Final Report: AI-Driven Dynamic Storytelling Game

~Hayden Komasz, Doug Hoover

Abstract:

Dungeons & Dragons (D&D) is a role-playing game centered around collaborative storytelling and gameplay created by dice mechanics and character decisions. Our project's goal was to develop a traditional Dungeon Master storyteller using large language models (LLMs) to build an intelligent and dynamically generated response in real time. The objective is to allow users to input a simple prompt and receive an entirely generated and branching story that includes characters, companions, encounters, and quests. Our comparative analysis of DeepSeek and Gemini lets us look deeper into modern LLMs' creativity, speed, grammar, and consistency.

Introduction:

The game's core begins with the user entering a narrative prompt, which will be used to generate the campaign. The player is then asked to select a character from among four options. The user can see the different skills and starting items within this player selection with the character. In addition, the user can see a short description of the character, their health, attack, and defense stats. These stats will determine how much damage is taken or output. At this point, the user continues with the story. They will be presented with a setting, a main quest, and three additional side quests that must be completed to win the game.

At every turn, the player will be shown the quest log and seven game options within the game. This includes exploring a location, traveling to new locations, checking quest logs, talking to companions, resting and healing, checking inventory, and exiting the game. In each environment, a user can select to explore the current location. After doing this, the user will be given the option to interact with the area. This interaction is random and may have only four options, with as many as twelve. In the encounters, the player is always allowed to have a battle. Each environment's goal is to find the item/defeat the enemy that solves the quest. After completing an encounter, the player is again prompted with the main script of options. The travel to a new location option will prompt the user to choose from four to five locations. By following the quest log, the user will see that each quest is mapped to a separate location. This allows each story to include three to four side quests with one main quest. After completing the side quests, the user can enter the final area to complete the main quest and finish the game. Within the game, it can be helpful to talk to companions who may have knowledge or items they can give the user to assist in the story. As mentioned before, the player will be given the option of battle in each environment.

Battling works based on a dice roll format. Depending on whether the player uses a basic attack, skill, or item, the dice rolls have different chances of landing. This dice roll system is also implemented for companions and enemies. During battle sequences, the game will generate between one and four enemies for the user and companions to battle. After completing a battle, the user will be gifted an item and have the chance to complete a related quest. At the end of each battle sequence, once the user has returned to the main menu, they can use the rest and heal function, which will allow the players and companions to heal for another encounter. Lastly, the user can check the inventory for items they can use during future battles.

Introduction to LLMs and the D&D Model:

In developing our system for Dungeons & Dragons (D&D) narrative generation and interactive gameplay support, we integrated two Large Language Models: DeepSeek and Gemini. These models were chosen due to their ease of accessibility and ease of integration. These models were tested on the same code and evaluated on specific criteria to evaluate the performance and effectiveness of each model for the use of this project.

Deepseek served as the primary model for our system. It was implemented and used when developing the game. Its accessible API and straightforward implementation made it an appealing model to build our game around. In creating the code, we initially used a free key that allotted a certain amount of tokens per day, which allowed us to use it fairly frequently, given enough time had passed between uses. Towards the end of developing the code, as we were working on it more frequently, the limited tokens per day became a burden, which prevented us from further developing and testing. For the entirety of development, the free key was used; however, for our final testing, we switched to a paid key, so we could run more tests at a time and receive better responses.

Gemini was not used until we began our final testing. Switching to this model was fairly simple, as all it required were small modifications to our config file, and in the main project code, all that needed to be changed was the model name. We wanted to use this so we could compare how a different model performs given the same code. Before testing, we were unsure of whether this change would actually work or entirely break the code. After changing the model and running, we found that the functionality was approximately the same as when using DeepSeek.

Design Motivation:

Traditional games rely on fixed scripts and limit the players' choice to what is already hard-coded. Our motivation was to push storytelling to another level by letting the user shape the world dynamically. This was inspired by reading studies of how AI dialogue can shape an engaging and immersive feeling that alters the game state according to the user's prompt.

Another motivation was to remove the need for a dungeon master, allowing new players to easily enjoy the game without needing another person who is familiar with the game. The way we have designed the model allows for consistent gameplay regardless of group size. We also aimed to find a balance between creative language generation and setting rules to guide the model. This allows for much innovation in the model design that we could not show off because of testing purposes on the LLM models. These innovations included changing the prompt mid-game and giving the player characters stats to make them more user-friendly in combat encounters.

Implementation:

The model's backend is built using Python and utilizes tools such as OpenAI for Gemini and DeepSeek's API to generate responses. A structured game loop handles narrative generation, combat mechanics, and quest tracking. At the start of the game, the user is asked to enter a game prompt and is asked questions such as which character they would like to select after the generation of the story. These prompts and actions are parsed and structured into function calls that relay data to the LLM. The LLM returns a response that is interpreted and displayed in the terminal.

First, we created the character and enemy classes to manage health, skills, inventory, and dedicated methods to handle attack logic, damage, and healing. Combat is implemented using a turn-based system where players and companions take turns attacking or using items/skills. Enemies are generated dynamically and target active characters based on random decision-making. Each skill and item is sent and interpreted using an API call that returns a JSON describing the effects. Players' and companions' data are stored in lists that update each turn. We use a consistent structure to manage active entities and location data. This allows the entire game flow to be driven by the user's choices and easy accessibility to make changes to the code structure to allow for more creative choices.

The story consists of four to five locations, each tied to a quest or set of enemy encounters. The explore function triggers a new API call to generate dynamic environments and interactions for the players. Using global variables and flags, we can track quest completion stats that will trigger a quest conclusion script and allow the user to reach the final location and boss after completing all side quests.

Experiment Setup:

In order to properly test the effectiveness of each model, we compared the responses generated when using the same story premise on each model. The story premises tested were “Surprise Me”, “Fantasy”, “Western”, and “Space”. After entering the prompt, we tested our core gameplay functionalities involving calls to the model before reaching the end of the game. Each run ran the story, character, and quest generation immediately after the premise was entered. The exploration location is called immediately when traveling to a new location. In the first area, a combat encounter was always chosen to test the enemy generation, which afterwards would complete the quest linked to that area, generating a quest conclusion. Subsequent areas used interact with the environment to test that function, and complete the quest more quickly and easily. Once the final quest is completed, a conclusion to the full story is called for, ending the test run. Other functions in the game use a call to our models, like talking to companions and use of skills and items in battle. These, however, were not run in all test runs to prevent overuse of tokens on the DeepSeek model. When such functions were used in a test run, they were omitted from our scoring.

To score our test runs, we evaluated the responses we received based on four categories: timing, grammar, creativity, and consistency. Timing was simply the average of all the calls to the model in each test run. Grammar was a score given by running the responses through Grammarly. Creativity scores were judged upon how well we believe the model generated creative and interesting responses based on the original premise. Consistency was how well the responses fit the user-entered story premise, or, in the case of "Surprise Me", how well it fit the initial story generation. Creativity and consistency are ultimately subjective; however, in the context of our project, necessary to evaluate to determine which model creates a more engaging game.

Results and Discussion:

Through our testing, Gemini proved to be the quicker and more consistent model for response times(See Figure 1). The DeepSeek calls for certain processes were often multiple times greater than the same calls to Gemini. Occasionally, DeepSeek calls would be received in around the same time as Gemini; however, in our testing, even at its best times, DeepSeek was slower than Gemini's worst.

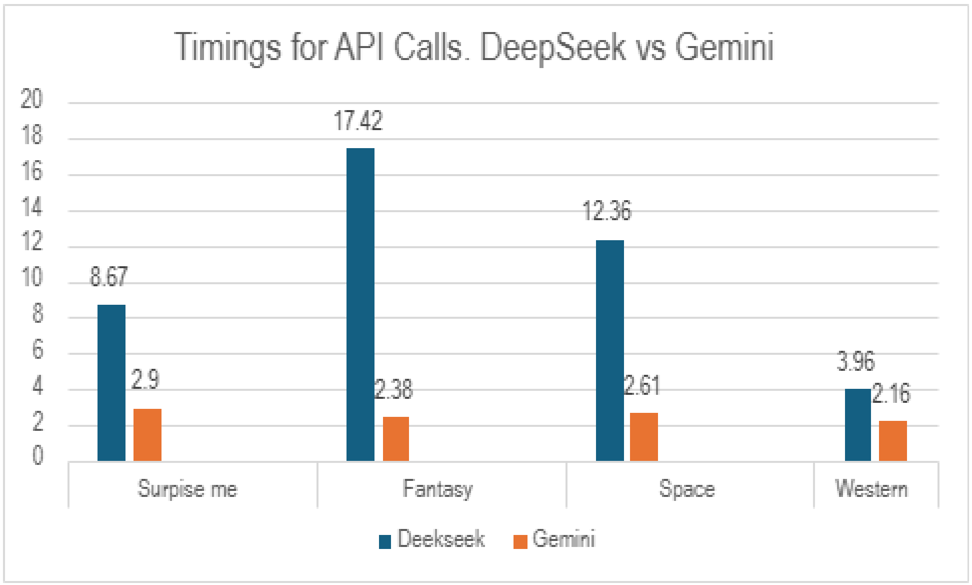


Figure 1: Graph for average time in seconds for each api call. DeekSeek vs Gemini

For creativity, both models were fairly equal, but in our tests, we believed Gemini to be slightly superior to DeepSeek(See Figure 2). Usually, responses for things such as the main story, characters, quests, and exploring environments were quite creative. Given the “Surprise Me” premise, both models could come up with many varying stories across genres and settings. Where both models often lacked creativity was the enemy generation. We often saw names like “Void” being used for space, or “Outlaw” for western, and “The” across all prompts, but especially for surprise me.

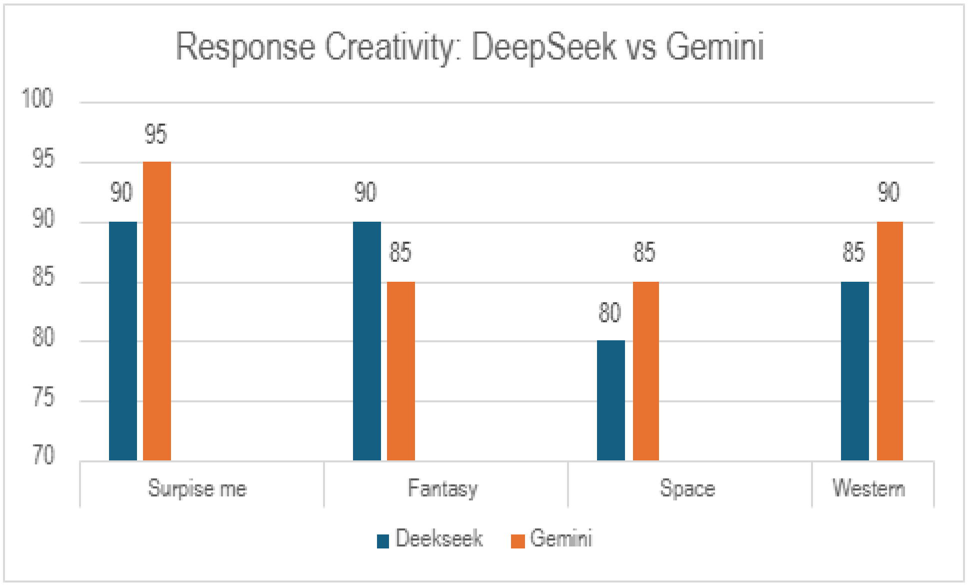


Figure 2: Graph for the creativity of API response. DeepSeek vs Gemini

Consistency was fairly good for both models, except for enemy generation for both models. As mentioned above, names like “The” would appear for enemies often. This happened more often with Gemini, which gave us lower scores(See Figure 3). Both models also lost some consistency with the quests. Although the quests fit the premise fairly well, the actions of the quests as well as their conclusion text often had little to do with the main story.

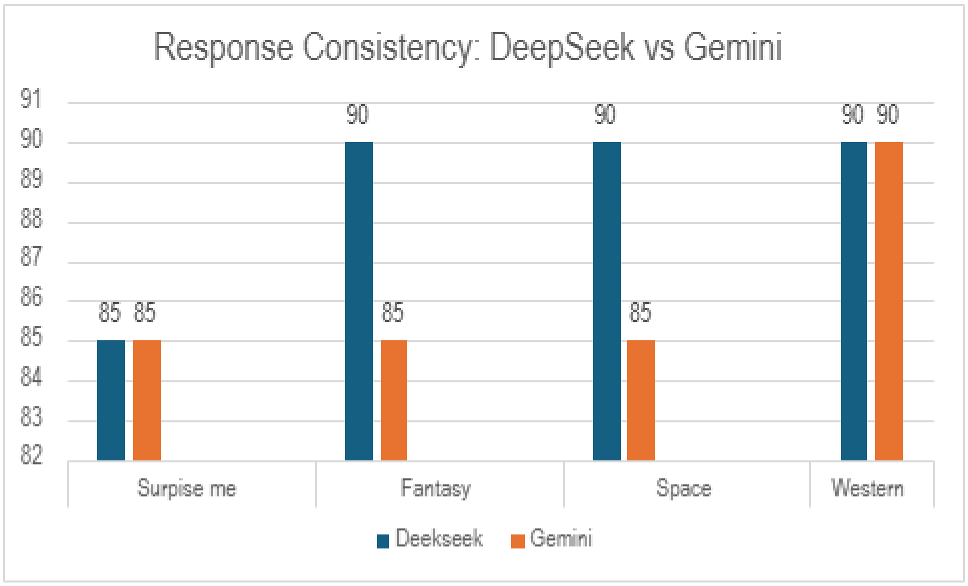


Figure 3: Graph for the consistency of API response. DeepSeek vs Gemini

Grammar for both models was usually quite good, with there only being one score below 90 on a Gemini run(See Figure 4). Overall, DeepSeek had better grammar on average. Both models, however, usually generated grammatically correct and coherent responses.

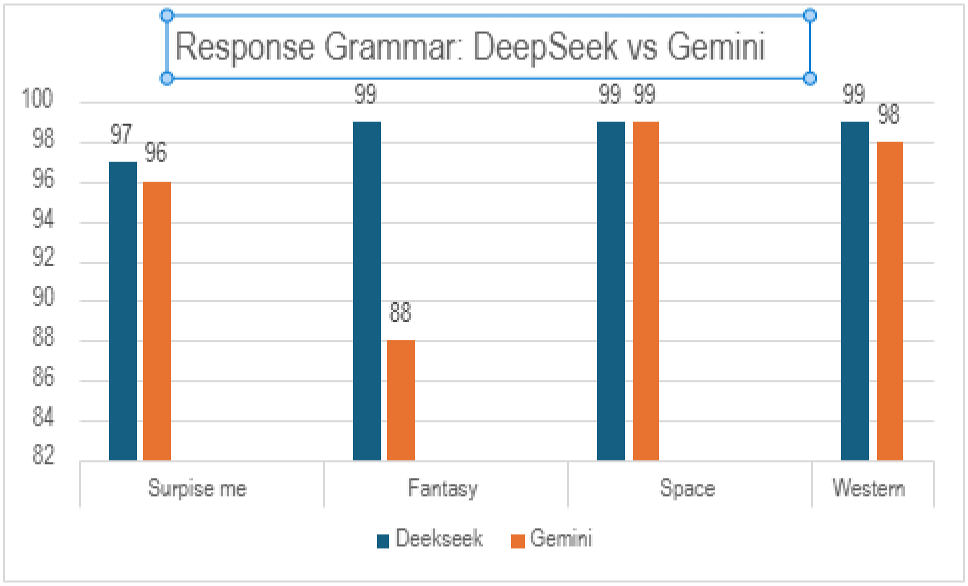
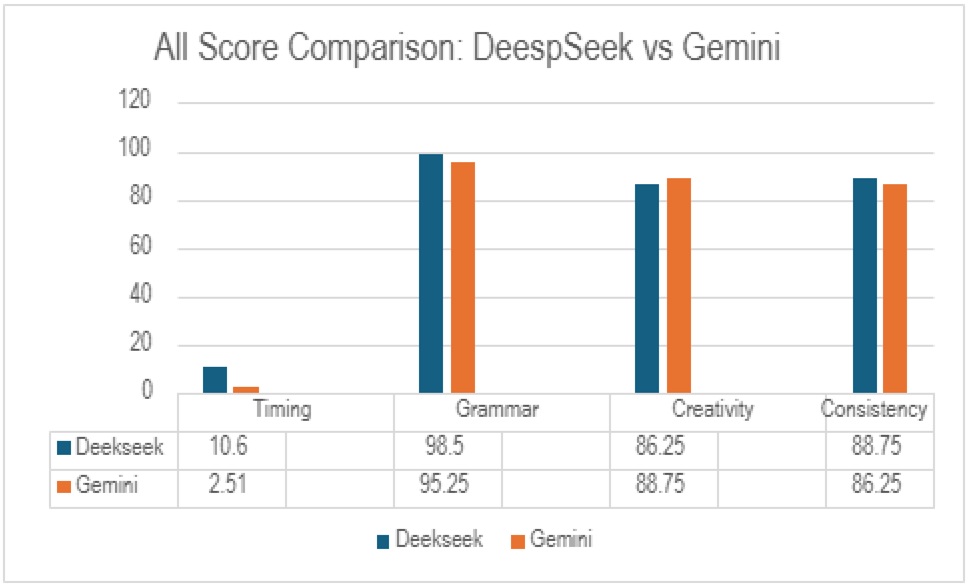


Figure 4: Graph for the grammar of API response. DeepkSeek vs Gemini

Averaging the scores for each criterion, Gemini had the best timing and creativity, while DeepSeek had the best Grammar and Consistency(See Figure 4). Given this, for the actual generated content of the game, Deepseek may be slightly better than Gemini, although the difference is fairly negligible. Gemini is the definitive answer for quicker responses as it is much quicker and more consistent than DepSeek.



	Timing	Grammar	Creativity	Consistency
🟦 Deekseek	10.6	98.5	86.25	88.75
🟧 Gemini	2.51	95.25	88.75	86.25

Figure 5: All criteria comparison of API response. DeekSeek vs Gemini

Across both models, we discovered difficulties in making a game with so many elements reliant on API responses. Often, we would run the code, and a response wouldn't be received from the call. Depending on what the call was for, this could entirely break the game and cause errors. We attempted to fix this by trying multiple times if a response isn't received. This helped, but still the run often ended in an error. Ultimately, if a response can't be received, there is no way for the game to work the way it was envisioned, which is the biggest problem with the game running on both models.

Conclusion:

Our project aimed to create a dynamic, AI-driven Dungeons & Dragons experience by using LLMs to generate adaptive storytelling without a human Dungeon Master. We explored this by integrating DeepSeek and Gemini into a Python-based game framework, enabling real-time generation of quests, characters, and dialogue based on user input.

To evaluate performance, we compared both models using identical game premises and scored them on timing, grammar, creativity, and consistency. Gemini consistently responded faster and offered more creative responses, while DeepSeek produced slightly better grammar and consistency. However, both models struggled with enemy generation and quest consistency.

While many core features worked well, the game's heavy dependency on API calls meant the game would not function when API responses weren't received. These errors show the major problem with a dynamically created game; if one thing fails, the rest of the game may too.

Problems like not receiving a response can be fixed by retrying, but that causes the gameplay experience to suffer as the player would be waiting indefinitely for something to happen. Lack of creativity or consistency may also be resolved through better prompts. Overall, this project demonstrates the potential of LLMs in interactive storytelling. Improving error handling and prompts could improve the gameplay experience; however, regardless of the changes made, the game will always be at the mercy of the models used.

Contributions:

Doug Hoover:

- Created the initial model script and prompt to be used in the LLM.
- Created character, companions and enemy classes.
- Created the API key and config files for Gemini and DeepSeek.
- Created the main game loop class.

Hayden Komasz:

- Created environment exploration and interactions.
- Implemented combat encounters.
- Made functioning quests and logic to complete the game.
- Added features like companion interaction, item drops, skill and item use.
- Tested and evaluated both models.

References:

Chris Callison-Burch, Gaurav Singh Tomar, Lara J. Martin, Daphne Ippolito, Suma Bailis, David Reitter. 2022. Dungeons and Dragons as a Dialog Challenge for Artificial Intelligence. arXiv:2210.07109

Andrew Zhu, Lara J. Martin, Andrew Head, Chris Callison-Burch. 2023. CALYPSO: LLMs as Dungeon Masters' Assistants. arXiv:2308.07540