Systemic Analysis and LLM Integration for SVG Generation in a Kaggle Competition

Systematic Development of SVG Generation Framework with Performance Optimization

Nelson David Posso Suarez
ndpossos@udistrital.edu.co

Edward Julian Garcia Gaitan
ejgarciag@udistrital.edu.co

Jaider Camilo Carvajal Marin
jccarvajalm@udistrital.edu.co

Universidad Distrital Francisco Jose de Caldas
Systems Analysis and Design Course
Faculty of Engineering
Systems Engineering Department
Bogotá D. C., Colombia

July 12, 2025

Abstract

This paper presents a systemic analysis and architectural design of an AI-based solution for the Kaggle competition "Drawing with LLMs". The project integrates the principles of systems analysis and design by evaluating the interactions between components, modeling data flow, and incorporating a large language model (LLM) to enhance SVG image generation from natural language prompts. A simulation-based evaluation is performed to assess performance and system behavior under varied conditions.

1 Introduction

The advancement of large language models (LLMs) has opened new avenues for generating visual content from textual descriptions. The Kaggle competition "Drawing with LLMs" presents a unique challenge: converting natural language descriptions into syntactically valid and semantically coherent Scalable Vector Graphics (SVG) code.

This task requires more than just linguistic translation. It involves the design and orchestration of a complete system that integrates natural language processing, SVG code generation, syntactic validation, and performance analysis. From a systems analysis and design perspective, this competition offers a fertile ground for applying principles such as systemic thinking, modularity, emergent behavior evaluation, and uncertainty management.

This technical report documents the development of a system based on LLMs that tackles this challenge using a systemic approach. Over a series of workshops, we defined and refined key components, evaluated behavior under controlled conditions, and made architectural decisions aimed at sustainability, adaptability, and operational efficiency.

Unlike purely algorithmic approaches, this work adopts a holistic view, analyzing both structural elements (inputs, outputs, actors, boundaries) and the internal dynamics of the system (feedback, sensitivity, complexity, stability). The integration of a local language model via Ollama and real-time performance monitoring exemplifies how design principles focused on resilience and systemic transparency were applied.

The report is structured into sections that cover, respectively: the systemic analysis of the competition, the architectural design of the solution, the LLM integration strategy, simulation and evaluation results, and a critical discussion of the system's performance, scalability, and emergent behavior.

2 Systemic Analysis of the Competition

The Kaggle competition "Drawing with LLMs" serves as a rich case study for systemic exploration. From a systems engineering standpoint, the task of generating SVG images from natural language prompts can be interpreted as an open, adaptive system composed of human and computational actors, inputs, outputs, feedback loops, and evolving internal structures. This section applies systemic analysis to deconstruct the challenge and understand its dynamic behavior, functional scope, and emergent complexity.

2.1 System Boundaries and Key Actors

We define the system boundaries as the end-to-end process starting from a textual prompt and concluding with a validated SVG output. Within these boundaries, several actors play critical roles:

- **Prompt Authors:** Users who generate or provide natural language descriptions of desired images.
- **Developers and Model Designers:** Engineers and researchers who design, fine-tune, and submit LLM-based models to the competition.
- **Evaluation Platform (Kaggle):** Acts as an execution environment, supplying test prompts and evaluating the submitted SVGs based on predefined metrics and rules.

The systemic view emphasizes that the system does not function in isolation. Its behavior and success are tightly coupled to the feedback provided by external evaluators and the stochastic outputs of internal generative models.

2.2 Inputs, Outputs, and Internal Processes

The system's primary input is a natural language prompt—typically a sentence that describes a simple visual composition. During training, the system also processes input–output pairs from the provided dataset (`train.csv`). During evaluation, new prompts from `test.csv` are fed without ground-truth SVGs, requiring the system to generalize.

The expected output is a syntactically valid SVG string that accurately and creatively reflects the semantics of the input prompt. This SVG is both machine-parseable and visually interpretable.

Internally, the system transforms prompts into SVGs through a multistage pipeline:

1. **Interpretation:** Text normalization and prompt enrichment.
2. **Generation:** An LLM translates the prompt into SVG format.
3. **Validation:** Post-processing ensures compliance with competition syntax rules.
4. **Feedback (implicit):** Observed outputs may guide model fine-tuning or prompt reformulation.

2.3 System Diagram

Figure 1 illustrates the systemic diagram developed during Workshop 1. It models the input-output flow, major components, and actor roles in the process.

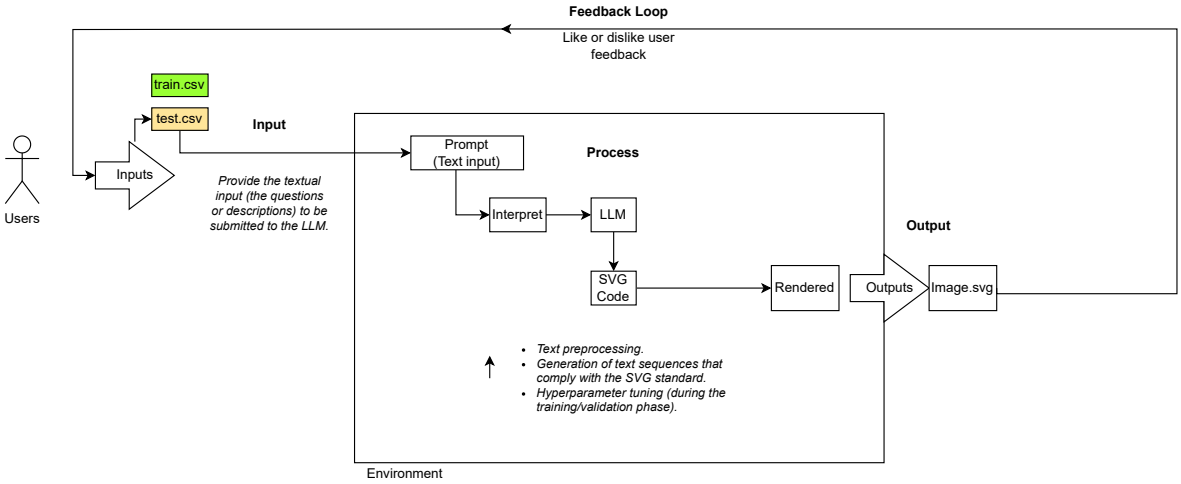

Figure 1: Systemic diagram outlining key actors, processes, and feedback within the LLM-based SVG generation system.

2.4 System Characteristics: Complexity, Sensitivity, Variability

This system exhibits the defining characteristics of complex adaptive systems:

- **Sensitivity to Input:** Due to the non-deterministic nature of LLMs, minor variations in prompt phrasing can lead to significant differences in SVG outputs.
- **Ambiguity and Creativity:** The model is expected to "interpret" vague descriptions, introducing both opportunity for creativity and risk of semantic drift.
- **Variability and Non-linearity:** Outputs are shaped by an interplay between the input, model weights, decoding parameters (e.g., temperature), and prompt constraints—yielding a system with nonlinear behavior.

While the system is evaluated through static metrics during the competition, its actual behavior is dynamic and emergent, shaped by iterative training, prompt refinement, and system-level tuning. This reveals the gap between deterministic expectations and probabilistic operation—a key insight in systemic analysis.

2.5 Holistic Insight

By understanding the competition through the lens of General Systems Theory, we not only identify components but also uncover interactions and dependencies. The system's behavior is not just a sum of parts—it emerges from coordination, constraint, adaptation, and feedback. These insights guide the architectural and simulation decisions in later sections.

3 System Architecture

To address the technical and systemic demands of the Kaggle competition "Drawing with LLMs," a modular, scalable, and resilient architecture was designed. The system integrates natural language processing with generative modeling and validation stages, ensuring both performance and adaptability under competition constraints.

3.1 Architectural Overview

The system is composed of four core subsystems:

- **Description Processor**
- **LLM Connector**
- **SVG Validator**
- **Performance Analyzer**

Each subsystem operates independently and communicates through well-defined interfaces. This architectural choice facilitates maintainability and enables targeted improvements without compromising the system as a whole.

3.2 Subsystem Descriptions

3.2.1 Description Processor

This component processes raw user prompts. It performs cleaning, keyword extraction, and semantic normalization. In this stage, domain-specific constraints (such as size, shape, or color hints) may be added to clarify the input and improve the LLM's response quality.

3.2.2 LLM Connector

Implemented via a local Ollama instance, this module interacts with the selected LLM (e.g., `llama3:8b`) using a structured prompt that constrains the model's output to valid SVG. It manages communication through API calls, controls inference parameters (like temperature and token limits), and includes retry mechanisms for robustness.

3.2.3 SVG Validator

This module checks the syntactic and semantic correctness of the SVG output. It ensures that the generated code adheres to XML/SVG standards, sanitizes dangerous tags, and discards malformed or empty results. When invalid outputs are detected, fallback SVGs are used, or regeneration is triggered.

3.2.4 Performance Analyzer

The performance analyzer tracks system behavior in real-time. It logs CPU/GPU usage, generation time, memory load, and output complexity. This data is stored and visualized for later analysis. It supports evaluation of the system under different prompt types and stress levels, providing essential feedback for design refinement.

3.3 Execution Flow

Figure 2 illustrates the system workflow, from user input to SVG output and analysis.

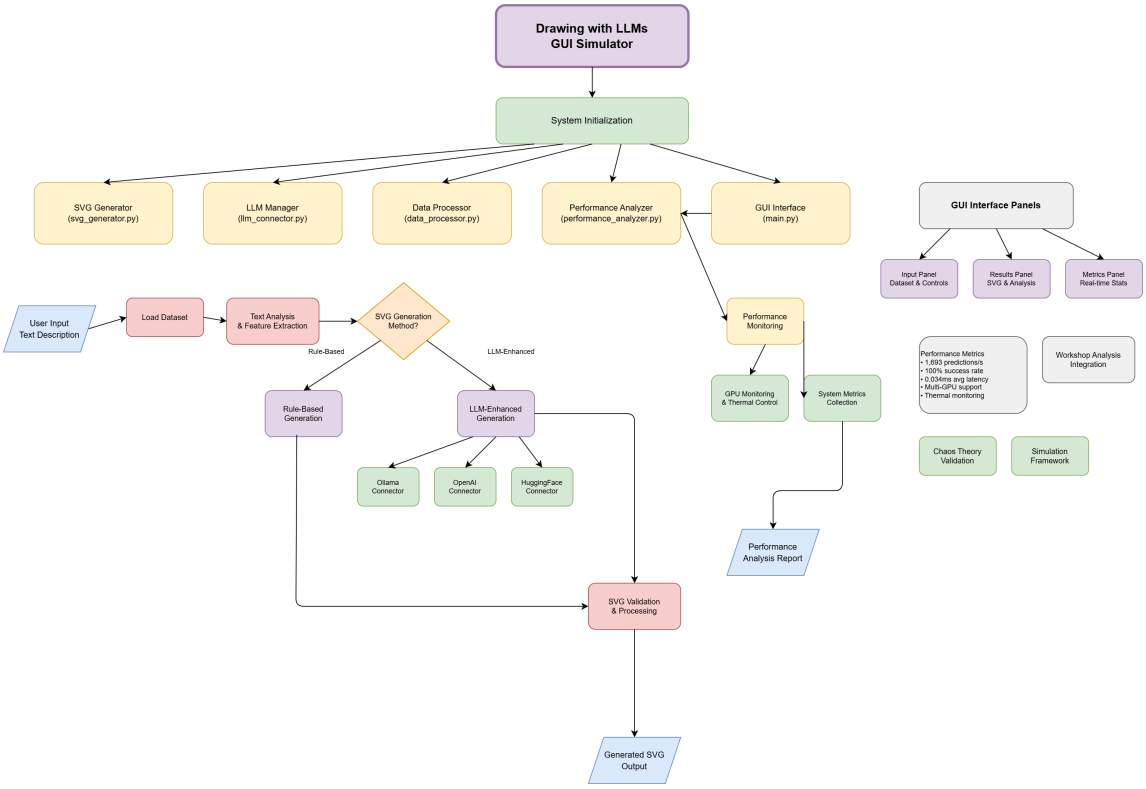

Figure 2: Execution pipeline of the SVG generation system.

1. The user submits a prompt to the system.
2. The description processor prepares the text.
3. The prompt is sent to the LLM via the connector.
4. The LLM generates a raw SVG candidate.
5. The validator checks and sanitizes the SVG.
6. The performance analyzer logs metrics.
7. The final SVG is returned, visualized, or stored.

3.4 Constraints and Assumptions

The system assumes:

- The LLM is locally available via a running Ollama server.
- Sufficient computational resources (RAM, GPU) are accessible.
- Prompts are in English and follow basic descriptive grammar.

To mitigate potential failures, the system includes timeouts, fallback mechanisms, and prompt-enhancing routines.

3.5 Systems Thinking Perspective

From a systems thinking point of view, the architecture balances centralized generation (via LLM) with decentralized analysis and validation. Subsystems act semi-autonomously but are interconnected through feedback loops. The use of performance logging and modular design supports adaptability, learning, and optimization—key characteristics of resilient and complex systems.

This modular pipeline enables component replacement (e.g., a new LLM) and opens the door for future extensions, such as multi-model ensembles, advanced semantic validation, or cross-modal training strategies.

4 LLM Integration Strategy

The integration of a Large Language Model (LLM) into the SVG generation system was guided by principles of modularity, local execution, and alignment with systemic requirements identified during the early phases of system analysis. The LLM operates as the core generative module in the architecture, translating structured prompts into syntactically valid Scalable Vector Graphics.

4.1 Model Selection and Deployment

To balance performance, accessibility, and cost, the `llama3:8b` model was selected for local deployment using the Ollama framework. This approach ensured:

- Independence from paid APIs or usage quotas.
- Full control over inference parameters and system load.
- Reduced latency and improved system availability.

Unlike external APIs (e.g., OpenAI, Claude, or DeepSeek), local inference offered greater flexibility for testing, debugging, and performance optimization without incurring financial or operational restrictions.

4.2 Prompt Engineering

A key aspect of successful LLM integration was the design of a robust system prompt. This prompt served to constrain the model's generative behavior and enforce output specifications required by the competition. The engineered prompt:

- Instructed the LLM to return only valid SVG code.
- Specified accepted SVG tags (`<svg>`, `<rect>`, `<circle>`, `<line>`, etc.).
- Defined structural constraints (e.g., root `<svg>` tag, no explanatory text).
- Emphasized brevity to avoid exceeding token or character limits.

This prompt acted as a boundary controller, guiding model output toward valid and meaningful visual structures while reducing the risk of hallucination or malformed syntax.

4.3 Connector Implementation

The file `ollama_connector.py` encapsulates the interaction logic between the local system and the Ollama server. It provides:

- Prompt injection and user prompt combination.
- Automatic retries in the case of failed or incomplete outputs.
- Timeout handling and exception recovery to ensure continuity.
- JSON parsing and response validation.

The connector also supports flexible switching between different LLM backends by abstracting model-specific configurations.

4.4 Execution Pipeline

The LLM integration fits into a broader pipeline as follows:

1. User submits a natural language prompt.
2. The system prompt and user input are merged and sent to the LLM.
3. The model returns an SVG string candidate.
4. The candidate is passed to the validator module.
5. Upon validation, the SVG is either accepted, retried, or discarded.

This pipeline supports fully automated generation cycles, enabling rapid prototyping and extensive simulation scenarios.

4.5 Systemic Implications

From a systemic engineering standpoint, the LLM is treated as a non-deterministic subsystem. Its behavior depends not only on internal parameters (weights, temperature) but also on external prompt structure and context. As a result:

- The system must tolerate output variability.
- Robust validation and feedback mechanisms are essential.
- The system benefits from runtime diagnostics and iterative prompt refinement.

By balancing generative power with systemic controls, the integration strategy supports a harmonious interaction between human input, model creativity, and competition constraints. The design reflects the holistic thinking inherent in systems engineering: anticipating uncertainty and ensuring resilience through modular oversight and feedback loops.

5 Simulation and Evaluation

The system evaluation was conducted in two stages: an initial manual simulation using a high-capacity external LLM, followed by a fully automated simulation with a locally hosted, smaller model. This two-phase approach highlights the trade-offs between output quality, control, cost, and system-level automation.

5.1 Manual Simulation Using Copilot

During the early phase of experimentation, GitHub Copilot was used to manually generate SVG outputs based on input prompts. This high-capacity, proprietary model demonstrated the potential for:

- Translating natural language into SVG commands.
- Producing complex and visually coherent drawings.
- Validating prompt engineering practices.

However, this method required human input at every stage, and automating the pipeline would have required access to commercial APIs (e.g., OpenAI, DeepSeek). These services often incur costs and impose usage limits, limiting flexibility and scalability.

5.2 Automated Simulation with Local LLM

To overcome these limitations, a local deployment of `llama3:8b` via Ollama was adopted. While the model has fewer parameters than commercial LLMs, it offers:

- Full control over inference logic and runtime configuration.
- Cost-free, unrestricted usage suitable for experimentation.

- Seamless integration into the system pipeline.

This shift allowed the system to be evaluated under realistic, repeatable conditions using a fully scripted pipeline.

5.3 Evaluation Metrics

The evaluation focused on core systemic metrics that reflect runtime behavior and resource performance:

- **Generation Time:** The average time required to produce and validate an SVG was approximately 20 seconds per prompt using the local LLM.
- **Complexity Score:** SVGs were analyzed for number of elements, nested structures, and tag diversity.
- **Resource Consumption:** System-level metrics (CPU, GPU, memory usage) were recorded per generation cycle.

These results allowed the team to profile and optimize system performance across different stages of the pipeline.

5.4 Systemic Findings

The use of a local model led to important insights:

- Model performance is sufficient for controlled environments despite the lack of generative richness.
- Full automation ensures consistency and reliability without human intervention.
- Resource load is predictable and manageable, enabling deployment on mid-range hardware.

The system favors autonomy and replicability over high-end creativity. This balance is more appropriate for competition settings, where constraints, traceability, and reproducibility are prioritized.

6 Discussion

The results of the system evaluation highlight several systemic behaviors and architectural implications that emerged throughout the development and simulation phases. These insights extend beyond simple performance metrics and reveal deeper patterns in the system's design, operation, and adaptability.

6.1 System Scalability and Modularity

Thanks to its modular architecture, the system can adapt to future changes, such as swapping out the LLM for a more powerful one or integrating new validation techniques. Each subsystem operates independently with clear input/output interfaces, supporting horizontal scaling and easier debugging. This modularity enhances maintainability and promotes experimentation without risking system collapse.

6.2 Emergence and Unexpected Behavior

Despite having deterministic logic within each module, the system occasionally produced outputs that were unexpected in form or content. These emergent behaviors were often the result of interactions between modules (e.g., between prompt engineering and model decoding). From a systems theory perspective, this demonstrates that global behavior cannot always be predicted from local rules—a hallmark of complex adaptive systems.

6.3 Variability and Chaos

A small change in the wording of a prompt often led to dramatically different SVG outputs, confirming the system's sensitivity to initial conditions. This aligns with chaos theory, where small perturbations in input can yield disproportionately large effects. Such sensitivity is not necessarily a flaw—it can also be a creative strength—but it must be monitored, especially in automated pipelines.

6.4 Systemic Monitoring and Feedback

The integration of a performance analyzer provided real-time diagnostics and historical logs. This enabled early detection of bottlenecks, unusual resource consumption, or invalid output patterns. Although the system does not yet implement a full feedback loop for self-correction, it lays the groundwork for a future digital twin capable of forecasting behavior based on previous simulations.

6.5 Trade-offs: Creativity vs. Control

Using a locally hosted LLM involves a trade-off: the outputs may be less stylistically rich compared to those from commercial-grade models, but they are more predictable, controllable, and cost-effective. This favors scenarios where systemic stability and repeatability are more critical than creative expressiveness.

6.6 Holistic Implications

Ultimately, the project underscores that building AI-driven generative systems is not just about maximizing model performance. It involves orchestrating many components—each with its constraints and behaviors—into a coherent and responsive system. Systems thinking provided the lens through which these interactions could be understood, tested, and improved.

7 Conclusion

The development of a generative SVG system using local LLMs has demonstrated both the potential and challenges of integrating language models into modular, automated pipelines. Through a systems-oriented methodology, we observed that the system exhibits high levels of variability and noise in its outputs—an expected behavior given the stochastic nature of LLMs.

However, this chaotic behavior can be mitigated through prompt refinement and, potentially, fine-tuning the model on domain-specific data. While Ollama's lightweight deployment of LLaMA 3.1 does not match the creative range of large commercial models, it provides an effective balance between control, interpretability, and resource efficiency.

The open-source nature of the LLM and its infrastructure makes the system not only functional but also educational. Developers gain a deeper understanding of how the models operate internally and how systemic components interact dynamically to shape the final output.

Overall, the project highlights the importance of treating AI systems not just as tools, but as interconnected ecosystems. Success depends not solely on model performance, but on systemic alignment, adaptability, and observability. Future work may explore reinforcement learning, adaptive prompting, and self-healing mechanisms to further enhance the system's robustness and creativity.

References

[1] T. Brown *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[2] M. Chen *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[3] E. Nijkamp *et al.*, "Codegen: An open large language model for code with multi-turn program synthesis," *arXiv preprint arXiv:2203.13474*, 2022.

[4] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proc. of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650, 2019.

[5] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2414–2423, 2016.

[6] A. Ramesh *et al.*, "Zero-shot text-to-image generation," in *International Conference on Machine Learning*, pp. 8821–8831, 2021.

[7] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[8] J. Devlin *et al.*, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[9] OpenAI, "GPT-4 Technical Report," *arXiv preprint arXiv:2303.08774*, 2023.

[10] H. Touvron *et al.*, "LLaMA 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[11] M. C. Jackson, *Systems Thinking: Creative Holism for Managers*, Wiley, 2003.

[12] L. von Bertalanffy, *General System Theory: Foundations, Development, Applications*, New York: George Braziller, 1968.

[13] Meta AI, "Llama 3.1: Open Foundation and Fine-Tuned Chat Models," Meta AI Blog, Jul. 2024. [Online]. Available: `https://ai.meta.com/blog/meta-llama-3-1/`