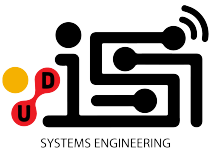
FRANCISCO JOSÉ DE CALDAS DISTRICT
UNIVERSITY
FACULTY OF ENGINEERING

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Workshop No. 1 Kagle Systems Analysis of Drawing with LLMs competition



Edward Julian Garcia Gaitan
Jaider Camilo Carvajal Marin
Nelson David Posso Suárez

Systems Engineering

April 14, 2025

Abstract

In this document, an analysis of the systems from the Kaggle competition 'Drawing with LLMs' will be conducted. In this competition, a prompt is given to generate a Scalable Vector Graphics (SVG) file. The objective of the competition is to create a reusable Kaggle Package, however, this is not the focus of this document.

1 Competition Overview

"This competition challenges you to build practical, reusable solutions for image generation that follow robust software engineering patterns. Given a text description of an image, your task is to generate SVG code which renders it as closely as possible. Scalable Vector Graphics (SVG) is a vector image format that uses XML to describe two-dimensional graphics which can be scaled in size without quality loss

The submission, created using the new Kaggle Packages feature, will be a with a function which generates SVG code for a given prompt. The end results will be a set of deployable model packages evaluated on their ability to deeply reason about abstract descriptions and translate them into precise, executable code.class Modelpredict()" [1].

The challenge is to design a model capable of receiving a natural language description (prompt) as input and generating valid SVG code. This code, when rendered, must produce an image that faithfully corresponds to the provided description. In other words, the model is expected to interpret the text and translate it into graphical instructions in SVG format, challenging the ability of language models to generate high-quality, visually coherent content.

1.1 Dataset Structure

The dataset consists of examples that associate:

1. Generation and validation of SVGs: The `svg_constraints.py` file defines constraints for validating SVG documents, ensuring they meet certain size limits, allowable elements, and valid attributes.

2. Gateway for evaluation: `svg_gateway.py` implements a gateway that interacts with participant models. This gateway:

   – Reads input data (text descriptions).
   – Sends the descriptions to the participant model to generate SVG images.
   – Validates the generated SVG images.
   – Writes the results to a submission file (`submission.csv`).

3. Communication infrastructure: It uses gRPC and Protobuf for communication between the gateway and the participant's

model inference server. Protobuf-generated files, such as `kaggle_evaluation_pb2.py` and `kaggle_evaluation_pb2_grpc.py`, define the messages and services for this communication.

4. Local tests: The `svg.py` file includes a `test` function that allows participants to test their models locally before submitting them to the competition. This ensures that their models meet the input and output requirements.

5. Input data: Files like `train.csv` and `test.csv` contain text descriptions that models should use to generate SVG images.

1.1.1 General flow

1. Participants implement a model with a `predict` function that takes text descriptions and returns SVG images.

2. The gateway sends the descriptions to the model, validates the generated images, and creates a submission file.

3. During the competition, Kaggle uses this system to automatically evaluate the models submitted by participants.

The test set includes approximately 500 descriptions, allowing us to evaluate the model's ability to generalize and generate consistent images based on varied descriptions. The training set, meanwhile, provides data pairs (text and SVG) so participants can train and fine-tune their solutions.

2 System Analysis Report

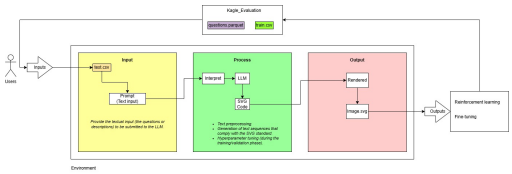


Figure 1: Diagram of the system

This diagram represents the system, its elements, and the relationships between them. Further details regarding the above will be provided below. The creation was assisted by the article: How to Build a Large System Diagram [2].

2.1 Elements

- **Input (Text Descriptions):** The system begins with natural language prompts that describe everyday objects or scenes. These textual inputs serve as the starting point for the image generation process.

- **Processing Model (LLMs & SVG Generator):** Participants are challenged to design models—often based on large language models (LLMs)—that translate these text prompts into SVG code. The model must integrate natural language processing with a graphics-generation component, ensuring that the resulting code accurately represents the described image.

- **Output (SVG Code):** The final output is Scalable Vector Graphics (SVG) code that, when rendered, should visually match the initial description. This output must adhere to strict formatting and content constraints.

- **Dataset:** The training dataset provides paired examples (text prompt and corresponding SVG code), while the test dataset (approximately 500 descriptions) is used for evaluation. This structured pairing establishes clear relationships between the descriptive language and its visual representation.

- **Evaluation Metrics and Feedback:** The competition utilizes evaluation metrics to assess how well the generated SVG images correspond to the given prompts. The scoring may involve comparing rendered images or analyzing the SVG structure, with feedback loops intended for iterative model improvement.

2.1.1 Relationships

- **Input-to-Output Pipeline:** The model translates textual input into a visual output by mapping natural language constructs to vector graphic elements.

- **Data Dependency:** The effectiveness of the model is directly related to the quality and representativeness of the training dataset, which governs the relationship between text and SVG code.

- **Constraints and Validation:** The system enforces strict SVG format rules, ensuring that all output adheres to predefined specifications. This adds an additional layer where the generated code is validated against both aesthetic and technical criteria.

2.2 Complexity and sensibity

In the competition, a series of restrictions or parameters are given to be considered for the implementation of the system; although these restrictions are part of the evaluation, they require the system to have a specific behavior to meet these requirements, thus interfering with it. Among these, the following stand out:

1. No SVG may more than 10,000 bytes long.
2. Each SVG may only include elements and attributes from an allowlist. Note that CSS style elements are not allowed.
3. No SVG may include any rasterized image data or data from external sources.

2.2.1 Potential Limitations

- Ambiguity in Text Descriptions: Natural language can be inherently ambiguous. Slight variations in phrasing may lead to significant differences in generated outputs, challenging the model's consistency.
- SVG Format Constraints: Strict restrictions on allowed SVG elements (prohibiting direct text rendering) require the model to use creative workarounds. This can limit the solution space and introduce potential conflicts between natural expression and technical feasibility.
- Model Generalization: Given the limited size of the test set, the system may be sensitive to overfitting, where the model performs well on training data but struggles with novel or highly variable prompts.

2.2.2 Variability Factors

Training Data Quality: The diversity and quality of the training examples will heavily influence the system's robustness, with potential biases or gaps affecting performance.

2.3 Chaos and randomness

In the competition, a series of restrictions or parameters are given to be considered for the implementation of the system; although these restrictions are part of the evaluation, they require the system to have a specific behavior to meet these requirements, thus interfering with it. Among these, the following stand out:

1. Variability in the LLM Output: Large Language Models (LLMs) such as Gemma or similar are stochastic (probabilistic) models. This means that even when provided with the same prompt, they can produce different outputs due to the ambiguity of natural language—where the model may interpret the same prompt in multiple ways. For example, two executions of the model using the prompt "a red triangle inside a blue circle" might result in SVGs with different sizes, positions, or shapes of the triangle. This introduces nondeterminism, which makes the system's behavior difficult to replicate exactly.
2. Sensitivity to Small Changes in the Prompt: A minor change in the wording of the text prompt can lead to a completely different SVG output. This reflects a "butterfly effect" often seen in chaotic systems, where small perturbations can produce large-scale changes. For instance, changing "a big red triangle" to "a large red triangle" might cause the model to generate a smaller triangle or one with a different orientation.
3. Complexity in the Interactions Between Elements: The system consists of multiple interconnected layers: text input, generation, visual representation, semantic evaluation, text-based penalties, and aesthetics. The interactions between all these components can lead to emergent behaviors that are difficult to predict or control.

3 Conclusion

The systems analysis of the Drawing with LLMs challenge helped reveal the true complexity behind a problem that initially seems simple. Generating SVG images from text involves multiple interconnected elements, such as the model, the data, the

execution environment, and the evaluation metrics.

Throughout the analysis, the system showed high sensitivity: small changes in the prompt can lead to completely different outputs. This variability reflects chaotic behavior, making the process less predictable and harder to reproduce.

It also became clear that performance depends not only on the model itself but on how all system components interact. This holistic perspective is essential for anticipating errors, proposing improvements, and designing more robust and realistic solutions.

References

[1] Kaggle. Drawing with LLMs Competition Overview, 2025. Accesed: 04/4/2025.

[2] E. Pettis. How to build a large system diagram, 2020. Accessed: 04/3/2025.