SCI066: Can large language models generate SVG code for vector graphics?

Victor Qiu

Supervisor: Professor Michael Witbrock and Dr. Trung Nguyen

Host Unit: Department of Computer Science, Faculty of Science, The University of Auckland

Degree/Programme: Bachelor of Engineering (Honours)/Bachelor of Science, Software Engineering, Mathematics

Statement of Scholarship Impact

Under the guidance of Professor Michael Witbrock and Dr. Trung Nguyen, the Summer Research Scholarship has been a transformative experience, in both my academic and professional journey. It has provided me with the unique opportunity to engage with advanced technologies and experts in the field of artificial intelligence (AI). This experience has not only enhanced my technical skills but also fostered critical thinking skills. It has solidified my interest in pursuing postgraduate studies, particularly in the realm of computer science.

Summary of Research and Significance

Vector graphics, which are created through code, are notable for their lossless quality; they maintain their clarity regardless of zoom level. This feature makes them invaluable in the field of graphic design. This research project explores the development of how vector graphics can be generated using machine learning models. Key aspects of the research include testing how capable current models are in creating simple to complex graphics, understanding the limitations of current AI technologies in this domain, and exploring reinforcement learning methods to improve the model's performance. This project highlights the seldom-explored intersection of AI and graphic design, showcasing it's potential and challenges.

Abstract

Vector graphics have become a staple in modern-day graphic design. Able to be scaled to any resolution, they offer several advantages compared to their raster counterpart. Despite this being the case, AI generation of vector graphics, although rising in traction, remains a rarity. The following report aims to critically assess the proficiency of AI models in generating vector graphics from textual descriptions. This research will primarily evaluate GPT-4's performance in producing Scalable Vector Graphics (SVG), highlighting its adeptness in producing basic shapes to more complex designs. To overcome current limitations in vector graphic generation, the study proposes the idea of an automated feedback loop mechanism. This is an iterative refinement process where vector graphics are converted to a raster format. Leveraging the multimodal capabilities of GPT-4 Vision, feedback is able to be given to the generated graphics, allowing them to be improved. At the end, we explore the Contrastive Language-Image Pre-training (CLIP) model, and how it can be used to produce a methodology for fine-tuning a model which generates vector graphics.

Contents

1 Introduction

1.1 What are Vector Graphics?

1.1.1 Raster vs Vector Graphics

Graphics primarily fall into two categories:

- **Raster graphics**, and
- **Vector graphics**

The difference between the two is how they are created and rendered. **Raster graphics** are composed of pixels, small squares of colour, which together form an image. Being pixel-based, means that raster graphics lose clarity, and become pixelated when scaled up. On the contrary, **vector graphics** are defined by mathematical equations. This approach allows them to be scaled to any size without losing clarity.

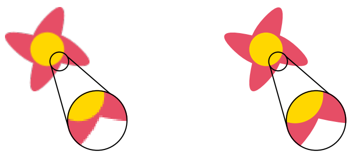


Figure 1: Raster Graphic (Left) and Vector Graphic (Right)

1.1.2 Technical Composition of Vector Graphics

Vector graphics are created using mathematical formulas and can be represented in various formats. Scalable Vector Graphics (SVG), a widely-used Extensible Markup Language (XML) format, facilitates graphic generation using geometric shapes or paths [1].

SVG Command	Description
`<line>`	Straight line
`<circle>`	Centered circle
`<rect>`	Rectangle shape
`<ellipse>`	Ellipse shape
`<polygon>`	Multi-sided shape
`<polyline>`	Connected lines
`<path>`	Complex shapes

Table 1: Basic SVG Commands

It encodes information such as coordinates, colour, and line thickness, enabling the precise rendering of vector graphics.

```
<svg width="200" height="200" xmlns="
    ↪ http://www.w3.org/2000/svg">
  <path d="M 100,10 L 120,80 L 190,80
      ↪ L 130,120 L 150,190 L 100,150
      ↪  L 50,190 L 70,120 L 10,80 L
      ↪ 80,80 Z"
        fill="yellow"
        stroke="orange"
        stroke-width="5"/>
</svg>
```

Figure 2: SVG and Vector Graphic of a Star

1.2 Aims and Objectives

This project encompasses three core objectives, which are outlined below:

- Evaluating AI in Vector Graphics,
- Feedback Loops, and
- Reinforcement Learning for Fine-Tuning

The goals outlined above will be delved into with a specific focus on generating vector graphics represented by SVG code. These objectives will be thoroughly examined in the subsequent sections.

2 Evaluating AI in Vector Graphics

2.1 Current Methods

The advent of AI in graphic design has predominantly been marked by its success in raster graphic generation, with groundbreaking projects like DALL-E [2] showcasing the potential of AI in this field. However, vector graphic generation using AI has been less prevalent compared to it's raster counterpart, where it has not seen the same level of exploration and development. Currently, while there are limited methods available for generating vector graphics, two notable approaches include:

- Direct SVG code generation, and
- Diffusion models in raster to vector conversion

This research project will focus on direct SVG code generation, primarily due to it being more primitive in nature, which provides a wider scope for exploration. However, a high-level summary for both approaches will be provided in the subsequent subsections.

2.1.1 Direct SVG Code Generation

A textual description of an object is input into a Large Language Model (LLM) such as GPT. This model then generates SVG code, which when rendered, outputs an vector graphic.

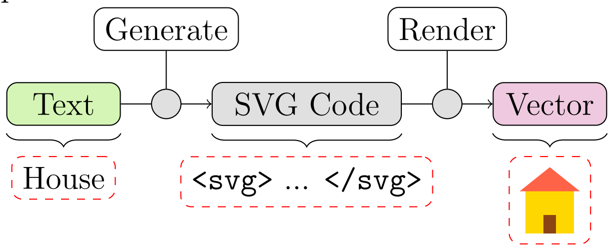


Figure 3: Text to SVG Code to Vector Graphic

2.1.2 Diffusion Models in Raster to Vector Conversion

A textual description of an object is input into a diffusion model, which outputs a raster graphic. This raster graphic is then vectorised into a vector graphic. A notable application of this methodology is VectorFusion [3].

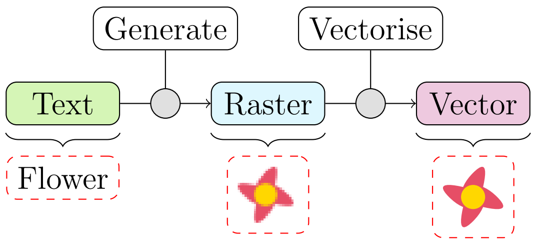


Figure 4: Text to Raster to Vector Graphic

2.2 Analysis of Direct SVG Code Generation with GPT

The exploration of direct SVG code generation in this project involves using OpenAI's GPT-4 Model.

2.2.1 Simple Shapes

In the case of simple shapes, GPT models have shown a high degree of accuracy. It can interpret basic instructions for shapes to construct circles, rectangles, and polygons, converting these into precise SVG code.

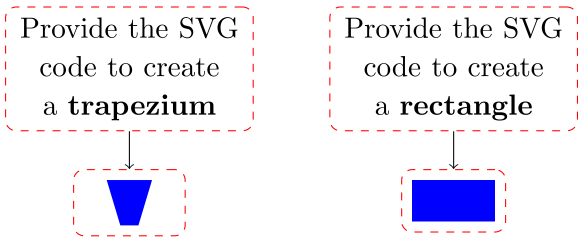

Provide the SVG code to create a **trapezium**
Provide the SVG code to create a **rectangle**

Figure 5: Prompt and Output for a Trapezium (Left) and a Rectangle (Right)

2.2.2 Complex Shapes

When it comes to complex shapes, the challenges increase. The GPT models, while capable, will struggle with the intricacies and details of more complex designs.

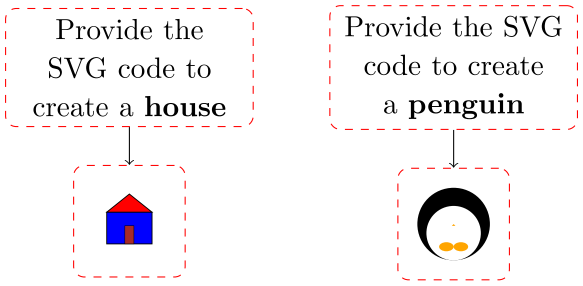

Provide the SVG code to create a **house**
Provide the SVG code to create a **penguin**

Figure 6: Prompt and Output for a House (Left) and a Penguin (Right)

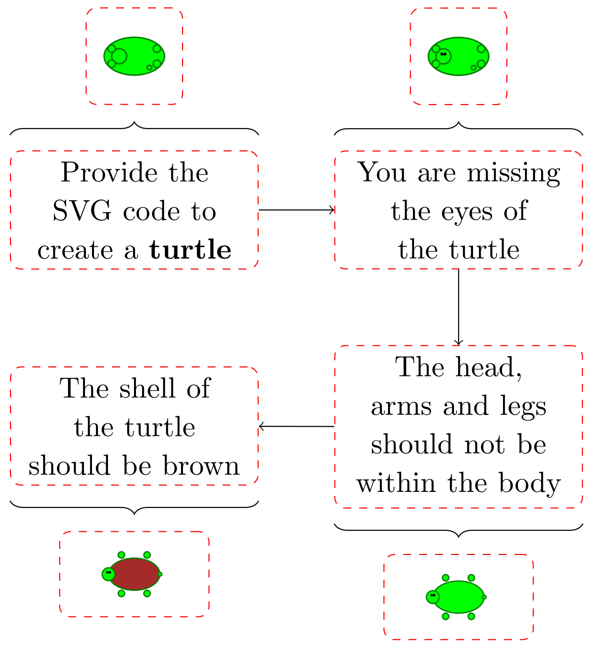
3 Feedback Loops

While GPT-4 is adequate for generating simple shapes, its performance on complex vector graphics often results in missing details, misplaced elements, or even outright inaccuracies. To address these concerns, we propose the implementation of a feedback loop, which would provide corrective inputs over a series of generation iterations. In the following subsections, we will examine a theoretical, idealised approach alongside a practical and easy to implement solution.

3.1 Ideal Approach

Ideally, the vector graphic generation process would involve identifying and correcting each incorrect detail iteratively until all inaccuracies are eliminated, resulting in a high-quality output.

For instance, when generating a **turtle**, should there be errors, the ideal response would be to prompt akin to the following, until the desired result:



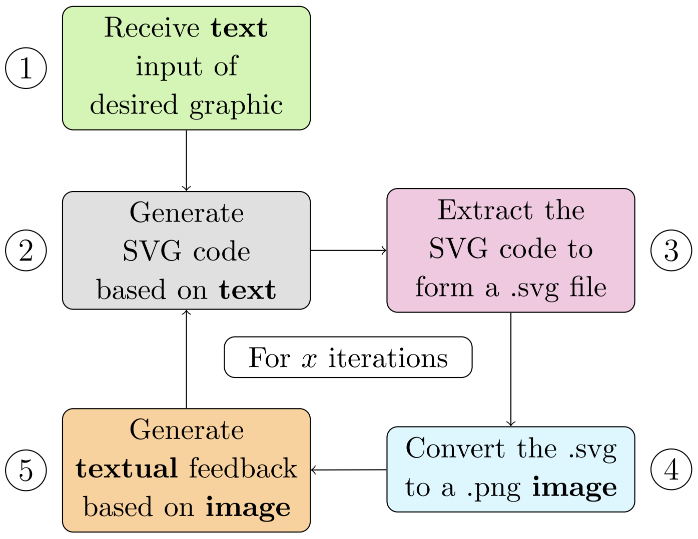
Provide the SVG code to create a **turtle**
You are missing the eyes of the turtle
The head, arms and legs should not be within the body
The shell of the turtle should be brown

Figure 7: Ideal Approach via Manual GPT Prompting

However, this approach would require intensive human labour, making it impractical, necessitating the need of a more automated alternative.

3.2 Automated Approach

In light of this, we propose the following methodology:



1
Receive **text** input of desired graphic
2
Generate SVG code based on **text**
3
Extract the SVG code to form a .svg file
For x iterations
4
Convert the .svg to a .png **image**
5
Generate **textual** feedback based on **image**

Figure 8: Automated Feedback Loop

The steps are as follows:

1. We first receive a string input of what the desired graphic is.

   Repeat steps 2 - 4 for x iterations

2. Based on the textual description provided, a model such as GPT-4 is then used to generate the corresponding SVG code.

3. We then take the output, and extract the text between the first `<svg>` tag and the last `</svg>`. We then save this as an .svg file.

4. Following, we then convert this .svg file to a .png file, converting the vector graphic to a raster graphic. This step leverages the advanced multimodal capabilities of GPT models, which have shown significant improvements in processing image inputs [4].

5. The raster graphic is then passed into a model that accepts image inputs, such as GPT-4 Vision, where it then evaluates and issues feedback on one thing that needs to be improved.

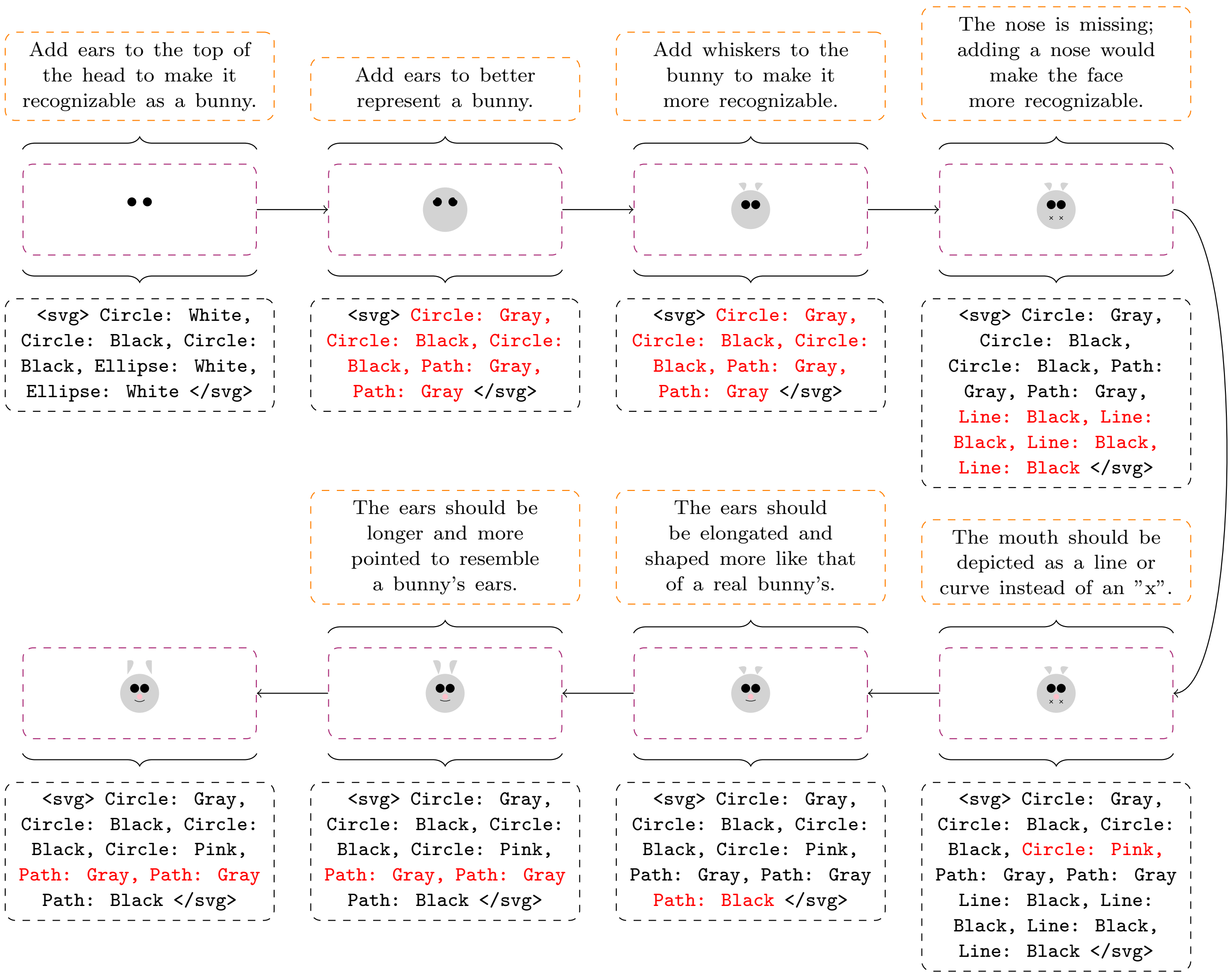
The exact prompts utilised for this methodology are as follows:

Step	Prompt
2.1	Generate simplistic SVG code representing a `<textInput>`. Ensure the SVG is of size 128x128.
2.2	Generate simplistic SVG code representing a userInput. Ensure the SVG is of size 128x128. The feedback provided for the graphic generated in the prior iteration was `<feedback>`.
5	The attached image is the current output for a `<textInput>` generated by GPT using SVG code. Please provide concise feedback focusing on a single key aspect that needs improvement to better represent the described object. In particular, if a certain feature is missing, or a feature has been drawn in the incorrect location, suggest to fix the feature. Provide only one very short sentence.

Table 2: Figure 8 Steps and Associated Prompts: In the first iteration, Step 2 uses Prompt 2.1. For all subsequent iterations, Prompt 2.2 is utilised.

3.2.1 Experiments

Evaluation shows the feedback loop enhances vector graphic generation, though outcomes can still be rudimentary and deteriorate over an excessive number of iterations. A representative illustration of a standard procedural execution is presented below, using a **bunny** as example:



Add ears to the top of the head to make it recognizable as a bunny.

```
<svg> Circle: White, Circle: Black, Circle: Black, Ellipse: White, Ellipse: White </svg>
```

Add ears to better represent a bunny.

```
<svg> Circle: Gray, Circle: Black, Circle: Black, Path: Gray, Path: Gray </svg>
```

Add whiskers to the bunny to make it more recognizable.

```
<svg> Circle: Gray, Circle: Black, Circle: Black, Path: Gray, Path: Gray </svg>
```

The nose is missing; adding a nose would make the face more recognizable.

```
<svg> Circle: Gray, Circle: Black, Circle: Black, Path: Gray, Path: Gray, Line: Black, Line: Black, Line: Black, Line: Black </svg>
```

The mouth should be depicted as a line or curve instead of an ”x”.

```
<svg> Circle: Gray, Circle: Black, Circle: Black, Circle: Pink, Path: Gray, Path: Gray Line: Black, Line: Black, Line: Black, Line: Black </svg>
```

The ears should be elongated and shaped more like that of a real bunny's.

```
<svg> Circle: Gray, Circle: Black, Circle: Black, Circle: Pink, Path: Gray, Path: Gray Path: Black </svg>
```

The ears should be longer and more pointed to resemble a bunny's ears.

```
<svg> Circle: Gray, Circle: Black, Circle: Black, Circle: Pink, Path: Gray, Path: Gray Path: Black </svg>
```

```
<svg> Circle: Gray, Circle: Black, Circle: Black, Circle: Pink, Path: Gray, Path: Gray Path: Black </svg>
```

Figure 9: SVG Bunny Generation: Illustrations in pink, feedback in orange, and pseudo-SVG code (Command: Colour) in grey. Code modifications such as colour, positioning, and etc across iterations are highlighted in red.

4 Reinforcement Learning for Fine-Tuning

The following looks at an approach which could be utilised to fine-tune a vector graphic generating model using reinforcement learning techniques.

4.1 Contrastive Language-Image Pre-training

Open AI's Contrastive Language-Image Pre-Training (CLIP) model is a neural network that has been trained on image and text pairs. This model is therefore able to assess the relevance or similarity between a given text and image [5]. Hence, for any generated vector graphic representation of an object x, once converted into a PNG, we can then categorise it with two distinct labels:

- Graphic of a x
- Not a graphic of a x

This binary labelling system covers the entire sample space of possibilities, assigning a probability to each label such that their combined total sums to one.

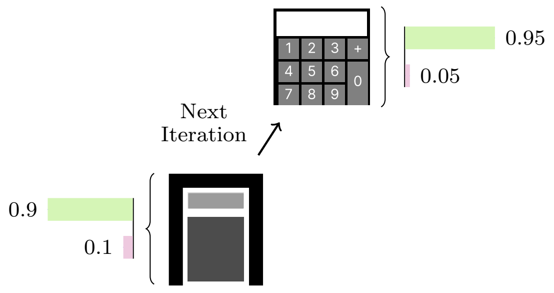


Figure 10: SVG Calculator Generation: Under the CLIP model, the probability that the graphic is of a calculator is represented by the green bar. Probability that it is not a calculator is represented by the pink bar.

As such, utilising the CLIP model, we are able to conform to the reinforcement learning paradigm [6], allowing us to set an:

- **Environment**: The environment in this context is defined as the space where vector graphic generation occurs.
- **Reward**: The reward is quantified as the probability assigned by the CLIP model to the label "Graphic of a x." A higher probability correlates with a more accurate representation of the object x, thereby yielding a higher reward. This probabilistic reward, serves as feedback for the model.
- **Agent**: The agent is the model tasked with vector graphic generation. It operates within the environment, using textual descriptions to produce a corresponding graphic. The reward it receives after each iteration informs and influences its subsequent generations, fine-tuning the model.

This results in a methodology for fine-tuning models that generate vector graphics, in forms such as SVG code.

5 Conclusion

This report concentrated on three primary objectives. We first analysed the current state of vector graphic generation, which although promising, is behind its raster counterpart. Following, we proposed an automated feedback loop to help improve results. This proved to be promising, albeit, has a lot of room left to improve. Lastly, we presented a methodology for fine-tuning a vector graphic generation model using reinforcement techniques, leveraging the CLIP model. Overall, we believe this report will be able to help contribute to the space that is at the intersection of graphics and AI.

References

[1] Quint, A. (2003). Scalable vector graphics. *IEEE MultiMedia, 10*(3), 99-102. `https://doi.org/10.1109/MMUL.2003.1218261`

[2] Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., & Sutskever, I. (2021). Zero-Shot Text-to-Image Generation. arXiv. `https://arxiv.org/abs/2102.12092`

[3] Jain, A., Xie, A., & Abbeel, P. (2022). VectorFusion: Text-to-SVG by Abstracting Pixel-Based Diffusion Models. arXiv. `https://arxiv.org/abs/2211.11319`

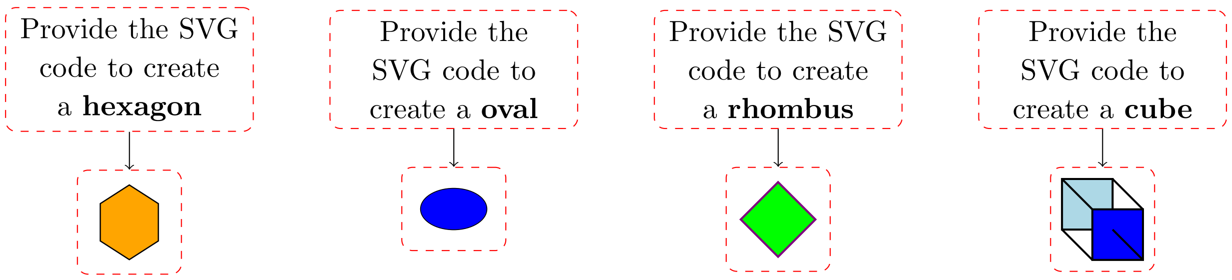
[4] OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., et al. (2023). GPT-4 Technical Report. arXiv. `https://arxiv.org/abs/2303.08774`

[5] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., & Sutskever, I. (2021). Learning Transferable Visual Models From Natural Language Supervision. arXiv. `https://arxiv.org/abs/2103.00020`

[6] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT Press.

Appendix

A Direct SVG Code Generation with GPT of Simple Shapes



Provide the SVG code to create a **hexagon**

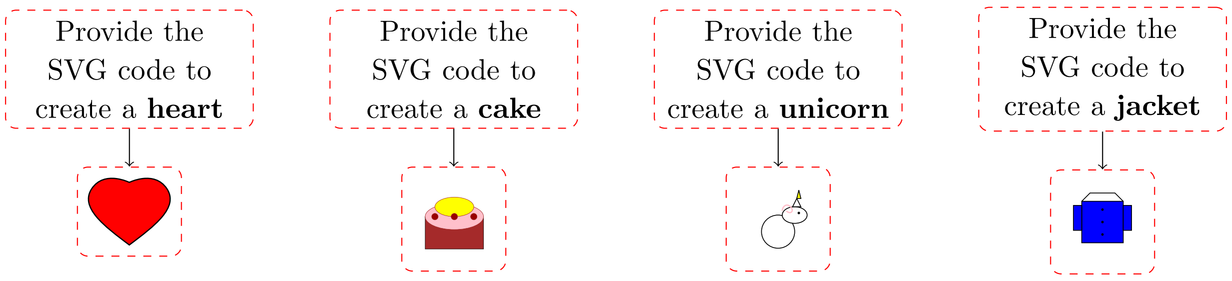
Provide the SVG code to create a **oval**

Provide the SVG code to create a **rhombus**

Provide the SVG code to create a **cube**

Figure 11: Prompt and Output for Various Simple Shapes

B Direct SVG Code Generation with GPT of Complex Shapes



Provide the SVG code to create a **heart**

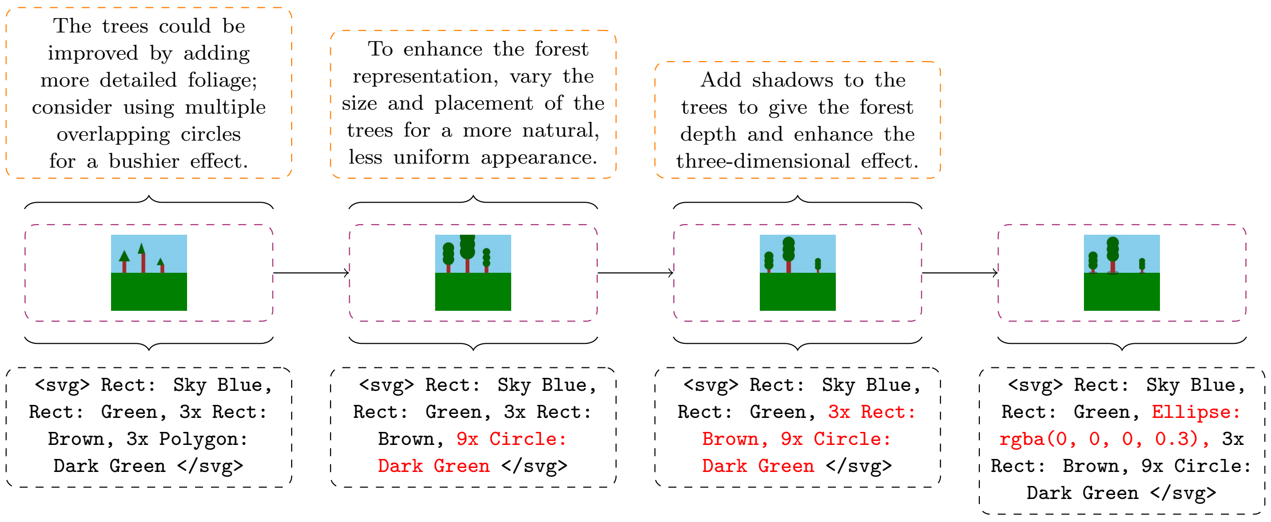
Provide the SVG code to create a **cake**

Provide the SVG code to create a **unicorn**

Provide the SVG code to create a **jacket**

Figure 12: Prompt and Output for Various Complex Shapes

C Feedback Loop SVG Code Generation



The trees could be improved by adding more detailed foliage; consider using multiple overlapping circles for a bushier effect.

To enhance the forest representation, vary the size and placement of the trees for a more natural, less uniform appearance.

Add shadows to the trees to give the forest depth and enhance the three-dimensional effect.

```
<svg> Rect: Sky Blue, Rect: Green, 3x Rect: Brown, 3x Polygon: Dark Green </svg>
```

```
<svg> Rect: Sky Blue, Rect: Green, 3x Rect: Brown, 9x Circle: Dark Green </svg>
```

```
<svg> Rect: Sky Blue, Rect: Green, 3x Rect: Brown, 9x Circle: Dark Green </svg>
```

```
<svg> Rect: Sky Blue, Rect: Green, Ellipse: rgba(0, 0, 0, 0.3), 3x Rect: Brown, 9x Circle: Dark Green </svg>
```

Figure 13: SVG Forest Generation: Illustrations in pink, feedback in orange, and pseudo-SVG code (Command: Colour) in grey. Code modifications such as colour, positioning, and etc across iterations are highlighted in red.