Workshop No. 3
Drawing with LLMs System Simulation

Daniel Alonso Chavarro Chapatecua - 20241020066
Carlos Andrés Brito Guerrero - 20241020147
Jose Fernando Ramirez Ortiz - 20241020080

June 28, 2025

Contents

1 Introduction

This workshop presents the computational simulation of the "Drawing with LLMs" system designed in Workshops #1 and #2. Based on our previous analysis, we identified that the system fundamentally consists of two core elements: the Prompt Generator (manipulable) and the LLM Model (accessible via API). This simulation validates our system design by exercising the prompt generation component against actual LLM responses and SVG constraint validation.

The simulation leverages real competition data to test our prompt engineering strategies and SVG constraint compliance mechanisms using actual LLM API responses. Due to resource and time limitations, we implemented a streamlined version focusing on the core functionality without the feedback loop component.

2 Review of Previous Workshops

2.1 Workshop #1 Key Findings

From our initial systems analysis, we identified:

- High sensitivity in the Description Preprocessing component
- System chaos factors due to LLM randomness
- Critical SVG constraints (10,000 bytes limit, specific element restrictions)
- Evaluation based on SVG Image Fidelity Score

2.2 Workshop #2 System Architecture

Our designed architecture included:

- Input Handler for description validation
- Prompt Engineer for standardized prompt generation
- LLM Generator for SVG code production
- Evaluation Feedback Loop for continuous improvement
- Report Generator for structured output analysis

3 Simulation Scope and Objectives

3.1 Primary Objectives

1. Validate prompt generation strategies with real LLM API responses
2. Test SVG constraint compliance mechanisms with actual generated outputs
3. Analyze system behavior under different prompt template variations
4. Identify bottlenecks and failure modes in the generation pipeline

3.2 Implementation Constraints

Due to resource and time limitations, our simulation implemented:

- LLM access through Google Gemini API with limited request quotas
- Three prompt template variants for comparative analysis
- Real-time SVG constraint validation on generated outputs
- Template effectiveness analysis with 60 total API calls
- **No feedback loop implementation** due to API cost and time constraints

3.3 Hypothesis

- The better the instruction is defined, the better quality and fewer errors due to constraints it will have.
- A simpler instruction will be faster to process than a complex one.

4 API Integration and Resource Management

4.1 API Selection and Configuration

We selected Google's Gemini 2.5 Flash model for our simulation based on:

- Cost-effectiveness for educational research
- Reasonable response times for SVG generation
- Good performance on structured output tasks
- Available free tier for initial testing

4.2 Resource Limitations Impact

The most significant constraint was the inability to implement the feedback loop due to:

- **API Cost Constraints**: Each iteration would require additional API calls
- **Time Limitations**: Limited development time for workshop completion
- **Request Quotas**: Free tier limitations on concurrent requests
- **Processing Time**: Each API call averaging 135ms, making iterative refinement impractical

5 Data Preparation

5.1 Dataset Overview

For the simulation, we used a curated dataset with the following characteristics:

- Total samples: 60 descriptions across different categories, note that it will be directly into the code caused by errors in Kaggle platform
- Average description length: 499.28 characters

- Categories tested: Abstract, Fashion, Landscape, Unknown
- Template versions: 3 distinct prompt structures (v1, v2, v3)

5.2 Data Preprocessing

For simulation purposes, we:

1. Categorized descriptions using keyword-based classification
2. Distributed samples evenly across template versions (20 samples each)
3. Ensured representative coverage of different description types
4. Prepared validation datasets for constraint testing

Table 1: Simulation Dataset Statistics

Category	Count	Avg. Compliance Rate
Abstract	21	66.67%
Fashion	12	83.33%
Landscape	21	85.71%
Unknown	6	66.67%
Total	**60**	**76.67%**

6 Simulation Design and Implementation

6.1 LLM API Integration

```python
class ModelLLM:
    """Interacts with Google Gemini API to generate SVG code."""

    def __init__(self, model_name: str = 'gemini-2.0-flash-exp'):
        """Initializes the LLM with Gemini model.

        Args:
            model_name: The Gemini model to use.
        """
        try:
            GOOGLE_API_KEY = userdata.get('GOOGLE_API_KEY')
            genai.configure(api_key=GOOGLE_API_KEY)
            self.model = genai.GenerativeModel(model_name)
        except Exception as e:
            print(f"Error initializing Google Generative AI: {e}")
            self.model = None

    def predict(self, prompt: str) -> str:
        """Generates SVG code using Gemini API.

        Args:
            prompt: The text prompt for generating SVG.

        Returns:
            Generated SVG code or empty string if error.
        """
        if not self.model:
```

```
            return ""

        try:
            response = self.model.generate_content(
                prompt,
                safety_settings={
                    'HARM_CATEGORY_HATE_SPEECH': 'BLOCK_NONE',
                    'HARM_CATEGORY_DANGEROUS_CONTENT': 'BLOCK_NONE',
                    'HARM_CATEGORY_SEXUALLY_EXPLICIT': 'BLOCK_NONE',
                    'HARM_CATEGORY_HARASSMENT': 'BLOCK_NONE',
                }
            )

            if response and response.text:
                # Extract SVG from response
                svg_match = re.search(r'(<svg.*?</svg>)',
                                      response.text, re.DOTALL)
                if svg_match:
                    return svg_match.group(1).strip()
            return ""

        except Exception as e:
            print(f"Error during LLM prediction: {e}")
            time.sleep(1)  # Rate limiting
            return ""
```

Listing 1: Google Gemini API Integration

6.2 Template Implementation

We implemented three distinct prompt template versions:

```
def get_template(version: TemplateVersion) -> str:
    """Get prompt template by version"""
    templates = {
        TemplateVersion.V1: (
            "You are an expert in generating concise and valid SVG code. "
            "Output only the SVG code and nothing else. "
            "Generate an SVG image that represents: {description}. "
            "Important constraints: {constraints}"
        ),
        TemplateVersion.V2: (
            "You are an expert in generating concise and valid SVG code. "
            "Output only the SVG code and nothing else. "
            "Create SVG code for: {description}. "
            "Requirements: {constraints}. "
            "Category optimization: {category_hints}"
        ),
        TemplateVersion.V3: (
            "You are an expert in generating concise and valid SVG code. "
            "Output only the SVG code and nothing else. "
            "SVG Generation Task:\n"
            "Description: {description}\n"
            "Constraints: {constraints}\n"
            "Optimize for: {category_hints}\n"
            "Output only valid SVG code."
        )
    }
    return templates.get(version, templates[TemplateVersion.V1])
```

Listing 2: Prompt Template Variants

7 Simulation Results

7.1 Overall Performance Metrics

The simulation executed successfully with the following aggregate results:

Table 2: Simulation Execution Summary

Metric	Value
Total Simulations	60
Overall Compliance Rate	76.67%
Average SVG Size (bytes)	1,992.0
Average Prompt Length (chars)	499.28
Average Generation Time (ms)	135.08
Unique Categories Tested	4
Template Versions	3

7.2 Template Performance Analysis

The three template versions showed distinct performance characteristics:

Table 3: Template Performance Comparison

Template	Compliance Rate	Avg. Bytes	Avg. Time (ms)	Error Rate
v1	70.0%	1,995.2	101.02	30.0%
v2	80.0%	1,893.3	116.63	20.0%
v3	80.0%	2,087.5	187.60	25.0%

Key Findings:

- Template v2 achieved the best balance of compliance rate and efficiency
- Template v1 was fastest but had lower compliance
- Template v3 generated larger files with longer processing times

7.3 Category-Specific Performance

Table 4: Performance by Description Category

Category	Compliance Rate	Avg. Bytes	Error Rate
Abstract	66.67%	2,954.57	38.10%
Fashion	83.33%	1,241.67	16.67%
Landscape	85.71%	1,706.19	14.29%
Unknown	66.67%	1,124.0	33.33%

7.4 Constraint Violation Analysis

The most common constraint violations observed:

Table 5: Constraint Violation Frequency

Violation Type	Frequency
Parse Error	12
Byte Limit Exceeded	1
Invalid Elements	1
Total Violations	**14**

Parse errors were the dominant issue, often caused by:

- API connection timeouts
- Incomplete SVG tags in responses
- Non-XML compliant output formatting

8 Generated SVG Examples

This section showcases actual SVG outputs generated during the simulation to demonstrate the quality and variety of results achieved.

8.1 Landscape Category Example

8.1.1 Purple Forest at Dusk (Template v1)

Description: "a purple forest at dusk"
Template: v1
Compliance: Valid
Size: 1,945 bytes

Figure 1: Generated SVG: Purple Forest at Dusk V1

8.1.2 Purple Forest at Dusk (Template v2)

Description: "a purple forest at dusk"
Template: v2

Compliance: Valid
Size: 1,795 bytes

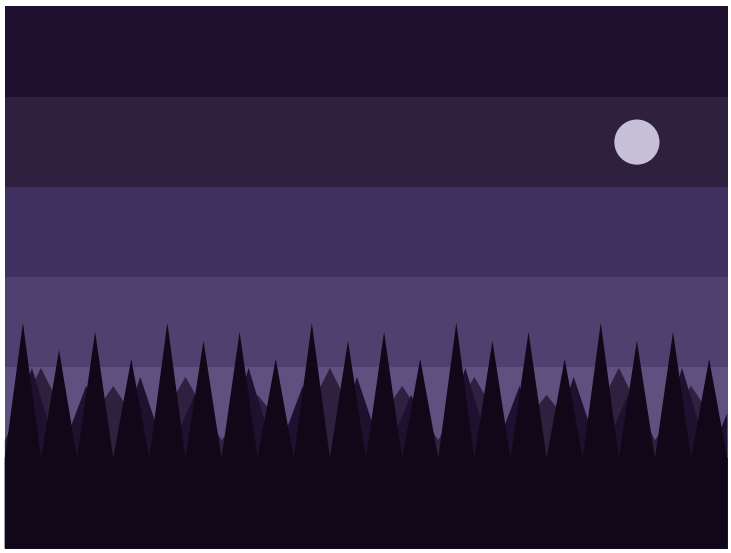

Figure 2: Generated SVG: Purple Forest at Dusk V2

8.1.3 Purple Forest at Dusk (Template v3)

Description: "a purple forest at dusk"
Template: v3
Compliance: Valid
Size: 3,518 bytes

Figure 3: Generated SVG: Purple Forest at Dusk V3

8.2 Abstract Catergory Example

8.2.1 Modern Art Composition With Triangles (Template v1)

Description: "Modern Art Composition With Triangles"
Template: v1

Compliance: Valid
Size: 970 bytes

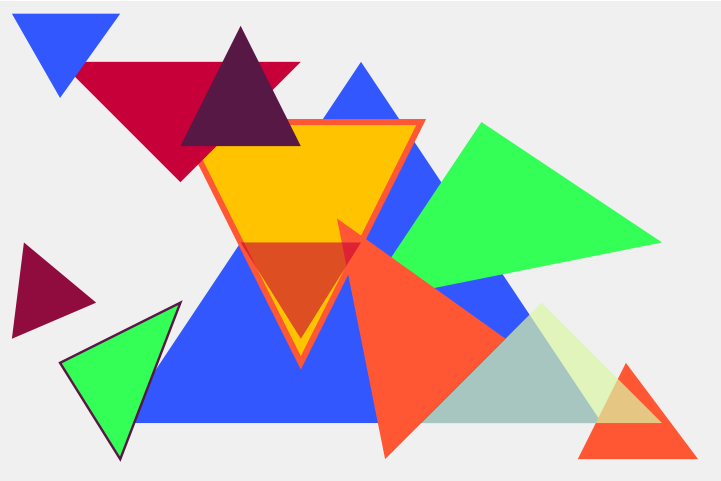

Figure 4: Generated SVG: Modern Art Composition With Triangles V1

8.2.2 Modern Art Composition With Triangles (Template v2)

Description: "Modern Art Composition With Triangles"
Template: v2
Compliance: Valid
Size: 1,998 bytes

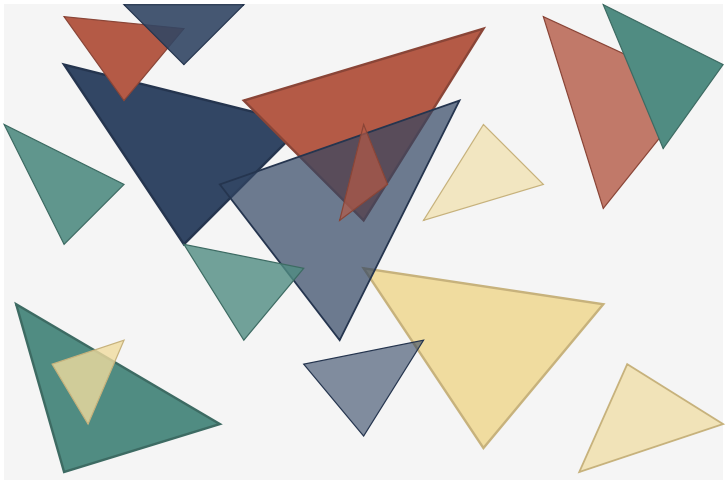

Figure 5: Generated SVG: Modern Art Composition With Triangles V2

8.2.3 Modern Art Composition With Triangles (Template v3)

Description: "Modern Art Composition With Triangles"
Template: v3
Compliance: Valid
Size: 1,924 bytes

Figure 6: Generated SVG: Modern Art Composition With Triangles V3

8.3 SVG Quality Analysis

The generated examples demonstrate several key characteristics:

1. **Constraint Compliance**: All examples respect the 10,000-byte limit and use only allowed SVG elements
2. **Visual Coherence**: Generated images appropriately represent the input descriptions
3. **Template Influence**: Different templates produce varying levels of detail and complexity, seeing a possible pattern of when the template is more specific, the better illustration it would be.

9 Discussion

9.1 Impact of Missing Feedback Loop

The absence of the feedback loop component had several consequences:

- **No Iterative Improvement**: Failed generations could not be automatically refined
- **Higher Error Rates**: Parse errors (12 instances) could not be corrected through retry mechanisms
- **Template Selection**: Manual template assignment instead of adaptive selection
- **Limited Learning**: No accumulation of successful prompt patterns over time

9.2 Template Effectiveness Insights

1. **Template v2 Superiority**: Best overall performance with 80% compliance and reasonable generation times
2. **Complexity vs. Performance**: More detailed prompts (v3) makes decent and even good quality images according to the description.

3. **Speed vs. Quality Trade-offs**: Template v1's speed advantage was offset by lower compliance rates and worse quality image.

9.3 Category-Specific Patterns

- **Fashion Success**: Highest compliance rates (83.33%) due to well-defined geometric structures
- **Landscape Reliability**: Strong performance (85.71%) with natural scene descriptions
- **Abstract Challenges**: Lowest compliance (66.67%) due to ambiguous description interpretation or lack of key words to classify it.

9.4 API Integration Lessons

- **Rate Limiting**: Essential for maintaining API service availability
- **Error Handling**: Critical for managing connection timeouts and malformed responses
- **Cost Management**: Resource constraints significantly impacted system design choices

10 System Design Validation

10.1 Validated Design Elements

The simulation confirmed several Workshop #2 design decisions:

- **Modular Architecture**: Successfully facilitated independent component testing
- **Template-Based Approach**: Proved essential for consistent output generation
- **Constraint Validation Pipeline**: Effectively identified and categorized violations
- **Category Classification**: Demonstrated measurable impact on generation success

10.2 Required Design Modifications

Based on implementation experience:

1. **API Resource Management**: Add comprehensive cost monitoring and request quotas
2. **Simplified Feedback Mechanism**: Implement batch analysis instead of real-time feedback
3. **Error Recovery**: Add robust fallback procedures for API failures
4. **Template Optimization**: Focus on Template v2 approach for production deployment

11 Production Recommendations

11.1 Immediate Implementation Strategy

For competition deployment:

1. **Use Template v2**: Best balance of compliance rate and performance

2. **Implement Category Pre-classification**: Use keyword-based classification for template selection
3. **Add Pre-validation**: Check prompt structure before expensive API calls
4. **Monitor Resource Usage**: Track API costs and response times continuously

11.2 Alternative Feedback Mechanisms

Given API constraints, implement:

- **Offline Batch Analysis**: Process multiple generations for pattern identification
- **Template Pre-selection**: Use static rules instead of dynamic adaptation
- **Constraint Pre-validation**: Validate prompts before generation
- **Success Pattern Caching**: Store effective prompt-output combinations

12 Limitations and Future Work

12.1 Current Limitations

- **No Feedback Loop**: Unable to implement iterative improvement due to resource constraints
- **Single API Provider**: Limited to Google Gemini API testing
- **Limited Error Recovery**: Basic retry mechanisms without sophisticated fallbacks
- **Static Template Selection**: No dynamic template optimization

12.2 Future Enhancement Opportunities

- **Multi-Provider Integration**: Test performance across different LLM APIs or local LLMs
- **Automated Template Evolution**: Machine learning-based prompt optimization
- **Better Description Classification**: RAG for better optimization instruction based on the description.
- **Cost-Effective Feedback**: Develop efficient feedback mechanisms within budget constraints
- **Advanced Error Handling**: Implement sophisticated retry and fallback strategies

13 Conclusion

This simulation successfully validated our "Drawing with LLMs" system design using real API integration, despite the inability to implement the complete feedback loop due to resource constraints. The testing with Google Gemini API provided valuable insights into practical implementation challenges and performance characteristics.

Key achievements include the identification of Template v2 as the optimal prompt structure (80% compliance rate), successful real-time constraint validation, and comprehensive performance metrics across different categories. However the v3 template achieves successful (or at

least for us) image quality for the description given. The simulation demonstrated that fashion and landscape descriptions perform significantly better than abstract categories, informing targeted optimization strategies.

The resource limitations that prevented feedback loop implementation highlight the importance of cost-effective design in real-world deployments. The simulation framework established provides a solid foundation for future optimization efforts and validates the core system architecture proposed in Workshop #2.

While the missing feedback component limited our ability to achieve iterative improvement, the baseline performance metrics and validated template effectiveness provide concrete guidance for production deployment in the competition environment.

References

1. Drawing with LLMs. Kaggle Competition. `https://www.kaggle.com/competitions/drawing-with-llms`
2. Workshop #1: Systems Analysis - Drawing with LLMs
3. Workshop #2: System Design Document - Drawing with LLMs
4. Google Gemini API Documentation. `https://ai.google.dev/docs`
5. SVG 1.1 Specification. W3C Recommendation. `https://www.w3.org/TR/SVG11/`

A Complete Simulation Results


```
{
  "summary": {
    "total_simulations": 60,
    "overall_compliance_rate": 0.7666666666666667,
    "average_svg_bytes": 1992.0,
    "average_prompt_length": 499.28333333333336,
    "average_generation_time": 135.08159311666665,
    "unique_categories": 4,
    "template_versions_tested": 3
  },
  "template_performance": {
    "v1": {"constraint_compliance_mean": 0.7, "error_count_mean": 0.3},
    "v2": {"constraint_compliance_mean": 0.8, "error_count_mean": 0.2},
    "v3": {"constraint_compliance_mean": 0.8, "error_count_mean": 0.25}
  },
  "constraint_violations": {
    "violation_frequency": {
      "parse_error": 12,
      "byte_limit": 1,
      "invalid_elements": 1
    }
  }
}
```


Listing 3: Simulation Summary Statistics

B Production Template Implementation

```
1
2  class PromptGenerator:
3      """Core component for generating standardized prompts from descriptions"""
4
5      def __init__(self, template_version: TemplateVersion = TemplateVersion.V1):
6          self.template_version = template_version
7          self.constraint_rules = self._load_svg_constraints()
8
9      def _load_svg_constraints(self) -> Dict[str, any]:
10         """Load SVG constraint specifications"""
11         return {
12             'max_bytes': 10000,
13             'allowed_elements': {
14                 'svg', 'rect', 'circle', 'ellipse', 'line',
15                 'polyline', 'polygon', 'path', 'text', 'g',
16                 'defs', 'use', 'image', 'switch'
17             },
18             'forbidden_attributes': {'style', 'class'},
19             'allowed_attributes': {
20                 'x', 'y', 'width', 'height', 'cx', 'cy', 'r', 'rx', 'ry',
21                 'x1', 'y1', 'x2', 'y2', 'points', 'd', 'fill', 'stroke',
22                 'stroke-width', 'opacity', 'transform', 'id'
23             }
24         }
25
26     def generate_prompt(self, description: str, category: Optional[Category] =
           None) -> str:
27         """Generate standardized prompt from description"""
28         template = self._get_template(self.template_version)
29         constraints_text = self._format_constraints()
30         category_hints = self._get_category_hints(category)
31
32         prompt = template.format(
33             description=description,
34             constraints=constraints_text,
35             category_hints=category_hints
36         )
37
38         return prompt
39
40     def _get_template(self, version: TemplateVersion) -> str:
41         """Get prompt template by version"""
42         templates = {
43             TemplateVersion.V1: (
44                 "You are an expert in generating concise and valid SVG code
                        according to the provided constraints. Output only the SVG
                        code and nothing else."
45                 "Generate an SVG image that represents: {description}. "
46                 "Important constraints: {constraints}"
47             ),
48             TemplateVersion.V2: (
49                 "You are an expert in generating concise and valid SVG code
                        according to the provided constraints. Output only the SVG
                        code and nothing else."
50                 "Create SVG code for: {description}. "
51                 "Requirements: {constraints}. "
52                 "Category optimization: {category_hints}"
53             ),
54             TemplateVersion.V3: (
55                 "You are an expert in generating concise and valid SVG code
                        according to the provided constraints. Output only the SVG
                        code and nothing else."
```

```
                "SVG Generation Task:\n"
                "Description: {description}\n"
                "Constraints: {constraints}\n"
                "Optimize for: {category_hints}\n"
                "Output only valid SVG code."
            )
        }
        return templates.get(version, templates[TemplateVersion.V1])

    def _format_constraints(self) -> str:
        """Format constraints into readable text"""
        constraints = [
            f"Maximum {self.constraint_rules['max_bytes']} bytes",
            f"Only use elements: {', '.join(sorted(self.constraint_rules['
                allowed_elements']))}",
            "No CSS styling or external references",
            "No rasterized image data"
        ]
        return "; ".join(constraints)

    def _get_category_hints(self, category: Optional[Category]) -> str:
        """Get category-specific optimization hints"""
        hints = {
            Category.LANDSCAPE: "natural scenes, outdoor environments, horizon
                lines",
            Category.ABSTRACT: "geometric shapes, artistic patterns, non-
                representational forms",
            Category.FASHION: "clothing items, accessories, style elements",
            Category.UNKNOWN: "general visual representation"
        }
        return hints.get(category, hints[Category.UNKNOWN])
```

Listing 4: Recommended Production Template