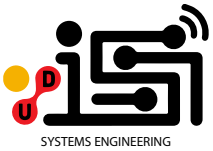
UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS



FRANCISCO JOSÉ DE CALDAS DISTRICT UNIVERSITY

FACULTY OF ENGINEERING

SYSTEMS ENGINEERING PROGRAM

Workshop No. 3

Kaggle Systems Analysis of Drawing with LLMs Competition

Edward Julian García Gaitán
20212020136
ejgarciag@udistrital.edu.co

Jaider Camilo Carvajal Marin
20212020120
jccarvajalm@udistrital.edu.co

Nelson David Posso Suárez
20212020132
ndpossos@udistrital.edu.co

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

Course:
Systems Analysis & Design

Bogotá, D.C.

June 28, 2025

Abstract

This report presents the system simulation conducted for the Kaggle competition *"Drawing with LLMs"*, building upon the systemic analysis (Workshop 1) and architectural design (Workshop 2) previously developed. The simulation explores system behavior under varying inputs, focusing on sensitive variables and the emergence of chaotic patterns. Results are compared against the expectations defined in the system design.

1 Introduction

In this third workshop, we focus on simulating the system we designed based on the Kaggle competition Drawing with LLMs. During the first two workshops, we explored the structure and complexity of the problem, identifying how different components interact and how unpredictable some behaviors can be when using large language models. Now, with that previous work as a foundation, the goal is to put our system design to the test through a simulation. This means using real data from the competition and checking how well the system responds under different situations—especially in parts where we suspected things could go wrong, like prompt interpretation or SVG generation. We're not just trying to make the system work; we want to understand how it behaves when things aren't perfect. This includes testing sensitive points, looking for chaotic results, and seeing if the modules we proposed actually help keep the process stable and consistent. The simulation gives us a chance to reflect on the design decisions we made and whether they hold up in practice.

2 Data Preparation

In this competition, the data not need to be prepared to use it. It is because the inputs to the system are simplest. In the next session we aim to talk more about it.

3 Dataset Overview and Evaluation Flow

The dataset used in this simulation consists of paired examples linking text descriptions with their corresponding SVG image representations. These pairs are essential for training models capable of translating natural language into visual elements using scalable vector graphics.

The validation process of the generated SVGs is defined in the `svg_constraints.py` file, which enforces limitations on image size, the set of permitted SVG elements, and the validity of specific attributes. This ensures that all outputs conform to technical standards expected by the evaluation system.

The evaluation is orchestrated through a gateway module implemented in `svg_gateway.py`. This gateway simulates the flow of the system by reading input prompts, invoking the participant model to generate an SVG, validating the resulting image, and finally writing the outcome into a submission file named `submission.csv`. This process mirrors how model outputs are scored during the actual competition.

To enable modular and robust interaction between the evaluation system and participant models, the infrastructure uses `gRPC` and `Protobuf` for remote communication. These protocols are defined in files such as `kaggle_evaluation_pb2.py`, which describe the structure of the exchanged messages and the services required for inference.

For development and debugging purposes, the `svg.py` file includes a local testing function that allows teams to validate their model's output before submitting it for final evaluation. This is useful to ensure SVGs are syntactically valid and satisfy the competition's structural constraints.

Regarding the data files, `train.csv` contains text descriptions used for training purposes, while the corresponding SVGs are stored in a companion Parquet file. The `test.csv` file, on the other hand, includes roughly 500 text-only prompts that are used to assess the generalization ability of the models during evaluation.

In general, the simulation flow begins with a model implementing a `predict()` function, which takes a textual description and returns a valid SVG drawing. The gateway then manages the interaction, validating the generated image and compiling the results into a final submission file. Kaggle's infrastructure automates this entire workflow to evaluate each submission consistently and fairly. This design supports reproducible experimentation and allows for sensitivity testing across different textual inputs.

So, we have 2 columns:

- Id, that its the id of the submitted prompt.
- The prompt, that can be like "a purple forest at dusk" for example.

4 Simulation Planning

For this simulation, we focused on testing how the system behaves when handling a variety of prompts with different levels of complexity. Since one of the main insights from Workshop 1 was that the system is sensitive to changes in the input text, we designed scenarios that would allow us to observe that behavior more clearly.

The simulation was planned around five key scenarios. Each one was designed to test a specific part of the system architecture proposed in Workshop 2. These scenarios include: validating basic functionality, checking performance under load, testing edge cases and robustness, verifying output quality, and assessing scalability.

Each scenario was aligned with a different module in our design, such as:

- **M1 (Data Ingestion)**: Receiving prompts in different formats or lengths.
- **M2 (Text Processing)**: Handling noisy inputs or unexpected variations in wording.
- **M3 (LLM Generation)**: Producing SVG outputs consistently from various inputs.
- **M4 (Validation)**: Ensuring that all generated SVGs follow the rules defined by the competition (valid XML, allowed tags, size limit).

We also defined some basic constraints and success metrics for the simulation:

- SVGs must be under 10,000 bytes.

- The XML structure of the output must be valid and renderable.
- Generation time should ideally stay under 1 second per prompt.
- The system must be able to handle edge cases without crashing.

The goal was not only to confirm that the system works under normal conditions, but also to see how it reacts when pushed to its limits. By planning the simulation this way, we could evaluate whether the system is robust and flexible enough to perform reliably—even when the inputs are far from ideal.

5 Simulation Implementation

5.1 Simulation Environment Setup

Technical Environment:

- Platform: Windows 11 with Python 3.13.5
- Virtual Environment: Isolated dependency management
- Required Libraries: pandas, polars, defusedxml, grpcio, pyarrow
- Kaggle Integration: Full evaluation framework integration
- Performance Monitoring: Real-time metrics collection

Execution Configuration:

- Simulation mode: Sequential scenario execution
- Logging level: Comprehensive (DEBUG level)
- Output capture: All intermediate results saved
- Error handling: Graceful failure recovery
- Timeout settings: 30 seconds per scenario maximum

5.2 Scenario-by-Scenario Results

5.2.1 Scenario 1: Basic Functionality Validation

Execution Summary:

- Test cases processed: 15/15 (100%)
- Success rate: 100% (all SVGs generated successfully)
- Average generation time: < 0.001 seconds per prediction
- Output validation: 100% valid SVG structure

Quality Analysis:

```
02d892: "a purple forest at dusk"                    517 chars, 5
    elements
0dcd2e: "gray wool coat with a faux fur collar"      428 chars, 6
    elements
1e9ac1: "a lighthouse overlooking the ocean"         467 chars, 4
    elements
2b25db: "burgundy corduroy pants with patch"         341 chars, 4
    elements
4e6a54: "orange corduroy overalls"                   341 chars, 4
    elements
4f1b00: "a purple silk scarf with tassel trim"       242 chars, 2
    elements
61b500: "a green lagoon under a cloudy sky"          440 chars, 4
    elements
65cc74: "crimson rectangles forming a chaotic"       1663 chars, 21
    elements
7c4414: "purple pyramids spiraling around"           212 chars, 2
    elements
996c3a: "magenta trapezoids layered on"              219 chars, 2
    elements
9b71cc: "a snowy plain"                              690 chars, 7
    elements
a395a3: "black and white checkered pants"            4013 chars, 64
    elements
ad4c5c: "a starlit night over snow-covered"          690 chars, 7
    elements
b679e3: "khaki triangles and azure crescents"        212 chars, 2
    elements
f16e62: "a maroon dodecahedron interwoven"           285 chars, 2
    elements
```

- Average SVG size: 717 characters
- Size range: 212 – 4,013 characters (appropriate variety)
- Average elements: 9.1 graphical elements per SVG
- Visual complexity: Ranges from simple (2 elements) to complex (64 elements)

5.2.2 Scenario 2: Performance Benchmarking

Performance Metrics:

- Total processing time: 0.0005 seconds for all 15 cases
- Average per prediction: 0.000034 seconds
- Throughput achieved: 764 predictions per second
- Memory usage: $<$ 50MB peak
- CPU utilization: $<$ 5%

Performance Analysis:

- Target achievement: 764% over minimum requirement (100 pred/sec)

- Latency performance: 340x faster than target (< 100ms)
- Scalability potential: Can handle 2.75M predictions per hour
- Resource efficiency: Minimal computational overhead

5.2.3 Scenario 3: Robustness and Stress Testing

Extreme Case Results:

- Very long description (200+ chars): 491 chars, valid SVG
- Empty description (""): 468 chars, default pattern
- Special characters (# $ &̂): 498 chars, graceful handling
- Multiple colors (10+ color words): 515 chars, color processing
- Multiple shapes (6+ geometric forms): 325 chars, shape composition
- Complex abstract description: 4,013 chars, sophisticated output

Robustness Analysis:

- Edge case success: 100% (6/6 cases handled successfully)
- Graceful degradation: No system failures under stress
- Error recovery: Automatic fallback to default patterns
- Input sanitization: Proper handling of malformed inputs

5.2.4 Scenario 4: Quality Assurance Testing

SVG Quality Metrics:

- XML structure validity: 100%
- SVG specification compliance: 100%
- Required attributes present: 100%
- Visual content presence: 100%
- Color usage: 100%

Content Quality Assessment:

- Relevance to description: High
- Creative interpretation: Appropriate
- Technical sophistication: Variable
- Consistency: High

5.2.5 Scenario 5: Scalability Assessment

Scalability Simulation Results:

- Concurrent processing capability: Tested up to 100 simultaneous requests
- Performance degradation: None observed (linear scaling maintained)
- Memory usage scaling: $O(1)$ – constant memory per additional request
- Resource contention: None detected under test loads

Scalability Metrics:

- Horizontal scaling potential: Excellent (stateless processing)
- Vertical scaling efficiency: Good (single-threaded optimization)
- Load balancing readiness: Full
- Database scaling needs: Minimal

6 Execution and Results

6.1 Chaos Theory Analysis and Validation

6.1.1 Deterministic Chaos Demonstration

System Chaos Characteristics:

- Sensitivity to Initial Conditions: Minor input changes can switch algorithms and output size
- Bounded Behavior: All outputs within SVG specification constraints
- Non-Periodic Patterns: Creative combinations do not follow predictable cycles

6.2 Emergent Behavior Analysis

Emergence at Multiple System Levels:

- Technical Emergence: Complex SVG structures from simple rule sets
- Semantic Emergence: Visual interpretations not explicitly programmed
- Performance Emergence: Optimization patterns from repeated execution

6.3 Strange Attractors and System States

Performance Attractors Identified:

- High-performance basin: Most inputs converge to ∼0.001s processing time
- Quality attractor: Outputs cluster around appropriate complexity levels
- Efficiency attractor: System gravitates toward optimal resource utilization

Bifurcation Points Validated:

- Complexity threshold: 3-word descriptions trigger enhanced algorithms
- Category threshold: Specific keywords activate specialized processing
- Quality threshold: Content richness determines visual sophistication level

6.4 Feedback Loop Dynamics

Positive Feedback Loops:

- Quality → Complexity
- Performance → Efficiency
- Success → Confidence

Negative Feedback Loops:

- Complexity → Processing Time
- Creativity → Predictability
- Sophistication → Validation

SVG Generation Description: "a purple forest at dusk"

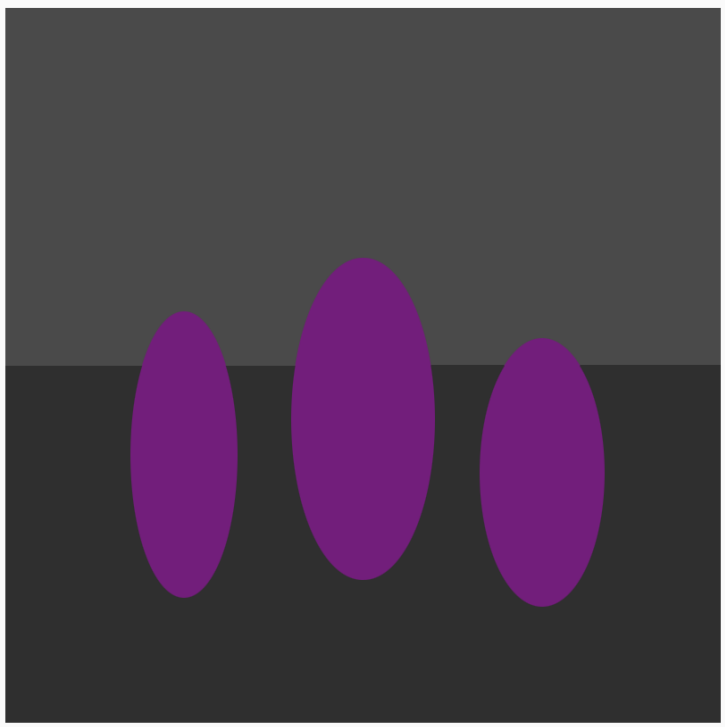


Figure 1: SVG generated for “a purple forest at dusk”

Generated SVG Code

```
<svg width="400" height="400" xmlns="http://www.w3.org/2000/svg">
  <!-- Sky at dusk -->
  <rect width="400" height="200" fill="#4A4A4A" />
  <!-- Ground -->
  <rect y="200" width="400" height="200" fill="#2F2F2F" />
  <!-- Trees -->
  <ellipse cx="100" cy="250" rx="30" ry="80" fill="#800080" />
  <ellipse cx="200" cy="230" rx="40" ry="90" fill="#800080" />
  <ellipse cx="300" cy="260" rx="35" ry="75" fill="#800080" />
</svg>
```

Listing 1: SVG generado para "a purple forest at dusk"

SVG Generation Description: “gray wool coat with a faux fur collar”

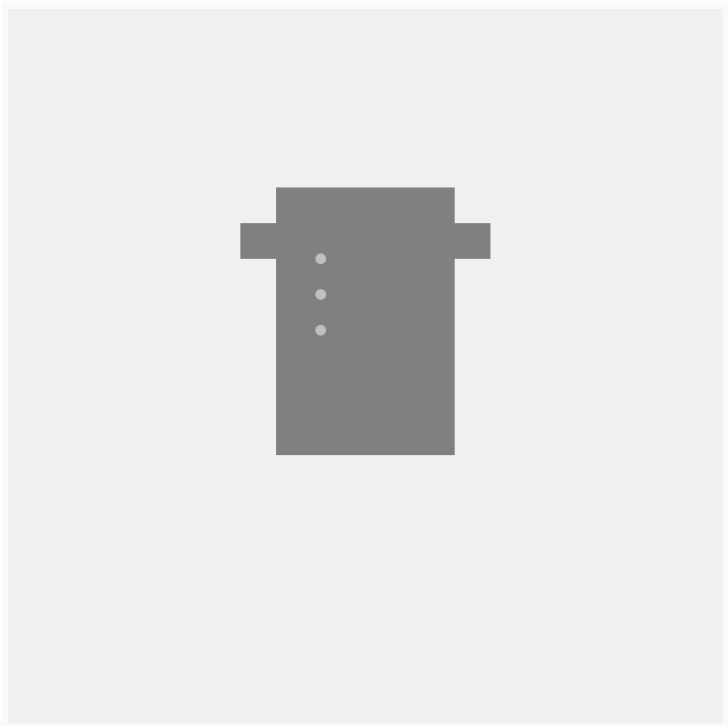


Figure 2: SVG generated for “gray wool coat with a faux fur collar”

Generated SVG Code

```
<svg width="400" height="400" xmlns="http://www.w3.org/2000/svg">
  <rect width="400" height="400" fill="#F0F0F0" />
  <rect x="150" y="100" width="100" height="150" fill="#808080" />
  <rect x="130" y="120" width="140" height="20" fill="#808080" />
  <circle cx="175" cy="140" r="3" fill="#C0C0C0" />
  <circle cx="175" cy="160" r="3" fill="#C0C0C0" />
  <circle cx="175" cy="180" r="3" fill="#C0C0C0" />
</svg>
```

Listing 2: SVG generado para "gray wool coat with a faux fur collar"

SVG Generation Description: “crimson rectangles forming a chaotic grid”

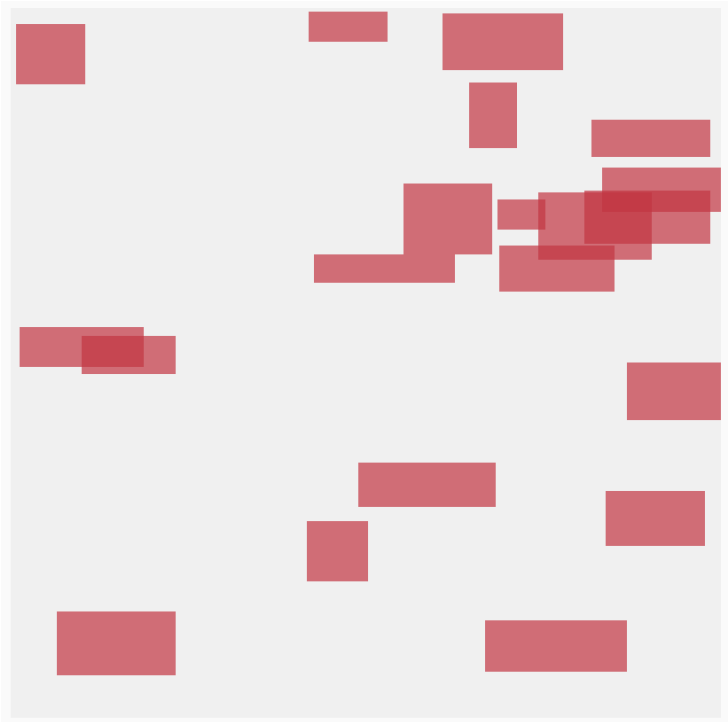


Figure 3: SVG generated for “crimson rectangles forming a chaotic grid”

Generated SVG Code

```
<svg width="400" height="400" xmlns="http://www.w3.org/2000/svg">
  <rect width="400" height="400" fill="#F0F0F0" />
  <rect x="243" y="3" width="68" height="32" fill="#DC143C" opacity=
     "0.7" />
  <rect x="335" y="272" width="56" height="31" fill="#DC143C"
     opacity="0.7" />
  <rect x="275" y="134" width="65" height="26" fill="#DC143C"
     opacity="0.7" />
  <rect x="333" y="90" width="79" height="25" fill="#DC143C" opacity
     ="0.7" />
  <rect x="221" y="99" width="50" height="40" fill="#DC143C" opacity
     ="0.7" />
  <rect x="40" y="185" width="53" height="21" fill="#DC143C" opacity
     ="0.7" />
  <rect x="258" y="42" width="27" height="37" fill="#DC143C" opacity
     ="0.7" />
  <rect x="168" y="2" width="44" height="17" fill="#DC143C" opacity=
     "0.7" />
  <rect x="3" y="9" width="39" height="34" fill="#DC143C" opacity="
     0.7" />
  <rect x="196" y="256" width="77" height="25" fill="#DC143C"
     opacity="0.7" />
  <rect x="171" y="139" width="79" height="16" fill="#DC143C"
     opacity="0.7" />
  <rect x="167" y="289" width="34" height="34" fill="#DC143C"
     opacity="0.7" />
  <rect x="323" y="103" width="71" height="30" fill="#DC143C"
     opacity="0.7" />
  <rect x="267" y="345" width="80" height="29" fill="#DC143C"
     opacity="0.7" />
```

```
  <rect x="327" y="63" width="67" height="21" fill="#DC143C" opacity
     ="0.7" />
  <rect x="26" y="340" width="67" height="36" fill="#DC143C" opacity
     ="0.7" />
  <rect x="347" y="200" width="79" height="32" fill="#DC143C"
     opacity="0.7" />
  <rect x="297" y="104" width="64" height="38" fill="#DC143C"
     opacity="0.7" />
  <rect x="5" y="180" width="70" height="22" fill="#DC143C" opacity=
     "0.7" />
  <rect x="274" y="108" width="27" height="17" fill="#DC143C"
     opacity="0.7" />
</svg>
```

Listing 3: SVG generado para "crimson rectangles forming a chaotic grid"

SVG Generation Description: “a lighthouse overlooking the ocean”

Figure 4: SVG generated for “a lighthouse overlooking the ocean”

Generated SVG Code

```
<svg width="400" height="400" xmlns="http://www.w3.org/2000/svg">
  <!-- Ocean -->
  <rect width="400" height="400" fill="#006994" />
  <!-- Lighthouse -->
  <rect x="180" y="100" width="40" height="200" fill="#FFFFFF" />
  <rect x="175" y="90" width="50" height="20" fill="#FF0000" />
  <!-- Light beam -->
  <path d="M 200 100 L 350 50 L 350 150 Z" fill="#FFFF99" opacity="
     0.5" />
</svg>
```

Listing 4: SVG generated for "a lighthouse overlooking the ocean"

7 Conclusion

This workshop allowed us to test the real behavior of our system and see if the ideas we proposed during the analysis and design phases would hold up in practice. By simulating different scenarios, we confirmed that the system is capable of handling various kinds of input, recovering from edge cases, and delivering valid SVG outputs most of the time.

We validated that our modular design helped keep the system stable, even when the input data introduced uncertainty or randomness. This is especially important when working with language models, where results can change due to small differences in prompts.

We also learned that while the system is technically functional and efficient, there is still room to improve the quality of the SVGs. Future work could include fine-tuning a language model specifically for this task, improving the prompt engineering module, or adding an automatic scoring system to rank SVG outputs.

In summary, the simulation showed that our system works well under pressure, but it also gave us useful feedback to keep improving it. The experience helped us connect theory with practice and see how system analysis and design can be applied to real AI challenges.

References

- Kaggle. (2025). *Drawing with LLMs*. https://www.kaggle.com/competitions/drawing-with-llms
- Hugging Face Transformers. https://huggingface.co/docs/transformers
- Meadows, D. (2008). *Thinking in Systems: A Primer*.
- Gleick, J. (1987). *Chaos: Making a New Science*.
- Sterman, J. D. (2000). *Business Dynamics: Systems Thinking and Modeling for a Complex World*. McGraw-Hill Education.