

Generating Simulated Data with a Large Language Model

Jeffrey Kerley, Derek T. Anderson, Andrew R. Buck, and Brendan Alvey

Electrical Engineering and Computer Science Department, University of Missouri, Columbia, MO 65211, USA

ABSTRACT

Enabling abstraction within a programming language has benefits. However, the associated complexity of such abstractions often pose a steep learning curve for users. While user interfaces or visual scripting can help alleviate this to some extent, they often lack readability and reproducibility, especially as complexity grows. Herein, we explore the use of Large Language Models (LLMs) as an intermediate between the nuanced, syntactical programming language and the natural (human) way of describing the world. Our formal language LSCENE is a way to procedurally generate realistic synthetic scenes in the Unreal Engine. This tool is useful because artificial intelligence (AI) typically requires large volumes of labeled data with variety. To generate such data for training and evaluating AI, we employ an LLM to interpret and sample LSCENEs that are compatible with user input. Through this approach, we demonstrate a reduction in abstract complexity, elimination of syntax complexity, and the ability to tackle complex tasks in LSCENE using natural language. To illustrate our findings, we present three experiments with quantitative results focused on spatial reasoning, along with a more intricate qualitative example of automatically generating an environment for a specific biome.

Keywords: Artificial Intelligence, LCAP, LSCENE, Large Language Model, Procedural, Simulation

1. INTRODUCTION

The manual creation or specification of digital content constitutes a resource-intensive endeavor, demanding significant time and financial investments. In the past few decades people have sought new ways to streamline tasks across domains, namely in entertainment. Early examples date back to the 1970s, with procedural generation techniques employed in tabletop role-playing games, and Perlin noise in the 1980s,¹ which revolutionized texture and terrain generation in computer gaming and film. Since then, a plethora of procedures have been proposed, from cellular automata to fractals, Brownian motion, decision trees, L-systems, and more, addressing diverse needs like textures/materials, objects (e.g., trees, buildings), organisms (e.g., characters, kinematics, animation), environments (e.g., dungeons, terrain, mountains, cities), to game play itself (Valve’s AI Director in Left 4 Dead). While historically these tools have been primarily confined to closed academic or entertainment circles, recent years have witnessed the emergence of low-to-no cost professional-grade tools. Notable examples include Houdini and new advancements in the Unreal Engine 5. The pursuit of streamlining content creation processes remains an academic and financial topic of paramount interest across disciplines.

In this article, we explore the role of large language models (LLMs) to streamline the procedural generation process. Previously, we introduced LSCENE, a structured formal language for specifying 3D scenes, and LCAP, a language for data collection.^{2,3} A limitation of our prior work is its tedious nature. There often exists complexity in expressing high-level concepts and relationships via the low-level syntax used in LSCENE. Manual editing of an LSCENE JSON can be challenging. The objective of the present article is to leverage natural language descriptions as input, e.g., “make a scene with a few small wooden boxes on an open grassy field with a few medium bushes.” An LLM is used to understand and translate the user’s request into a LSCENE JSON, which is then consumed by a procedural engine to sample and build a 3D environment. This article serves as both an exploration into the feasibility of using LLMs for this purpose and an investigation into effective methods to achieve high-quality results with minimal user interaction.

Our focus is the automated generation of scenes (via LSCENE) and data (via LCAP) that facilitate the training, evaluation, and closing the loop on artificial intelligence (AI). Our aim is to establish a streamlined process for automatically sourcing content from large, high-quality photometrically scanned assets to construct computer vision image datasets that enable tasks like 3D reconstruction, object localization and detection,

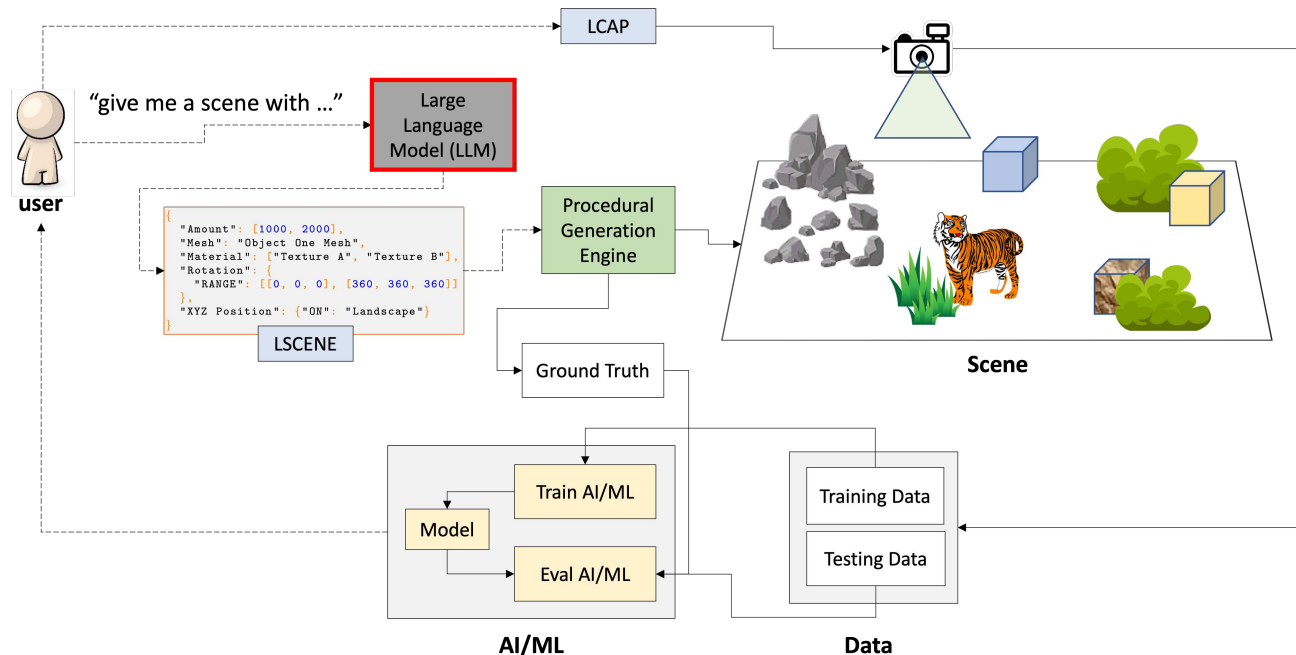


Figure 1: Illustration of the current article. Users interact with an LLM to more easily construct a formal specification of a scene (LSCENE) that is interpreted by a procedural generation engine. This scene is sampled to produce a data set (via LCAP), with associated ground truth, to train and evaluate an AI model.

semantic segmentation, fusion, and dynamic interrogation. Simulation has numerous advantages in this regard. Notably, data collection in the real world poses significant challenges, particularly in acquiring precise metadata and ground truth, often serving as a bottleneck for many applications. Simulation offers the ability to rapidly prototype, gather data for augmenting real-world datasets, or directly train AI models. However, a key challenge lies in specifying the desired data to be collected and ensuring its generation at scale, in terms of volume, variety, and quality. Employing vague, high-level natural language-based descriptions simplifies human input and output, enhances explainability, and aids in the closed-loop process, where insights gained from experiments inform future sampling strategies. Recent research has shown that harnessing language as a bridge between different domains such as imagery and text can lead to foundation models that enhance generality. This approach facilitates deeper and more robust reasoning, bolstered by these versatile representations.

Our article contributes in several key ways. Firstly, we explore multiple LLMs and employ different strategies for framing our task to them. Secondly, we conduct multiple examples to assess the overall effectiveness of each LLM and interaction strategy. Thirdly, we provide a reflective analysis, highlighting the pros and cons of these various approaches. Our aim is to offer insights into promising pathways and practical limitations, thereby informing the reader and directing future work. Figure 1 is a high level overview of the proposed article.

2. RELATED WORK

2.1 Procedural Generation

The realm of procedural generation has been a subject of extensive research within the video game and film industries for decades. Dating back to 1985, Perlin pioneered the development of algorithms based on noise functions to craft procedural materials, initially showcased through the creation of a marble vase.⁴ This procedural approach evolved to encompass a variety of methodologies such as fractals and L-systems,^{5,6} utilized for generating intricate structures like plants, trees, and vegetation. Subsequent research delved into real-time procedural terrains⁷ and city generation,⁸ with companies like SideFX building on these concepts to realize commercial tools like Houdini. While focus has predominantly been on algorithms for asset generation, including creatures and animation, recent endeavors have expanded to prioritize user experience. Notably, Valve’s introduction of the AI

Director in Left4Dead⁹ exemplifies this shift, employing algorithms to dynamically adjust entity spawning and properties within maps, enhancing gameplay pacing based on user engagement. Moreover, procedural text-based games and maps, such as dungeons, have a longstanding presence in video games, dating back even further.

In recent years, Epic Games has made strides in the realm of procedural content generation with a suite of tools centered around the Unreal Engine (UE). While UE has historically offered support for procedural features such as terrain creation and foliage spawning, the latest iterations of the engine have expanded upon this foundation. For instance, with the release of UE5.2, Epic introduced the Procedural Content Generation (PCG) plugin, which boasts support for a range of procedural functionalities including mesh generation, texture creation, foliage placement, level generation, and animation. Notably, these capabilities have been seamlessly integrated with UE's Blueprint Visual Scripting system, enhancing accessibility and ease of use. Furthermore, in UE5.4, Epic unveiled support for an additional PCG tool, the Biome Creation plugin. This innovative tool adopts a flexible, data-driven approach, leveraging systemic methodologies to empower creators with the latest advancements in procedural content generation.

Despite the abundance of procedural concepts in content generation for user experiences, relatively limited attention has been directed towards procedural algorithms catering to the needs of AI. The UE PCG tool, for example, has poor integration with generative AI since a LLM can not yet interface with the blueprint medium in a sufficient manner. This raises critical questions regarding the types, volumes, varieties, and attributes of data essential for AI applications, marking AI as a relatively new "consumer" in the procedural domain.

2.2 Large Language Models (LLMs)

Using large language models (LLMs) as translators, for programming or normal written languages, has been one of their most successful applications see Zhu 2023,¹⁰ Jiao 2023 (GPT-4),¹¹ and Briako 2023(PALM).¹² These findings suggest that LLMs can be used for novel languages which have the same principles as the other languages that the LLMs have been trained and exposed to. For example, Google's Gemini pro was tested on Kalamang, which has less than 200 speakers worldwide, and it was able to translate English to Kalamang at a human level with the amount of content given. This idea of taking English sentences and turning them into a never before seen language, or vice versa is a powerful feature of LLMs. Understanding how LLMs can connect general ideas to each specific language is important. For instance, Bhattamishra 2020¹³ discusses how LLMs are capable of learning general rules for formal languages that do not have "periodicity and modular counting." Positional encoding solves most of the issues of the self attention layer in transformers for learning more complex languages. Hierarchical and recursive languages fall under this realm. The metrics that confirm this emergent behavior, however, often have their own problems.

Testing LLMs on spatial reasoning tasks such as CALVIN, Sharma2023,¹⁴ or seeing what topological or mathematical a model understands, yamada2024¹⁵ are additional techniques to understand the limits of the model's reasoning, knowledge, and general intelligence. The use of standardized tests to evaluate LLM's capabilities is not well understood. Even though benchmark datasets exists, contamination of these tests in the training are wide spread. For example, OpenAI stated in their technical report on GPT-4, that "the degradation (non-contaminated percent minus contaminated) is generally small and as often positive as negative, from which we conclude that contamination is not a substantive confounder on the overall results."¹⁶ So although they had full tests in the training data, such as the GRE or LSAT, etc., they claim the data shows no correlation between success rate on the test (GRE is contaminated but GPT-4 preforms worse on this than other standardized tests with less contamination). This is an inherently flawed assumption to make. Comparing accuracy on one test with contamination to another (with less), is not a legitimate method to dismiss the role contamination may have played on the model's performance.

Given the absence of a single LLM showcasing superior performance across all tasks and the rapidly evolving nature of the field, it becomes imperative to examine multiple LLMs to derive conclusions for the current article. Herein, we delve into the specifics of the following LLMs (see Table 1):

- M1 (Gemini-1.5-pro): (Google) A mixture of experts model that boasts a 1 million token context length.
- M2 (GPT-4): (OpenAI) Scores high on most standardized tests and currently one of the most consistent LLM for problem solving and creative tasks, only 32k context length.

Table 1: LLM Models

Model Label	Full Model Name
M1	Gemini-1.5-pro
M2	GPT-4
M3	GPT-3.5-turbo
M4	Claude-3-opus-20240229
M5	Claude-3-sonnet-20240229
M6	Claude-3-haiku-20240307

- M3 (GPT-3.5-turbo): A faster but simpler model than GPT-4, good for most everyday tasks.
- M4 (Claude-3-opus-20240229): (Anthropic) LLM that rivals OpenAI’s GPT-4 in abstract and creative problem solving, as well as 200k context length with good recall. The most capable model from Anthropic.
- M5 (Claude-3-sonnet-20240229): (Anthropic) Comparable to GPT-4, this model demonstrates lower performance on various mathematical and graduate-level assessments, rendering it less proficient overall.
- M6 (Claude-3-haiku-20240307): (Anthropic) Comparable to GPT-3.5.

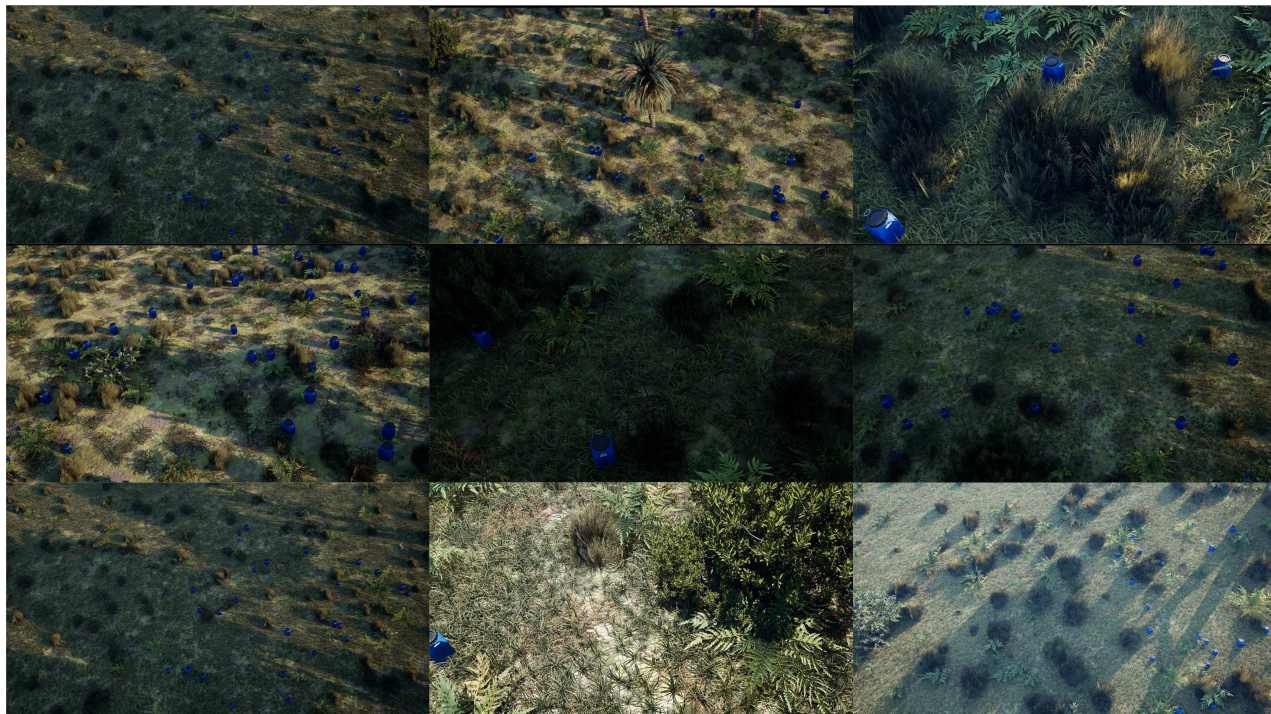


Figure 2: Example of procedurally generated imagery in the Unreal Engine 5 for a simple LSCENE corresponding to “a few blue water cans in an open grassland biome at different times of day.”^{2,3}

3. LSCENE

We recommend referring to our previous articles on LSCENE and LCAP for full details.^{2,3} In this section, we present a succinct overview essential for understanding key concepts discussed in this article. Figure 2 is LSCENE imagery for a simple scene with *a few* target objects (blue water cans) in an open grassland biome at different times of day. Specifically, the LSCENE specified foliage objects (bushes) on a grassy terrain (different textures/materials) with sparse grass. The ability to create procedural scenes like these typically involves establishing rules and constraints governing the generation, placement, and attributes of objects. These mechanisms

allow each procedural instance to draw from a defined space based on specific conditions. However, approaches relying on simplistic rules and limited expressiveness often result in unrealistic and/or repetitive arrangements. Here, we delve into a formal language that utilizes simplified JSON markup to enhance human editing and readability. LSCENE is as a powerful way to specify a scene that can then be sampled to procedurally generate complex and diverse synthetic environments.

We start by considering a trivial LSCENE (see Listing 1). For computer vision, we often desire to maintain certain visual characteristics like object texture or size while granting freedom in sampling spatial arrangement and orientation. This can be succinctly expressed as, “I want a scene with many instances of object One that have Texture A or B with random rotations and placement ON the Landscape.”

Listing 1: *Simple LSCENE specifying 1,000 to 2,000 sampled instances of an object with varying textures, random orientations, and positions in a user constrained region of space (“on” the landscape).*

```
{
  "Amount": [1000, 2000],
  "Mesh": "Object One Mesh",
  "Material": ["Texture A", "Texture B"],
  "Rotation": {
    "RANGE": [[0, 0, 0], [360, 360, 360]]
  },
  "XYZ Position": {"ON": "Landscape"}
}
```

While LSCENE operates at an abstract level with various implementation possibilities, we have predominantly used it in conjunction with the Unreal Engine (UE). Specifically, our implementation involves leveraging material parameters within UE5 to facilitate seamless texture swapping on materials. UE employs this method to amalgamate multiple textures—such as normals, base color, and displacement maps—into a unified material, culminating in a final output. Moreover, the availability of free assets from Quixel has proven advantageous. We adhere to the same naming conventions and employ identical master materials to ensure consistency in parameter manipulation across materials. Despite this standardization, LSCENE remains flexible enough to accommodate different parameter names for third-party materials, allowing for customization, modification, and sampling.

For an exhaustive list of capabilities, please refer to our previous work in Ref. 2, which delves into comprehensive functionalities and practical applications. The key takeaway is that LSCENE boasts a minimalist syntax conducive to sampling, yet possesses enough expressiveness to generate intricate and photorealistic scenes, thanks to the capabilities of UE5 and the availability of photometrically scanned assets from Quixel and other sources.

4. LLM DRIVEN LSCENE GENERATION

Current generation of scenes with LSCENE requires the manual markup of a JSON, a parser in C++, and UE which serves as the 3D rendering software and data extractor. A user who can navigate UE’s Editor has a majority of the skills needed to produce data via LSCENE. What is left to learn is the LSCENE’s syntax, which simply serves as a translation between the 3D scene a user is thinking of and creating that in UE. LSCENE is comparable to CSS, XML, HTML, Markup, and other Markup languages, only with the syntax of JSON gluing LSCENE together. The ideas present in all of these markup languages serve to label or describe properties of a web page, document, or other abstract data. Although LSCENE is a new and unseen language for an LLM, based on the literature, (Zhao, 2023),¹⁷ (Bubeck, 2023)¹⁸ and similarity between other known languages and markup, these models should perform adeptly at generating LSCENE JSON markup.

Creating a 3D scene using user-defined assets is made straightforward by employing Retrieval Augmented Generation (RAG) methods, as detailed in Gao (2024)¹⁹ and Lewis (2021).²⁰ In RAG, the embedding vector derived from a description of assets serves as the foundation for searching a database containing asset descriptions. A distance metric is computed between the embedding vectors, returning all vectors within a user-defined threshold as matches. This approach does not utilize generative AI; instead, it relies on the accuracy of embedding vectors in capturing information from natural text descriptions.

A challenge with these methods, however, is that while data retrieval is automated, overall context and decision-making still rests with the LLM after data retrieval. For relatively small databases, managing data through text documents remains a viable approach. OpenAI, Anthropic, and Google all endorse the practice of uploading files and inputting tokens at the prompt layer to enable models to reason and understand user-provided data. LSCENE will leverage this method for asset selection, feeding information into the prompt layer. Moreover, techniques like prompt engineering (Sahoo, 2024),²¹ chain of thought (Wei, 2022)²² and prompt tuning (Nori et al., 2023)²³ enhance performance with LLMs. They also provide valuable insights into effective communication with LLMs via structured prompting guidelines, RAG, agent ensembles, and similar methodologies. Let it be known, that although LLMs perform well, they do not respect formalism, grammars, or Automata completely, see (Liu,2023).²⁴

In contrast, we delve into (refer to Table 2) a straightforward method aimed at discerning the essential information requisite for a model’s success. We inquire whether the level of abstraction in the prompt correlates with the efficacy of more adept agents. Is it more advantageous to present a single example encompassing all use cases, or to provide five examples with more detailed labeling but distributed information? Each configuration is ingested as system directives or prompts. Our objective is to present robust examples embodying common spatial or relational requisites encountered during scene construction. By ensuring that the LSCENE generated aligns with the user’s specifications, we instill confidence in the accuracy of the output.

Table 2: LLM configurations explored herein for LSCENE generation

Configuration Label	Meaning
EX1:A	Generic LSCENE JSON with documentation
EX1:B	LSCENE JSON example with all use cases
EX1:C	EX1:A and EX1:B
EX1:D	Five shot LSCENE examples

An illustrative instance of a generic LSCENE configuration is provided in Listing 2, accompanied by corresponding text or natural language documentation in Listing 3. This example, denoted as EX1:A in Table 2, offers insight into the structure and content of a typical LSCENE setup. For a comprehensive listing of prompts and directive information, please refer to the appendix.

Listing 2: *Generic structure of an LSCENE JSON that we provide to an LLM.*

```

"Object_Identifier": {
  "Amount": ["low_num", "high_num"],
  "A_Class": [
    {"class_name1": "probability1"},
    {"class_name2": "probability2"}
  ],
  "Overlap": "boolean_value",
  "Scale": ["low_scale", "high_scale"],
}

```

Listing 3: *Generic LSCENE documentation that we provide to an LLM.*

```

"A_Class": "Specifies the class or Mesh of the object.",
"Overlap": "Specifies whether objects can overlap each other.",
"Amount": "Specifies the range of instances of an object.",
"Scale": "Specifies the range for the scaling of the object.",

```

5. EXPERIMENTS

Numerous aspects of LSCENE warrant exploration and validation. These include an LLM’s proficiency in generating syntactically correct programs, its capability to navigate an asset library and retrieve appropriate

content, and its ability to manipulate attributes such as color and shape, among others. While there are myriad cases to consider, we narrow our focus to a singular yet crucial attribute: spatial relations.

Generating procedural LSCENEs requires an LLM to possess some level of spatial intelligence. For example, users frequently desire to place objects in particular relative configurations, e.g., “ON the ground,” “IN a grid,” or “NEAR the edge.” While these are relatively simple configurations, more complicated composite examples exist, e.g., in a spiral, fractal, or stacked pyramid formation. Ideally, an LLM should be capable of generating a variety of scenes from straightforward prompts. Consider a scenario in which we want to train a deep neural network for image classification or semantic segmentation, which typically demands vast amounts of labeled data. To enhance the network’s generalization capabilities, extensive sampling is essential. Considering variation across variables like texture, shape, occlusion and relative viewing is imperative for achieving robustness. Thus, the creation of complex scenes, along with associated collection specifics outlined in LCAP, derived from procedural language rules is indispensable. To assess the spatial reasoning abilities of current generation LLMs and their capacity to integrate these concepts into a formal LSCENE, we propose a set of experiments.

Utilizing the directives outlined in Table 2 as input variations for each model described in Table 1, we instruct the LLMs to generate LSCENEs featuring random spatial placement, grid placement, and spiral placements. Each LLM configuration is designed to test different skills. In EX1:A, abstract rules and generic markup, along with simple documentation explaining each function in LSCENE, are provided. For instance, a function like A.Class is defined with a description indicating its purpose and expected arguments. EX1:B takes a literal approach, offering an instanced LSCENE JSON with all possible use cases of LSCENE functions. However, no documentation accompanies these examples, only literal instances of function usage. Combining elements from EX1:A and EX1:B, EX1:C creates a comprehensive prompt/directive containing literal examples, documentation, and abstract definitions for each function. In EX1:D, we present five examples with labeled input and output. Input descriptions represent scene descriptions, while output showcases generated LSCENE JSON corresponding to the given natural language description. Last, human evaluators are tasked with assessing these tasks, assigning a pass or fail based on the criteria outlined in Tables 3, 4, and 5. Syntax errors may arise from incorrect function usage, hallucinations, or structural issues within the LSCENE JSON. While a model may get part of a task correct, such as placing an object on the ground, it may also have a syntax error in the same output when hallucinating a “stack” function for example. Thus we have divided tasks into 3 categories of completeness, with error tracking to handle undefined behavior from the model.

5.1 Experiment 1: Random Spatial Placement

Placing objects within, on, or stacked atop one another may seem like straightforward requests, but there exists a crucial connection between the objects defined in the LSCENE and the spatial properties required for each arrangement. In LSCENE, the syntax accommodates spatial constraints such as ON or IN, simplifying the task for any LLM by enabling the correct invocation of these functions on the objects it positions. A LLM lacking this capability would struggle to use previously defined objects in the LSCENE correctly within the 3D space it constructs. Moreover, since objects in LSCENE can derive their position and dimensions from other objects, there’s also an *algebraic component* involved in calculating the final bounds or positions of certain objects.

Place Any entails the model’s ability to accurately position objects on landscapes or other objects, demonstrating its capability to reference LSCENE objects correctly. *One and In* introduces the constraint that an object must be placed on another object while remaining within the bounds of yet another object. This presents a double spatial constraint where both a boundary must be established and adhered to, and an object must be created and placed within this defined zone. *Stacked* represents a more abstract requirement. Initially, an object must be placed on another, such as the ground. Then, after these objects are positioned, an additional object must be stacked atop the already placed object. Although the LSCENE syntax for stacked may seem only marginally different from the other two, it necessitates a more meticulous interpretation of the scene description (the user’s request for a stacked object) and requires careful consideration of how to implement this in the LSCENE syntax (e.g., Object B ON Ground, Object A on Object B).

As an example, Figure 3 is an example of GPT-4 ability to support the *Place Any* request. The full set of results across LLMs is reported in Figure 4. Last, Table 3 reports the pass and fail behavior of our four different progressively more challenging constraint tasks for the different LLMs.

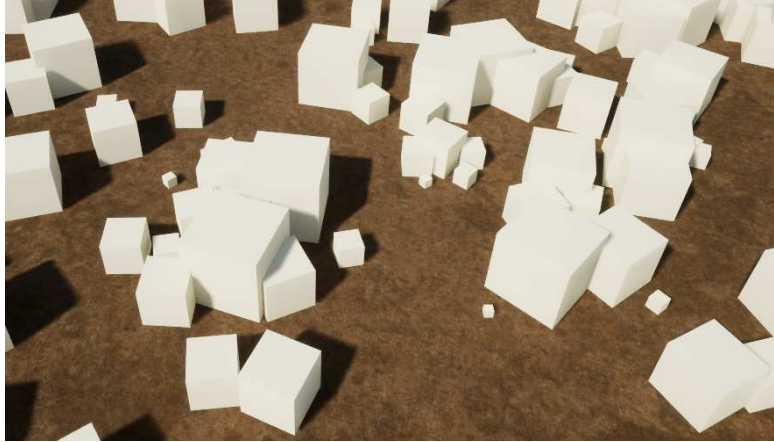


Figure 3: LSCENE generation of cubes via EX1:A for the random spatial placement experiment using GPT-4.

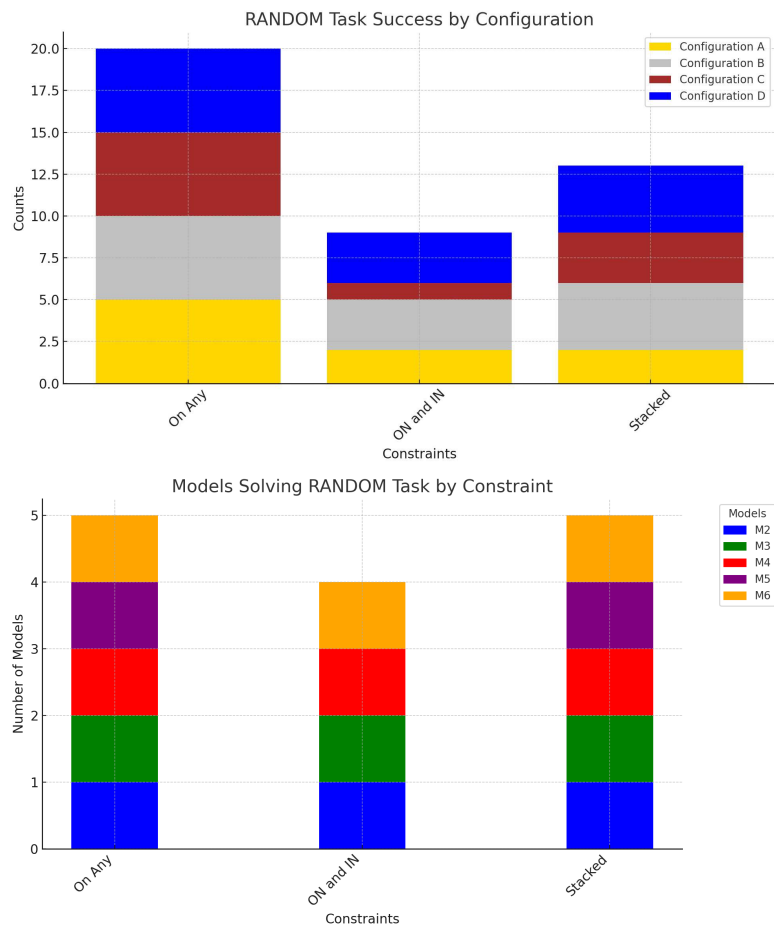


Figure 4: Random placement completion results across LLMs for Experiment 1.

Table 3: LLM LSCENE Generation for Random Spatial Placement (see Experiment 1)

Model	Place Any	On and In	Stacked	Syntax Error
M2	ABCD	AD	D	x
M3	ABCD	ABD	ABD	x
M4	ABCD	BCD	BC	ACD
M5	ABCD	x	ABCD	A
M6	ABCD	B	BCD	x

5.2 Experiment 2: Grid Placement

The category for the grid experiment, labeled *Grid ANY*, serves to evaluate and help us understand the model’s capacity to connect concepts in LSCENE with shape construction. If a model can generate an LSCENE, it should theoretically be able to produce one that generates specific 3D shapes, given that foundational models have demonstrated their ability to reason about mathematical concepts. Table 4 is our findings.

The initial surprising finding from the grid generation experiment is that GPT-4 barely outperforms GPT-3.5. Upon analyzing the generation from GPT-3.5, it becomes apparent that it was capable of producing a complete grid, albeit with spacing issues attributable to a misinterpretation of how the **extent** parameter affected the size of the cubes. Figure 5 show how reducing the size of the cubes by half results in a correctly spaced 3D grid. For comprehensive results by model or configuration, refer to Figure 6.

Table 4: LLM LSCENE Generation for Grid Placement (see Experiment 2)

Experiment Label	Grid ANY	Complete Grid	Complete Grid with Space	Syntax Error
M2	ACD	D	x	x
M3	CD	C	x	B
M4	ABCD	BCD	C	x
M5	ABCD	ACD	x	x
M6	ACD	A	x	x

Gemini’s comprehensive understanding is evident in the output it produced after being provided with EXP:C. Access to the Google generative AI API was not available at the time of this paper, so a simple case study was performed on Gemini-1.5-pro. Gemini was able to fully understand LSCENE and use it to configure a grid and spiral, as depicted in Figure 7. The reader can see that not only do we get the LSCENE JSON, we also get an explanation of the parts; supposedly how the LLM thinks about the parts and why it selected them. It should be noted that Gemini exhibited some laziness as it failed to complete the coordinates without a second agent pass that requested the full coordinates. This behavior is consistent across all LLMs (Table 1).

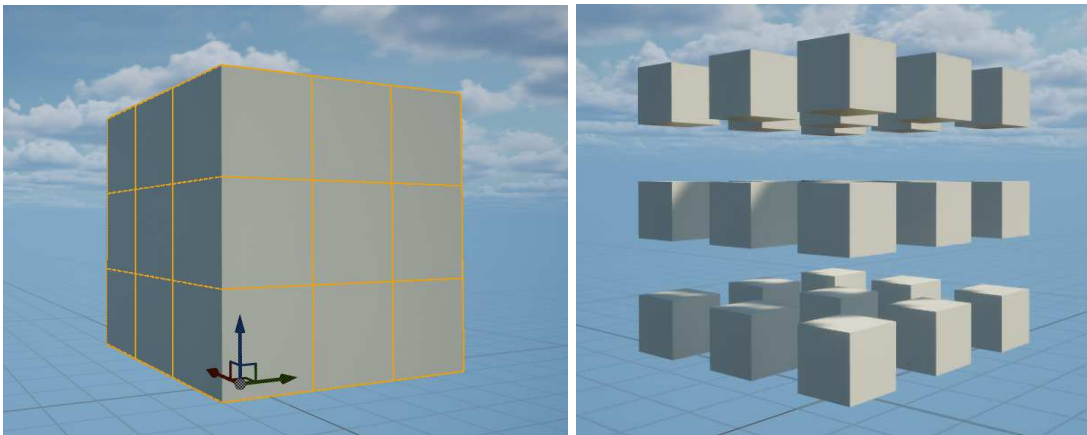


Figure 5: Gpt3.5 with configuration EXP:C for (left) unspaced grid versus edited extent (right).

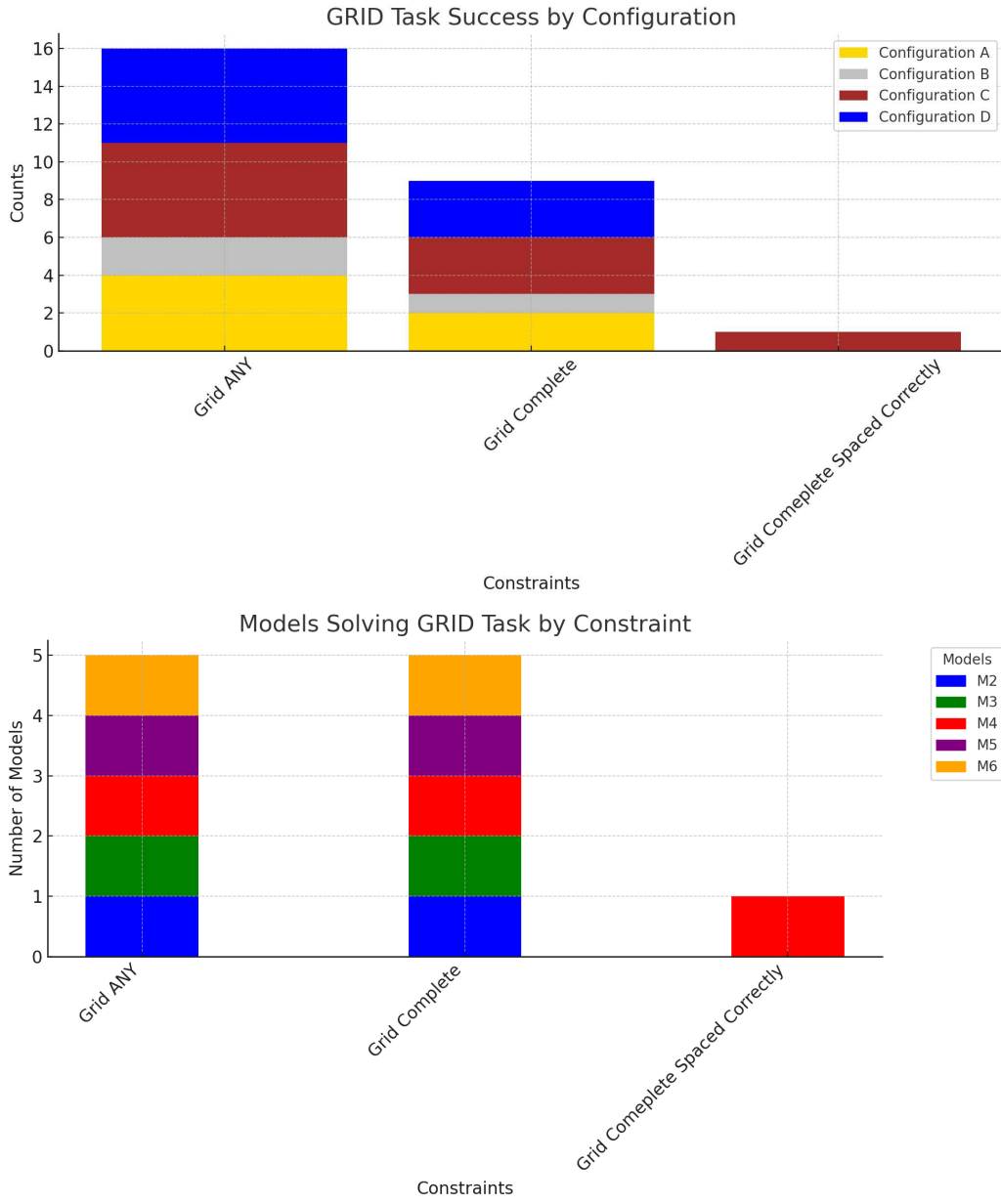


Figure 6: Grid tasks completion results across LLMs for Experiment 2.

LSCENE for Two Objects with Shape_Cube and Extents 10,10,10

This LSCENE creates two objects with the specified properties, placing one in a spiral shape and the other in a 3D grid, ensuring at least 10 units of separation between shapes.

```
{
  "LSCENE": {
    "MANY": {
      "Object": {
        "Spiral": {
          "Amount": 30,
          "A_Class": "Shape_Cube",
          "Extent": [10, 10, 10],
          "XYZ_Position": {
            "coords": [
              [0, 0, 0],
              [10, 0, 10],
              [20, 0, 20],
              [30, 0, 30],
              ... // Continue spiral coordinates with 10 unit increments
            ]
          }
        },
        "Grid": {
          "Amount": 30,
          "A_Class": "Shape_Cube",
          "Extent": [10, 10, 10],
          "XYZ_Position": {
            "coords": [
              [0, 0, 0],
              [10, 0, 0],
              [20, 0, 0],
              [0, 10, 0],
              [10, 10, 0],
              [20, 10, 0],
              ... // Continue grid coordinates with 10 unit increments
            ]
          }
        }
      }
    }
  }
}
```

 Use code with caution.

Json

Explanation:

- The **MANY** section indicates that we want to create multiple instances of an object.
- We define two objects within the **Object** section: "Spiral" and "Grid".
- Both objects have the following properties:
 - **Amount** : 30 instances of each object will be created.
 - **A_Class** : "Shape_Cube" specifies the shape of the object.
 - **Extent** : [10, 10, 10] defines the dimensions of the object.
- The **XYZ_Position** section defines the placement of the objects:
 - For the "Spiral" object, we use the **coords** function to specify coordinates that form a spiral pattern. You need to fill in the remaining coordinates with 10 unit increments in each dimension.
 - For the "Grid" object, we use the **coords** function to specify coordinates that form a 3D grid pattern. Similarly, fill in the remaining coordinates with 10 unit increments.

Figure 7: Example Gemini-1.5-pro labeled output, i.e., LSCENE with explanation.

Table 5: LLM LSCENE Generation for Spiral Placement (see Experiment 3).

Experiment Label	Spiral Any	Spiral CLOSE	Spiral Space Constraints	Syntax Error
M2	AC	x	x	D
M3	B	B	x	D
M4	A	x	x	BCD
M5	AD	A	x	B
M6	D	x	x	BC

5.3 Experiment 3: Spiral Placement

The spiral test is a step up in challenge compared to the grid generation task. While a 3D grid is a well-defined concept with little room for interpretation, a spiral presents a fuzzier or more ambiguous concept, particularly in 3D space. Gemini-1.5-pro managed to construct a spiral, as illustrated in Figure 8, albeit with moderate success. The figure captures the essence of the spiral, with curves looping around a central point. By testing LSCENE requests with abstract tasks like spirals, we can identify areas where an LLM may encounter difficulties. Overall results can be found in Table 5, with comparative analysis provided in Figure 10.

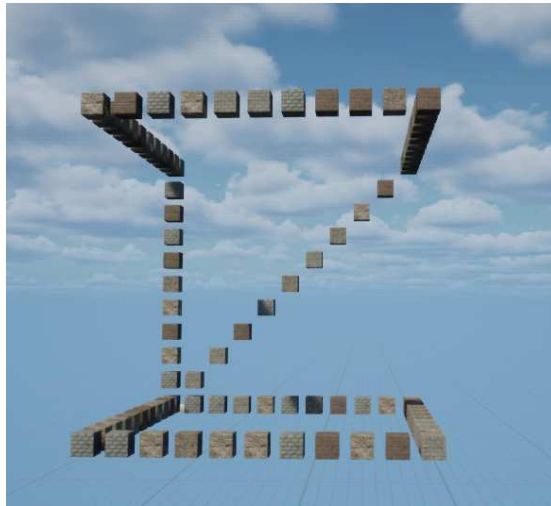


Figure 8: Gemini-1.5-pro spiral LSCENE generation attempt (see Experiment 3).

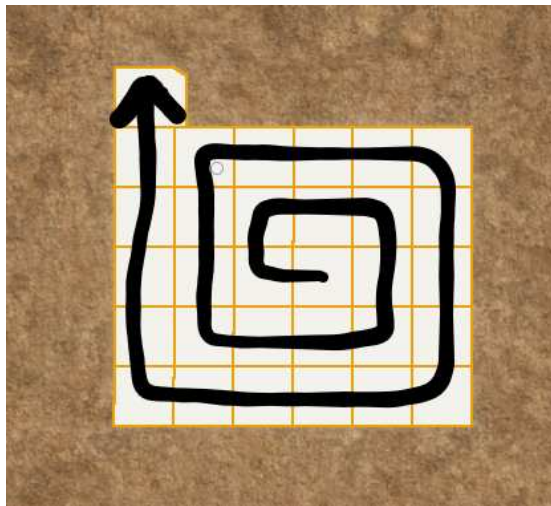


Figure 9: GPT-3.5 spiral attempt with EXP:B. The arrow is the order in which the spiral was created.

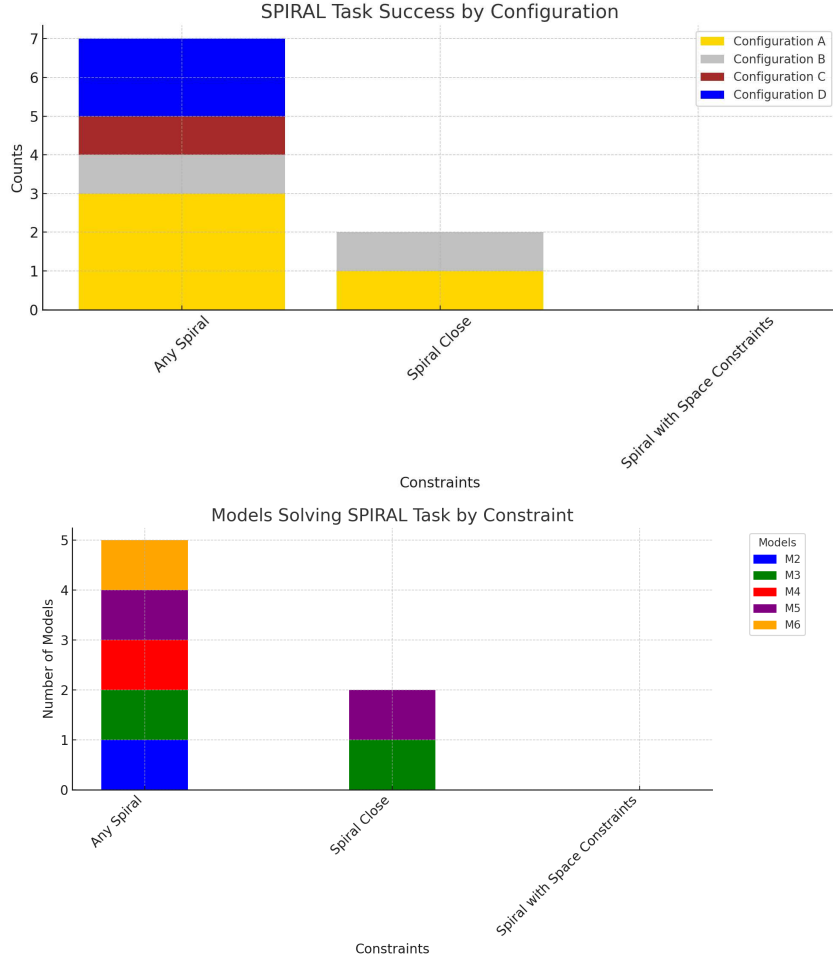


Figure 10: Spiral completion results across LLMs for Experiment 3.

5.4 Summary

The Anthropic Models demonstrated strong performance overall, with Claude-Opus completing the highest number of tasks, as depicted in Figure 12. However, it’s noteworthy that Claude-Opus also exhibited the most violations of the LSCENE syntax or general hallucinations. This raises the question: does this model tend to take risks on uncertain tasks, resulting in more correct responses overall? If we can filter out incorrect guesses while retaining accurate answers, this would arguably make it the most desirable model. Given that LSCENE can reject invalid or incorrect markup through parsing alone, receiving occasional invalid or incorrect LSCENE outputs isn’t a significant issue.

Upon analyzing the results, it’s apparent that all models performed similarly, as shown in Figure 13. However, a closer examination of how each model utilized the LLM configuration, as illustrated in Figure 11, reveals some interesting insights. For example, GPT-4 (M2) struggled to utilize the literal example (B) effectively, whereas GPT-3.5 (M3) made more frequent use of it. Similarly, GPT-4 tended to employ the more abstract general configuration (A) more often than GPT-3.5, a trend also observed in Claude-Haiku (M6). This suggests that models with less *creativity* tend to rely more on literal examples than on abstract definitions or examples. Notably, Claude-Opus (M4) excelled at making connections using configuration C, which combined elements from both A and B. In fact, it leveraged this additional information to accurately generate a 3D grid with proper spacing and the correct number of cubes, a feat unmatched by any other model.

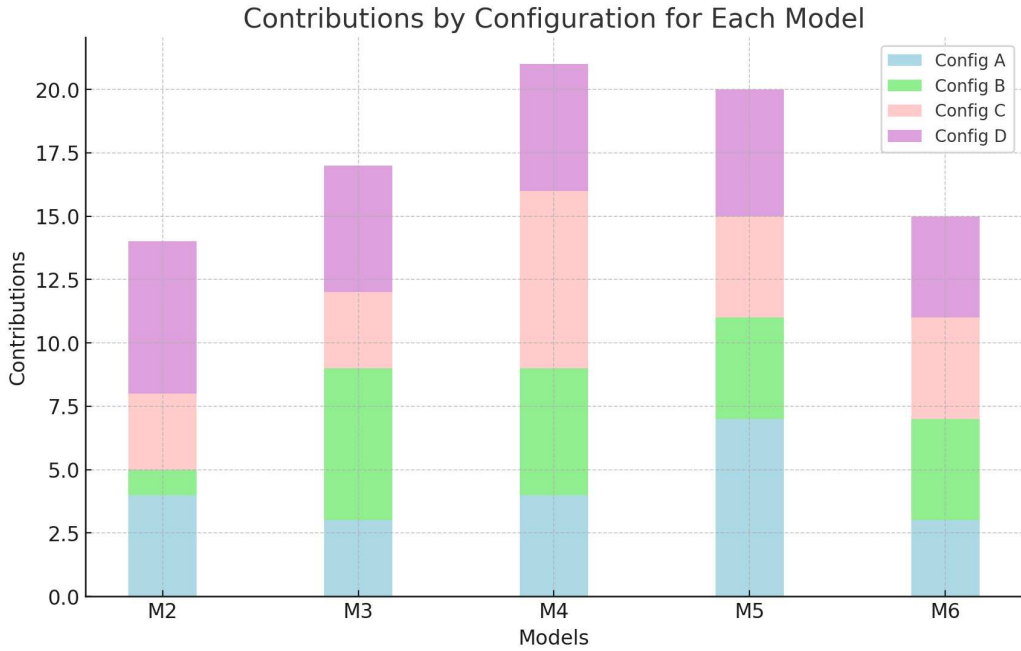
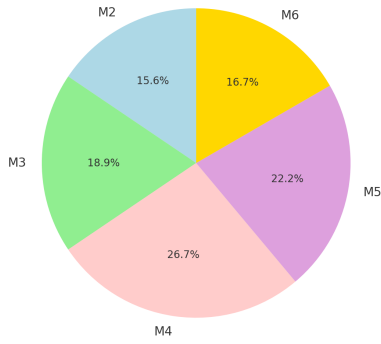


Figure 11: Final LLM task completions by configuration.

Percentage of Total Task Completions by Model (Lighter Colors)



Percentage of Syntax Errors by Model

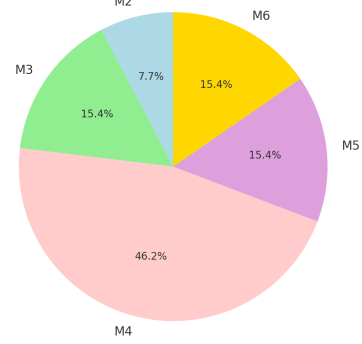
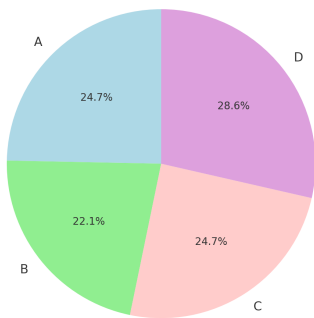


Figure 12: Final LLM task completions.

Percentage of Total Task Successes by Configuration



Percentage of Syntax Errors by Configuration

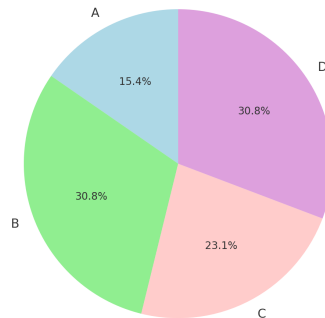


Figure 13: LLM configuration success vs error.

6. MORE COMPLEX LSCENE EXAMPLE

Experiments 1 through 3 helped us understand various LLM pros and cons relative to generating controlled targeted cases of relative spatial relations and syntactically correct LSCENE JSONs. Those experiments delved into the intricacies of prompt-to-output for each LLM. In this section, we simply aim to qualitatively observe an LLMs’ proficiency in generating more complex themed scenes, e.g., what we showed in Figure 2.

To this end, we utilize assets from the Quixel Megascan library. Specifically, the LLM is expected to accurately reference assets in the prompt, utilize these referenced assets correctly within the LSCENE, and sample from assets randomly based on the LSCENE description. To achieve this, we vary the temperature hyperparameter – a scale supported by most models in Table 1 – which affects the determinism of the LLM (with 0 being fully deterministic and 1.0 being most “creative”). More specifically, we assume than a user inputs a targeted biome, such as grassy or rocky and arid. The LLM must consider this prompt when selecting relevant assets. Assets are provided in name-only format, although advanced techniques may involve fully labeled descriptions for each asset; such as Quixel labels or other AI-generated labels.

Example 1 is a grassy biome, where the object of interest in this dataset is teddy bear. Meaning, this data could be used to train and/or evaluate an AI-based computer vision algorithm like object detection and localization or semantic segmentation. This data was generated with Claude-Opus at a temperature of 0.9, with a user request that asked for a grassy biome with man made clutter (2 wooden bears), with a list of paths to Uassets in the current UE project. Four frames were arbitrarily grabbed, see Figure 14.

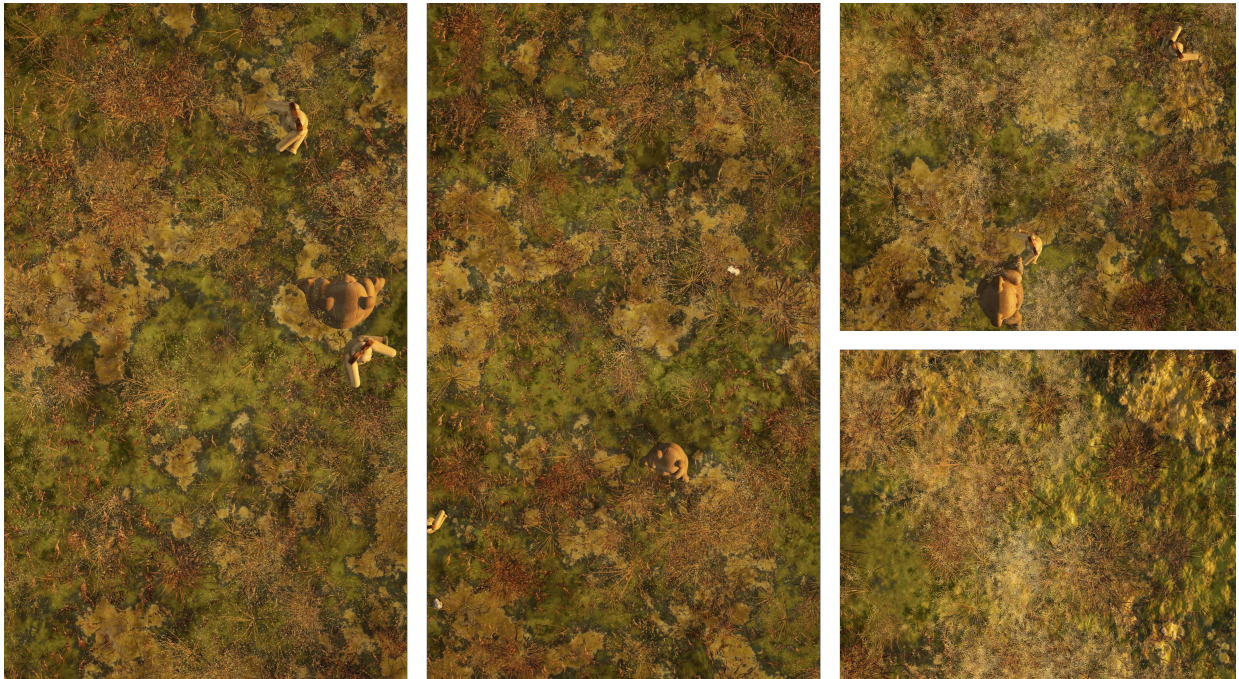


Figure 14: Example 1; Claude-Opus LSCENE with Configuration C, requesting a grassy biome, man made objects (teddy bear), and given a single asset pack.

Example 2, a rocky biome, was run under the same conditions, only without man made clutter requested (the teddy bear), but still referencing the same asset paths. Figures 15 and 16 show the results of Opus generating a rocky biome LSCENE. The controlled variability of each biome is apparent, and with a data collection of random camera poses and 1000s of frames to sample, the dataset becomes even more robust from a simple prompt. Furthermore, Opus was given the same assets with each request, so the ability to pick and choose assets corresponding to the prompt was achieved (while referencing in LSCENE).



Figure 15: Example 2, sample 1; Claude-Opus LSCENE with Configuration C, requesting a rocky biome and given a single asset pack.



Figure 16: Example 2, sample 2; Claude-Opus LSCENE with Configuration C, requesting a rocky biome and given a single asset pack.

7. CONCLUSION

In this article, we explored if large language models (LLM) can be used to translate user prompts into formal 3D scene descriptions for procedural synthesis of environments. Our focus centered on exploring different state-of-the-art LLMs and methodologies for interacting with these models to generate LSCENE descriptions and visualize them as datasets for computer vision using the Unreal Engine.

Through an example, we showcased the creation of different biomes—grassy versus rocky—leveraging high-quality photometrically scanned assets from the Quixel Megascans library. This analysis was predominantly qualitative. Additionally, we conducted three other experiments to quantitatively explore the performance of our approach relative to the spatial placement of objects. Our findings highlighted notable differences among various LLMs. For instance, it became evident that LLMs with lower levels of *creativity* tend to rely more on literal examples rather than abstract definitions or examples. Notably, Claude-Opus emerged as a standout performer, particularly when provided with generic LSCENE JSONs accompanied by documentation and examples featuring various use cases. However, we note that these are specific findings and the LLM field is evolving constantly. Additional research is required to discover more general philosophies of interacting with certain types of LLMs or targeted papers are needed, e.g., an optimal method for just Claude-Opus.

In summary, our exploration underscores the potential of current-generation LLMs in generating formal scene definitions, thereby facilitating the procedural generation of environments. This capability holds significant relevance across a spectrum of tasks, ranging from film and gaming to more pertinent applications such as generating controlled volume and diverse imagery for training and testing AI techniques.

8. FUTURE WORK

In conclusion, while this paper is an initial exploration of LLM-driven procedural generation, it highlights several areas ripe for improvement. Future research will expand our methodology beyond LSCENE to include generating LCAP JSONs for data capture. We also desire to develop more intricate scene (and capture) descriptions that consider contextual nuances. To date, our current focus has been on generating scenes with more random and diverse attributes, which aids in AI training and evaluation, particularly in computer vision contexts where generalization is the goal. However, enhancements are required to tackle more complicated scenarios effectively where context matters, e.g., cities, mazes, etc.

Moreover, our research was initiated with the aim of closing the loop on training and evaluating AI. Presently, we have one component of that puzzle in place. Future endeavors will be geared towards bridging the gap between these formal experiment definitions and formal language-based explanations of black box model performance. In our previous work,²⁵ Alvey et al. outlined a process for generating natural language-based descriptions of model performance. Our aim is to intertwine these two research strands, enabling the effective utilization of LLMs as search tools in both design and evaluation processes. Achieving this goal will necessitate the development of a summarizer to consolidate findings across experiments and a sampler to guide future experimentation.

Finally, numerous other exciting avenues await exploration. One possibility is leveraging LLMs not just for generating LSCENE descriptions, but also for generating the underlying sampler code, such as Unreal Engine C++ code. Additionally, future work may involve incorporating RAG or specific LLM agents to effectively handle large-scale asset databases. Another, and last, idea is to introduce additional features and constraints to enhance the expressiveness and controllability of the language, thus further expanding its capabilities.

REFERENCES

- [1] Perlin, K., “An image synthesizer,” *SIGGRAPH Comput. Graph.* **19**, 287–296 (jul 1985).
- [2] Kerley, J., Anderson, D. T., Alvey, B., and Buck, A., “How should simulated data be collected for AI/ML and unmanned aerial vehicles?,” in [*Synthetic Data for Artificial Intelligence and Machine Learning: Tools, Techniques, and Applications*], Howell, C. L., Manser, K. E., and Rao, R. M., eds., **12529**, 125290J, International Society for Optics and Photonics, SPIE (2023).

- [3] Kerley, J., Fuller, A., Kovaleski, M., Popescu, P., Alvey, B., Anderson, D. T., Buck, A., Keller, J. M., Scott, G., Yang, C., Yasuda, K. E., and Ryan, H. A., “Procedurally generated simulated datasets for aerial explosive hazard detection,” in [*Chemical, Biological, Radiological, Nuclear, and Explosives (CBRNE) Sensing XXIII*], Guicheteau, J. A. and Howle, C. R., eds., **12116**, 1211611, International Society for Optics and Photonics, SPIE (2022).
- [4] Perlin, K., “An image synthesizer,” in [*Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*], *SIGGRAPH '85*, 287–296, Association for Computing Machinery, New York, NY, USA (1985).
- [5] Church, E. M. and Semwal, S., “Simulating trees using fractals and l-systems,” (2006).
- [6] Prusinkiewicz, P., Hanan, J., Hammel, M., and Mech, R., “L-systems: from the theory to visual models of plants,” (2001).
- [7] Olsen, J., “Realtime procedural terrain generation,” (2004).
- [8] Kelly, G. and McCabe, H., “A survey of procedural techniques for city generation,” **14** (01 2006).
- [9] Booth, M., “The ai systems of left 4 dead,” in [*Artificial Intelligence and Interactive Digital Entertainment Conference*], Stanford University (2009).
- [10] Zhu, W., Liu, H., Dong, Q., Xu, J., Huang, S., Kong, L., Chen, J., and Li, L., “Multilingual machine translation with large language models: Empirical results and analysis,” (2023).
- [11] Jiao, W., Wang, W., tse Huang, J., Wang, X., Shi, S., and Tu, Z., “Is chatgpt a good translator? yes with gpt-4 as the engine,” (2023).
- [12] Briakou, E., Cherry, C., and Foster, G., “Searching for needles in a haystack: On the role of incidental bilingualism in palm’s translation capability,” (2023).
- [13] Bhattamishra, S., Ahuja, K., and Goyal, N., “On the Ability and Limitations of Transformers to Recognize Formal Languages,” in [*Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*], Webber, B., Cohn, T., He, Y., and Liu, Y., eds., 7096–7116, Association for Computational Linguistics, Online (Nov. 2020).
- [14] Sharma, M., “Exploring and improving the spatial reasoning abilities of large language models,” (2023).
- [15] Yamada, Y., Bao, Y., Lampinen, A. K., Kasai, J., and Yildirim, I., “Evaluating spatial understanding of large language models,” (2024).
- [16] OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Al-tenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., Bello, I., Berdine, J., Bernadett-Shapiro, G., Berner, C., Bogdonoff, L., Boiko, O., Boyd, M., Brakman, A.-L., Brockman, G., Brooks, T., Brundage, M., Button, K., Cai, T., Campbell, R., Cann, A., Carey, B., Carlson, C., Carmichael, R., Chan, B., Chang, C., Chantzis, F., Chen, D., Chen, S., Chen, R., Chen, J., Chen, M., Chess, B., Cho, C., Chu, C., Chung, H. W., Cummings, D., Currier, J., Dai, Y., Decareaux, C., Degry, T., Deutsch, N., Deville, D., Dhar, A., Dohan, D., Dowling, S., Dunning, S., Ecoffet, A., Eleti, A., Eloundou, T., Farhi, D., Fedus, L., Felix, N., Fishman, S. P., Forte, J., Fulford, I., Gao, L., Georges, E., Gibson, C., Goel, V., Gogineni, T., Goh, G., Gontijo-Lopes, R., Gordon, J., Grafstein, M., Gray, S., Greene, R., Gross, J., Gu, S. S., Guo, Y., Hallacy, C., Han, J., Harris, J., He, Y., Heaton, M., Heidecke, J., Hesse, C., Hickey, A., Hickey, W., Hoeschele, P., Houghton, B., Hsu, K., Hu, S., Hu, X., Huizinga, J., Jain, S., Jain, S., Jang, J., Jiang, A., Jiang, R., Jin, H., Jin, D., Jomoto, S., Jonn, B., Jun, H., Kaftan, T., Lukasz Kaiser, Kamali, A., Kanitscheider, I., Keskar, N. S., Khan, T., Kilpatrick, L., Kim, J. W., Kim, C., Kim, Y., Kirchner, J. H., Kiros, J., Knight, M., Kokotajlo, D., Lukasz Kondraciuk, Kondrich, A., Konstantinidis, A., Kosic, K., Krueger, G., Kuo, V., Lampe, M., Lan, I., Lee, T., Leike, J., Leung, J., Levy, D., Li, C. M., Lim, R., Lin, M., Lin, S., Litwin, M., Lopez, T., Lowe, R., Lue, P., Makanju, A., Malfacini, K., Manning, S., Markov, T., Markovski, Y., Martin, B., Mayer, K., Mayne, A., McGrew, B., McKinney, S. M., McLeavey, C., McMillan, P., McNeil, J., Medina, D., Mehta, A., Menick, J., Metz, L., Mishchenko, A., Mishkin, P., Monaco, V., Morikawa, E., Mossing, D., Mu, T., Murati, M., Murk, O., Mély, D., Nair, A., Nakano, R., Nayak, R., Neelakantan, A., Ngo, R., Noh, H., Ouyang, L., O’Keefe, C., Pachocki, J., Paino, A., Palermo, J., Pantuliano, A., Parascandolo, G., Parish, J., Parparita, E., Passos, A., Pavlov, M., Peng, A., Perelman, A., de Avila Belbute Peres, F., Petrov, M., de Oliveira Pinto, H. P., Michael, Pokorny, Pokrass, M., Pong, V. H., Powell, T., Power, A., Power, B., Proehl, E., Puri, R., Radford,

- A., Rae, J., Ramesh, A., Raymond, C., Real, F., Rimbach, K., Ross, C., Rotsted, B., Roussez, H., Ryder, N., Saltarelli, M., Sanders, T., Santurkar, S., Sastry, G., Schmidt, H., Schnurr, D., Schulman, J., Selsam, D., Sheppard, K., Sherbakov, T., Shieh, J., Shoker, S., Shyam, P., Sidor, S., Sigler, E., Simens, M., Sitkin, J., Slama, K., Sohl, I., Sokolowsky, B., Song, Y., Staudacher, N., Such, F. P., Summers, N., Sutskever, I., Tang, J., Tezak, N., Thompson, M. B., Tillet, P., Tootoonchian, A., Tseng, E., Tuggle, P., Turley, N., Tworek, J., Uribe, J. F. C., Vallone, A., Vijayvergiya, A., Voss, C., Wainwright, C., Wang, J. J., Wang, A., Wang, B., Ward, J., Wei, J., Weinmann, C., Welihinda, A., Welinder, P., Weng, J., Weng, L., Wiethoff, M., Willner, D., Winter, C., Wolrich, S., Wong, H., Workman, L., Wu, S., Wu, J., Wu, M., Xiao, K., Xu, T., Yoo, S., Yu, K., Yuan, Q., Zaremba, W., Zellers, R., Zhang, C., Zhang, M., Zhao, S., Zheng, T., Zhuang, J., Zhuk, W., and Zoph, B., “Gpt-4 technical report,” (2024).
- [17] Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J., and rong Wen, J., “A survey of large language models,” *ArXiv* **abs/2303.18223** (2023).
- [18] Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro, M. T., and Zhang, Y., “Sparks of artificial general intelligence: Early experiments with gpt-4,” (2023).
- [19] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., and Wang, H., “Retrieval-augmented generation for large language models: A survey,” (2024).
- [20] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D., “Retrieval-augmented generation for knowledge-intensive nlp tasks,” (2021).
- [21] Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., and Chadha, A., “A systematic survey of prompt engineering in large language models: Techniques and applications,” (2024).
- [22] Wei, J., Wang, X., Schuurmans, D., Bosma, M., hsin Chi, E. H., Xia, F., Le, Q., and Zhou, D., “Chain of thought prompting elicits reasoning in large language models,” *ArXiv* **abs/2201.11903** (2022).
- [23] Nori, H., Lee, Y. T., Zhang, S., Carignan, D., Edgar, R., Fusi, N., King, N., Larson, J., Li, Y., Liu, W., Luo, R., McKinney, S. M., Ness, R. O., Poon, H., Qin, T., Usuyama, N., White, C., and Horvitz, E., “Can generalist foundation models outcompete special-purpose tuning? case study in medicine,” (2023).
- [24] Liu, B., Ash, J. T., Goel, S., Krishnamurthy, A., and Zhang, C., “Transformers learn shortcuts to automata,” (2023).
- [25] Alvey, B., Anderson, D., Keller, J., and Buck, A., “Linguistic explanations of black box deep learning detectors on simulated aerial drone imagery,” *Sensors* **23**(15) (2023).