# Real-Time World Crafting: Generating Structured Game Behaviors from Natural Language with Large Language Models

#### Austin Drake\*

University of Exeter Department of Computer Science austin.drake.343@gmail.com

## **Hang Dong**

University of Exeter
Department of Computer Science
h.dong2@exeter.ac.uk

#### **Abstract**

We present a novel architecture for safely integrating Large Language Models (LLMs) into interactive game engines, allowing players to "program" new behaviors using natural language. Our framework mitigates risks by using an LLM to translate commands into a constrained Domain-Specific Language (DSL), which configures a custom Entity-Component-System (ECS) at runtime. We evaluated this system in a 2D spell-crafting game prototype by experimentally assessing models from the Gemini, GPT, and Claude families with various prompting strategies. A validated LLM judge qualitatively rated the outputs, showing that while larger models better captured creative intent, the optimal prompting strategy is task-dependent: Chain-of-Thought improved creative alignment, while few-shot examples were necessary to generate more complex DSL scripts. This work offers a validated LLM-ECS pattern for emergent gameplay and a quantitative performance comparison for developers.

#### 1 Introduction

A long-standing goal in Human-Computer Interaction (HCI) and in interactive art forms is to allow users to control complex simulations in real-time using only natural language, which requires translating between ambiguous human intent and the rigid logic of computer systems (Manaris, 1998). While Large Language Models (LLMs) offer a promising approach to this problem, allowing an LLM to generate and execute arbitrary code in a general-purpose language poses significant risks to system stability and security (Majumdar and Vogelsang, 2024). To mitigate these risks, we propose a novel software architecture that uses an LLM to interpret natural language commands into a constrained, human-readable Domain-Specific Language (DSL). This intermediate layer provides

a safe and verifiable instruction set for a video game simulation, addressing a gap in research on the real-time translation of creative instructions and producing physically plausible behaviors in interactive systems. We use a 2D physics-based video game as a testbed to evaluate this architecture.

While many games use LLMs for narrative generation, our work focuses on driving more realized, physically-based systems. Our proposed architecture relies on the Entity-Component-System (ECS) pattern (Romeo, 2016; Wiebusch and Latoschik, 2015), whose flexible, compositional design is well-suited for dynamically modifying object behaviors at runtime based on non-deterministic LLM outputs. Our prototype, *Latent Space*, features a spell-crafting system and a cellular automata sandbox to test the LLM's ability to generate both compositional and logic-based DSL code.

This paper investigates how effectively LLMs can translate creative commands into structured, executable instructions. We experimentally evaluate the performance of models from the Gemini, GPT, and Claude families with various prompt engineering strategies to determine which factors most influence the generation of logically consistent and physically plausible game mechanics. Our contributions include: (1) a validated, DSL-mediated architecture for safely integrating LLMs into interactive systems; (2) a quantitative comparison of model performance on creative and logical generation tasks; (3) an analysis of prompting strategies for maximizing creative throughput in languagedriven interfaces; and (4) an evaluation of the architecture, including both human and automated LLM-as-judge ratings. Our demo, experiment implementation, and generated data are available as a GitHub repository<sup>1</sup>.

<sup>\*</sup>Corresponding Author

Ihttps://github.com/austin-the-drake/
real-time-world-crafting-wordplay-demo

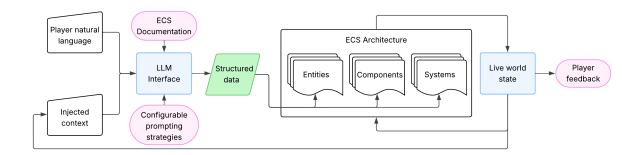


Figure 1: A flow diagram of the developed architecture. The LLM interface (left) handles translating Natural Language (NL) into DSL code. The ECS (center) consists of data output by the LLM and logical systems operating on that data. The I/O container (outside edges) is the mode of content delivery and interaction for the player.

## 2 Background

## 2.1 Historical Challenges in Games

Natural language interfaces in interactive systems and art forms have historically been limited by the challenges of linguistic ambiguity and the prohibitive authoring cost required to support widereaching player freedom. Early parser-based systems such as Zork (Lebling et al., 1979), while revolutionary, were also very rigid, and frustratingly rejected any inputs that did not exactly match their pre-programmed vocabularies. Later NLP-driven experiences including Façade (Mateas and Stern, 2003, 2005) addressed this problem by mapping free-form user input to the closest available prewritten options, but this comes with an immense workload of manually authoring responses to maintain the illusion of natural conversation. These challenges have led modern games to often rely on systems such as "dialogue wheel", which restrict player input to lessen the authoring burden at the cost of player expressiveness (Taylor-Giles, 2020).

#### 2.2 LLM-Driven Architectures

LLMs can simultaneously interpret ambiguous text and generate novel content, which presents a potential solution to both historical problems at once. Contemporary research is moving from using models to select from pre-written options (as in *Façade*) towards generating fully dynamic outcomes in realtime (Gallotta et al., 2024). Applications range from generating dynamic narratives in co-creative storytelling and role-playing games (Sun et al., 2023; Sanders, 2024; Triyason, 2023) to driving complex agent behaviors in simulations and strategy games (Park et al., 2023; Yim et al., 2024; Ito and Takahashi, 2024). A promising direction for LLMs is to support natural language as a free-form

scripting interface for game mechanics. Several existing projects constrain player input to make this task more tractable (BigFrogg, 2025); However, this project investigates the challenge of translating fully unconstrained language into game logic.

Architectural inspiration for this task can be drawn from domains like robotics and data science (Liu et al., 2024; Li et al., 2025a; Sah et al., 2024). A common pattern involves an LLM translating a high-level command into a structured, machine-readable representation (e.g., API calls, JSON), which is then executed by a deterministic back-end (Perrault and Grosz, 1988; Sah et al., 2024). Robotics frameworks like Interactive Task Planning (ITP) use this to break abstract goals down into concrete actions or update plans based on new context (Li et al., 2025a). A major focus is grounding the LLM's output in the state and capabilities of the target system (Bernardi et al., 2025).

# 3 System Architecture and Implementation

Our demo, *Latent Space*, is designed to translate natural language into executable game mechanics. The primary design challenge was to safely and reliably use the non-deterministic output of an LLM within a deterministic game engine. To achieve this, we developed a modular architecture (Figure 1) that sandboxes the LLM's influence. Instead of generating general-purpose code, which is unconstrained and error-prone, the LLM's role is strictly limited to generating structured data in a custom JSON-based DSL.

The architecture's layers are: (1) an LLM interface that translates a player's request into the DSL; (2) a custom ECS framework that parses the DSL to instantiate and configure the correspond-

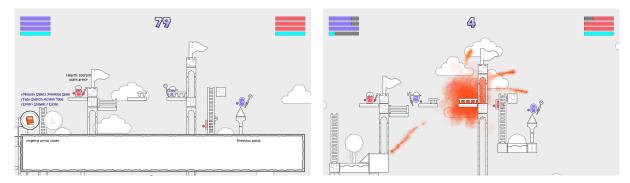


Figure 2: A pair of screenshots from *Latent Space's* Battle Mode. Natural language magic descriptions, e.g., "Singeing arrow volley," entered into the text interface (left), can have explosive consequences (right). Each team's remaining players and their health (red/purple) are visible above, as is the amount of time remaining for each turn.

ing game entities and their components; and (3) a commercial game engine that acts as an I/O container, rendering the ECS state and capturing player input. This separation of concerns improves system stability and also allows the LLM's translation performance to be tested in isolation.

#### 3.1 Prototype and DSL Design

Latent Space is designed to provide players with unconstrained, free-form natural language input, in contrast to systems that rely on combining predefined elements (BigFrogg, 2025). The prototype features two distinct game modes to evaluate the LLM's reasoning on different types of tasks. Each mode requires its own DSL; both of which are represented as JSON, behaving like an abstract grammar (Wang et al., 2023) that simplifies parsing.

Battle Mode (Figure 2) is a turn-based artillery game where players interact by simply describing magical spells. This mode tests the LLM's ability to perform creative, semantic translation, mapping a user's description to a set of components in our compositional spell DSL. This DSL defines spells as unordered collections of data components (e.g., projectile, element) and event triggers that can embed other spell definitions.

Alchemy Mode (Figure 3) is a cellular automata sandbox modeled after *The Powder Toy* (Skowronek, 2008) and *Sandspiel* (Bittker, 2023). This mode tests the LLM's ability to generate complex, ordered logical rules. Players describe new materials, and the LLM must generate a valid ruleset in our procedural cellular automata DSL. This DSL uses a nested structure of conditional checks to define grid-based interactions where the order of operations is critical.

#### 3.2 Grounding and Robustness

To ensure system stability, we utilize a DSL as an intermediate layer, which constrains the LLM's output to a set of pre-validated operations, reducing the risks associated with arbitrary code generation (Majumdar and Vogelsang, 2024). Since a pre-trained LLM has no knowledge of a novel DSL, this presents an out-of-distribution problem (Yuan et al., 2023), which can lead to poor performance and hallucinated function calls (Joel et al., 2024). Rather than extensive fine-tuning, we use prompt-based knowledge injection, providing the DSL documentation to the model in prompts to ground its output, applying recommendations from literature on DSL code generation (Gu et al., 2025).

We employ several prompt engineering techniques to ensure the generated DSL code is syntactically and logically sound. To ground the LLM's output in the current game state, each API request is extended with dynamic context. For the spell DSL, this includes a list of currently active magical elements, while for the more complex automata DSL, the prompt includes the full DSL representation of all existing materials. We also use few-shot examples and Chain-of-Thought (CoT) prompting, where the LLM first generates a plan mapping user phrases to DSL components before producing the final JSON (Wei et al., 2022; Li et al., 2025b, 2024). Our system's backend follows the ECS pattern, a compositional design where an object's behavior is defined by its data components rather than a fixed class hierarchy (Romeo, 2016; Cheng et al., 2023). In this model, an Entity is a unique identifier for an object, which holds various Components, consisting of raw data chunks such as position or health. Systems then contain the logic, iterating over entities that possess a specific set of compo-

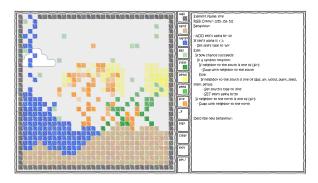


Figure 3: A screenshot from *Latent Space's* Alchemy Mode. Natural language material descriptions are translated into actionable cellular automata rules (right) by the LLM. Players can observe the generated behaviors in the workspace (left).

nents to apply behaviors such as movement. This separation of data and logic provides the runtime flexibility necessary to dynamically alter game objects by modifying their data components, which is done simply by following the instructions contained in the generated DSL scripts, making it ideal for encouraging emergent gameplay (Wiebusch and Latoschik, 2015; Hatledal et al., 2021). The prompts for spell and automata scripting are available in Appendix A.1 and A.2, respectively.

To handle the non-deterministic nature of LLMs, the system also includes several error-handling measures. A validation layer checks all generated JSON for syntactic correctness, valid component types, and in-range parameter values. The parser strips extraneous text, and for minor errors such as a missing parameter, the system applies default values. If a script is completely unusable, the game defaults to a harmless "fizzle" effect, providing feedback to the player without crashing the game.

## 4 Experimental Methodology

We evaluated our framework through two primary experiments, using a combination of quantitative algorithmic metrics and automated qualitative ratings from a validated LLM judge. To support this, we generated a corpus of 2600 DSL scripts comprised of three distinct subsets, described below.

**Test Data Generation** First, a naturalistic set (N=2400) was created to test the models' ability to reason with novel concepts that may not have a perfect correspondence to the available DSL components, as would be expected during real gameplay. This set was generated by prompting the *Gemini* 2.5 *Pro* model (Comanici et al., 2025) to create

100 unique, creative task descriptions, which were then used as inputs for the main experiment. Secondly, a bidirectional set (N=120) was created to test information preservation. It consists of 30 random, procedurally-generated "source" scripts and 120 corresponding natural language descriptions of varying styles (a creative narrative vs. a technical document) and lengths (summary length vs detailed report), which were also generated by prompting the Gemini 2.5 Pro model. Finally, a handcrafted ground-truth set (N=80) of expert-authored "good" (well-formed and effective) and "bad" (syntactically valid but logically flawed) scripts was created to rigorously validate the LLM judge. The latter scripts were created by randomly replacing components or by shortening the former. These consisted of 50 spell and 30 cellular automata examples, for a total of 25 "good" and 25 "bad" spell scripts, and 15 "good" and 15 "bad" automata scripts.

#### 4.1 Experiment 1: NL-to-DSL Translation

This experiment assessed the main translation pipeline using the naturalistic test set in a 4×3×2 fully-crossed, repeated-measures design. We tested three factors: Model (*Gemini 2.5 Flash, GPT-4.1 mini, Claude 4 Sonnet*, and *Gemma 3 4B* as a baseline); Shot Strategy (zero-, one-, and few-shot); and Prompting Technique (standard vs. Chain-of-Thought). Outcome measures included the Average Success Rate (ASR) of generating syntactically valid DSL and, for all valid scripts, automated qualitative ratings from our LLM judge.

#### 4.2 Experiment 2: Bidirectional Translation

This experiment evaluated how well a user's creative intent is preserved in a "round-trip" translation, simulating the process of a player articulating a mental concept. Using the bidirectional set, an LLM had to recreate an original, procedurallygenerated DSL script using only its corresponding natural language description. Description length and style were compared in a 2x2 factorial design. We assessed the difference between the original and final scripts using two algorithmic similarity measures: (1) tree edit distance, computed between the Abstract Syntax Trees (ASTs) of the scripts using the APTED algorithm to quantify structural changes (Pawlik and Augsten, 2016, 2015), and (2) Jaccard similarity, calculated as the intersectionover-union of component names in the scripts, to measure the overlap in pure semantic content. This experiment was limited to the spell DSL.

```
{
friendlyName: "Wind scout",
count: 1,
components: [
{componentType: "projectile",
    radius: 2, speed: 15, gravity: 0},
{componentType: "element",
    element: "wind"},
{componentType: "controllable",
    mana_cost: 0.1},
{componentType: "buttonTrigger",
    payload_components: [
    {componentType: "teleportCaster"}
    ]}
]}
```

```
{
name: "gas",
color_hex: "#CCCCCC",
behavior: {
   actions: [{
     type: "in_rand_rotation",
     actions: [{
       type: "if_neighbor_is",
       direction: "south",
       options: ["air"],
       actions: [{
       type: "do_swap",
       direction: "south"
       }],
...
```

Figure 4: An example of the magical spell scripting DSL (left) and the cellular automata DSL (right). In natural language, these are "A controllable wind pixie that warps me when I call," and "a cloudy gas that diffuses randomly."

#### 4.3 Automated Qualitative Assessment

Because large-scale human evaluation was infeasible, we used an automated LLM judge based on *GPT 4.1* to provide qualitative ratings at scale, justifying its use with a rigorous validation process. For all valid outputs, the judge assigned 1-5 Likert scores for the 4 criteria described below.

- (i) Creative Alignment: This scale assesses how well the generated output captured the user's core creative intent, with an emphasis on theme.
- (ii) Instruction Following: This scale measures how accurately the output adhered to specific, direct commands or constraints in player requests.
- (iii) Emergence: Measures whether the output surpassed the user's request in a positive and unexpected way, such as environmental interaction.
- (iv) Structural Coherence: Evaluates adherence to the DSL's logical rules. This was designed to provide a more comprehensive measure after basic syntactic validity was found to be very high.

The LLM judge (*GPT 4.1*) was instructed to produce a textual evaluation of each script before assigning a numerical score, a form of Chain-of-Thought prompting used to improve reliability (Wei et al., 2022). To protect against a common leniency bias in LLM judges, a large model and careful prompting were used in accordance with established guidelines (Thakur et al., 2025). The prompt for the LLM judge is available in Appendix A.3.

**Validating the LLM judge** The judge was validated against the handcrafted ground-truth dataset (N=80) as introduced earlier in the Section. A

series of paired Wilcoxon Signed-Rank tests confirmed it could reliably distinguish between "good" and "bad" examples. Scores for "good" items were statistically significantly higher across all criteria: Creative Alignment (V=147, p=0.00058), Instruction Following (V=166, p=0.00037), Emergence (V=271, p<.0001), and Structural Coherence (V=195.5, p<.00057). The judge also had strong classification performance; for the 50 spell scripts, maximum F1 scores were 0.77, 0.81, 0.90, and 0.80, and Area Under the Curve (AUC) values were 0.82, 0.85, 0.92, and 0.82, respectively, on the four criteria. For automata scripts, F1 scores were 0.83, 0.93, 0.80, and 0.77, and AUC values were 0.88, 0.96, 0.83, and 0.70. This accuracy demonstrates alignment with a human-defined quality standard.

To test for consistency, a second, equivalentlycapable LLM (Gemini 2.5 Pro) rated the same outputs to calculate inter-rater reliability using Spearman's rank-order correlation ( $\rho$ ), quadratically weighted Cohen's Kappa ( $\kappa_w$ ), and a two-way Intraclass Correlation Coefficient (ICC). Agreement was moderate to substantial; For magical spells, Spearman's  $\rho$  ranged from .59 to .70 across scales,  $\kappa_w$  from .56 to .64, and ICC from .56 to .65. For cellular automata, agreement was higher ( $\rho$  from .76 to .81,  $\kappa_w$  from .74 to .77, and ICC from .75 to .77), with the exception of Structural Coherence, which showed lower agreement ( $\rho$ =.36,  $\kappa_w$ =.23, ICC=.24). The full metrics are available in Appendix B (Table 4). Substantial agreement indicates the criteria are objective enough to be scored consistently, reducing the concern of single-model bias. However, we acknowledge self-enhancement bias (Zheng et al., 2023) as a potential limitation and revisit it in the discussion section.

Table 1: Percentage of Success (ASR, score in %) Across Models and Prompting Strategies

	Zero-Shot		One-Shot		Few-Shot	
Model	Standard	СоТ	Standard	СоТ	Standard	СоТ
Magical Spells						
Claude 4 Sonnet	100	100	100	100	100	100
Gemini 2.5 Flash	96	100	100	98	100	100
Gemma 3 (4B)	100	98	100	100	100	100
GPT-4.1 Mini	100	100	100	100	100	100
Cellular Automata						
Claude 4 Sonnet	100	100	100	100	98	100
Gemini 2.5 Flash	82	92	100	100	94	98
Gemma 3 (4B)	92	76	98	100	100	100
GPT-4.1 Mini	90	88	98	98	100	100

#### 5 Experimental Results

The overall reliability of the translation pipeline, measured by Average Success Rate (ASR), was high. For the compositional spell DSL, success was nearly perfect, whereas the procedural automata DSL proved slightly more challenging (Table 1). This high syntactic validity can be attributed to both the maturity of modern LLMs and the pipeline's error-handling features, which salvaged most flawed outputs. However, this large ceiling effect means that a simple pass/fail metric is limited in utility compared to the qualitative measures.

#### 5.1 NL-to-DSL Translation

An ANOVA revealed that model choice was the strongest predictor of performance (p<.001 on all outcomes), as shown in Table 3 in Appendix A. Post-hoc tests showed that for the creative spell task, Claude 4 Sonnet significantly outperformed all other models on the automated qualitative ratings. For the logical automata task, this gap narrowed, but Claude 4 Sonnet still significantly outperformed other models on most outcomes. The small baseline model, Gemma 3 (4B), performed significantly worse than all larger models across both tasks. The effect of prompting strategy was also task-dependent. For the compositional spell task, in-context examples had no significant effect. For the procedural automata task, however, few-shot and one-shot prompting significantly improved all scores over zero-shot. Chain-of-Thought (CoT) prompting was a more generally effective strategy, significantly improving Creative Alignment and Emergence for both DSLs. We also observed several significant interaction effects (Figure 6 in Appendix A), showing that the most effective strategy is highly model-dependent.

#### 5.2 Bidirectional Translation

To assess how well a user's creative intent is preserved, this experiment performed a round-trip translation from a DSL script to natural language and back again. We specifically investigated how the style (narrative vs. technical) and length (summary vs. detailed) of the natural language description impacted the accuracy of the final, regenerated script, using a linear mixed-effect model to handle the mixture of discrete and continuous variables and 2x2 factorial design.

The linear mixed-effect model results for the bidirectional experiment are shown in Table 2, and visualized in Figure 5. The most influential factor was the style of the natural language description; Technical descriptions resulted in significantly better information preservation than creative narrative descriptions, with scripts that were both more structurally similar (lower tree edit distance, p < .001) and had greater semantic overlap (higher Jaccard Similarity, p < .001) with the originals. The complexity of the original script was also a significant factor; scripts with greater nesting complexity were harder to reproduce, resulting in a higher tree edit distance (p<.001). The length of the description was observed to influence semantic content, as short descriptions resulted in significantly lower Jaccard Similarity (p <.001). A significant interaction effect for tree edit distance (p <.001) suggests that the combination of a summary description and a narrative style was particularly detrimental. These results suggest that detailed and technically-phrased inputs are most effective for accurately translating a user's creative intent into a functional DSL script, which may run counter to most players' natural speaking or writing styles.

Table 2: Linear Mixed-Model Results for the bidirectional translation experiment

	Tree Edit Distance			Jaccard Similarity			
Predictor	Estimate	SE	p	Estimate	SE	p	
Fixed Effects							
(Intercept)	4.34	2.19	.056	0.36	0.05	<.001‡	
Description Style (Technical)	-7.40	1.10	<.001‡	0.13	0.02	<.001‡	
Description Detail (Summary)	0.87	1.10	.433	-0.11	0.02	<.001‡	
Component Complexity	-0.31	0.44	.485	-0.00	0.01	.854	
Nesting Complexity	1.86	0.45	<.001‡	0.02	0.01	.096	
Style × Detail Interaction	5.60	1.56	<.001‡	-0.03	0.03	.305	
Random Effects							
••	Variance	Std. Dev.		Variance	Std. Dev.		
Procedural Code (Intercept)	6.85	2.62		0.005	0.070		
Residual	18.15	4.26		0.009	0.092		

*Note.* Significance codes: \* p < .05, † p < .01, ‡ p < .001. Intercept contains Style (Narrative) and Detail (Detailed)

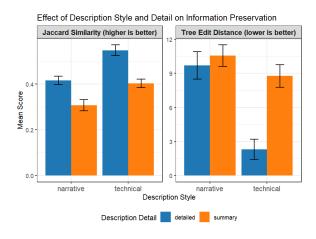


Figure 5: Clustered bar chart for the bidirectional translation experiment. Left: Jaccard Similarity (higher is better). Right: Tree Edit Distance (lower is better).

#### 5.3 Auxiliary Findings

Comparing our data generation methods, we found that scripts generated from grounded, procedurally-based inputs scored significantly higher on Creative Alignment (W=16968, p<.001, d=1.06), Instruction Following (W=16766, p<.001, d=1.00), and Emergence (W=24681, p<.001, d=0.55). Interestingly, the naturalistic inputs scored significantly higher on structural coherence (W=41737, p<.001, d=-0.58). We also found a clear trade-off between model performance and inference latency, with the fastest model (*Gemma 3 4B*, M=3.35s) producing the lowest quality outputs and the slowest (*Gemini 2.5 Flash*, M=12.2s) performing better.

## 6 Human Pilot Study

To complement and extend the automated evaluation, we conducted a small-scale pilot study with 6 human participants. The study was designed to capture a holistic player experience, which means there are significant methodological differences between the LLM judge's analysis of static code and the players' live play. Participants interacted with the prototype without knowledge of the underlying DSL and possessed genuine creative intent, whereas the LLM judge was provided with full documentation and could only infer intent from the provided input text.

Participants rated the generated mechanics on the same 1-5 scales for Creative Alignment, Instruction Following, and Emergence as the LLM judge. Ratings were generally high (median scores of 4, 4, and 3, respectively). We found a weak but statistically significant positive correlation between human and LLM ratings for Creative Alignment ( $\rho$ =.37, p<.01) and Instruction Following ( $\rho$ =.34, p<.01). This suggests that the programmatic qualities assessed by the LLM have a measurable, though modest, overlap with the subjective experience of a human player.

In contrast, no significant correlation was found for Emergence, implying that a player's subjective feeling of surprise is distinct from the judge model's technical measure of novelty. Qualitative feedback showed that participants were often more lenient than the LLM judge, expressing enjoyment and surprise at the creative outcomes in the game, sometimes despite subpar interpretations of their inputs. However, some also described difficulty in determining the underlying capabilities of the system and what kind of requests they could expect it to handle. These findings indicate that while the LLM judge is a useful proxy for output quality, it cannot fully capture the subjective, interactive nature of human play.

#### 7 Discussion

Our results indicate that the proposed LLM-ECS architecture is a promising method for translating natural language into executable game mechanics; Using an intermediate DSL proved effective for maintaining system stability by constraining the LLM to generate structured data. The architecture does not entirely eliminate the authoring burden, but changes the type of work necessary; developers must move from creating exponential content (e.g., every possible spell) to designing an expressive underlying system (the DSL and its components).

The choice of LLM was the most significant predictor of quality. Our findings show that a certain competence threshold, met by large state-of-the-art models, is required to handle the creative and logical reasoning for a novel DSL without fine-tuning, while the best prompting strategy depended on the structure of the target DSL. The procedural automata DSL, with its strict ordering, required fewshot examples to generate coherent scripts, whereas the more forgiving compositional spell DSL benefited more from a Chain-of-Thought reasoning step. This suggests that an important design consideration is that the nature of the task and structure of the DSL should inform the prompting strategy.

Potential for Emergent Gameplay One of the primary goals of this work was to develop a system that could enable dynamic and emergent gameplay, and a finding after implementation is that the nature of this emergence is closely tied to the level of abstraction that the LLM is permitted to operate. In the ECS-driven Battle Mode, novelty arose from surprising combinations of high-level, preauthored capabilities. Instead of eliminating the historical authoring burden problem entirely, the developer's task moves from scripting every possible outcome to engineering a flexible and expressive set of foundational components. The resulting gameplay, while novel, is therefore constrained by the creative limits of that initial ECS component design, which requires consideration.

In contrast, Alchemy Mode demonstrated a more classic, "bottom-up" form of emergent gameplay by tasking the LLM with generating simple, local rules for material interactions from first principles. This approach produced complex and unpredictable large-scale patterns that were not present in the system's low-level parts. The relative success of this mode was likely driven by two key factors: a more granular, low-level DSL, and the

extensive contextual grounding the model received on the simulation's current state. This implies that to encourage more complex, systemic emergence, a system must provide the model with enough detailed context and control over the most foundational building blocks of the simulation. This would shift designer's role from content creation into architecting a sufficiently expressive and robust environment, where the player can construct new high level rules through the LLM system.

**Input Style & Translation Fidelity** The finding that the system performs best with precise, technical language presents a challenge for an interface designed to be intuitive; the system most accurately matched the proxy for creative intent when given these structured descriptions, suggesting that users may need to adopt a more technical phrasing to achieve predictable results. For creative applications, this is not necessarily a flaw; misinterpretations can lead to surprising and enjoyable outcomes, framing the LLM interaction as a creative, experimental process that can yield unexpected results. However, for scientific or other high-stakes applications requiring precision, this ambiguity presents a significant barrier. This suggests several directions for future HCI research, such as developing adaptive interfaces that include player feedback or systems that can outline plans or request clarification when faced with ambiguous commands.

#### 8 Conclusion

This research introduces the LLM-ECS pattern as a reliable template for building safer, languagedriven interactive systems. We provide a quantitative comparison of LLM performance on simultaneously creative and logical generation tasks and an analysis of how prompting strategies can be tailored to compositional versus procedural DSLs. Future work could focus on adaptive and multi-modal interfaces that teach users more effective interaction styles. A more ambitious direction would be to have the LLM determine the actual outcomes of gameplay on a semantic or logical basis, and provide interoperable hooks for other game systems to ground these outcomes in the virtual world. For game development, the immediate next step is to integrate this architecture into a full game with player progression, while a longer-term goal is to enable the LLM to alter more fundamental game systems, a significant step toward solving the historical authoring burden in interactive media.

#### Limitations

The primary limitation of this work is the use of an LLM judge for qualitative evaluation. While validated, it is not a perfect substitute for human assessment, is subject to potential self-enhancement bias, and notably struggled to reliably evaluate the structural coherence of the procedural automata scripts. A methodological disconnect still remains, where a script can appear well-reasoned in code yet result in a poor gameplay experience, or vice versa. Therefore, the judge ratings are only interpreted on a relative basis between experimental groups, and cannot speak to broader success in a live game. The small-scale human pilot study, while encouraging, may not generalize to a broader player base.

Our experiments also relied on synthetic data. Inputs for the bidirectional experiment were indirectly grounded to the DSL's feature space, likely simplifying the translation task compared to ambiguous, authentic user input. Additionally, the system was only tested with "good faith" (at least moderately aligned with design goals) inputs in the automated evaluation; a production system would require more comprehensive input sanitization beyond the simple "fizzle out" spell failure fallback used here to handle adversarial or nonsensical prompts. Finally, these results represent a narrow snapshot of a rapidly evolving field; Using state-of-the-art models whose architectures are proprietary and frequently updated poses a significant challenge to the reproducibility of this work.

#### **Ethical Considerations**

The human pilot study in Section 6 has been approved by the FESE Engineering Mathematics and Physical Sciences Ethics Committee at the University of Exeter.

While our application in a recreational game is low-risk, deploying a similar architecture in safety-critical domains like robotics would require more substantial controls to handle misaligned LLM outputs. While modern LLM providers have robust safety features, there remains a risk of insensitive or offensive content being generated. However, since the model is primarily producing DSL code for this prototype, the negative impact of unsafe outputs is minimized. Additionally, the prototype's reliance on user-provided, external API keys introduces data privacy and ownership concerns, as user data becomes subject to the terms of third-party model providers. This may be mitigated by opting

out the human review and data sharing features in those providers. For many sensitive applications, privately-hosted models would be preferable.

#### References

Raffaella Bernardi, Julia Hockenmaier, Udo Kruschwitz, Prashant Jayannavar, and Massimo Poesio. 2025. Human in the Loop Learning through Grounded Interaction in Games (Dagstuhl Perspectives Workshop 24492). *Dagstuhl Reports*, 14(12):28–45.

BigFrogg. 2025. Wizard cats. Game Software. Accessed: 2025-07-18.

Max Bittker. 2023. sandspiel studio. https://github.com/maxbittker/sandspiel-studio. Software. Live version available at: https://studio.sandspiel.club/.

Tianshi Cheng, Tong Duan, and Venkata Dinavahi. 2023. Ecs-grid: Data-oriented real-time simulation platform for cyber-physical power systems. *IEEE Transactions on Industrial Informatics*, 19(11):11128–11138.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-Jiang Jiang, and 3290 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *Preprint*, arXiv:2507.06261.

Roberto Gallotta, Graham Todd, Marvin Zammit, Sam Earle, Antonios Liapis, Julian Togelius, and Georgios N. Yannakakis. 2024. Large language models and games: A survey and roadmap. *IEEE Transactions on Games*, pages 1–18.

Xiaodong Gu, Meng Chen, Yalan Lin, Yuhan Hu, Hongyu Zhang, Chengcheng Wan, Zhao Wei, Yong Xu, and Juhong Wang. 2025. On the effectiveness of large language models in domain-specific code generation. *Preprint*, arXiv:2312.01639.

Lars I. Hatledal, Yingguang Chu, Arne Styve, and Houxiang Zhang. 2021. Vico: An entity-component-system based co-simulation framework. *Simulation Modelling Practice and Theory*, 108:102243.

Ray Ito and Junichiro Takahashi. 2024. Demo paper: A game agents battle driven by free-form text commands using code-generation llm and behavior branch. In 2024 IEEE Conference on Games (CoG), pages 1–2.

Sathvik Joel, Jie JW Wu, and Fatemeh H. Fard. 2024. A survey on llm-based code generation for low-resource and domain-specific programming languages. *Preprint*, arXiv:2410.03981.

- Lebling, Blank, and Anderson. 1979. Special feature zork: A computerized fantasy simulation game. *Computer*, 12(4):51–59.
- Boyi Li, Philipp Wu, Pieter Abbeel, and Jitendra Malik. 2025a. Interactive task planning with language models. *Preprint*, arXiv:2310.10645.
- Jia Li, Ge Li, Yongmin Li, and Zhi Jin. 2025b. Structured chain-of-thought prompting for code generation. *ACM Trans. Softw. Eng. Methodol.*, 34(2).
- Yinghao Li, Rampi Ramprasad, and Chao Zhang. 2024. A simple but effective approach to improve structured language model output for information extraction. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 5133–5148, Miami, Florida, USA. Association for Computational Linguistics.
- Rui Liu, Yibei Guo, Runxiang Jin, and Xiaoli Zhang. 2024. A review of natural-language-instructed robot execution systems. *AI*, 5(3):948–989.
- Subhabrata Majumdar and Terry Vogelsang. 2024. Towards safe llms integration. In Andrei Kucharavy, Octave Plancherel, Valentin Mulder, Alain Mermoud, and Vincent Lenders, editors, *Large Language Models in Cybersecurity: Threats, Exposure and Mitigation*, chapter 11, pages 243–257. Springer Cham.
- Bill Manaris. 1998. Natural language processing: A human-computer interaction perspective. *Advances in Computers*, 47:1–66.
- Michael Mateas and Andrew Stern. 2003. Façade: An experiment in building a fully-realized interactive drama. In *Game Developers Conference (GDC)*.
- Michael Mateas and Andrew Stern. 2005. Structuring content in the Façade interactive drama architecture. In *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE '05, pages 93–98. AAAI Press.
- Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. *Preprint*, arXiv:2304.03442.
- Mateusz Pawlik and Nikolaus Augsten. 2015. Efficient computation of the tree edit distance. *ACM Transactions on Database Systems*, 40:1–40.
- Mateusz Pawlik and Nikolaus Augsten. 2016. Tree edit distance: Robust and memory-efficient. *Information Systems*, 56:157–173.
- C. Raymond Perrault and Barbara J. Grosz. 1988. Chapter 4 natural-language interfaces. In Howard E. Shrobe and the American Association for Artificial Intelligence, editors, *Exploring Artificial Intelligence*, pages 133–172. Morgan Kaufmann.

- Vittorio Romeo. 2016. Analysis of entity encoding techniques, design and implementation of a multithreaded compile-time entity-component-system c++14 library. Bachelor's thesis, University of Messina, 07.
- Subham Sah, Rishab Mitra, Arpit Narechania, Alex Endert, John Stasko, and Wenwen Dou. 2024. Generating analytic specifications for data visualization from natural language queries using large language models. Presented at the NLVIZ Workshop, IEEE VIS 2024. *Preprint*, arXiv:2408.13391.
- Tyler Sanders. 2024. Machines of the absurd: leveraging generative ai for creativity, humor, and playfulness. Master's thesis, College of Computing and Digital Media, DePaul University, Chicago, Illinois, USA, June. Degree Project in Computing and Digital Media.
- Stanislaw K. Skowronek. 2008. The powder toy. Game Software. Accessed: 2025-07-18.
- Yuqian Sun, Zhouyi Li, Ke Fang, Chang Hee Lee, and Ali Asadipour. 2023. Language as reality: A cocreative storytelling game experience in 1001 nights using generative ai. *Preprint*, arXiv:2308.12915.
- Leanne Taylor-Giles. 2020. *Player-Centred Design in Role-Playing Game Branching Dialogue Systems*, pages 295–325. Springer International Publishing, Cham.
- Aman Singh Thakur, Kartik Choudhary, Venkat Srinik Ramayapally, Sankaran Vaidyanathan, and Dieuwke Hupkes. 2025. Judging the judges: Evaluating alignment and vulnerabilities in llms-as-judges. *Preprint*, arXiv:2406.12624.
- Tuul Triyason. 2023. Exploring the potential of chatgpt as a dungeon master in dungeons & dragons tabletop game. In *Proceedings of the 13th International Conference on Advances in Information Technology*, IAIT '23, New York, NY, USA. Association for Computing Machinery.
- Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A Saurous, and Yoon Kim. 2023. Grammar prompting for domain-specific language generation with large language models. *Advances in Neural Information Processing Systems*, 36:65030–65055.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.
- Dennis Wiebusch and Marc Erich Latoschik. 2015. Decoupling the entity-component-system pattern using semantic traits for reusable realtime interactive systems. In 2015 IEEE 8th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), pages 25–32.

Yauwai Yim, Chunkit Chan, Tianyu Shi, Zheye Deng, Wei Fan, Tianshi Zheng, and Yangqiu Song. 2024. Evaluating and enhancing llms agent based on theory of mind in guandan: A multi-player cooperative game under imperfect information. *Preprint*, arXiv:2408.02559.

Lifan Yuan, Yangyi Chen, Ganqu Cui, Hongcheng Gao, Fang Yuan Zou, Xingyi Cheng, Heng Ji, Zhiyuan Liu, and Maosong Sun. 2023. Revisiting out-of-distribution robustness in nlp: Benchmarks, analysis, and llms evaluations. In *Advances in Neural Information Processing Systems*, volume 36, pages 58478–58507. Curran Associates, Inc.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

## **A** Abridged Prompts

This appendix contains the abridged, plain-text versions of the three main prompts used in our experiments. To adhere to formatting constraints, lengthy definitions of components and evaluation rubrics were replaced with descriptive tags, while all instructional text remains unaltered where possible. The full unabridged prompts are available in our GitHub repository<sup>2</sup>.

#### A.1 Spell Scripting Prompt

The first prompt, shown in Listing 1, configures the LLM for *Latent Space's* Battle Mode. Its task is to translate a user's natural language description into a valid JSON object that conforms to a DSL, designed with a compositional, unordered structure where spells are defined as collections of components. This approach, which maps directly to the underlying ECS architecture, was found to be more forgiving for LLMs than strictly ordered logic.

Listing 1: Abridged Spell Scripting Prompt.

You are an AI game design assistant.
Your task is to generate a JSON
object that defines a magical spell
using a component-based system. The
spell will be based on a provided
description.

Overall Goal:

Create a single JSON object. The root of this object must always contain the key: "components". The value of " components" must be an array of individual component objects. The top-level spell may optionally contain the key: "count" when strictly appropriate for multi-cast. The top-level spell must finally contain the key: "friendlyName" containing a creative 2-3 word name for future reference.

Strict Output Requirements:

- The entire response must be a single, valid JSON object.
- Do not include any explanatory text, markdown formatting, or anything outside this single JSON object.
- Use only the componentTypes and their associated properties as defined below.
- For fields with "Possible Options" or specific enumerated values, you must choose a value from the provided list(s). Do not invent new string values for these fields.
- If a property is optional and not relevant to the spell concept, omit it
- Properties of a spell are never automatically inherited by subspells; they must be repeated as necessary when using triggers.
- Always think creatively, and always consider whether to add sub-spells or physical manifestations when it could strengthen the concept.
- Numerical values should be sensible for a game context and fall within suggested ranges if provided.

Component Definitions:

- A spell is defined by an array under the "components" key. Each object in this array is a component.
- I. Spell Class Components (Choose exactly one as the primary form of the spell or sub-spell. Using more than one requires that they be placed in different nested triggers .):
- projectile: Defines the spell as a projectile.
  - <... 7 more 'Spell Class' components were defined here (e.g., wallCrawl , aoe, shield, manifestation), each with specific properties and descriptions. Full list omitted for brevity. ...>
- II. General Spell Property Components (
   Add as needed):
- element: Assigns a magical element from a predefined list.
  - <... 3 more 'General Spell Property'
     components were defined here (e.g
     ., color, spawnAngle, manaCost),
     each with specific properties.
     Full list omitted for brevity.</pre>

<sup>2</sup>https://github.com/austin-the-drake/ real-time-world-crafting-wordplay-demo

- III. Behaviour Modifier Components (Can be stacked; primarily affect projectile or wallCrawl):
- homing: Causes the spell to seek enemies.
  - <... 2 more 'Behaviour Modifier'
    components were defined here (
    boomerang, controllable), each
    with specific properties. Full
    list omitted for brevity. ...>
- IV. Trigger Components (Define subspells or effects triggered by conditions):
- timerTrigger: Executes a sub-spell after a set time.
  - <... 3 more 'Trigger' components were
     defined here (buttonTrigger,
     impactTrigger, deathTrigger), each
     with specific properties to
     define a sub-spell payload. Full
     list omitted for brevity. ...>

<Few-shot examples may be provided here>

#### Vour Tack

Generate the JSON object containing a "
components" array for the spell
concept provided below. Ensure all
your choices and values adhere to
the definitions, constraints, and
suggested ranges listed above. The
user's description of a magical
spell will be provided first,
followed by a list of all magical
elements available in your toolbox.

<Dynamic context here>
<Player input here>

#### **A.2** Automata Scripting Prompt

The second prompt (Listing 2) is tailored to the cellular automata sandbox. It instructs the LLM to generate a behavior script using a node-based DSL with a strict, procedural structure. Unlike the spell DSL, the sequence of conditional checks and actions in this language is highly important to the final outcome.

Listing 2: Abridged Automata Scripting Prompt.

You are a game design assistant specialising in cellular automata. Your task is to generate a single, valid JSON object that defines a behavior script based on the user's description. The output must be a single JSON object containing three root keys: "name" (a creative, oneword, lowercase string), "color\_hex" (a hex string like "#RRGGBB"), and "behavior" (a struct containing the "actions" array).

The entire response must be a single, valid JSON object. Do not include any explanatory text, markdown

formatting, or anything outside of this object. Adhere strictly to the node definitions provided.

#### Important Notes:

- The term "direction" below is a placeholder for one of ["north", " northeast", "east", ...].
- "self" is a valid keyword for a particle's own cell.
- Actions are processed sequentially.
- The do\_swap node is special: it moves the cell, then can immediately run a nested actions list from the cell's new location. [...]
- If you are asked to update an existing cell type, do not alter its name field, or else it may break references.
- Available Node Types (for the actions array inside the behavior struct)
- I. Wrapper / Modifier Nodes (These
   contain other nodes):
- in\_rand\_rotation: Executes nested actions in one random of 8 directions.
  - <... 2 more 'Wrapper / Modifier' nodes were defined here (in\_rand\_mirror , in\_rand\_flip). Full list omitted for brevity. ...>
- II. Conditional Nodes (These check a condition and then run nested actions):
- if\_neighbor\_is: Checks if a neighbor is one of the types in options.
  - <... 4 more 'Conditional' nodes were defined here (e.g., if\_alpha, if\_neighbor\_count, if\_chance).
    Full list omitted for brevity.
    ...>
- III. Executor / Action Nodes (These perform an action and are usually the innermost nodes):
- do\_swap: Swaps position with a neighbor, then can immediately run a nested actions list from the cell's new location. This entire operation ends the cell's turn.
  - <... 4 more 'Executor / Action' nodes
     were defined here (e.g.,
     do\_set\_type, do\_spawn,
     do\_copy\_alpha). Full list omitted
     for brevity. ...>

<Few-shot examples may be provided here>

#### Your Task:

Be creative and thorough. [...] Now, generate the complete JSON object for the following element. You will be provided with a user input [...], and a list of all the existing materials and their respective behaviors.

<Dynamic context here>

## A.3 Judge Model Prompt

The third prompt (Listing 3) configures a separate LLM to act as an automated judge. This approach was justified by established literature on using LLMs for larger-scale qualitative assessments. To improve reliability and mitigate known issues like leniency bias, the prompt uses several strategies, including instructing the model to produce a rationale before its final score (a form of Chain-of-Thought) and using a capable state-of-the-art model, in line with recommended guidelines.

#### Listing 3: Abridged Judge Prompt.

- Your primary role is to evaluate a JSON-based (spell/automata) script. You will act as a strict technical reviewer. This is a difficult logical and creative task, and most scripts will not achieve a perfect score.
- This prompt provides all the information you need in a structured format. First, you will be presented with the task rules [...]. Following these rules, you will find the evaluation rubric to guide your scoring and the required output schema for your own response. Finally, you will see the full task [...].
- Please begin by carefully reviewing the documentation below.
- <Documentation from the original game
   prompt was provided to the model
   here.>
- You will evaluate the generated script based on the following four axes. This is a challenging task and you should evaluate critically. Score each axis from 1 (poor) to 5 (excellent).
- Creative Alignment (1-5):
- (This scale measures how well the code matched the core concept or "theme" of the user's description, with an emphasis on presentation.)
- 5 (Perfect Match): The code perfectly captured the core concept and all thematic details, feeling exactly like what was imagined.
- 1 (No Connection): The code's core function was the opposite of what was requested.
  - <... Detailed descriptions for scores 2-4 on this axis were also provided. ...>

Instructional Precision (1-5):

- (This scale measures how well the code obeyed any specific, direct commands or constraints the user included, with an emphasis on logic.)
- 5 (Followed All Instructions Perfectly
  ): Every single explicit instruction
  and constraint was implemented
  precisely as requested.
- 1 (Ignored All Instructions): The code failed to follow multiple explicit instructions.
  - <... Detailed descriptions for scores 2-4 on this axis were also provided. ...>

#### Emergence (1-5):

- (This scale measures if the code's behaviour went above and beyond the user's request in a positive way.)
- 5 (Delightful Surprise): The code added a significant, unrequested feature that was a perfect creative fit, making the entire concept much better and more interesting.
- 1 (Boring/Literal): The code was a completely literal and uninspired translation of the prompt with no creative flair whatsoever.
  - <... Detailed descriptions for scores 2-4 on this axis were also provided. ...>
- Structural & Logical Coherence (1-5):
  (This scale evaluates the syntactic and logical validity of the generated spell.) This is informed by the "Algorithmic Pre-Check". If the pre-check failed, this score cannot be higher than 2.
- 5: The entire response is a single, valid JSON object with perfect data types and structure, and it is completely free of any logical contradictions defined in the component rules.
- 3: Valid JSON, but with some data type errors or logical conflicts forbidden by the rules (e.g., a manifestation component having a deathTrigger).
- 1: Invalid JSON or major structural errors (e.g., components is not a list).
- <Calibration examples were provided to the model here.>
- You must respond with a single, valid JSON object and nothing else. Remember that these responses must be graded strictly. The majority of responses should achieve an average of 2-4. 4s and 5s should be reserved for truly exceptional responses [...].
- The root object of your response must contain two keys, in this exact order: "rationales" and "scores".

- "rationales": An object containing a brief, one-sentence text explanation for each of the four scores. [...]
   "scores": An object containing a
- "scores": An object containing a numerical score (1-5) for each of the four rubric axes. [...]

lower agreement on the Structural Coherence scale for the complex cellular automata scripts, suggesting this type of logical evaluation remains challenging even for capable models.

## **B** Full NL-to-DSL ANOVA Results

The full ANOVA results for the NL-to-DSL translation experiment are presented in Table 3. The analysis shows that the choice of LLM was the most significant predictor of output quality across all four qualitative measures for both the compositional spell DSL and the procedural automata DSL (p<.001). The effects of prompting strategies were highly task-dependent; Chain-of-Thought (CoT) prompting generally improved creative alignment and emergence, while the inclusion of few-shot examples was only significant for the more logically complex cellular automata task.

Figure 6 visualizes these results for the magical spell DSL, demonstrating the significant interaction effects found between the model, shot strategy, and prompting technique. This illustrates that the optimal prompting strategy is highly dependent on the specific model being used, as the impact of CoT and few-shot examples was not uniform across all models.

#### C LLM Judge Validation

This appendix details the validation process for the LLM judge used in our experiments. Figure 7 presents the Receiver Operating Characteristic (ROC) curves, which visualize the performance of the *GPT-4.1* judge in discriminating between expert-authored "good" (well-formed and effective) and "bad" (logically flawed) scripts from our handcrafted ground-truth dataset (N=80). The high Area Under the Curve (AUC) values across most criteria demonstrate the judge's strong alignment with a human-defined standard of quality.

To further ensure consistency and mitigate single-model bias, we calculated the inter-rater reliability between the primary judge and a second, equivalently-capable model, *Gemini 2.5 Pro*. Table 4 displays these metrics, including Spearman's  $\rho$ , weighted Cohen's Kappa  $(\kappa_w)$ , and the Intraclass Correlation Coefficient (ICC). The results show moderate to substantial agreement across most criteria, indicating that the evaluation scales are objective enough to be scored consistently by different state-of-the-art models. A notable exception was

Table 3: ANOVA Results for the NL-to-DSL Experiment

	Predictor	Magical Spells			Cellular Automata		
Outcome		F	p	$\overline{\eta_G^2}$	F	р	$\eta_G^2$
Creative Alignment	Model (M)	58.43	<.001‡	.12	117.88	<.001‡	.35
C	Shot Strategy (S)	0.53	.581	.00	21.13	<.001‡	.03
	Planning (P)	9.04	.004†	.01	15.85	<.001‡	.01
	$M \times S$	2.31	.040*	.01	1.14	.340	.00
	$M \times P$	2.33	.080	.00	0.78	.499	.00
	$S \times P$	0.29	.744	.00	0.91	.400	.00
	$M\times S\times P$	0.95	.447	.00	1.05	.386	.00
Instruction Following	Model (M)	86.86	<.001‡	.14	115.53	<.001‡	.31
Č	Shot Strategy (S)	1.47	.235	.00	23.75	<.001‡	.03
	Planning (P)	2.78	.102	.00	7.98	.007†	.01
	$M \times S$	2.54	.024*	.01	1.47	.199	.01
	$M \times P$	2.77	.044*	.01	2.14	.107	.00
	$S \times P$	1.91	.153	.00	1.17	.316	.00
	$M\times S\times P$	1.04	.398	.00	0.77	.571	.00
Emergence	Model (M)	35.63	<.001‡	.10	128.81	<.001‡	.33
	Shot Strategy (S)	1.81	.170	.00	16.08	<.001‡	.02
	Planning (P)	5.78	.020*	.01	17.57	<.001‡	.01
	$M \times S$	2.66	.021*	.01	1.62	.156	.01
	$M \times P$	0.81	.487	.00	1.00	.386	.00
	$S \times P$	3.63	.032*	.00	0.23	.790	.00
	$M\times S\times P$	2.66	.020*	.01	0.83	.521	.00
Structural Coherence	Model (M)	192.28	<.001‡	.41	18.81	<.001‡	.05
	Shot Strategy (S)	2.88	.062	.00	22.32	<.001‡	.05
	Planning (P)	0.38	.540	.00	1.55	.220	.00
	$M \times S$	2.56	.031*	.01	7.20	<.001‡	.03
	$M \times P$	6.31	.001†	.02	0.78	.486	.00
	$S \times P$	1.48	.233	.00	0.01	.968	.00
	$M \times S \times P$	0.79	.558	.00	0.91	.450	.00

*Note.*  $\eta_G^2$  = generalized eta-squared. Significance codes: \* p < .05, † p < .01, ‡ p < .001

Table 4: Inter-Rater Reliability Metrics Between GPT-4.1 and Gemini 2.5 Pro

	Magical Spells			Cellular Automata			
Metric	Spearman's ρ	Weighted $\kappa$	ICC	Spearman's ρ	Weighted $\kappa$	ICC	
Creative Alignment	.70	.57	.57	.81	.74	.75	
Instruction Following	.68	.57	.57	.79	.75	.75	
Emergence	.68	.64	.65	.76	.77	.77	
Structural Coherence	.59	.56	.56	.36	.23	.24	

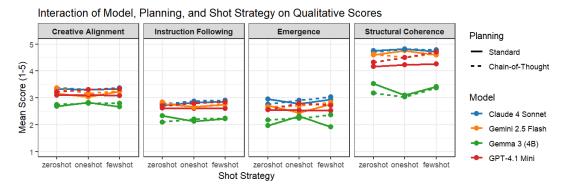
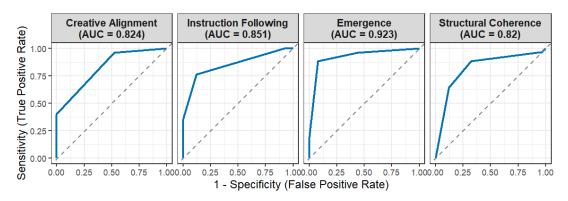


Figure 6: Interaction plots for the NL-to-DSL ANOVA for magical spell DSL scripts.

## Magical Spells (N=50)



#### Cellular Automata (N=30)

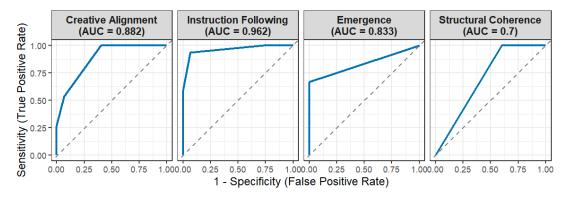


Figure 7: ROC plots of *GPT-4.1* performance in discriminating between manually-authored "good" and "bad" DSL scripts. Performance was evaluated separately for the 4 Likert scales on the ground-truth data.