Divination: Applying Retrieval-Augmented Generation with LLMs to generate content for D&D

Luiz Carlos Costa da Silva

Monograph for the Completion of Course
presented to the course
MAC0499
(Capstone Project)

Advisor: Renato Cordeiro Ferreira

São Paulo, January 2025

Contents

Chapter 1

Introduction

The term "artificial intelligence" has gained prominent media attention; projects like ChatGPT from OpenAI and StyleGAN from NVIDIA (Foster, 2019) exemplify the glowing success. This capstone project was inspired by a similar approach introduced by (A. Zhu and Callison-Burch, 2023), which implements a system based on Dungeons and Dragons (D&D) to help players manage the narrative and monster statistics (e.g. health points, skills, background history, etc.) in their game sessions.

The system described in A. Zhu and Callison-Burch (2023) generates random encounters with monsters to help DMs. It also provides a "conventional brainstorming method," where the DM can ask the system for ideas for the adventure.

This project inspired Divination, which wants to create an application that uses Dungeons and Dragons' open-source information to help DMs manage players' adventures. Unlike Calypso, which focuses on monster encounters and brainstorming, this project provides detailed information from the game sourcebooks to help build characters, create adventures, and so on.

1.1 What is AI e LLM?

Artificial Intelligence (AI), according to the book by Russell and Norvig (2009), can be divided into two dimensions: "thought processes and reasoning" and "behavior."

In the thought process and reasoning dimension, AI is the effort to make the machine think like humans (Thinking Humanly), or the system achieves a rational thought process (Thinking Rationally).

Regarding the behavior dimension, the system can act like a human (Acting Humanly), which is measured with the Turing test, or try as much to follow a logical path to the answer (Acting Rationally).

LLMs, or Large Language Models, are large-scale implementations of pre-trained Language Models (PLMs) W. X. Zhao and Wen. (2024), which aim to predict the likelihood of the next word (token) in a sentence using pre-trained data as a base.

1.2 What is D&D?

According to Dungeons and Dragons Player's Handbook Mearls and Crawford (2024), D&D is a "game that focuses on storytelling"; players make an adventure party to explore a world created by another played called Dungeon Master (known as DM).

The DM creates a new world called a campaign. In this campaign, the player decides the lore of the campaign, e.g., an adventure in which the players should save a princess locked in a castle surrounded by monsters.

After creating a campaign, the players create their characters using the book as a reference, e.g. the first player creates a warrior, and the second creates a mage. Each player has information such as skills and attributes like intelligence, strength, and charisma. After creating the characters, the DM leads the players to follow the history created by him, using the example of the castle, the players need to defeat monsters on the path to the princess. The scenarios, stories, and monsters are all controlled by DM, which sometimes can make it difficult to administrate a large amount of information.

1.3 Objective

The main goal of the capstone project is to create a ChatBot that uses LLM to answer specific questions about the game. The answer should be based on local archives sent to LLM using the provided system.

1.4 Text Structure

This monograph is organized as follows: chapter 2 defines keywords relevant to the project, such as Smart Systems, LLMs, and RAG definitions; chapter 3 discuss the system architecture e code structure; chapter 4 presents the results given inputs related to DnD; and chapter 5 analyze the results obtained.

Chapter 2

Fundamentals

This section provides the necessary background on artificial intelligence to understand the proposal of this capstone project.

The topics include Large Language Models (LLMS) and Retrieval-Augmented Generation (RAG)

2.1 Large Language Models (LLMs)

Machines, unlike humans, cannot naturally understand and communicate unless equipped with powerful artificial intelligence (W. X. Zhao and Wen., 2024). One of the AI techniques used to achieve this goal is *Language Modeling* (LM). This technique, according to the article by W. X. Zhao and Wen. (2024), "aims to model the generative likelihood of word sequences, to predict the probabilities of future (or missing) tokens." This LM can be pre-trained with a large-scale unlabeled *corpus* to be "effective as general-purpose semantic features."

LLMs, or Large Language Models use *Transformers language models* with a hundred of billions of parameters, trained with text data, i.e., GPT-4 (OpenAI, 2025), Gemini (Google, 2025) and LLaMA (Meta, 2025). These LLMs are one of the major development stages of LMs. LLM is a term used by the research community to refer to large-sized pre-trained language models. These LLMs can generate text or complete complex task-solving. (W. X. Zhao and Wen., 2024).

2.2 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is an architectural approach that aims to enhance LLMs' output using external knowledge sources. This external knowledge is divided into chunks with relevant context information. This approach mitigates LLMs "hallucinations", i.e., when they try to answer a user without having the necessary data and output false information (Y. Gao, 2024).

The simplest version of RAG consists of three major steps: Indexing, Retrieval, and Generation.

- Indexing - Extract raw data from external sources and separate it into chunks. The chunks are then stored in the vector store after being "encoded into vector representations using an embedding model."
- Retrieval - The user query is converted into a vector representation and compared with the chunks ranked for similarity. The K most similar chunks are selected as part of the expanded

context.

- Generation - An output will be generated, considering the most similar chunks and user Queries. Some systems may use conversational history to maintain a line of dialogue.

2.3 Hexagonal Architecture

The Hexagonal Architecture, according to Cockburn (2005), is a software design paradigm that focuses on separating business logic from external dependencies. The nature of this solution is the "code pertaining to the 'inside' part should not leak into the ''outside" part.", the application communicates with the external dependencies using ports (similar to computer physical ports), and these ports can be implemented by adapters that implement the functionality of a port.

This design paradigm is represented by a hexagon, the innermost layer represents the core that handles business logic, the middle layer represents the ports handling external data abstraction, and the outermost layer represents adapters that implement the ports.

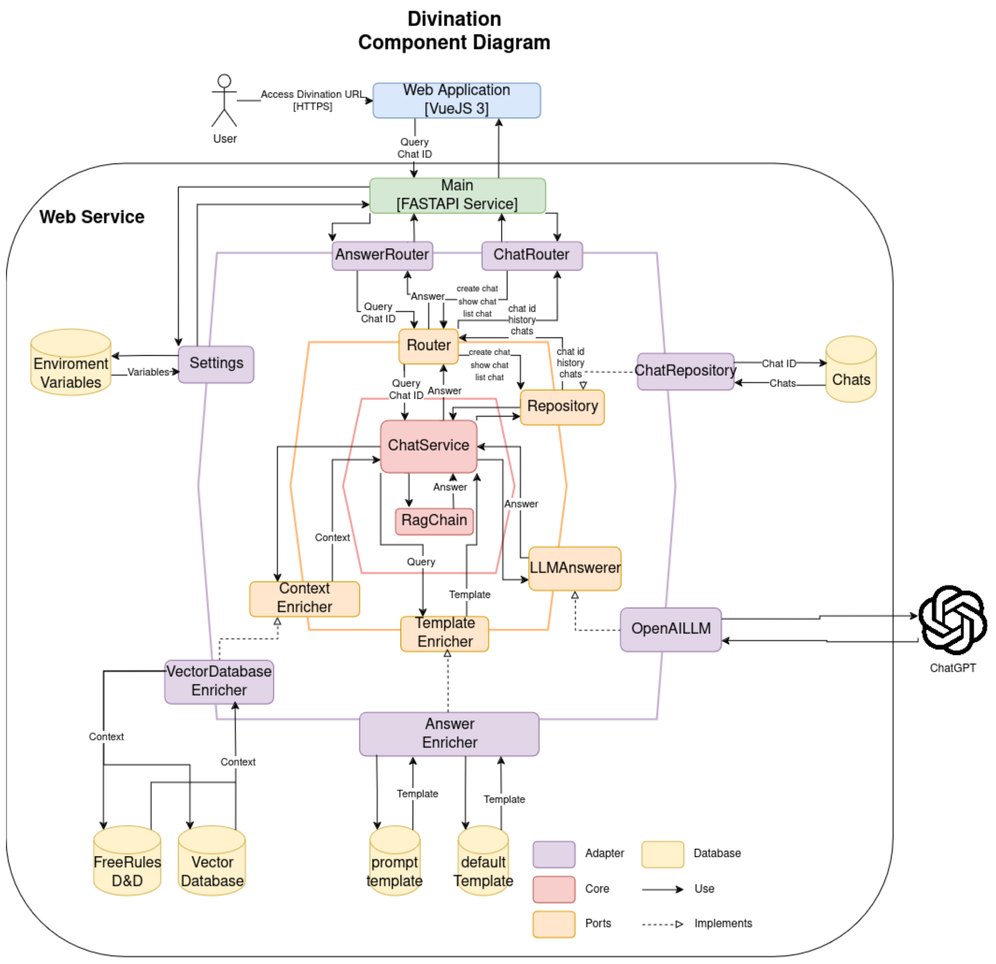


Figure 2.1: *Divination Architecture following the Hexagonal Architecture Pattern: Core, ports, and adapters are separated by concentric layers*

The advantage of the hexagonal architecture is due to the core isolation from external dependencies which makes the code more maintainable, while the use of adapters to implement ports provides flexibility to change dependencies.

Chapter 3

Divination

This chapter describes the capstone project DIVINATION. The architecture and code will be shown using the C4 model diagramming technique (Brown, 2024).

3.1 System

The Divination application receives a text related to D&D game from an user and returns an answer using ChatGPT-4o powered by RAG.

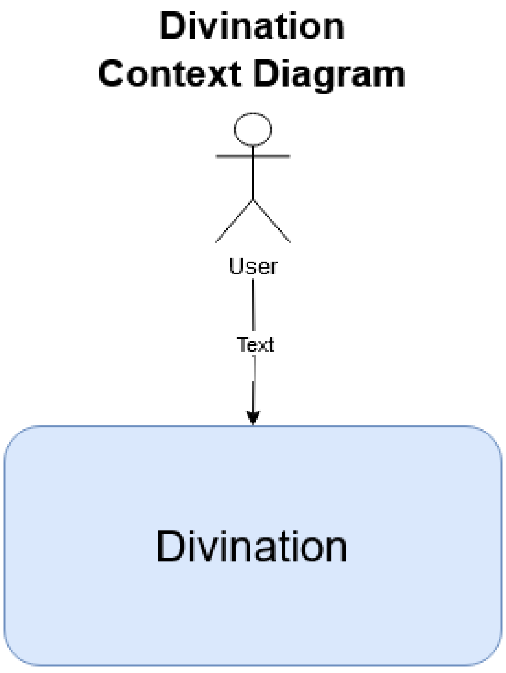


Figure 3.1: *Divination Context Diagram: This diagram shows the first layer of the C4 model, it contains the user of the application that sends a text message to Divination*

3.2 Containers

The Divination Application is divided by Web Application, Web Service, and Vector Database.

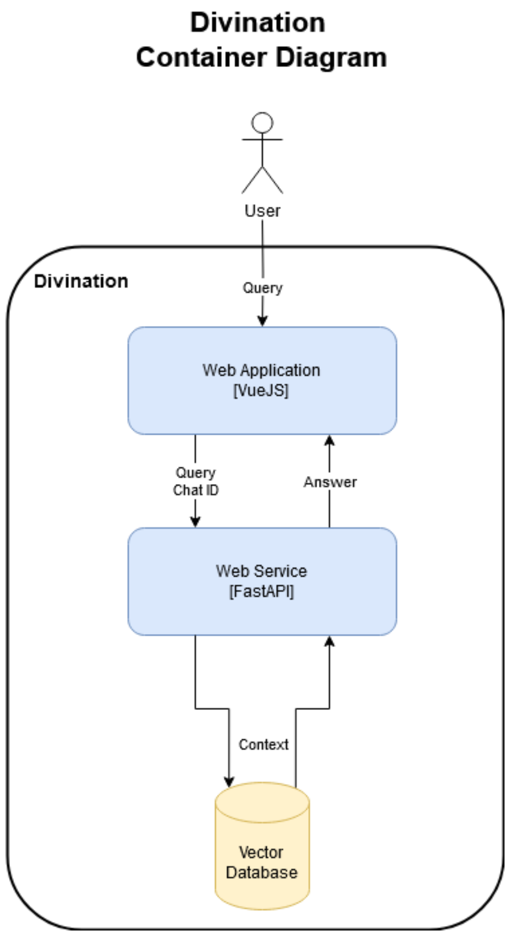
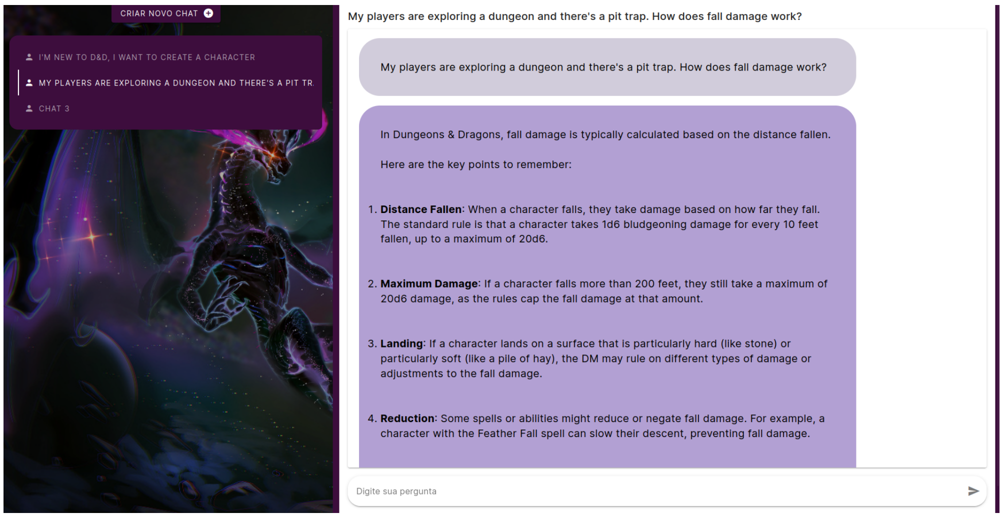


Figure 3.2: *Divination Container Diagram: This diagram shows the second layer of the C4 model, it zooms in on the Divination application showing the Web Application, WebService, and VectorDatabase*

1. **Web Application**: Made with VueJs, this container provides the user interface as in figure 3.3, allowing the user to create a new chat, select a chat, and send a query to the Web Service to receive an answer for a D&D question

2. **Web Service**: Made with FastAPI, this container is responsible for receiving the query from the user and using RAG-enhanced ChatGPT-4o to respond.

3. **Vector Database**: The D&D chunks of text are transformed into vectors and stored in a vector database. This database is on RAG to give context to the LLM considering the n most similar vectors to the answer.


CRIAR NOVO CHAT
I'M NEW TO D&D, I WANT TO CREATE A CHARACTER
MY PLAYERS ARE EXPLORING A DUNGEON AND THERE'S A PIT TR
CHAT 3
Digite sua pergunta

My players are exploring a dungeon and there's a pit trap. How does fall damage work?

My players are exploring a dungeon and there's a pit trap. How does fall damage work?

In Dungeons & Dragons, fall damage is typically calculated based on the distance fallen.

Here are the key points to remember:

1. **Distance Fallen**: When a character falls, they take damage based on how far they fall. The standard rule is that a character takes 1d6 bludgeoning damage for every 10 feet fallen, up to a maximum of 20d6.

2. **Maximum Damage**: If a character falls more than 200 feet, they still take a maximum of 20d6 damage, as the rules cap the fall damage at that amount.

3. **Landing**: If a character lands on a surface that is particularly hard (like stone) or particularly soft (like a pile of hay), the DM may rule on different types of damage or adjustments to the fall damage.

4. **Reduction**: Some spells or abilities might reduce or negate fall damage. For example, a character with the Feather Fall spell can slow their descent, preventing fall damage.

Figure 3.3: *Divination Chat interface: the user interface allows the user to create a chat, select a chat, and send a query*

3.3 Components

The Web Service architecture was made using Hexagonal Architecture, separated by core, ports, and adapters.

3.3.1 Core

The core hexagon is responsible for dealing with the program business logic and represents the central part of the application. The divination core is made up of ChatService and RagChain.

1. `ChatService`: This component is responsible for receiving all the dependencies and orchestrating the interaction between them.

2. `RagChain`: This component is responsible for providing the answer to the user query

3.3.2 Ports

The port hexagon defines how the core communicates with external dependencies, specifying the possible interactions between them using abstract methods.

1. `LLMAnswerer`: Defines how the provided LLM (e.g. ChatGPT-4o) should answer the query.

2. `TemplateEnricher`: Defines how the configuration templates should be, it contains the LLM personality template and how the LLM should read the current chat history.

3. `ContextEnricher`: Defines how the context external data such as D&D books should be retrieved.

4. `Router`: Defines how the routes should be created.

5. `Repository`: Defines the methods for create and retrieve chat history for the user.

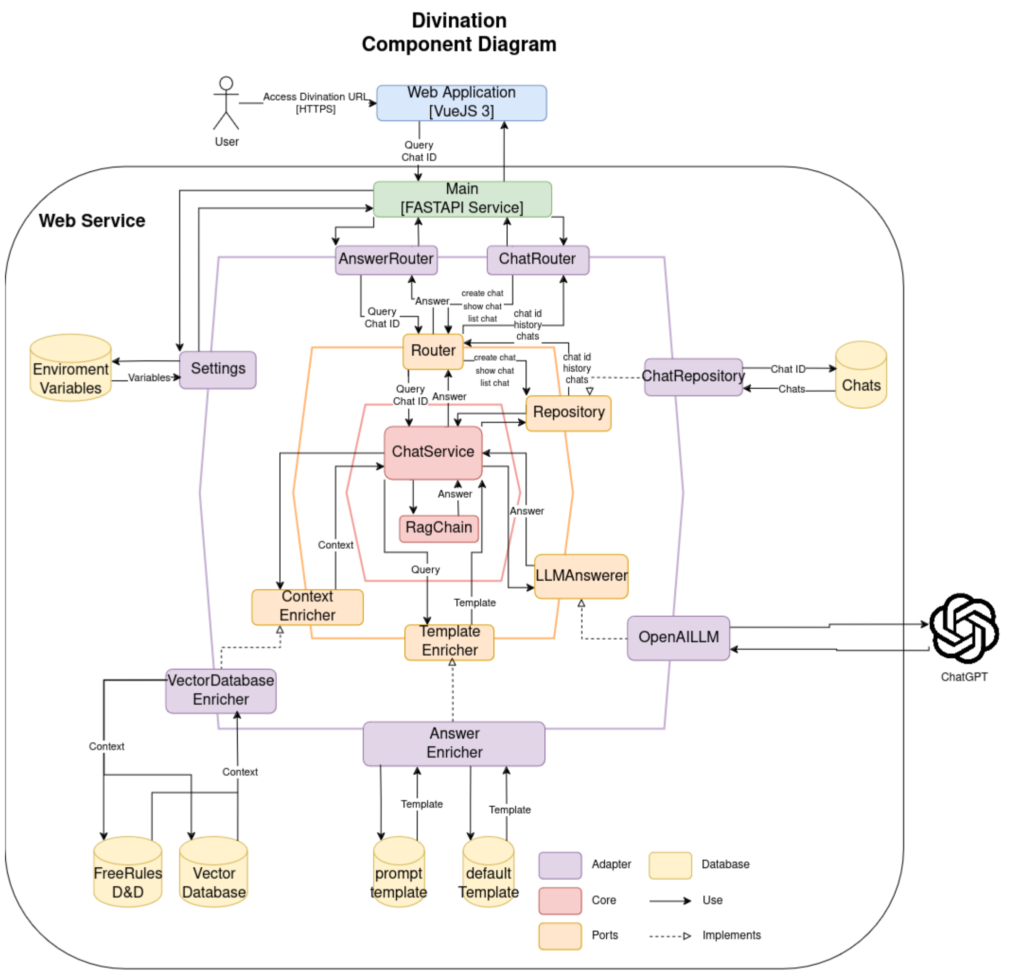


Figure 3.4: *Divination Web Service: Hexagonal Architecture component diagram of Web Service, displaying core, ports, adapters and database.*

3.3.3 Adapters

1. `OpenAILLM`: Implements the LLMAnswerer port, answering the user query using chatGPT-4o.

2. `ChatRepository`: Implements the Repository port. It creates the chats dictionary and returns char history from a specific chat or returns the dictionary.

3. `AnswerEnricher`: Implements the TemplateEnricher port. It's responsible for retrieving the context files from a database such as how the LLM should read the chat history and personality.

4. `VectorDatabaseEnricher`: Implements the ContextEnricher, part responsible for reading the context data, split into chunks and embed into a vector datastore.

5. `Settings`: Responsible for providing configuration variables to the project, such as OpenAi API key, security variables, allowed origins, etc.

6. `AnswerRouter`: Implements the Router port. It provides the Answer Routers, the first route is `/v1/context`, giving the option to change the template used for LLM personality, and the second route is the `v1/answer`, it receives a query and returns an answer.

7. `ChatRouter`: Implements the Router port. It provides the Chat Router, the first route `v1/chats` responsible for creating a new chat, the `/v1/chats/:id` route returns a chat history given a chat id, and the last route `v1/chats` returning all the chats stored in the program.

3.3.4 Database

The database is composed of configuration texts, it's composed by context texts (FreeRules D&D, VectorDatabas, promptTemplate, DefaultTemplate), environment variables, and chats.

1. `Enviroment Variables`: Composed by configuration variables, it contains the OpenAI API key, LangChain API Key, Allowed Origins, Allowed methods, Allowed Headers, Host and Port.

2. `FreeRules D&D`: Main Context used for Divination, it contains all the free rules of the game provided by the dndbeyond site.

3. `Vector Database`: After loading the FreeRules, the text is embedded into vectors and stored in a vector store. When a query is received, the query and the vectors are compared, and the nth most helpful vectors are selected to help the LLM answer the question.

4. `Prompt template`: This template is responsible for using the last user question and reformulating it with the historical messages.

5. `Default template`: instructs how the LLM should answer the question following a given personality, some of the current instructions include "You're a dungeon master assistant, and you need to follow the context as much as possible to answer the question"

6. `Chats`: Responsible for storing the program chats history into a dictionary.

3.4 Code

This section presents the code of the most important business logic code, composed of `ChatService` and `RagChain`.

3.4.1 ChatService

The ChatService component receives each dependency and orchestrates them to LLM Answer. When called by `answer route`, it gets the user query and uses it to retrieve the nth most useful vectors from `vectorstore`, retrieve the `answer template` (e.g. the bot should always try to answer according to book information and always say "thanks for asking" in the end), retrieve the `history template` (defines how to bot should use old messages) and pass the data for the next step, the `RagChain`.

```python

class ChatService:
    def __init__(
        self, context_enricher, answerer, template, chat_repository, settings
    ):
        self.context_enricher = context_enricher
        self.llm_answerer = answerer
        self.answer_template = template
        self.chat_repository = chat_repository
        self.project_settings = settings

    def get_answer(self, query, chat_id):
        context = self.context_enricher.getData(query)
        template = self.answer_template.get_template()
        history_template = self.answer_template.get_history_template()
        return self.llm_answerer.get_answer(
            chat_id,
            query,
            context,
            self.chat_repository,
            template,
            history_template,
            self.project_settings,
        )

```

3.4.2 RagChain

This component's main responsibility is to answer the user's question. It receives all the conversation context, how the bot should create a context using the chat history messages (history_retriever), and how it should respond to the user's question (question_answer_chain).

```

class RagChain:
    def __init__(
        self, chat_repository, history_retriever, question_answer_chain
    ):
        self.chat_repository = chat_repository
        self.history_retriever = history_retriever
        self.question_answer_chain = question_answer_chain

    def answer(self, query, chat_id):
        rag_chain = create_retrieval_chain(
            self.history_retriever, self.question_answer_chain
        )

        conversational_rag_chain = RunnableWithMessageHistory(
            rag_chain,
            self.chat_repository.get_history,
            input_messages_key="input",
            history_messages_key="chat_history",
            output_messages_key="answer",
        )

        answer = conversational_rag_chain.invoke(
            {"input": query},
                config={"configurable": {"session_id": chat_id}},
            )["answer"]

            return answer

```

Chapter 4

Results

The chapter will introduce the project results after testing with d&d players with Persona (Skand, 2024) scenarios.

4.1 Persona

The Persona Scenarios is a tool to help focus on who the target users are. In this project two personas were used; the player persona, and the dungeon master persona. Each persona has associated test scenarios. The test users were put to compare these scenarios using ChatGPT and the `Divination` project.

4.1.1 Player Persona

This persona focuses on a player who doesn't know yet all the mechanics of the game and uses the application to understand better

The scenario given to this persona was:

"You're a new D&D player and just discovered the existence of an auxiliary tool of D&D called `divination`. You want to create a new chat and ask how to create a character after you try to understand the interface, and then create a character with race, class, and background."

4.1.2 Dungeon Master Persona

This persona focuses on a player who knows how to DM a D&D campaign but can't manage a large amount of information, so they need to use an application to help remember some of the rules.

The scenario given to this persona was:

"Your players are exploring a dungeon and there's a pit trap. How does fall damage work? and how can they get across?

4.2 Experiments

The persona tests were performed by **seven**(7) users using Divination and ChatGPT-4o to get answers; each user acted as the Player persona and the Dungeon Master persona. After the tests, the feedback was collected using Google Forms, which are presented below. The Google Forms was composed by 7 questions:

- "How do you evaluate the usability of the tools? (Very Bad to Very Good)"
- "How do evaluate the quality of the tool's responses? (Very Bad to Very Good)"
- "When asking consecutive questions, how do you evaluate the tool's ability to reuse the context of previous interactions? (Very Bad to Very Good)"
- "In your perception, which tool provided the best results? (Very Bad to Very Good)"
- "At any point, did ChatGPT's responses generate hallucinations? (false content presented as true) (Text input)"
- "At any point, did Divination's responses generate hallucinations? (false content presented as true) (Text input)"
- "Do you have some feedback for improvement of Divination? (Text input)"

The questions can be seem in the appendix A.

4.2.1 Usability

About the usability, was asked the question "How do you evaluate the usability of the tools?". ChatGPT showed no visible major errors, a few users complained about `Divination` chat interface and suggested changes to the message history box e.g. autoscrolling, and visual changes.

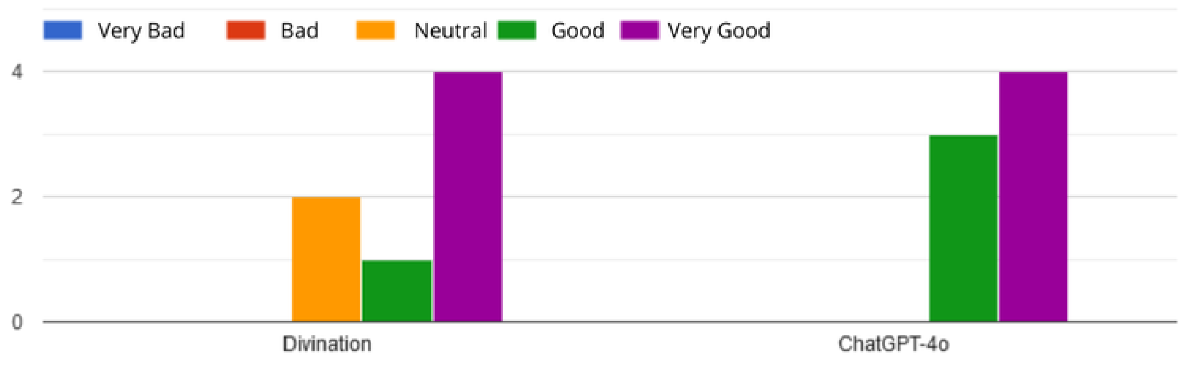


Figure 4.1: *Chart representing the user's answer for the question 'How do you evaluate the usability of the tools?'*

4.2.2 Answer Quality

About the quality of the answers, was asked the question "How do evaluate the quality of the tool's responses?". Divination gave more concise responses compared to ChatGPT-4o. The major problem pointed out by users for ChatGPT-4o was the extension of the answer, comparing it to a "wall of text".

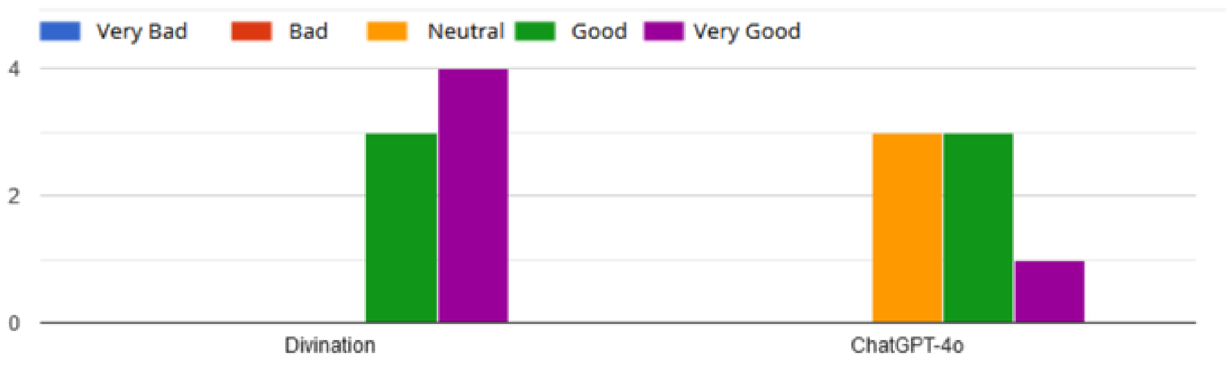


Figure 4.2: *Chart representing the user's answer for the question 'How do you evaluate the quality of the tool's responses?'*

4.2.3 Context Quality

About the context quality, was asked the question "When asking consecutive questions, how do you evaluate the tool's ability to reuse the context of previous interactions?". The chart below showed that the Divination was better than ChatGPT-4o when it comes to use already answered questions in the same conversation.

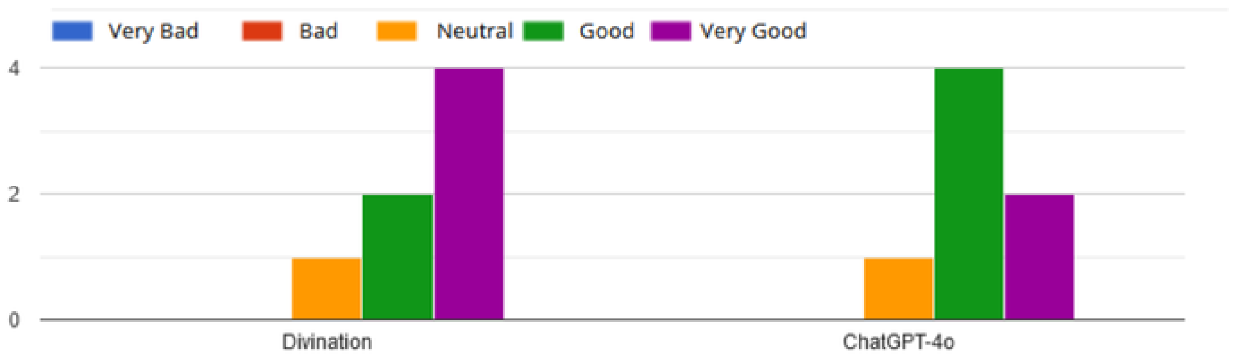


Figure 4.3: *Chart representing the user's answer for the question 'When asking consecutive questions, how do you evaluate the tool's ability to reuse the context of previous interactions?'*

4.3 Hallucinations

`Divination` showed no hallucinations, but ChatGPT-4o have mistaken some concepts about the game, such as:

1. wrong translation of the skill "lay on hands".
2. Wrong Answer stating "Surprised" is a magic effect, which is not; and
3. wrong information about "fall damage" and the number of magic spells of a mage,

All three items are in the D&D rules book.

4.3.1 General Feedbacks

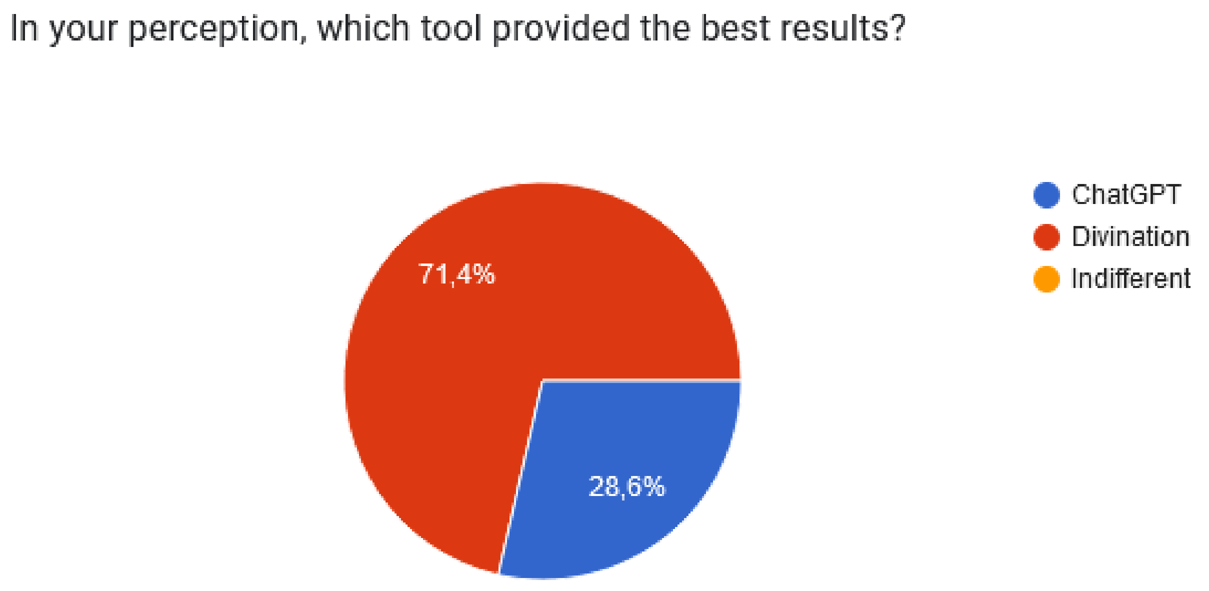
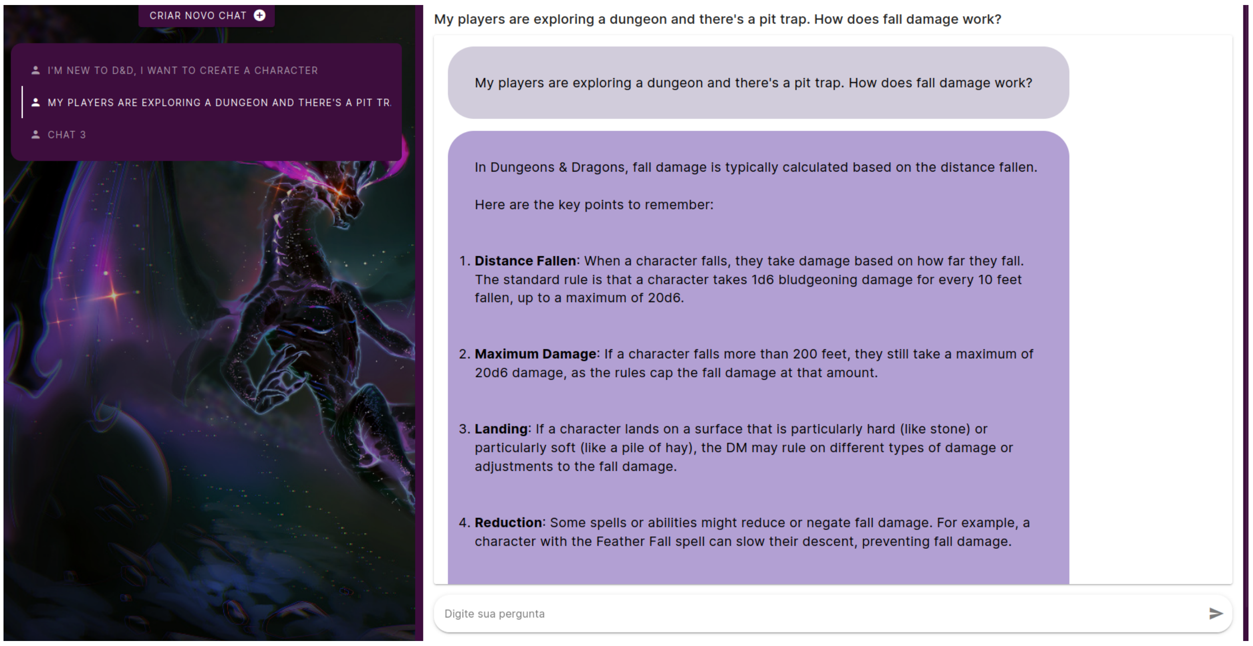


Figure 4.4

In general, the tested users preferred `Divination`, mostly because the project gave more concise answers than ChatGPT-4o, allowing the user to quickly find the necessary data, Furthermore ChatGPT showed minor hallucinations in its answers.

4.4 Examples

In the following figure 4.5, a user is illustrated using Divination to answer questions regarding the Dungeon Master Persona.


CRIAR NOVO CHAT
I'M NEW TO D&D, I WANT TO CREATE A CHARACTER
MY PLAYERS ARE EXPLORING A DUNGEON AND THERE'S A PIT TR
CHAT 3
Digite sua pergunta

My players are exploring a dungeon and there's a pit trap. How does fall damage work?

My players are exploring a dungeon and there's a pit trap. How does fall damage work?

In Dungeons & Dragons, fall damage is typically calculated based on the distance fallen.

Here are the key points to remember:

1. **Distance Fallen**: When a character falls, they take damage based on how far they fall. The standard rule is that a character takes 1d6 bludgeoning damage for every 10 feet fallen, up to a maximum of 20d6.

2. **Maximum Damage**: If a character falls more than 200 feet, they still take a maximum of 20d6 damage, as the rules cap the fall damage at that amount.

3. **Landing**: If a character lands on a surface that is particularly hard (like stone) or particularly soft (like a pile of hay), the DM may rule on different types of damage or adjustments to the fall damage.

4. **Reduction**: Some spells or abilities might reduce or negate fall damage. For example, a character with the Feather Fall spell can slow their descent, preventing fall damage.

Figure 4.5: *User testing the Divination app following the Dungeon Master Persona*

Chapter 5

Discussion

This chapter discusses the results obtained with users' feedback on ChatGPT and Divination (4).

5.1 Divination answers quality

In general, the users preferred divination answers because they were more concise and showed fewer hallucinations than ChatGPT, but due to the database being restricted to only one source of information, some answers were too short. Improvements in the interface were other points appointed by users.

5.2 ChatGPT answers quality

ChatGPT was superior to `Divination`, being more responsive and having fast answers. However, its biggest problem was the hallucinations and the over-answers to the user questions.

5.3 Limitations

`Divination` is a single-user tool. Therefore, the project could not be hosted online for test purposes. Tests were made along with the users running the project.

Another limitation was the D&D sourcebook. The project uses the open game license book, but some information only can be found on the paid versions.

`Divination` answerers were slower compared to ChatGPT answers due to the larger computational power of OpenAI.

Chapter 6

Conclusion

The goal of this capstone project was to create a tool based on LLM with RAG to help players of the game Dungeons and Dragons(D&D). This tool should answer users' questions related to the game, considering the content of it's sourcebooks.

The final tool, Divination, successfully achieved its proposed goals; it delivers an interface to create a chat and ask questions about the game. The question is sent to the backend and processed by an LLM with RAG fed with game rules. Then, the answer is shown to the user. Its answers showed less hallucinations compared to ChatGPT4-o due to the use of the RAG technique.

Some limitations were discovered during the development, such as relying on licensed data instead of the paid versions of the books, Moreover, lots of machine power are required for embedding and retrieving data from the vector store to give faster answers.

6.1 Future work

This project has open opportunities for improvements, considering the users feedback. Firstly, one way to improve the project is to turn the single-user application into a multi-user application. This change will allow better testability and the option to host on a website.

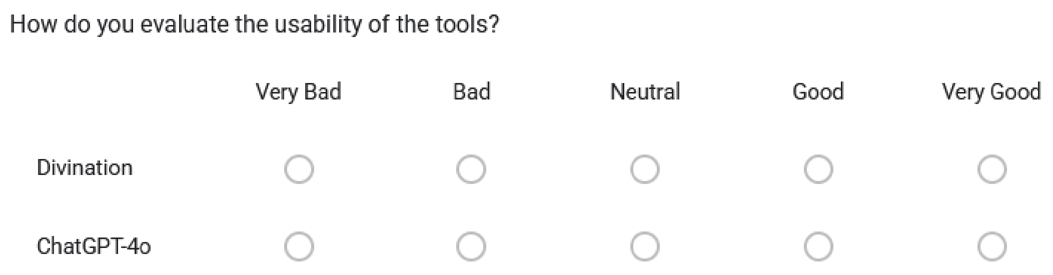
Secondly, adjusting the configuration files, e.g., the temperature of the underlying LLM, the context vector length, etc., and analyzing how these changes affect the project's output may significantly improve `Divination`'s results.

Appendix A

Survey

A.1 Google Forms Questionnaire

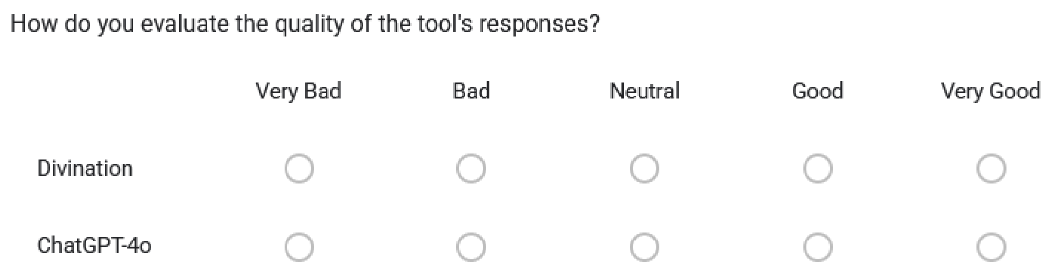
A.1.1 Question #1



How do you evaluate the usability of the tools?

	Very Bad	Bad	Neutral	Good	Very Good
Divination	○	○	○	○	○
ChatGPT-4o	○	○	○	○	○

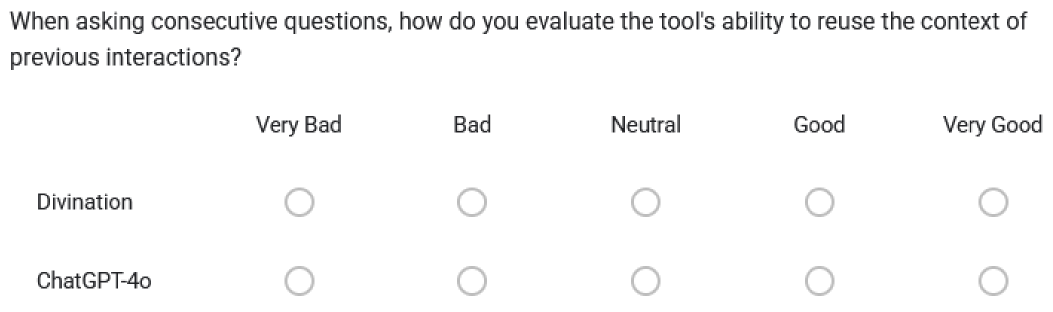
A.1.2 Question #2



How do you evaluate the quality of the tool's responses?

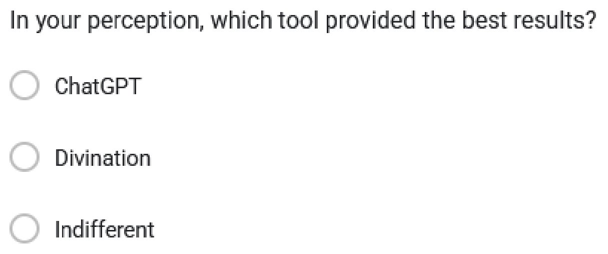
	Very Bad	Bad	Neutral	Good	Very Good
Divination	○	○	○	○	○
ChatGPT-4o	○	○	○	○	○

A.1.3 Question #3


When asking consecutive questions, how do you evaluate the tool's ability to reuse the context of previous interactions?

	Very Bad	Bad	Neutral	Good	Very Good
Divination	○	○	○	○	○
ChatGPT-4o	○	○	○	○	○

A.1.4 Question #4

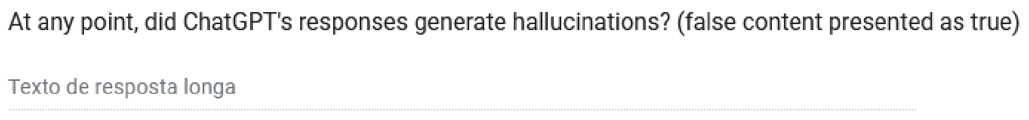

In your perception, which tool provided the best results?

○ ChatGPT

○ Divination

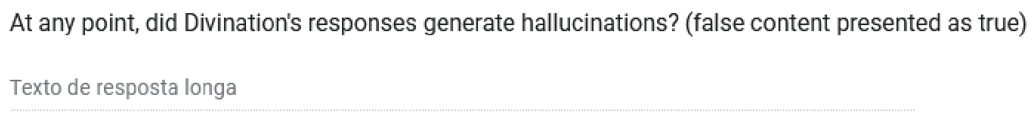
○ Indifferent

A.1.5 Question #5


At any point, did ChatGPT's responses generate hallucinations? (false content presented as true)

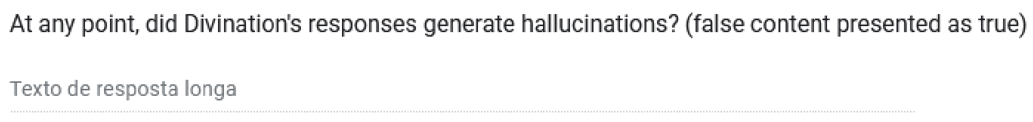
Texto de resposta longa

A.1.6 Question #6


At any point, did Divination's responses generate hallucinations? (false content presented as true)

Texto de resposta longa

A.1.7 Question #7


At any point, did Divination's responses generate hallucinations? (false content presented as true)

Texto de resposta longa

Bibliography

A. Head A. Zhu, L. Martin and C. Callison-Burch. Calypso: Llms as dungeon master's assistants. **AIIDE**, 2023. URL https://ojs.aaai.org/index.php/AIIDE/article/view/27534. 1

Simon Brown. The c4 model for visualising software architecture, 2024. URL https://c4model.com/. 5

Alistair Cockburn. Hexagonal architecture, 2005. URL https://alistair.cockburn.us/hexagonal-architecture/. 4

David Foster. **Generative Deep Learning**. O'Reilly, 1st edition, 2019. 1

Google. Gemini, 2025. URL https://gemini.google.com/app. 3

Mike. Mearls and Jeremy Crawford. **Dungeons Dragons Player's Handbook 5th edition**. Wizards of the Coast, 2024. 1

Meta. Llama, 2025. URL https://www.llama.com/. 3

OpenAI. Chatgpt, 2025. URL https://chatgpt.com/. 3

Stuart. Russell and Peter. Norvig. **Artificial Intelligence: A Modern Approach**. Pearso, 2009. 1

Kumar Skand. Crafting winning personas, 2024. URL https://www.uxmatters.com/mt/archives/2019/09/crafting-winning-personas.php. 12

J. Li T. Tang X. Wang Y. Hou Y. Min B. Zhang J. Zhang Z. Dong Y. Du C. Yang Y. Chen Z. Chen J. Jiang R. Ren Y. Li X. Tang Z. Liu P. Liu J.-Y. Nie W. X. Zhao, K. Zhou and J.-R. Wen. A survey of large language models. 2024. URL https://arxiv.org/abs/2303.18223. 1, 3

X. Gao K. Jia J. Pan Y. Bi Y. Dai J. Sun M. Wang H. Wang Y. Gao, Y. Xiong. Retrieval-augmented generation for large language models: A survey. 2024. URL https://arxiv.org/abs/2312.10997. 3