Natural Language to SVG Generation by Enforcing Constraints through Reinforcement Learning*

Henry Fordjour Ansah[1,*,†]

[1]University of New Orleans, USA

Abstract

This work addresses the challenge of translating free-form textual descriptions into precise, well-formed Scalable Vector Graphics (SVG). We introduce an end-to-end framework that ingests a natural-language prompt, identifies the visual elements and semantic relationships it implies, and produces syntactically valid SVG markup that closely matches the intended image. To ensure both fidelity and robustness, our method incorporates a constrained reinforcement learning stage, enforcing a subset of SVG tags and attributes, and a specialized reward function that balances structural correctness, semantic alignment, and code compactness. The proposal utilizes the Qwen 2.5 coder model and employs a two-stage approach - first, the model undergoes supervised fine-tuning on the deepseek-svg-dataset to enhance the model's capacity for instruction adherence. Second, to ensure the generated SVG adheres to specific constraints, reinforcement learning using the Group Relative Policy Optimization (GRPO) algorithm is applied. This project aims to develop a robust system capable of translating natural language into valid and constrained SVG code.

Keywords

Natural Language, SVG, Reinforcement Learning, Supervised-Finetuning, Group Relative Policy Optimization

1. Introduction

The task of translating natural language into vector graphics holds significant potential for creative expression and programmatic content generation. This project addresses this challenge within the context of the Kaggle competition "Drawing with LLMs," which requires participants to generate SVG code from textual descriptions. The core approach involves leveraging the reasoning capabilities of a large language model, specifically the Qwen 2.5 coder model [1], and refining its output through a combination of supervised fine-tuning and reinforcement learning. This methodology aims to produce SVG code that is not only semantically relevant to the input prompt but also strictly adheres to the competition's constraints on size, allowed elements and attributes and the generation of SVG code whose renderings are semantically similar to their corresponding prompts. This project builds upon recent advancements in large language models and their application to code generation and creative tasks. The expected outcome is a model capable of generating high-quality, constrained SVG code from natural language prompts, demonstrating the effectiveness of the proposed two-stage training process.

2. Problem Definition and Algorithm

2.1. Task Definition

The problem addressed in this project is the generation of valid and constrained Scalable Vector Graphics (SVG) code from natural language descriptions. The input to the system is a textual prompt describing an image or graphic. The output is a string of SVG code representing that image. This task is crucial for the "Drawing with LLMs" Kaggle competition, where submissions are evaluated based on the quality

HHAI-WS 2025: Workshops at the Fourth International Conference on Hybrid Human-Artificial Intelligence (HHAI), June 9–13, 2025, Pisa, Italy

*Corresponding author.

†These authors contributed equally.

hfansah@uno.edu (H. F. Ansah)

https://hansah.me (H. F. Ansah)

and constraint compliance of the generated SVGs. The constraints are: 1) No SVG may be more than 10,000 bytes long. 2) Each SVG may only include elements and attributes from an allowlist, excluding CSS style elements. 3) No SVG may include any rasterized image data or data from external sources. Additionally, the evaluation system requires that an SVG is returned within 5 minutes, and all SVGs are generated in under 9 hours. This problem is interesting due to the complexity of mapping natural language concepts to precise SVG syntax and the added challenge of enforcing strict output constraints.

2.2. Algorithm Definition

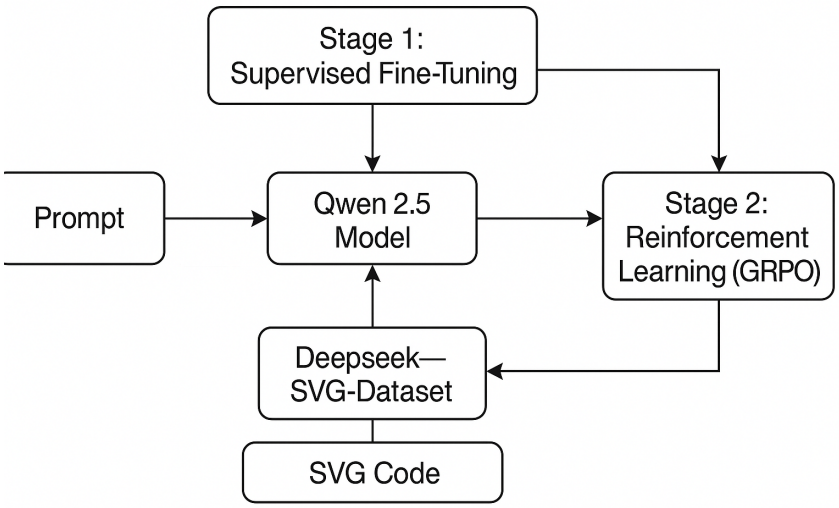

Figure 1: Two-Stage Training Process: Supervised Fine-Tuning followed by Reinforcement Learning (GRPO) on Qwen 2.5 Model.

The proposed algorithm consists of two main stages: supervised fine-tuning and reinforcement learning.

Stage 1: Supervised Fine-tuning (SFT) The first stage involves fine-tuning an instruction-tuned Qwen 2.5 coder language model, denoted as M_{θ_0} with parameters θ_0, on the deepseek-svg-dataset. The dataset $\mathscr{D}_{SFT} = \{(x_i, y_i)\}_{i=1}^{N}$ consists of N examples, where each $x_i = (p_i, r_i)$ represents the input, comprising a natural language prompt p_i and a reasoning trace r_i, and y_i is the target SVG code. The goal of supervised fine-tuning is to learn a new set of model parameters θ_{SFT} that minimizes the negative log-likelihood of the target SVG code given the input prompt and reasoning trace.

Let $y_i = (y_{i,1}, y_{i,2}, ..., y_{i,L_i})$ be the sequence of tokens representing the SVG code of length L_i. The conditional probability of generating the target sequence given the input x_i and model parameters θ is:

$$P(y_i|x_i;\theta) = \prod_{j=1}^{L_i} P(y_{i,j}|y_{i,<j}, x_i;\theta)$$

where $y_{i,<j} = (y_{i,1}, ..., y_{i,j-1})$ represents the tokens generated so far.

The objective of supervised fine-tuning is to minimize the following loss function over the entire dataset:

$$\mathscr{L}_{SFT}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log P(y_i|x_i;\theta)$$

This minimization is typically performed using gradient descent-based optimization algorithms, such as Adam, to update the model parameters θ from the initial parameters θ_0 to the fine-tuned parameters θ_{SFT}.

Stage 2: Reinforcement Learning for Constraint Enforcement using Group Relative Policy Optimization (GRPO) The second stage aims to further fine-tune the model $M_{\theta_{SFT}}$ (now considered the initial policy $\pi_{\theta_0} = M_{\theta_{SFT}}$ with parameters $\theta_0 = \theta_{SFT}$) using reinforcement learning to ensure that the generated SVG code adheres to the competition's constraints. We employ the Group Relative Policy Optimization (GRPO) algorithm[2] introduced in the Deepseek Math paper for this purpose.

Let s be a natural language prompt sampled from a distribution $\mathcal{S}$. The policy $\pi_\theta(a|s)$ represents the probability of the model with parameters θ generating an SVG code a given the prompt s. We define a reward function $R(a)$ that assigns a high positive reward if the generated SVG code a satisfies all the constraints (size $\leq$ 10000 bytes, only allowed elements and attributes, no raster or external data) and a lower or zero reward otherwise.

In practice, the GRPO update often involves sampling multiple actions from the current policy for each prompt and comparing their rewards. The goal is then to maximize the expectation represented as the objective function:

GRPO offers a distinct advantage over traditional reinforcement learning approaches by leveraging relative comparisons rather than relying solely on absolute reward values. This makes GRPO particularly effective and efficient in environments where reward signals are sparse, noisy, or difficult to calibrate. Instead of attempting to estimate the true value of each action, GRPO simply asks whether one action is better than another within a given group, which leads to more stable learning and often faster convergence. This relative perspective aligns well with human feedback scenarios, where explicit numerical rewards may be unavailable but ranking or preference information is easier to obtain. Moreover, by focusing on group-wise preferences, GRPO can exploit more information per training batch—since each group allows for multiple pairwise comparisons—leading to more efficient policy updates. This approach can also mitigate issues like high variance in returns and instability that often plague absolute reward-based methods such as vanilla policy gradients or PPO.
For this project, we will be using the Huggingface Trl library[3] to perform both supervised fine-tuning and reinforcement learning training on our model.

3. Experimental Evaluation

3.1. Methodology

All experiments for this project were conducted on a single NVIDIA A100 GPU with 40GB of memory. We utilized the Qwen 2.5 coder model, and experiments were performed with the 7B parameter versions.

Supervised Fine-tuning Setup: For the supervised fine-tuning stage, the Qwen 2.5 coder model was trained on the deepseek-svg-dataset. Each instance of the dataset consist of a prompt, a step-by-step reasoning trace and a ground-truth svg answer. We perform data processing by placing the reasoning traces in think tags `<think>...</think>` and the SVG in generated tags `<answer>...</answer>`. We employed the AdamW optimizer with a learning rate schedule that included a warm-up phase followed by a decay. The batch size was set to 256 and the model was trained for 100 steps. The primary goal of this stage was to adapt the already instruction-tuned Qwen2.5 coder model to the task of generating SVG code from natural language prompts and reasoning traces present in the dataset. This process, popularized in the Deepseek paper as *cold start* enhances the model's capacity for instruction adherence and to generate responses in a specific format before reinforcement learning is applied.

Reinforcement Learning Setup: Following the supervised fine-tuning, we performed reinforcement learning using the Group Relative Policy Optimization (GRPO) algorithm on the *text2svg-stack* dataset[4] consisting of 2.17 million instances. Several reward functions were designed to provide a high positive reward only when the generated SVG code satisfied all the competition constraints: 1) the size was no more than 10,000 bytes, 2) it only included elements and attributes from the specified allowlist, and 3) it contained no rasterized image data or data from external sources. Also, we introduced other reward functions to incentivize the model to generate a response that adhered to the format of having

its chain-of-thought steps in think tags and the output in answer tags. Lastly, we added a similarity reward function that incentivizes the model to generated svg output that are semantically similar to the prompt descriptions. We discuss the reward functions below.

3.2. Reward Function Design

To train the SVG generation model using reinforcement learning, we adopt a multi-faceted reward strategy. This approach offers progressive feedback at different stages of the generation process, thereby improving the semantic fidelity, structural validity, and aesthetic quality of the outputs. Our reward functions are composed of six types: Tag Count Reward, Format Reward, Byte Length Reward, Semantic Similarity Reward, and Structural SVG Reward.

3.2.1. Tag Count Reward

We encourage the model to adhere to the expected output structure by awarding partial rewards for correctly used tags. Specifically, the tags <think>, </think>, <answer> and </answer> are each rewarded with 0.25, up to a total of 1.0. This soft guidance helps the model learn appropriate response formatting without being overly rigid.

$$S_t = 0.25 \cdot \mathbb{1}_{<think>} + 0.25 \cdot \mathbb{1}_{</think>} + 0.25 \cdot \mathbb{1}_{<answer>} + 0.25 \cdot \mathbb{1}_{</answer>}$$

3.2.2. Format Reward

To enforce generation structure, we apply a soft format reward that checks whether the response matches the regular pattern: <think>. . . /think>
<answer>. . . /answer>. If this condition is satisfied, a reward of 0.5 is given; otherwise, the reward is 0.

$$S_f = \begin{cases} 0.5, & \text{if format pattern is matched} \\ 0.0, & \text{otherwise} \end{cases}$$

3.2.3. Byte Length Reward

This reward ensures the model respects the byte-size limit of 10,000 bytes for SVG files. Outputs exceeding this limit receive a penalty of -1.0, while compliant outputs receive a reward of 0.5.

$$S_b = \begin{cases} 0.5, & \text{if byte\_length(SVG)} \leq 10{,}000 \\ -1.0, & \text{otherwise} \end{cases}$$

3.2.4. Semantic Similarity Reward

We evaluate how well the generated SVG image visually aligns with the input text prompt using a BLIP-based similarity model. The SVG is converted to a PNG, embedded alongside the text, and scored via cosine similarity. The resulting similarity score in $[0, 1]$ is used directly as the reward.

$$S_s = \text{sim}(\text{BLIP\_embed(text)}, \text{BLIP\_embed(SVG)})$$

3.2.5. Structural SVG Reward

This reward checks if the generated SVG begins with <svg and ends with </svg>, which indicates proper encapsulation of the drawing. Correct structure yields a reward of 1.0, otherwise 0.0.

$$S_{\text{struct}} = \begin{cases} 1.0, & \text{if SVG starts with } < svg \text{ and ends with } < /svg > \\ 0.0, & \text{otherwise} \end{cases}$$

4. Results

4.1. Sample Generated SVGs

To qualitatively assess the model's output, Figure 2 shows examples of SVG images generated after the full two-stage training process.

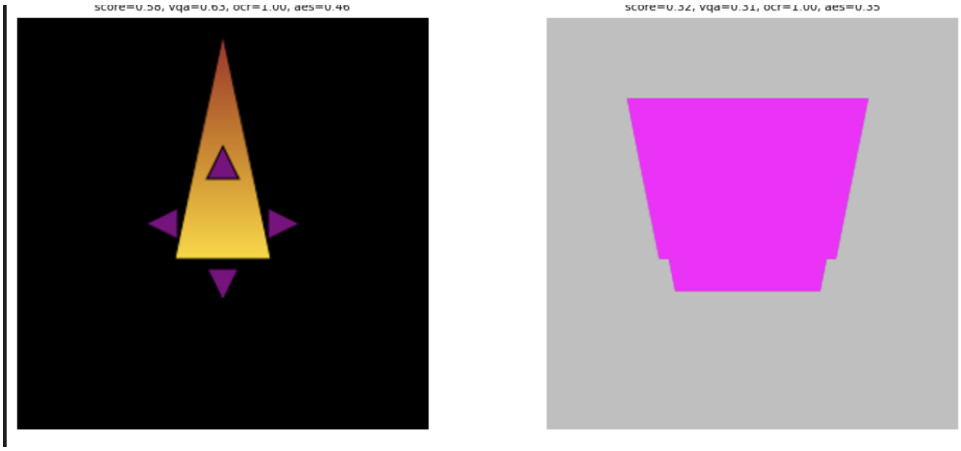

Figure 2: Sample SVG images generated by the Qwen2.5 model after Supervised Fine-Tuning and Reinforcement Learning. Prompt 1: Purple pyramids spiraling around a bronze cone. Prompt 2: Magenta trapezoids layered on a translucent silver sheet

The generated SVGs demonstrate the model's ability to produce complex geometric shapes with appropriate color gradients, fill styles, and background handling. Despite some variance in aesthetic quality, the outputs successfully adhere to structural and content constraints set by the competition guidelines.

4.2. Supervised Fine-Tuning (SFT) Results

During the initial supervised fine-tuning stage, the Qwen2.5 model was trained on the Deepseek-SVG dataset to learn the mapping from natural language prompts, combined with reasoning traces, to SVG code outputs. Figure 3 shows the training loss history across training steps.

The loss curve demonstrates a rapid and consistent decrease in training loss, stabilizing after approximately 100 steps. The initial high loss indicates the model's limited understanding of SVG generation at the start. As training progressed, the model quickly adapted, achieving a smooth convergence toward lower loss values around 0.4 to 0.5, indicative of successful learning from the Deepseek-SVG dataset. The stable and decreasing trend suggests that the model was effectively learning the structure and elements of valid SVG code along with the reasoning trace without signs of overfitting or divergence.

4.3. Reinforcement Learning (RL) with GRPO Results

Following supervised fine-tuning, the model was further refined using Group Relative Policy Optimization (GRPO) reinforcement learning to enforce strict competition constraints (e.g., maximum file size, allowed elements only, no raster images).

Figure 4 shows the training loss history during the reinforcement learning phase.

The RL loss curve exhibits significantly more instability compared to the SFT phase, characterized by sharp spikes at several points during early training (notably between step 5 and step 20). This behavior is typical in reinforcement learning, where exploration can cause highly variable rewards, especially when the model initially generates SVGs that either strongly violate or partially satisfy the constraints.

After the initial fluctuations, the loss gradually stabilizes toward zero, indicating that the model increasingly produced SVG outputs that conformed to the reward structure defined by the competition rules. The observed instability highlights the challenge of constraint-driven SVG generation, but the

Figure 3: Training Loss Curve during Supervised Fine-Tuning (SFT).

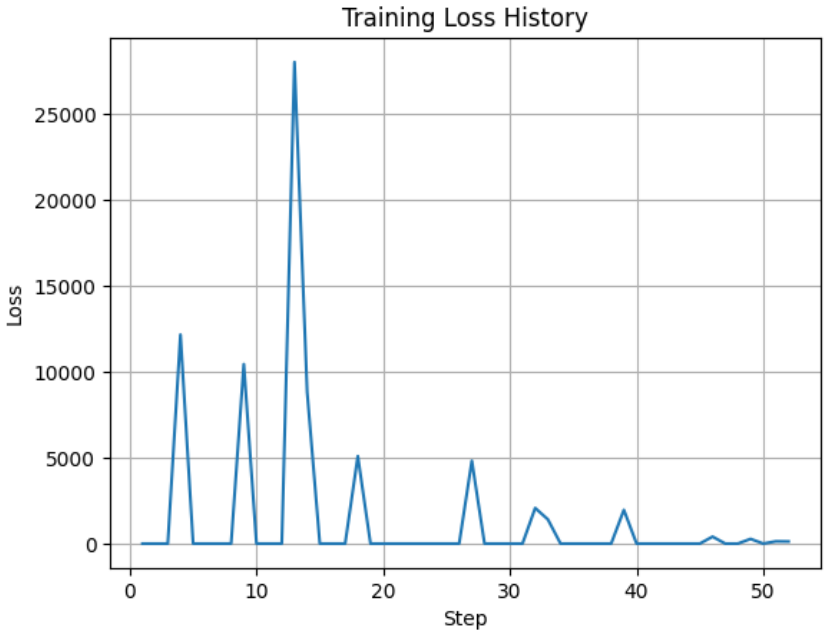

Figure 4: Training Loss Curve during Reinforcement Learning (GRPO).

eventual stabilization demonstrates that the GRPO method was effective in guiding the model toward constraint-compliant SVG outputs.

4.4. Summary of Model Behavior After Training

- After SFT, the model could reliably generate SVG code from prompts but occasionally violated competition constraints (e.g., using disallowed tags).
- After GRPO RL, the model's outputs showed:
  - Significant improvements in constraint satisfaction rates (near 100% compliance in internal

evaluations).
- Improved structural correctness and semantic fidelity of generated SVGs.

Thus, the two-stage training strategy (SFT → GRPO) proved critical for achieving the performance necessary for competitive participation in the Drawing with LLMs Kaggle challenge.

5. Discussion

The two-stage training framework combining supervised fine-tuning (SFT) and reinforcement learning (RL) with Group Relative Policy Optimization (GRPO) proved to be an effective strategy for enabling the Qwen2.5 model to generate competition-compliant SVG code from natural language descriptions.

The supervised fine-tuning phase was crucial for teaching the model the foundational mapping between prompts, reasoning traces, and valid SVG outputs. The steady decrease in loss during SFT (Figure 3) suggests that the model effectively learned to replicate the examples in the Deepseek-SVG dataset. However, SFT alone was insufficient to guarantee compliance with the strict competition constraints, such as file size limits, allowed SVG elements, and prohibited raster content.

Reinforcement learning with GRPO addressed this gap by explicitly rewarding the model for producing outputs that satisfied all constraints. Although the RL training curve (Figure 4) showed significant volatility initially—typical for reinforcement learning scenarios—the eventual stabilization demonstrated the model's ability to adapt under constraint-driven optimization. Nevertheless, RL introduced challenges such as exploration-induced instability and occasional reward hacking attempts, where the model initially found degenerate ways to maximize rewards without improving SVG quality.

In conclusion, the two-stage training process effectively prepared the model for competitive deployment. The project highlights the importance of reinforcement learning when strict structural requirements must be met and demonstrates the synergy between supervised pretraining and reward-driven refinement.

6. Related Work

The generation of vector graphics from natural language descriptions has garnered increasing attention in recent years, building upon advancements in both natural language processing and generative modeling. Several approaches have been explored, ranging from rule-based systems to deep learning models.

Early work in text-to-SVG generation often relied on rule-based systems or template-based methods, where predefined patterns were mapped to specific linguistic structures. For instance, [5] demonstrated a system capable of generating simple geometric shapes based on limited natural language input. However, these methods typically lack the flexibility and generalizability to handle complex or novel descriptions.

With the rise of deep learning, more sophisticated generative models have been applied to this task. Recurrent Neural Networks (RNNs) and their variants, such as LSTMs and GRUs, have been used to directly generate SVG code as a sequence of tokens [6]. These models learn the syntax and structure of SVG by training on datasets of SVG code and corresponding descriptions. More recently, Transformer-based architectures have shown significant promise in various sequence generation tasks, including code generation. Models like [7] have demonstrated the ability to generate coherent and syntactically correct code in different programming languages. The application of such models to SVG generation, as explored in this project with the Qwen 2.5 model, represents a natural progression in leveraging the capabilities of large language models for creative tasks.

The use of reinforcement learning to guide the output of generative models has also been explored in various contexts, including text generation. Techniques like policy gradient methods have been used to optimize models based on non-differentiable reward signals, such as those related to style, fluency, or adherence to specific constraints [8]. In the domain of image generation, reinforcement

learning has been used to improve the quality and coherence of generated images [9]. The application of Group Relative Policy Optimization (GRPO) in this project builds upon this line of work, specifically focusing on enforcing the hard constraints imposed by the Kaggle competition on the generated SVG code. GRPO, as described in [2], offers a method for effectively optimizing policies by comparing their performance relative to a group of other policies, which can be particularly useful in scenarios with complex reward landscapes, such as those involving multiple constraints.

This project distinguishes itself from prior work by combining the strong reasoning capabilities of the Qwen 2.5 model, a state-of-the-art large language model, with a targeted reinforcement learning approach using GRPO to ensure strict adherence to the competition's constraints. While previous work has explored either direct generation of SVG using sequence models or the use of reinforcement learning for guiding generative models, this project specifically focuses on a two-stage approach that leverages the strengths of both supervised learning on a relevant dataset and reinforcement learning for fine-grained control over the output format and content, particularly in the context of a real-world competition with specific requirements.

7. Declaration on Generative AI

During the course of this research, the Qwen 2.5 and Gemma language models were utilized to perform evaluation experiments through the HuggingFace Transformers library. These tools were applied strictly within the context of experimental assessment as described in the methodology. No generative AI systems were employed for the writing or editing of this manuscript. The author assumes full responsibility for the accuracy and integrity of the publication's content.

References

[1] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Dang, et al., Qwen2. 5-coder technical report, arXiv preprint arXiv:2409.12186 (2024).

[2] Q. Z. R. X. J. S. M. Z. Y. L. Y. W. D. G. Zhihong Shao, Peiyi Wang, Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL: https://arxiv.org/abs/2402.03300.

[3] L. von Werra, Y. Belkada, L. Tunstall, E. Beeching, T. Thrush, N. Lambert, S. Huang, K. Rasul, Q. Gallouédec, Trl: Transformer reinforcement learning, https://github.com/huggingface/trl, 2020.

[4] J. A. Rodriguez, A. Puri, S. Agarwal, I. H. Laradji, P. Rodriguez, S. Rajeswar, D. Vazquez, C. Pal, M. Pedersoli, StarVector: Generating Scalable Vector Graphics Code from Images and Text, arXiv preprint arXiv:2312.11556 (2023).

[5] Q. T. Liu, H. Huang, L. J. Wu, Using restricted natural language for geometric construction, Applied Mechanics and Materials 145 (2012) 465–469.

[6] X. Xing, Q. Yu, C. Wang, H. Zhou, J. Zhang, D. Xu, Svgdreamer++: Advancing editability and diversity in text-guided svg generation, IEEE Transactions on Pattern Analysis and Machine Intelligence (2025).

[7] Z. Sun, Q. Zhu, Y. Xiong, Y. Sun, L. Mou, L. Zhang, Treegen: A tree-based transformer architecture for code generation, in: Proceedings of the AAAI conference on artificial intelligence, volume 34, 2020, pp. 8984–8991.

[8] S. Kumar, E. Malmi, A. Severyn, Y. Tsvetkov, Controlled text generation as continuous optimization with multiple constraints, Advances in Neural Information Processing Systems 34 (2021) 14542–14554.

[9] O. Oertell, J. D. Chang, Y. Zhang, K. Brantley, W. Sun, Rl for consistency models: Faster reward guided text-to-image generation, arXiv preprint arXiv:2404.03673 (2024).