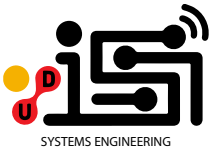
UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS



FRANCISCO JOSÉ DE CALDAS DISTRICT UNIVERSITY

FACULTY OF ENGINEERING

SYSTEMS ENGINEERING PROGRAM

Workshop No. 2

Kaggle Systems Analysis of Drawing with LLMs Competition

Edward Julian García Gaitán	Jaider Camilo Carvajal Marin	Nelson David Posso Suárez
20212020136	20212020120	20212020132
ejgarciag@udistrital.edu.co	jccarvajalm@udistrital.edu.co	ndpossos@udistrital.edu.co

Professor:
Eng. Carlos Andrés Sierra, M.Sc.

Course:
Systems Analysis & Design

Bogotá, D.C.

May 10, 2025

Abstract

This document presents the solution to Workshop No. 2, focusing on the systems analysis of the Kaggle competition "Drawing with LLMs." The analysis includes a summary of Workshop No. 1, system requirements, architecture, strategies to handle sensitivity and chaos, technological stack, implementation plan, and references.

1 Introduction

The "Drawing with LLMs" competition [1], promoted by the Kaggle platform, represents a contemporary challenge at the forefront of artificial intelligence: generating images in Scalable Vector Graphics (SVG) format [4] from textual descriptions (prompts) using Large Language Models (LLMs) [2]. This task lies at the confluence of Natural Language Understanding (NLU), automatic code generation, and computer graphics synthesis, inherently introducing notable systemic complexity.

Workshop No. 1 of this course was dedicated to a detailed analysis of this competition from a systems engineering perspective. This analysis unveiled the system's structure, identifying as crucial elements: input prompts (with their inherent semantic and syntactic variability), LLMs (as the core of intermodal translation), the resulting SVG code (with its strict formal grammar), datasets (for training and evaluation), and evaluation metrics (which define success and may include comparisons like SSIM [8]). The interactions between these elements proved to be highly sensitive to factors such as prompt quality and specificity, LLM choice and configuration (e.g., temperature, top-p), and the robustness of SVG parsers and renderers. Crucially, manifestations of chaos theory [6] were anticipated, such as sensitive dependence on initial conditions (the butterfly effect in prompt formulation) and the emergence of non-linear and unpredictable behaviors in generation, especially with ambiguous or novel prompts. These findings underscore the need for a design that is not only functional but also resilient and adaptable, as postulated by Meadows in her work on systems thinking [5].

This System Design Document (SDD) primarily aims to propose a software architecture and implementation methodology that directly and proactively address the identified challenges and requirements. A design is sought that satisfies the following macro-objectives, derived from the guidelines of Workshop No. 2 and best practices in software engineering:

- **Functional Effectiveness:** The system must be capable of generating SVG representations that are semantically coherent with the input prompt and syntactically valid according to the SVG format specification.
- **Robustness and Reliability:** The system must exhibit predictable behavior with valid inputs, be tolerant to failures in individual components, and appropriately handle anomalous or malformed inputs.
- **Operational Efficiency:** The system must meet the performance constraints imposed by the competition, particularly in terms of generation latency and processing capacity.
- **Scalability and Maintainability:** The architecture must facilitate adaptation to increasing volumes of data or requests and allow for system evolution (e.g., incorporating new LLMs, adjusting modules) with reasonable maintenance effort.

This design is articulated on fundamental principles of systems engineering, such as modularity, cohesion, low coupling, and separation of concerns, seeking a system that is both innovative in its generative capacity and disciplined in its engineering construction.

2 System Design Development

This section details the proposed design for the SVG generation system, starting with the consolidation of requirements derived from the Workshop No. 1 analysis and the competition specifications, followed by the presentation of the system architecture, strategies to address sensitivity and chaos, and the selected technology stack.

2.1 System Requirements

System requirements have been classified into functional (describing what the system must do) and non-functional (describing how the system must behave or the qualities it must possess). These stem directly from the prior systemic analysis and the guidelines of the "Drawing with LLMs" competition.

2.1.1 Functional Requirements (FR)

- **FR1: Textual Prompt Ingestion and Parsing:** The system shall accept textual descriptions (prompts) provided in the format specified by Kaggle (e.g., CSV, JSON files) as input. It must be able to parse and validate the structure of this input data.
- **FR2: Advanced Semantic Interpretation of Prompt:** The system shall perform a deep semantic interpretation of the prompt's content, identifying entities, attributes, spatial and temporal relationships, artistic styles, and other descriptive intentions relevant to graphic generation.
- **FR3: Generation of Coherent and Valid SVG Code:** The system shall dynamically generate source code in SVG format that visually represents the semantic interpretation of the prompt. The generated SVG must be syntactically correct according to the SVG 1.1 specification [4] or the relevant version.
- **FR4: Flexible Configuration of Generation Parameters:** The system must allow configuration of key parameters influencing the generation process, such as base LLM selection, inference hyperparameters (e.g., temperature, top-p, max tokens), and possible style or complexity constraints.
- **FR5: Results Persistence and Management:** The system shall be capable of storing generated SVGs, along with relevant metadata (source prompt, generation parameters, quality metrics if applicable), in an organized manner for subsequent analysis or submission.

2.1.2 Non-Functional Requirements (NFR)

- **NFR1: Performance (Generation Latency):** The system shall generate an SVG graphic from an average textual description in a time not exceeding 1 second, under the hardware conditions stipulated or inferred from the Kaggle platform. *(This is an ambitious target, and its feasibility will depend on the LLM and SVG complexity)*.

- **NFR2: Reliability and Accuracy:** The system must ensure that 100% of generated SVG files are syntactically valid and renderable by standard SVG viewers. Semantic coherence and visual fidelity with the prompt must be maximized, aiming for high performance on the official competition evaluation metric (e.g., SSIM [8] if compared against a reference image, or intrinsic SVG quality metrics).
- **NFR3: Scalability (Processing Capacity):** The system shall be capable of processing a volume of up to 10,000 textual descriptions (prompts) in batch or concurrent mode, within the time limits set by Kaggle for mass evaluation phases.
- **NFR4: Usability (for Developers and Experimentation):** The system's architecture and source code must be clear, well-structured, and adequately documented to facilitate understanding, maintenance, extension, and iterative experimentation by the development team. (Note: This refers to code/architecture usability, not end-user UI).
- **NFR5: Maintainability and Modularity:** The design must be eminently modular, allowing for the update, replacement, or addition of individual components (e.g., a new LLM, an improved SVG validation engine, different preprocessing strategies) with minimal and controlled impact on the rest of the system.
- **NFR6: Robustness and Error Handling:** The system shall manage anomalous inputs (e.g., empty, malformed, excessively long, ambiguous prompts) and internal failures (e.g., network errors when contacting an LLM API, memory failures) in a controlled and predictable manner, avoiding abrupt crashes and providing, where possible, informative feedback or detailed logs.

2.2 System Architecture

The proposed architecture for the "Drawing with LLMs" system is based on a modular, layered design with a well-defined data processing flow. This approach promotes separation of concerns, cohesion within modules, and low coupling between them, thereby facilitating maintainability and scalability.

2.2.1 Applied Architectural Design Principles

The conception of this architecture has been guided by the following fundamental principles of systems and software engineering:

- **Modularity and Componentization:** The system is decomposed into logically distinct and autonomous functional modules, each encapsulating a specific responsibility. This allows for parallel development, isolated testing, and maintenance of each component.
- **Separation of Concerns (SoC):** Distinct functionalities reside in different modules (e.g., LLM interaction logic is separate from SVG validation logic).
- **Interface Abstraction:** Modules interact with each other through well-defined and stable interfaces, hiding their internal implementation details. This reduces direct dependencies and facilitates the substitution of implementations.
- **High Cohesion:** Functionalities grouped within the same module are strongly related and contribute to a common purpose.

- **Low Coupling:** Dependencies between modules are minimized. When they exist, they are managed through the defined interfaces.
- **Dataflow-Oriented Design:** The architecture follows a pipeline pattern where data (prompts, intermediate representations, SVGs) flows through a sequence of processing stages.
- **Testability:** Each module is designed with testability in mind, allowing for the creation of unit tests to verify its isolated behavior, as well as integration tests to validate inter-module interactions.

2.2.2 High-Level Architecture Diagram

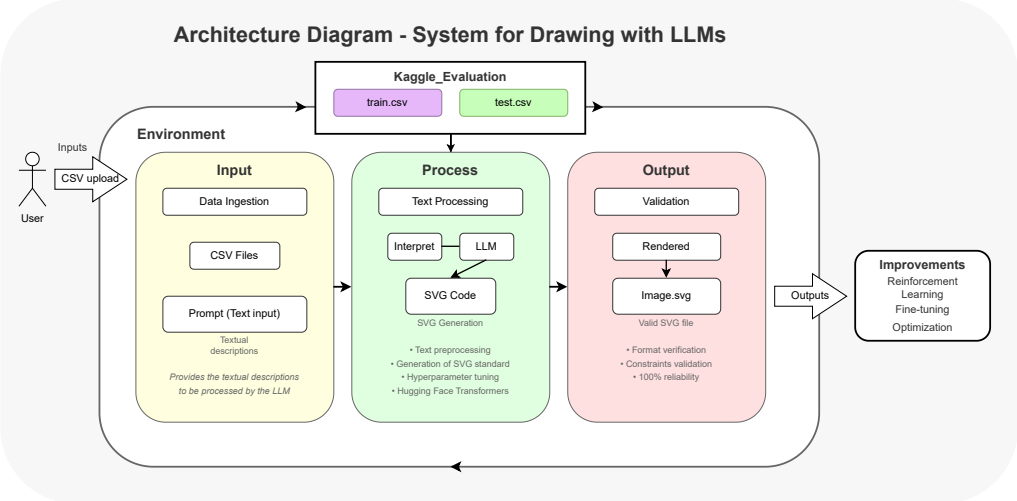


Figure 1: Conceptual Architecture Diagram for the "Drawing with LLMs" System.

2.3 Strategies for Sensitivity Management and Chaos Mitigation

The Workshop No. 1 analysis highlighted the system's high sensitivity to prompt formulation and the inherently unpredictable (chaotic, in the sense of Gleick [6]) nature of LLM outputs. A robust design must explicitly incorporate strategies to manage this variability and mitigate its undesirable effects. These strategies are inspired by systems thinking principles [5] for designing resilient systems.

- **Robust and Adaptive Prompt Engineering (in M2):**
  - **Normalization and Standardization:** Implement exhaustive cleaning and normalization routines for input prompts to reduce spurious variability that does not contribute useful semantic information.
  - **Prompt Templating:** Utilize structured templates that guide the LLM towards the desired output format (SVG or a controlled intermediate representation). This can include placeholders for entities, attributes, and relationships extracted from the original prompt.

  - **Dynamic Few-Shot Learning Techniques:** If latency permits, for complex or ambiguous prompts, a few-shot prompt could be dynamically constructed by including relevant examples of (similar prompt, good SVG) pairs extracted from a knowledge base or previous successful runs.

- **Control and Stabilization of LLM Generation (in M3):**
  - **Fine-tuning of Inference Parameters:** Systematically experiment with and select optimal values for LLM decoding parameters.
  - **Multiple Sampling and Selection Strategies:** For critical prompts, generate multiple SVG candidates (e.g., by slightly varying the prompt or using different sampling seeds) and then select the best candidate using the Quality Assessment Module (M5) or validation heuristics.
  - **Intelligent Retry Mechanisms:** If the LLM fails to generate a valid or coherent SVG, implement retry logic that might, for example, simplify the prompt, ask the LLM to "reflect" on its error and correct it, or use a backup LLM.
  - **Stop Sequences and Length Limits:** Define clear stop sequences and limits on LLM output length to prevent it from generating excessively long or incomplete code.

- **Rigorous Validation and Defensive Postprocessing (in M4):**
  - **Multi-level Validation:** Not only validate the SVG's XML syntax but also check for conformance with specific SVG profiles (e.g., SVG Tiny if compatibility with limited devices is sought), attribute validity (e.g., colors, units), and overall structure (e.g., presence of an ‘<svg>‘ tag).
  - **Heuristic Correction and Sanitization Rules:** Implement a set of rules to identify and automatically correct common error patterns observed in LLM output (e.g., badly closed tags, common missing attributes, out-of-range values). Also, to "sanitize" the SVG, removing empty elements or potentially unsafe scripts (although the LLM is not expected to generate these for this task).
  - **Complexity Limits (Circuit Breakers):** Establish thresholds for the complexity of the generated SVG (e.g., maximum number of elements, maximum DOM tree depth, file size). If an SVG exceeds these limits, it might be rejected or simplified.

- **Continuous Monitoring, Detailed Logging, and Error Analysis (in M0 and specific modules):**
  - **Structured Logging:** Implement detailed logging at each stage of the pipeline, recording the input prompt, generation parameters, raw LLM response, final SVG, processing times, and any errors or warnings. Use structured formats (e.g., JSON) to facilitate automated log analysis.
  - **Process and Quality Metrics:** Monitor key metrics such as the success rate of generating valid SVGs, average latency, and quality scores (if M5 is active).
  - **Root Cause Analysis:** Establish a process to analyze systematic failures or low-quality outputs to identify whether they are due to issues in prompts, LLM configuration, postprocessing, or unexpected interactions between modules. This is crucial for iterative system improvement.

- **Controlled Feedback Loops (in the development process):**
  - **Human-in-the-Loop Evaluation:** During development and experimentation, incorporate human evaluation of generated SVGs to refine prompt engineering strategies, adjust LLM parameters, and improve postprocessing rules.
  - **LLM Fine-tuning (Long-term Consideration):** If a sufficiently large and high-quality dataset of (prompt, excellent_SVG) pairs can be obtained, consider fine-tuning a pre-trained LLM to specialize it for the SVG generation task. This can reduce sensitivity and improve coherence.

The implementation of these strategies seeks to transform a potentially erratic system into one that, although inherently complex, operates within more predictable and reliable behavioral channels, thereby increasing the chances of success in the competition.

2.4 Technology Stack and Implementation Plan

2.4.1 Recommended Technology Stack

The selection of the technology stack is based on tool maturity, availability of relevant libraries, community familiarity, and suitability for system requirements.

2.4.2 Phased Implementation Plan

An iterative and incremental development approach is proposed, possibly following agile principles like Scrum, divided into phases or sprints with clear objectives.

Phase 1: Initialization and Core Prototyping (Weeks 1-2):

- *Activities:* Setup development environment (Git repository, Python virtual environment, base dependencies). Develop a minimal "end-to-end" prototype (very simple M1 Module, M3 with an accessible LLM via API or a small local one, M4 with basic validation, M6 to save).
- *Objectives:* Validate technical feasibility of the main flow. Establish basic project structure.
- *Key Deliverable:* Functional prototype that takes a prompt and generates a simple SVG.

Phase 2: Development of Central Modules (Weeks 3-5):

- *Activities:* Robust implementation of M1 (Ingestion), M2 (Basic Preprocessing), M3 (LLM Core with advanced configuration), and M4 (SVG Postprocessing and Rigorous Validation). Initial integration of these modules. Development of unit tests for each.
- *Objectives:* Have the main processing chain functional and tested. Capacity to generate valid SVGs for a diverse set of prompts.
- *Key Deliverable:* Alpha version of the system with integrated central modules.

Phase 3: Implementation of Robustness and Quality Strategies (Weeks 6-8):

Table 1: Proposed Technology Stack for the "Drawing with LLMs" System.

Category	**Technology/Tool**	**Justification**
Main Language	Python 3.9+	Mature ecosystem for AI/ML/NLP [7]; vast libraries.
NLP / LLMs	Hugging Face Transformers [2]	Access to SOTA LLMs; tools for inference and fine-tuning.
	Sentence Transformers	For embeddings if used in preprocessing/search.
SVG Gen./Manip.	'lxml' [9]	Efficient XML/SVG parsing, validation, and manipulation.
	'svgwrite' [3]	Programmatic and simple creation of SVG files.
	'cairosvg' / 'rsvg-convert'	SVG to raster rendering for preview/evaluation.
Data Handling	Pandas	Reading and manipulating tabular data (e.g., CSV of prompts).
Version Control	Git / GitHub	Standard for collaborative development and code versioning.
Dev. Environment	VS Code / PyCharm + JupyterLab	Powerful IDEs; Jupyter for interactive experimentation.
Data Validation	Pydantic	Declaration and validation of data models (e.g., for prompts).
Testing	PyTest	Robust and flexible testing framework for Python.

- *Activities:* Refinement of M2 with advanced prompt engineering techniques. Implementation of sensitivity handling and chaos mitigation strategies in M3 and M4 (LLM control, defensive validation). Development of M5 (Quality Assessment) for internal metrics. Implementation of structured logging and basic monitoring.
- *Objectives:* Significantly improve the quality, reliability, and predictability of generated SVGs.
- *Key Deliverable:* Beta version of the system with enhanced robustness capabilities.

Phase 4: System Optimization and Documentation (Weeks 9-10):

- *Activities:* Performance optimization of critical modules (especially M3 and M4). Scalability testing simulating competition load. Thorough documentation of code and system.

- *Objectives:* Functionally complete system, and optimized for performance.
- *Key Deliverable:* System ready for final testing.

Phase 5: Final Testing, Refinement, and Packaging (Weeks 11-12):

- *Activities:* Execution of a comprehensive test plan (functional, performance, robustness) using validation datasets (if provided by Kaggle or created internally). Bug fixing and final refinements based on test results. Preparation of code and artifacts for Kaggle submission according to their guidelines.
- *Objectives:* Ensure maximum system quality and compliance. Submission package ready.
- *Key Deliverable:* Final solution packaged for Kaggle. Finalized design document.

This plan is a guide; the exact duration and content of each phase may be adjusted based on progress and discoveries made during development.

3 Conclusion and Future Work

The system design presented in this document offers a structured and methodologically grounded solution for the complex challenge of the "Drawing with LLMs" competition. Drawing upon the findings of the Workshop No. 1 systemic analysis, this design not only addresses the functional requirements of SVG generation from text but also proactively integrates strategies to manage the inherent sensitivity and chaotic behaviors associated with Large Language Models.

The proposed modular architecture, underpinned by software engineering principles such as high cohesion and low coupling, is designed to be robust, maintainable, and scalable. The modules, from prompt ingestion and preprocessing to LLM generation, SVG validation, and persistence, are clearly defined in terms of responsibilities and interfaces. The delineated strategies for sensitivity management – including advanced prompt engineering, LLM inference control, and defensive output validation – along with a plan for logging and monitoring, are crucial for enhancing system reliability and predictability. The selected technology stack, centered on Python and the Hugging Face Transformers ecosystem, provides a powerful and flexible environment for implementation.

The phased and iterative implementation plan will allow for controlled development, with verification points and the ability to adapt the design as greater practical understanding of the system's behavior is gained. While comprehensiveness has been sought, it is inherent in projects of this nature that unforeseen challenges and opportunities for improvement will arise during the construction phase.

As **future work** or potential extensions, the following could be considered:

- **Fine-tuning of Specific LLMs:** If a high-quality dataset of (prompt, optimal_SVG) pairs can be collected, fine-tuning a pre-trained LLM could dramatically improve the quality and specificity of generations.
- **Reinforcement Learning with Human Feedback (RLHF) Mechanisms:** To refine SVG quality, an RLHF loop could be explored where human evaluations of generated SVGs are used to train a reward model and subsequently adjust the LLM.

- **Integration of External Knowledge Bases:** For improved understanding of specific entities or concepts in prompts, the system could consult structured knowledge bases (e.g., Wikidata, DBPedia).
- **Advanced SVG Optimization:** Implement more sophisticated algorithms for optimizing the size and complexity of the generated SVG without losing visual fidelity.
- **Exploration of Multimodal LLM Architectures:** Investigate the use of LLMs that have inherent visual understanding and generation capabilities, which might simplify parts of the pipeline.

In conclusion, this system design document provides a detailed roadmap for constructing a competitive and well-engineered solution for Workshop No. 2. The rigorous application of the principles outlined herein will be fundamental to transforming the concept into a functional and successful system.

References

[1] Kaggle. (n.d.). *Drawing with LLMs*. Retrieved from https://www.kaggle.com/competitions/drawing-with-llms/overview

[2] Hugging Face. (n.d.). *Transformers Documentation*. Retrieved from https://huggingface.co/docs/transformers/

[3] The svgwrite Core Developers. (n.d.). *svgwrite Documentation*. Retrieved from https://svgwrite.readthedocs.io/

[4] W3C. (2011). *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. Retrieved from https://www.w3.org/TR/SVG11/

[5] Meadows, D. H. (2008). *Thinking in Systems: A Primer*. Chelsea Green Publishing.

[6] Gleick, J. (1987). *Chaos: Making a New Science*. Penguin Books.

[7] VanderPlas, J. (2016). *Python Data Science Handbook*. O'Reilly Media.

[8] Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4), 600-612.

[9] lxml.de. (n.d.). *lxml - Processing XML and HTML with Python*. Retrieved from https://lxml.de/

[10] Streamlit Inc. (n.d.). *Streamlit Documentation*. Retrieved from https://docs.streamlit.io/