Informatyka techniczna i sztuczna inteligencja
Technical informatics and Artificial Intelligence
Engineer of XXI Century'2024

Automatization of narration in RPG games: application of advanced language models

Maksymilian Grygiel [1,*], Scientific Supervisor: Vasyl Martsenyuk [2]

[1] *Msc, Department of Computer Science and Automatics University of Bielsko-Biala, Bielsko-Biala, Poland, ma.grygiel@ubb.edu.pl*
[2] *Professor, Department of Computer Science and Automatics University of Bielsko-Biala, Bielsko-Biala, Poland, vmartsenyuk@ubb.edu.pl*
[*] *ma.grygiel@ubb.edu.pl*

Abstract: The article discusses an own solution aimed at automating the creation of scenarios in RPG games. The development of this solution involved the use of artificial intelligence technology based on large language models. Additional data was sourced from online resources. Knowledge base methods and prompt templates were employed to prepare the AI for the role of a narrator. All components of the project were created using professional open-access tools. The solution was compared with the ChatGPT tool, highlighting differences in response times and ways of interaction with chatbots. An analysis was also conducted on the compilation time of the project and the response time to user inputs, depending on the size of the knowledge base and the large language model used.

Keywords: chatbot; roleplaying game; RPG; large language model; language model; LLM; knowledge base;

Automatyzacja narracji w grach RPG: zastosowanie zaawansowanych modeli językowych

Maksymilian Grygiel [1,*], Opiekun Naukowy: Vasyl Martsenyuk [2]

[1] *Mgr, Wydział Budowy Maszyn i Informatyki, Uniwersytet Bielsko-Bialski, Polska, ma.grygiel@ubb.edu.pl*
[2] *Profesor, Wydział Budowy Maszyn i Informatyki, Uniwersytet Bielsko-Bialski, Bielsko-Biała, Polska, vmartsenyuk@ubb.edu.pl*
[*] *ma.grygiel@ubb.edu.pl*

Streszczenie W artykule omówiono własne rozwiązanie, którego głównym celem jest automatyzacja tworzenia scenariuszy w grach RPG. Do jego stworzenia posłużyło wykorzystanie technologii sztucznej inteligencji opartej na dużych modelach językowych. Dodatkowe dane zostały pobrane z zasobów sieciowych. Do nauczenia sztucznej inteligencji roli narratora wykorzystano metody bazy wiedzy oraz szablonów zapytań. Wszystkie części projektu utworzono przy użyciu profesjonalnych narzędzi typu open-access. Rozwiązanie porównano z narzędziem ChatGPT, gdzie przedstawiono różnice w czasach oczekiwania na odpowiedź a także w sposobie interakcji z chatbotami. Dokonano również analizy czasu kompilacji projektu oraz czasu udzielania odpowiedzi na wpisy użytkownika programu w zależności od rozmiaru bazy wiedzy oraz wykorzystanego dużego modelu językowego.

Słowa kluczowe: chatbot; gra fabularna; RPG; duży model językowy; model językowy; LLM; baza wiedzy;

1. Introduction

The role of narration in RPG games is crucial, as it drives the story, immerses players, and enhances the overall gaming experience. Traditionally, narration in RPGs has relied heavily on pre-written scripts and human input, limiting the flexibility and dynamism of the storytelling. However, with the advent of advanced language models, there is now potential to automate and enhance this aspect of game design. The evolution of chatbot technology, dating back to the 1960s, has laid the groundwork for this innovation [1]. Early chatbots only analyzed user input to generate responses, but it wasn't until the 21st century, with the rise of natural language processing and, more recently, transformer-based architectures, that AI could begin to approach human-like conversation [2] [3] [4] [5]. This study explores the application of these cutting-edge technologies to automate narration in RPG games, promising to revolutionize how stories are told and experienced in the gaming world.

The purpose of this work is to develop a chatbot application capable of serving as a scenario writer or narrator within a role-playing game (RPG). This approach aims to enhance the narrative flexibility and dynamism of RPGs by allowing the story to evolve in response to player actions in real time. To achieve this, a variety of advanced tools and methodologies were employed. The chatbot is hosted locally using the Ollama framework, ensuring seamless integration and performance within the gaming environment. Several state-of-the-art Large Language Models (LLMs), including Llama 2, Llama 3, Gemma, and Mistral, were utilized for their ability to generate complex, context-aware narratives. To fine-tune the chatbot's behavior and ensure it aligns with the desired narrative style, the LangChain framework was applied, allowing for customizable and adaptive responses. Additionally, custom data was incorporated to provide the necessary context for the chatbot, ensuring that it can generate content that is not only coherent but also deeply integrated with the game's world and storyline.

The article "An Overview of Chatbot Technology" by Eleni Adamopoulou and Lefteris Moussiades outlines the process of how chatbots function and how they are created from scratch [6]. When designing a chatbot, it is essential to decide which features it will offer. This decision allows AI developers to more easily select the appropriate solutions in the form of algorithms, platforms, and tools needed to create the chatbot. Simultaneously, this decision helps end-users understand what to expect from the chatbot [7].

When a user inputs a query into a chatbot, the Understanding Component undertakes the task of interpreting the received text [8]. Using this interpreted query, the chatbot then accesses its resources to retrieve the data needed to formulate a response. These resources may include databases known as Knowledge Bases or APIs.

The Response Generation Component uses Natural Language Generation (NLG) to create a human-like response based on the user's intent and context [9]. Generative models are the most advanced and are considered to simulate human responses effectively. They utilize machine learning algorithms and deep learning techniques. These models are based not only on the user's current query but also on their previous inputs. After generating the response using the selected method, the Dialogue Management Component updates the conversation's context.

Large Language Models (LLMs) are types of neural networks used in artificial intelligence. They are employed in the design of advanced systems for understanding and generating text that resembles human language. This is achieved by training these models on the statistical relationships between words in a text and teaching them to predict the next words or tokens. These processes are known as self-supervised learning (SSL) or semi-supervised learning (also referred to as weak supervision) [10].

A key term in the context of language models is neural networks. These are computational systems that use algorithms to process information. They utilize neurons that process input data using activation functions and pass the result to subsequent neurons. Additionally, they have the capability to learn from the available data [11].

Deep learning is another important topic in the context of large language models, as it plays a significant role in neural networks and is one of the methods employed in machine learning. It is a type of neural network in which artificial neurons are divided into layers. Each layer is responsible for inputs and outputs. In each layer, data is processed to generate information, which is then used by the subsequent layer for its own tasks. This process enables the artificial intelligence model to learn by processing its own data [12].

Neural networks for large language models are categorized based on their architecture. The most common is transformer-based architecture, often combined with decoder-only architecture. This approach involves learning the context of data while simultaneously retaining its meaning. It achieves this by identifying relationships within sequential data. For example, it connects words and their meanings by utilizing the entire context of the sentence in which they appear [13] [14].

In summary, the primary aim of this work is to develop a solution that functions as a scenario writer in RPG games by utilizing large language models to create an advanced chatbot. The principal conclusion is that an approach combining a knowledge base, selected large language models, and tailored prompt templates successfully achieves the objective of creating an immersive gaming experience.

2. Materials and Methods

To enable the chatbot to run locally, the Ollama tool was utilized, allowing for the download and implementation of selected large language models (LLMs) directly onto the machine. The LLMs used in this project were Llama 2, Llama 3, Gemma, and Mistral.

A custom application was developed in Python, a widely adopted programming language in the field of artificial intelligence, due to its robust libraries and community support. The LangChain library was integrated into the application to facilitate the personalization and fine-tuning of the models, providing flexible options for tailoring the chatbot's behavior to specific requirements. Additionally, the FAISS (Facebook AI Similarity Search) library was employed to create a language base, optimizing the chatbot's ability to retrieve and process relevant information.

The implementation process began with initializing the selected LLM within the application. The following code snippet demonstrates this initialization:

```
llm = Ollama(model="llama2",
callback_manager=CallbackManager([StreamingStdOutCallbackHandler()]))
```

This initialization step sets up the LLM to be used, with a callback manager configured to handle streaming outputs, ensuring real-time processing and interaction capabilities within the chatbot.

The next step involved loading the data that would serve as the chatbot's knowledge base. The prepared dataset contained information on role-playing games that were deemed valuable for the chatbot during user interactions. This dataset included details on games played between players, as well as inputs from game masters, providing a rich source of contextual information for the chatbot. The knowledge base was stored in a .txt file format within the solution.

The application was programmed to load this file, convert its contents into a usable format, and then split the text into manageable chunks to form vectors for efficient processing. The following code snippet demonstrates this process:

```
loader = UnstructuredFileLoader("./dataSmall.txt")
docs = loader.load()
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=500)
documents = text_splitter.split_documents(docs)
vector = FAISS.from_documents(documents, embeddings)
```

Here, the UnstructuredFileLoader is used to load the raw text data. The RecursiveCharacterTextSplitter then splits the document into chunks of 1,000 characters with a 500-character overlap, ensuring that the context is preserved across chunks. Finally, the FAISS (Facebook AI Similarity Search) library is used to create vectors from these chunks, enabling efficient retrieval and processing by the chatbot.

To define the chatbot's behavior at the launch of the project, a prompt template was employed. This was accomplished using the ChatPromptTemplate.from_messages() method, where key characteristics of the chatbot's behavior were specified. The following code illustrates this setup:

```
initial_prompt_template = ChatPromptTemplate.from_messages([
    ("system", '''Be my Dungeon Master in a Dungeons and Dragons game.
    Assume the role of an expert on the works and literary style of a high-quality novel.
    Give a narrative description of everything that follows, based on my input.
    Provide suitable names for other characters and places.
    Have characters always use dialogue when interacting with me.
    Always conduct all conversations and dialogues in quotation marks in the style of a high-
quality novel.'''),
    ("user", "{input}")
])
```

In this setup, the system prompt instructs the chatbot to act as a Dungeon Master in a Dungeons and Dragons game, emulating the narrative style of a high-quality novel. This configuration ensures that the chatbot generates immersive and contextually appropriate responses during user interactions.

The next step involves defining the chat function. In this method, the user's input and the conversation history are processed. The program then searches through the knowledge base using a similarity search to find relevant information, which is incorporated into the chatbot's prompt template to enhance the response.

```
def chat(user, conversation_history):
    playersAction = user
    conversation_history.append(playersAction)
    search = vector.similarity_search(playersAction, k=3)
    context = "\n".join([doc.page_content for doc in search])
    history = "\n".join(conversation_history[-5:])

    if len(conversation_history) == 1:
        prompt_template = initial_prompt_template
    else:
        prompt_template = default_prompt_template

    template = prompt_template.messages[0].prompt.template + '''
    Here is additional data to help you create a better answer.
    Context for player's action: {context}
    Player's action: {question}'''
```

In this function, the user's action is appended to the conversation history. The vector.similarity_search() method retrieves the top three most relevant pieces of context from the knowledge base, which are then compiled into a single context string. Depending on whether it's the first user input or a subsequent one, the appropriate prompt template (initial_prompt_template or default_prompt_template) is selected. The template is then enriched with additional context and the player's action to guide the chatbot's response generation.

The final component of the application is the chain definition, which specifies the sequence of resources that the chatbot should use to generate responses:

```
chain = initial_prompt_template | llm | output_parser
```

This chain ensures that the chatbot follows the prompt template, leverages the selected large language model (LLM), and parses the output appropriately before delivering a response.

3. Results

The custom solution successfully fulfilled the objective of acting as a story narrator for role-playing games. Three analyses were conducted to evaluate its performance. The first analysis compares the custom solution with a popular chatbot, ChatGPT, in its ability to act as a game master in an RPG setting. The second analysis focuses on comparing response times when using different large language models. The third analysis examines compilation times with varying sizes of the knowledge base.

3.1. Comparison with ChatGPT

In this analysis, a comparison was made between the custom solution and ChatGPT. The comparison focused on the initial stages of the game, where the user begins the role-playing session. For the chatbot to progress, it needs context regarding the game, particularly the setting of the campaign and the character the user is playing. The custom solution was designed to include this information, so the user does not need to input any data before starting the game. In contrast, ChatGPT, being a general-purpose chatbot, requires the user to manually provide information about their role-playing campaign before the game can begin.

Another aspect compared was the response time. Since the custom solution is hosted locally, its response time depends on the processing speed of the hosting PC. On average, a modern personal computer using Llama 2 as the chosen language model returns a response within approximately 15 seconds. On the other hand, ChatGPT typically responds immediately after the user submits their prompt.

3.2. Comparison of response times with different large language models

Response time is a crucial factor in a chatbot's performance, as prolonged delays can disrupt the user experience and hinder the natural flow of conversation. An analysis was conducted to evaluate the response times across different large language models. Four models were tested, each selected for their similarity in parameter count. The same prompt was used for all models, with each chatbot asked to respond ten times.

Table 1. Results of response time measurements for different large language models.

Large Language Model	Average Time (s)	Median (s)
Llama2 7B	12.1	12.3
Llama3 8B	27.73	28.3
gemma 7B	24.52	24.05
mistral 7B	22.94	20.4

The analysis shows that Llama 2 is the fastest LLM among those compared, with an average response time of 12 seconds, despite Llama 3 being a newer and more advanced model. The other models could not achieve response times lower than 20 seconds.

3.3. Comparison of project compilation times using different sizes of knowledge bases.

For certain use cases, the time required for the program to compile and launch may be a significant consideration. It was found that the size of the knowledge base has the most substantial impact on compilation time. Three knowledge base files were created, each with different sizes: 2 kilobytes (small), 65 kilobytes (medium), and 264 kilobytes (large). The program was launched 10 times with each knowledge base, and the compilation time was measured.

Table 2. Results of compilation time measurements for different sizes of knowledge bases.

Knowledge Base	Average Time (s)	Median (s)
Small (2kb)	14.813	14.395
Medium (65kb)	263.9	263.005
Large (264kb)	1118.097	1114.865

The results show a strong correlation between the size of the knowledge base and the program's compilation time. With the smallest knowledge base, the program launches in about 15 seconds. The medium-sized knowledge base (65 kb) increases this time to 264 seconds—an almost 19-fold increase in wait time for a 32-fold increase in data. The largest knowledge base (264 kb) results in a compilation time of 1118 seconds (18 minutes and 38 seconds), representing an 80-fold increase in time for 132 times more data.

Reference

1. TechTarget. "What is a chatbot?" Available online: https://www.techtarget.com/searchcustomerexperience/definition/chatbot (Accessed on 27.05.2024)
2. Tomáš Zemčík "A brief history of chatbots". DEStech Transactions on Computer Science and Engineering 10, 2019, pages 15-17.
3. Shawar, Bayan Abu, Atwell, Eric "Fostering language learner autonomy through adaptive conversation tutors". Proceedings of the The fourth Corpus Linguistics conference, 2007, Volume 3, pages 186-193.
4. Arisoy, E., Sainath, T. N., Kingsbury, B., & Ramabhadran, B. "Deep neural network language models". Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT, 06.2012, pages 20-28

5. Graves, A., & Graves, A.. "Long short-term memory". Supervised sequence labelling with recurrent neural networks, 2012, 37-45.
6. Adamopoulou, E., & Moussiades, L. "An overview of chatbot technology". "IFIP international conference on artificial intelligence applications and innovations", 2020, pages 373-383.
7. Nimavat, K., & Champaneria, T. "Chatbots: An overview types, architecture, tools and future possibilities". Int. J. Sci. Res. Dev, 5.7, 2017, pages 1019-1024.
8. Kucherbaev, P., Bozzon, A., & Houben, G. J. "Human-aided bots". IEEE Internet Computing, 22.6, 2018, pages 36-43.
9. Singh, S., Darbari, H., Bhattacharjee, K., & Verma, S. "Open source NLG systems: A survey with a vision to design a true NLG system". Int J Control Theory Appl, 9.10, 2016, pages 4409-4421.
10. Radford, A., Wu, J., Amodei, D., Amodei, D., Clark, J., Brundage, M., & Sutskever, I. "Better language models and their implications". 2019, *OpenAI blog*, *1*.2.
11. Tadeusiewicz, R., & Szaleniec, M. "Leksykon sieci neuronowych". Projekt Nauka. Fundacja na rzecz promocji nauki polskiej. Poland, 2015.
12. Microsoft Learn. "Deep learning vs. machine learning in Azure Machine Learning". Available Online: https://learn.microsoft.com/en-us/azure/machine-learning/concept-deep-learning-vs-machine-learning?view=azureml-api-2
13. Vaswani, A. (2017). Attention is all you need". 2017.
14. Min, E., Chen, R., Bian, Y., Xu, T., Zhao, K., Huang, W., ... & Rong, Y. "Transformer for graphs: An overview from architecture perspective". 2022.