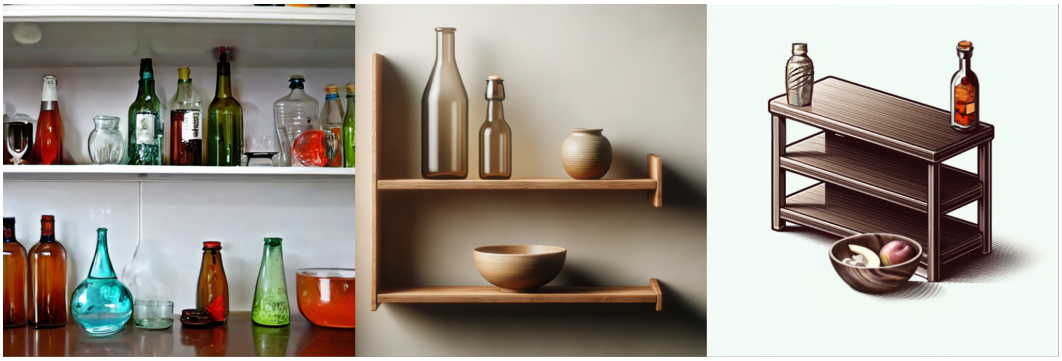
4.4 Results

The experimental results are summarized in Table 1, where we compare the performance of our model against the Stable diffusion[13] on two key metrics: generative numeracy and spatial reasoning. Our model demonstrated a lower generative numeracy value and a higher spatial reasoning accuracy compared to Stable Diffusion[13].

5 Analysis

The results indicate that our model significantly outperforms the Stable Diffusion[13] in both generative numeracy and spatial reasoning. Specifically, our model achieved a generative numeracy score of 0.31, which is lower than the 0.33 score of Stable Diffusion[13]. This demonstrates our model's superior ability to generate the correct number of objects as specified in the prompts, highlighting its effectiveness in understanding and implementing detailed numerical constraints.

Furthermore, our model achieved a spatial reasoning accuracy of 0.72, substantially higher than the 0.20 accuracy of Stable Diffusion[13]. This indicates that our model excels at accurately placing objects in the correct spatial relationships as specified by the prompts. The advanced techniques of subgoal decomposition, planning, reflection, and tool usage, combined with the iterative refinement process, contribute to this significant improvement in spatial reasoning.



Stable Diffusion DALL·E Ours

Figure 2: Qualitative comparison for the prompt: "Two bottles on the shelf and one bowl below that".

Figure 2 shows the images generated by the baseline model and PicassoGen. The baseline model fails to correctly match the number and positions of objects, whereas PicassoGen accurately matches both the number and positions of the objects.

6 Conclusion

Our experimental analysis demonstrates that the integration of LLM agents with advanced planning and inpainting techniques leads to substantial improvements in text-to-image generation tasks. Our model's lower generative numeracy and higher spatial reasoning accuracy compared to the previous model highlight its ability to more accurately generate images that adhere to detailed textual prompts. These findings suggest that our approach can set new benchmarks in the field of text-to-image generation, providing a robust framework for handling detailed and specific prompts with high fidelity. Future work could explore further enhancements and applications of this approach to other complex generation tasks.

References

[1] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):1–26, 2024.

[2] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2024.

[3] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36, 2024.

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

[5] Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.

[6] Chen Qian, Xin Cong, Wei Liu, Cheng Yang, Weize Chen, Yusheng Su, Yufan Dang, Jiahao Li, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development, 2023.

[7] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *In the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*, UIST '23, New York, NY, USA, 2023. Association for Computing Machinery.

[8] Kai Mei, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. Aios: Llm agent operating system. *arXiv:2403.16971*, 2024.

[9] Sen Li, Ruochen Wang, Cho-Jui Hsieh, Minhao Cheng, and Tianyi Zhou. Mulan: Multimodal-llm agent for progressive multi-object diffusion, 2024.

[10] Hanan Gani, Shariq Farooq Bhat, Muzammal Naseer, Salman Khan, and Peter Wonka. LLM blueprint: Enabling text-to-image generation with complex and detailed prompts, 2024.

[11] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[12] OpenAI. Gpt-4o. `https://openai.com/index/hello-gpt-4o/`, 2024.

[13] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, June 2022.

[14] OpenAI. Dall-e 2. `https://openai.com/index/dall-e-2/`, 2022.

[15] OpenAI. Dall-e 3. `https://openai.com/index/dall-e-3/`, 2023.

[16] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. 2023.

[17] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2023.

[18] Long Lian, Boyi Li, Adam Yala, and Trevor Darrell. Llm-grounded diffusion: Enhancing prompt understanding of text-to-image diffusion models with large language models. *arXiv preprint arXiv:2305.13655*, 2023.

A Appendix: Detailed Prompt Templates for LLM agents

As described in Section 3.2, various agents within our framework utilize LLMs to perform their tasks. For each input prompt p, the LLM is prompted using the following template:

User Proxy

> **Prompt:** "Make rectangle masks for the following prompt: 'p'. The position bot will decide the positions and sizes first, and the position verifier bot will verify the positions. If there are no issues, the mask generation bot will generate and save the masks. If there are problems, the position bot must reposition."

Position Bot

> **Prompt:** "You are responsible for determining the positions and sizes of rectangles surely within an image of size (self.image size). Ensure the rectangles surely do not overlap and stay within the image boundaries. Provide the 'position list' (List[List[int]]) to the position verifier bot. Each item in the list should be formatted as [center x, center y, width, height]. Make sure the bounding boxes do not overlap or go beyond the given image size boundaries. Adjust positions and sizes always to fit within the boundaries. All object names must be included in a single list. If necessary, make reasonable guesses to position the objects naturally within the scene described by the user prompt. Combine all object names and their positions into a single list before sending it to the position verifier bot. Small width and height are not good."

Position Verifier Bot

> **Prompt:** You are responsible for verifying the correctness and naturalness of the positions and sizes of the rectangles based on their names and given coordinates. The images are of size self.image size. Each bounding box should be in the format (object name, [center x, center y, width, height]). Make sure the bounding boxes do not overlap or go beyond the given image size boundaries. Adjust positions and sizes if necessary to fit within the boundaries. Ensure that all bounding boxes do not overlap and are sufficiently spaced apart. If any bounding boxes overlap, request repositioning from the position bot. All object names must be included in a single list, and verify that the positions and sizes do not exceed the boundaries of a self.image size image. Combine all object names and their positions into a single list before sending it to the mask generation bot. If the positions are correct, provide the combined position list to the mask generation bot. If the positions are not correct, make the position bot reposition and give the reason. Small width and height are not good.

Background Generation Bot

> **Prompt:** You are responsible for generating detailed and accurate background prompts for images of size self.image size. Ensure the backgrounds complement the positions and sizes of the rectangles based on their names and given coordinates. Each bounding box is in the format (object name, [center x, center y, width, height]). Make sure the background elements do not interfere with or obscure the bounding boxes. Adjust the background details to naturally fit within the image boundaries and to enhance the overall scene. Ensure that all background elements are sufficiently spaced apart from the bounding boxes to avoid overlap. The generated background should be consistent with the positions and sizes of the objects and should maintain the image boundaries of self.image size. Verify that the entire composition, including both background and foreground elements, remains within the boundaries of the image size self.image size. Ensure that the final prompt includes all necessary details to accurately render the background without causing overlaps or boundary issues with the foreground objects.

Mask Generation Bot

Prompt: You are responsible for verifying the correctness and naturalness of the positions and sizes of the rectangles based on their names and given coordinates. The images are of size self.image size. Each bounding box should be in the format (object name, [center x, center y, width, height]). Make sure the bounding boxes do not overlap or go beyond the given image size boundaries. Adjust positions and sizes if necessary to fit within the boundaries. Ensure that all bounding boxes do not overlap and are sufficiently spaced apart. If any bounding boxes overlap or exceed the image size, request repositioning from the position bot. All object names must be included in a single list, and verify that the positions and sizes do not exceed the boundaries of a self.image size image. Combine all object names and their positions into a single list before sending it to the mask generation bot. If the positions are correct, provide the combined position list to the mask generation bot.