Technical Report on the ”Drawing with LLMs” Kaggle Competition System Design

Daniel Alonso Chavarro Chapatecua Carlos Andrés Brito Guerrero Jose Fernando Ramirez Ortiz

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Technical Report on the ”Drawing with LLMs” Kaggle Competition System Design

Abstract—**This technical report presents a comprehensive system design for the “Drawing with LLMs” Kaggle competition, which focuses on using Large Language Models to generate SVG images from text descriptions. The report details our modular architecture designed to handle high-sensitivity components and incorporate continuous feedback loops for improvement. We analyze the system’s functional and non-functional requirements, propose technical implementations using open-source models, and provide strategies to address the identified sensitivity and chaos factors. Our design emphasizes modularity, performance optimization, and cost-effective implementation using Python-based frameworks and free-tier models like DeepSeek, Gemma, and Qwen. The system processes text descriptions (limited to 200 characters) through a pipeline that includes preprocessing, prompt engineering, SVG generation, and evaluation. Our approach aims to balance performance constraints while delivering high-quality SVG outputs that accurately represent textual descriptions within the competition’s technical limitations.**

I. Introduction

The “Drawing with LLMs” Kaggle competition presents a unique challenge at the intersection of natural language processing and computer graphics. The task requires designing a system capable of interpreting textual descriptions and generating corresponding images in SVG format. This technical report builds upon our findings from Workshop #1, where we analyzed the competition requirements and identified key system components and constraints.

The core challenge lies in creating a system that can effectively translate natural language into visual representations while adhering to strict technical limitations. The competition restricts input descriptions to 200 characters and requires outputs as SVG code under 10,000 bytes, using only permitted SVG elements and attributes without CSS styling, rasterized image data, or external fonts.

Our approach focuses on developing a modular architecture that efficiently processes text descriptions, engineers appropriate prompts for the Large Language Model (LLM), generates SVG code, and incorporates feedback for continuous improvement. The system aims to optimize the SVG Image Fidelity Score, which measures how well the generated images match their textual descriptions.

This report details our system design, including architecture, component interactions, strategies for addressing sensitivity and chaos factors, and a comprehensive implementation plan using open-source technologies. We emphasize cost-effective solutions that operate within Kaggle’s notebook constraints while delivering high-quality visual outputs.

II. Problem Definition

The “Drawing with LLMs” Kaggle competition presents the following challenges:

1) **Input Processing**: The system must interpret brief text descriptions (up to 200 characters) that describe various types of images, including landscapes, abstract art, and fashion items.
2) **SVG Generation**: The system must generate valid SVG code that accurately represents the described image while adhering to strict technical constraints:
   - Maximum size of 10,000 bytes
   - Use of only permitted SVG elements and attributes
   - No CSS styling, rasterized image data, or external fonts
3) **Image Quality**: The generated SVGs must achieve high scores on the SVG Image Fidelity Score, which evaluates how well the images match their text descriptions. The system must also avoid generating text in the images, as this incurs penalties in the evaluation.
4) **System Sensitivity**: Our Workshop #1 analysis identified high sensitivity in both the Description Preprocessing component and the LLM Model, requiring careful handling to ensure consistent results.
5) **Chaos Management**: Initial randomness in LLM outputs needs to be managed and gradually stabilized through continuous learning and feedback loops.
6) **Resource Constraints**: The system must operate within Kaggle’s notebook constraints and utilize open-source or free-tier technologies due to limited resources.

These challenges require a system design that balances technical constraints with creative output quality, emphasizing efficiency, reliability, and adaptability to feedback.

III. Background and Literature Review

A. SVG Format and Capabilities

Scalable Vector Graphics (SVG) is an XML-based vector image format for two-dimensional graphics that supports interactivity and animation. Unlike raster formats, SVG images remain crisp at any resolution, making them ideal for responsive design. The format supports basic shapes, paths, text, and various visual effects.

According to the W3C SVG 1.1 Specification, SVG provides a rich set of features including:

- Basic shapes (rectangles, circles, ellipses, lines, polylines, polygons)
- Path elements for complex shapes

- Text elements
- Transformations (translate, rotate, scale)
- Gradients and patterns
- Clipping and masking

For this competition, we are limited to a subset of these features, excluding CSS styling, external fonts, and rasterized image data.

B. *LLMs for Image Generation*

Recent advancements in Large Language Models have demonstrated surprising capabilities in generating structured outputs like code, including SVG. Unlike traditional image generation models that produce raster outputs, LLMs can generate vector graphics through code generation.

Key research in this area includes:

- Parameter-Efficient Fine-Tuning (PEFT) techniques, which enable adapting large models to specific tasks with minimal resources
- Few-shot learning approaches that allow models to generate complex outputs with limited examples
- Prompt engineering strategies that structure input to guide the model toward specific output formats

These approaches form the foundation of our system design, enabling us to leverage open-source LLMs for the SVG generation task within the competition's constraints.

IV. System Requirements

A. *Functional Requirements*

1) The system must accept text descriptions of images (up to 200 characters).
2) The system must preprocess descriptions into standardized prompts for the LLM.
3) The system must generate syntactically correct and constraint-compliant SVG code.
4) The system must optimize SVG output to score highly on the SVG Image Fidelity Score.
5) The system must ensure SVG outputs are under 10,000 bytes.
6) The system must support the generation of images across multiple categories (landscapes, abstract, fashion, etc.).
7) The system must generate structured reports for each image generation task based on the Kaggle's evaluation.

B. *Non-Functional Requirements*

1) **Performance**: Generate SVG responses within a reasonable timeframe (under 30 seconds per prompt).
2) **Reliability**: Maintain consistent quality across different description categories.
3) **Maintainability**: Design a modular architecture that allows for component replacement or enhancement.
4) **Adaptability**: Support the incorporation of feedback to improve generation quality over time.
5) **Efficiency**: Optimize memory usage during SVG generation to operate within Kaggle's notebook constraints.
6) **Cost Management**: Due to limited resources, the system must utilize cheap or open-source technologies.

V. System Architecture

A. *Architectural Diagram*

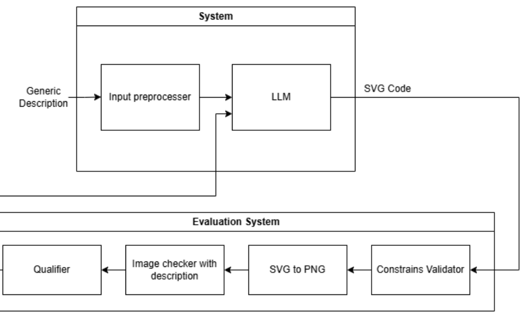

Fig. 1. High-Level Architecture of the Drawing with LLMs System

B. *Component Descriptions*

1) Input Handler: The Input Handler serves as the entry point for the system workflow, responsible for:

- Receiving and parsing input text descriptions
- Performing initial validation (e.g., checking length constraints)
- Standardizing inputs (e.g., removing special characters, normalizing whitespace)
- Categorizing the description type (landscape, abstract, fashion, etc.)
- Passing validated descriptions to the Prompt Engineer

2) Prompt Engineer: The Prompt Engineer transforms raw descriptions into structured prompts that optimize LLM understanding:

- Adds specific instructions about SVG constraints and requirements
- Applies templating and formatting based on the identified category
- Includes few-shot examples when appropriate
- Specifies output format requirements
- Optimizes prompts based on feedback from previous generations

This component is critical due to its high sensitivity; small changes in prompt structure can significantly impact output quality.

3) LLM Generator: The LLM Generator is the core component that interprets engineered prompts and produces SVG code:

- Loads the selected LLM model (DeepSeek, Gemma, Qwen, etc.)
- Processes the engineered prompt
- Generates initial SVG code
- Applies learned patterns from training data and feedback loops
- Controls generation parameters (temperature, max tokens, etc.)
- Optimizes memory usage during inference

4) Evaluation Feedback Loop:

The Evaluation Feedback Loop collects metrics to improve future generations:

- Collects scoring and performance metrics
- Analyzes patterns in high and low-scoring outputs
- Provides structured feedback to the LLM Generator

C. Systems Engineering Principles Applied

Our architecture applies the following systems engineering principles:

1) **Modularity**: Each component has a specific responsibility, allowing for isolated development, testing, and improvement.
2) **Separation of Concerns**: Clear division between input handling, prompt engineering, generation, validation, and reporting functions.
3) **Feedback Integration**: Explicit feedback loop to support continuous improvement and system learning.
4) **Interface Standardization**: Well-defined interfaces between components for seamless integration and potential component replacement.
5) **Graceful Degradation**: The system can fall back to simpler generation strategies when faced with challenging inputs.

VI. Addressing Sensitivity and Chaos

A. Prompt Engineering Strategy for Sensitivity

1) **Standardized Prompt Templates**: Develop category-specific templates to ensure consistent input formatting for the LLM.
2) **Prompt Versioning**: Implement version control for prompts to track changes and their impact on generation quality.
3) **Instruction Clarity**: Use explicit, detailed instructions within prompts to guide the LLM toward desired outputs.
4) **Constraint Embedding**: Directly incorporate SVG constraints into prompts, including element restrictions and size limitations.

B. LLM Optimization for Sensitivity

1) **Few-Shot Learning**: Include carefully selected examples in prompts to demonstrate desired output patterns.

This is meant to LLM's which are not fine tuned.

C. Chaos Mitigation Strategies

1) **Evaluation Weighting**: Weight feedback based on the reliability of evaluation metrics for every loop.
2) **Error Pattern Recognition**: Identify and systematically address common error patterns in SVG generation.

D. Monitoring and Error Handling

1) **Error Classification System**: Categorize errors (syntax, constraint violation, rendering issues) for targeted improvements.
2) **Graceful Degradation**: Implement fallback strategies for challenging inputs, such as simplifying the requested image.
3) **Exception Handling**: Robust exception handling at each stage of the pipeline to prevent cascading failures.

VII. Technical Stack and Implementation

A. Recommended Technologies

1) Core Technologies:

- **Programming Language**: Python 3.9+
- **LLM Framework**: Transformers (Hugging Face models)
- **SVG Processing**: lxml and svglib
- **Vector Operations**: NumPy

2) LLM Models:

- **Base Models**: DeepSeek, Gemma, Gemmini 2.5 Flash, Qwen models or Free tier tuned models
- **Model Adaptation**: PEFT for efficient fine-tuning when resources permit

3) Development and Testing:

- **Environment**: Jupyter Notebooks
- **Version Control**: Git
- **Testing Framework**: pytest or Kaggle testing package

B. Implementation Plan

1) Phase 1: Core Infrastructure :

- Set up model loading and inference pipeline
- Implement basic prompt engineering templates
- Develop input handling components and pre-trained LLM model
- Establish basic project structure and version control

2) Phase 2: Optimization and Refinement:

- Fine-tune model on category-specific training data (won't be applied due to the limited amount of time and resources)
- Develop advanced prompt engineering strategies
- Implement the feedback collection mechanism

3) Phase 3: Quality Assurance and Scaling:

- Comprehensive testing across image categories
- Performance optimization for Kaggle notebooks
- System integration testing
- Documentation and preparation for submission

C. Design Patterns

- **Factory Pattern**: For generating category-specific prompts based on input classification.
- **Observer Pattern**: For monitoring performance and metrics across the system.
- **Template Method Pattern**: For structuring the overall generation workflow while allowing specific steps to vary.

VIII. Methodology

Our methodology for developing and evaluating the system follows these steps:

1) **Model Selection and Baseline**:
   - Evaluate different open-source LLM models (DeepSeek, Gemma, Qwen) on a sample set of descriptions
   - Select the model with the best baseline performance for further development
2) **Prompt Engineering Experimentation**:
   - Develop a test suite of prompts with varying structures and instructions
   - Measure the impact of different prompt elements on output quality
   - Iterate on prompt design based on empirical results
3) **Performance Measurement**:
   - Track generation time across different description types and complexities
   - Measure memory usage during inference
   - Identify performance bottlenecks and optimization opportunities
4) **Feedback Loop Implementation**:
   - Develop automated evaluation metrics aligned with competition scoring
   - Test the effectiveness of feedback integration on subsequent generations
   - Measure improvement rates over iterations

All experiments are conducted in Kaggle notebooks to ensure consistency with the competition environment. We use version control to track changes and maintain documentation of experiment results.

IX. Results and Analysis

As this is a design document, we have not yet implemented the full system.

X. Conclusions and Recommendations

Based on our system design and preliminary findings, we draw the following conclusions:

1) A modular architecture with clear separation of concerns provides the flexibility needed to address the challenges of SVG generation from text descriptions.
2) The high sensitivity of prompt engineering and LLM components requires systematic experimentation and versioning to achieve consistent results.
3) Open-source models like DeepSeek and Gemma offer viable options for cost-effective implementation, though with performance trade-offs compared to larger commercial models.
4) Effective feedback loops are essential for continuous improvement, particularly to address the chaos factors identified in our analysis.

We recommend the following approaches for implementation:

1) **Prioritize Prompt Engineering**: Invest significant effort in developing and testing prompt templates, as this component shows the highest sensitivity and impact on output quality.
2) **Start Simple**: Begin with well-defined image categories (basic shapes, simple landscapes) and progressively advance to more complex descriptions as the system stabilizes.
3) **Leverage Few-Shot Learning**: Include carefully selected examples in prompts to guide the model, particularly for complex image types.
4) **Consider Ensemble Approaches**: If resources permit, implement an ensemble of smaller models rather than relying on a single large model, potentially improving both performance and reliability.

This system design provides a comprehensive framework for addressing the "Drawing with LLMs" competition challenge while operating within technical and resource constraints.

XI. References

1) Drawing with LLMs. Kaggle. https://www.kaggle.com/competitions/drawing-with-llms
2) Systems Analysis & Design Workshop #1 (April 13, 2025)
3) Systems Analysis & Design Workshop #2 (May 10, 2025)
4) Hugging Face Transformers Documentation. https://huggingface.co/docs/transformers/
5) SVG 1.1 Specification. https://www.w3.org/TR/SVG11/
6) Parameter-Efficient Fine-Tuning (PEFT). https://huggingface.co/docs/peft/

Appendix

The competition restricts SVG elements to a specific subset of the SVG specification. Permitted elements include:

- Basic shapes: `<rect>`, `<circle>`, `<ellipse>`, `<line>`, `<polyline>`, `<polygon>`
- Path element: `<path>`
- Group element: `<g>`
- SVG root element: `<svg>`

Permitted attributes include:

- Geometric attributes: `x`, `y`, `width`, `height`, `cx`, `cy`, `r`, `rx`, `ry`, `points`, `d`
- Styling attributes: `fill`, `stroke`, `stroke-width`, `opacity`, `fill-opacity`, `stroke-opacity`
- Transformation attributes: `transform`