Enhancing User Experience in Games with Large Language Models

Ciprian Paduraru[1], Marina Cernat[1] and Alin Stefanescu[1,2]

[1]*Department of Computer Science, University of Bucharest, Romania*
[2]*Institute for Logic and Data Science, Romania*

Keywords: Large Language Models, Retrieval Augmented Generation, Teacher Model, Fine-Tuning, Video Games, Active Assistance, Simulation Applications.

Abstract: This paper explores the application of state-of-the-art natural language processing (NLP) technologies to improve the user experience in games. Our motivation stems from the realization that a virtual assistant's input during games or simulation applications can significantly assist the user in real-time problem solving, suggestion generation, and dynamic adjustments. We propose a novel framework that seamlessly integrates large-scale language models (LLMs) into game environments and enables intelligent assistants to take the form of physical 3D characters or virtual background entities within the player narrative. Our evaluation considers computational requirements, latency and quality of results using techniques such as synthetic dataset generation, fine-tuning, Retrieval Augmented Generation (RAG) and security mechanisms. Quantitative and qualitative evaluations, including real user feedback, confirm the effectiveness of our approach. The framework is implemented as an open-source plugin for the Unreal Engine and has already been successfully used in a game demo. The presented methods can be extended to simulation applications and serious games in general.

1 INTRODUCTION

Following the current development in the field of Large Language Models (LLMs), our primary contribution in this work is to explore and develop approaches that allow the end user to benefit from their potential in games or simulation applications in general. The main problem we are addressing is how to support the user at runtime with the help of an assistant who can be asked either by text or voice. The entity assisting the user can be represented virtually in the simulation environment, e.g. a non-playable character (NPC) or a narrator/environmental listener.

Use Cases. Together with industry partners, we investigated use cases for the inclusion of such an assistant in games and simulation applications in general. The first category investigated relates to the classic *sentiment analysis* (Wankhade et al., 2022). An example of this is when a user is playing with an NPC and states either verbally or in writing that the difficulty of the simulation is too high or not challenging enough. In this case, one option would be to dynamically change the difficulty level so that the user has an engaging experience. Another concrete example: Imagine the user is playing an ice hockey game like NHL[1] and is unhappy with the referee's decision. Reactions that he expresses loudly could prompt the referee in the game to penalize the user's actions in order to entertain and motivate him.

Furthermore, we have discovered that NLP techniques can be used to construct NPC companions that physically exist in the simulated environment and can be prompted by a voice or text command to help the user. Specific examples are when the user is stuck at a destination or is unable to find the way to a particular destination. In this situation, the NPC companion can understand the user's request and guide them to a specific destination or give advice on how to solve various tasks. Another source of irritation for users when dealing with virtual worlds is their inability to understand the many designed dynamics. This is a common problem in the industry that leads to users abandoning the applications before the developers can make revenues or provide an adequate gaming experience. In this scenario, we see NLP as a viable option where the user can ask questions that a chatbot can answer live to assist them. Specific examples include questions about healing procedures, finding certain items in a particular room, determining what is needed or missing to achieve certain goals, etc.

[1]https://www.ea.com/en-gb/games/nhl

Paduraru, C., Cernat, M. and Stefanescu, A.
Enhancing User Experience in Games with Large Language Models.
DOI: 10.5220/0012854600003753
In *Proceedings of the 19th International Conference on Software Technologies (ICSOFT 2024)*, pages 293-304
ISBN: 978-989-758-706-1; ISSN: 2184-2833

Contributions. Our main research goal is to create a novel framework that enables game developers to use LLMs in their games in a reusable and efficient way in terms of computational power. To the best of our knowledge, this is the first work that addresses the problem of integrating LLMs into games for real-time inference that are locally deployed and fulfill developers' requirements in terms of use cases and technologies. The contributions are summarized below: Our contributions are summarized below.

- A reusable and flexible open-source framework for game developers (and simulation applications in general) to integrate LLMs and related processes into their products. We refer to this as *GameAssistant* and its open-source repository https://github.com/AGAPIA/NLPForVideoGames is intended to act as a plugin for the Unreal Engine[2], a game engine widely used in games, the simulation industry and academia. User input can be in the form of voice or text messages.
- Prototyping and evaluation of different techniques to reuse processes and tools in the LLM domain for simulation software and video games, such as Retrieval Augmented Generation (RAG), agents and tools, fine-tuning pipelines.
- A mechanism to incorporate a teacher LLM model to create a synthetic dataset with rigorous structure, input and output for customized game actions. This is used to reduce the cost of obtaining a fine-tuning dataset.
- Address the efficiency problem of integrating LLMs into games for end users without incurring additional cloud costs or requiring a dedicated GPU, running everything on the machine where the game is played. Our solution is based on evaluating and reusing small models that are further trained (fine-tuned) depending on the use case to achieve the required quality. The model currently used is Phi-3-mini (Abdin et al., 2024), a language model with 3.8 billion parameters recently released by Microsoft. It is integrated in a plugin-compatible form so that other extensions can also be integrated.

The rest of the work is structured as follows. Section 2 shows use cases of NLP applications and large language models in different domains that have explored similar solutions in different environments. Section 3 gives a sketch presentation of the field of LLMs, both theoretical and practical, including the motivation for the current model choice. A general overview of the supported features is shown in Section 4. The architecture and implementation of the GameAssistant framework are presented in Dection 5. The evaluation results are presented in Section 6, which includes both a quantitative and a qualitative evaluation as well as information on our setup, observations and datasets. The final section summarizes our findings and suggestions for future work.

[2]https://www.unrealengine.com

2 RELATED WORK

Conversational Agents (CA) are widely used in various industries, including medicine, the military and online shopping, as reported in the review study (Allouch et al., 2021). These agents are usually virtual agents that attempt to communicate with interested people and answer their queries, at least until they receive information from them, which is then passed on to actual human agents. Recently, LLMs have improved the capabilities of traditional NLP techniques in building virtual agents. The authors of (Guan et al., 2023) investigate the application of LLMs to improve the natural language understanding and reasoning capabilities of intelligent virtual assistants. The researchers present an innovative LLM-integrated virtual assistant capable of autonomously performing multi-step tasks within mobile applications in response to high-level user commands. The implemented system, which is at its core a mobile payment application, provides a comprehensive solution for interpreting instructions, deriving goals and performing actions. The article in (Cascella et al., 2024) explores the potential uses of LLMs in healthcare, focusing on their role in chatbots and systems that interact in the management of clinical documentation and medical literature summarization (also known as Biomedical NLP). The main hurdle in this area is research into their application in diagnostics, clinical decision support and patient triage. The reported results are from one year of using LLMs in real-life situations. CAs were also used for gamification reasons. In (Yunanto et al., 2019), the authors propose an educational game called *Turtle Trainer* that uses an NLP method for its non-playable characters (NPCs). In the game, the NPCs can automatically respond to other players' questions in English. Human users can compete against NPCs, with the player who answers the most questions correctly winning a round. While their methods for understanding and answering questions are based on standard NLP methods, we adopt their scoring strategy, which is based on qualitative feedback from two perspectives: (a) How do human users perceive how well their opponents,

i.e. NPCs, understand and answer the questions? (b) Does the existence of NPCs in this way indicate a stronger interest in the learning game itself? Duolingo (Munday, 2017) is a common platform for learning different languages. It supports the development of educational games, such as language learning through gamification , as well as machine learning-based approaches to performance testing. Using the platform itself and NLP methods, the authors recommend the use of automatically generated language tests that can be scored and psychometrically analysed without human labour or supervision.

LLMs in Games. The work that comes closest to our goals is (Paduraru et al., 2023), in which the author uses classical NLP techniques based on similarity metrics and intent models to implement a chat assistant in video games. However, they do not consider an ongoing conversation or any additional context and are strict regarding the phrases and natural language variations that can be used. In comparison, our work integrates a comprehensive language model that has the following features: a) fast adaptation to different user styles by training massive datasets containing different language formats, b) support for variable context that can be used as knowledge with the RAG method, and c) full support for the history of messages during a conversation. A motivating related work is presented in (Isaza-Giraldo et al., 2024), in which the authors explore LLMs as evaluators in serious games, focusing specifically on open-ended challenges. The researchers developed a prototype sustainability game about energy communities and tested it with ChatGPT-3.5, finding that it scored 81% of player responses correctly and provided valuable learning experiences. The results suggest that LLMs have the potential to act as mediators in educational games and that they can facilitate the development of game prototypes through natural language prompts.In (Cox and Ooi, 2024), the authors discuss how the capabilities of LLMs can be used to improve the dynamics and variety of conversational responses of NPCs in video games. The paper explores what guidelines can be created for designers to effectively incorporate LLMs into NPC dialogs and the potential impact from different perspectives to drive further research and development efforts. We follow their study and integrate their observations into our current work. In (Huber et al., 2024), the authors discuss the transformative potential and related challenges of LLMs in education and how these challenges could be addressed with the help of game and game-based learning. The paper explores how a new pedagogy of learning with AI could utilize LLMs for learning by generating games and gamifying learning materials. The work in (Gallotta et al., 2024) explores the diverse functions that LLMs can perform within a gaming context and provides an overview of the latest advancements in the various uses of LLMs for gaming purposes. The authors also delve into areas that have not been fully explored and suggest potential avenues for the future application of LLMs in gaming. The work strikes a balance between the capabilities and constraints of LLMs in the realm of gaming. It offers an understanding of how an LLM can be incorporated into a broader system for sophisticated gameplay. The authors express optimism that this article will lay the groundwork for pioneering research and innovation in this rapidly evolving field. In comparison to ours, their work is a survey or highglights of potential without ofering a technical deep solution or architecture. We reuse their presented ideas and future and ofer a technical framework that can satisfy their discussed requirements.

Agents and Tools. The main goal of these elements is to bridge the gap between user requests and backend systems through tasks expressed in natural language. There are numerous strategies in this area. The technique we have included in our project is one that embeds reasoning traces and task-specific actions into language models (Yao et al., 2023). This combination allows the model to gain a comprehensive understanding of the tasks and dynamically adapt its behavior based on the state of the backend system and the context of the conversation.

We have also experimented with another method, known as (Schick, 2023), which offers several advantages and disadvantages compared to the previous method. In their study, a language model (LLama 2 7B (Touvron et al., 2023)) is trained specifically for independent interaction with external tools via simple APIs. By carefully selecting APIs, incorporating calls at the right time, and seamlessly integrating their results into subsequent token predictions, Toolformer effectively reduces the distance between language modeling capabilities and practical tool usage. However, we could not fully implement this approach as it requires a dataset of API call examples, which we could not create in time. A detailed comparison is planned for future work.

When it comes to large systems and APIs, a more appropriate strategy might be to include Gorilla (Patil et al., 2023). In their research, the authors fine-tuned a language model that outperforms GPT-4, especially in the area of API request formulation. Gorilla, through the use of APIs and integration with a document retriever, outperforms other methods in terms of adaptability to document changes during testing. This adaptability significantly increases reliability and applicability across a wide range of tasks, especially for

large projects.

Retrieval Augmented Generation (RAG). To support RAG in our application, we use the Faiss library (Douze et al., 2024), (Johnson et al., 2019), which is designed for fast vector similarity search. It provides a comprehensive toolkit with indexing techniques and associated primitives for tasks such as search, clustering, compression and vector transformation.

3 A CONTEXTUAL INTRODUCTION TO LLMs

Language Models (LMs) are statistical models that assign probabilities to word sequences by capturing the inherent structure and patterns of natural language. An autoregressive language model (LLM), on the other hand, predicts the next token in a sequence based on the previous tokens. The main goal is to train a model that improves the likelihood of text data.

Consider a text sequence represented as $(s_1, s_2, ..., s_T)$, where s_t denotes the token at position t. The probability of predicting w_t considering the previous context $(s_1, s_2, ..., s_{t-1})$ is denoted by $P(s_t|s_1, s_2, ..., s_{t-1})$. The typical objective function in this case is to minimize the cross entropy loss, i.e., to maximize the conditional probability associated with a given text sequence, Eq (1).

$$L_{LM} = \frac{1}{T}\sum_{t=1}^{T} -\log P(s_t|s_1, s_2, ..., s_{t-1}) \quad (1)$$

Some of the well-known applications of autoregressive language models are: text generation, translation, code auto- completion, question-answering.

Architectures. Currently, the known LLMs are based on the transformer (Vaswani et al., 2023) architecture and different variants of self-attention mechanisms. There are three known architectures: encoder only, encoder-decoder and decoder only. The models we experiment with in this paper fall into the decoder-only architecture class, which uses only the decoder component of the traditional transformer architecture. In contrast to the encoder-decoder configuration, which contains both an encoder and a decoder, the decoder-only architecture focuses solely on the decoding process. It generates the tokens sequentially, taking into account the previous tokens in the sequence. This approach has proven itself for tasks such as text generation, as it eliminates the need for an explicit encoding phase. It is also suitable for our use case in which the chatbot receives a context, a user task, and has to generate a response in the form of a text.

Embeddings. The text representation must be converted to a sequence of floating point numbers to be useful for training LLMs with transformers. This representation is called *embedding*. It involves moving a sequence of N words into a space of $\mathcal{R}^{N\times D}$, where D is the dimension used to embed each word. Intuitively, the goal is to give each word a fluent representation such that the *distance* or *angle* between two similar words is small and large when the semantics of the words are different.

We briefly mention some other concepts used along Section 5. *Prompt Engineering* is a technique used to control the responses of Large Language Models (LLMs) by formulating specific statements or questions to elicit the desired results from the model. As shown in the literature, providing examples in a specific order and selecting the content for the prompt is important for the quality of the results (Marvin et al., 2024). *Retrieval Augmented Generation (RAG)*, is a method that extends the capabilities of LLMs by incorporating knowledge from external sources, generally outside the training data used by the LLM. This helps to improve the accuracy and credibility of the generated content and avoid hallucinations. (Zhao et al., 2024). *agents* refer to entities that can think, plan and act individually when prompted with a query by querying the LLM. The original problem is split into smaller parts, some of which are solved using *tools* or *functions*, which are interfaces provided by the backends of the system. In this way, the LLM is able to access external functionalities and provide better quality answers, again avoiding hallucinations.

Fine-Tuning. The LLM models are trained in a series of steps: a) preparing a training corpus of data, b) pre-training the LLM, c) fine-tuning the model using specific data or tasks. Since our work reuses the Phi-3 models, we start with step c) and use general supervised fine-tuning (SFT). Here, the baseline LLM obtained after steps a) and b) is further trained using a dataset of (instruction, output) pairs. This process of data set acquisition and fine-tuning is explained in detail in Section 5.5.

Motivation for Fine-Tuning. The efficiency of LLMs can be significantly improved by using domain-specific datasets (OpenAI et al., 2024) for applications that require a combination of general and domain-specific language as well as the understanding of technical terms. The efficiency of smaller fine-tuned models and their performance after fine-tuning is also described in the literature, and two main cases

are of interest for our work.

a) Training of models for code generation from general language models, (Rozière et al., 2024), (Li et al., 2023)

b) Fine-tuning of models for interaction with external agents and tools (Schick, 2023), (Schick, 2023), (Patil et al., 2023) on different data sets and use cases.

Our methods follow these two themes in fine-tuning the foundation LLM to achieve the proposed goals. The former is used by the framework to make the LLM aware of static content of a game that can be presented in a textual format (e.g. manuals, guides, Youtube transcripts, etc.), while the latter trains the LLM to invoke functionalities exposed by the game developer in a decoupled way. Sections 5.3 and 5.5 present the technical details.

4 FRAMEWORK OVERVIEW

Assistant Features. After discussions with the game developers, we have set up some concrete use cases where an assistant can be used in the game either in physical form (NPC character) or without one (narrative character). Note that the implementation of the proposed framework is not limited to the feature list, but that its architecture and components have been developed with generality in mind.

1. Question Answering.

   This category is about situations where the user needs help or support. Two subcategories have been identified:

   - Context-related situations during a game situation: e.g. *How can I defend myself efficiently against this fast attacker? - during a football match.*
   - General questions that are in a loaded manual and do not depend on the game: e.g. *In this UI window, where do I find the ammo upgrade?, "Which buttons do I have to press for a chip shot to trick the goalkeeper?".*

2. Explicit actions requests.

   These are requests from the user to the game wizard to perform various actions. Typical use cases are: (a) requesting hints or solutions to solve puzzles in contextual situations, (b) requesting a companion NPC (in this case the physical assistant) to show a path to a destination that the user has difficulty reaching, (c) helping to eliminate an enemy character, or even (d) changing settings to improve performance (e.g. rendering options) or volume options without going through menus.

3. Actions derived through sentiment analysis This category includes cases where the user requests or responds to a change in settings but does not specify exactly what they want. For example, the user could simply say by voice: *This opponent is too hard to defeat/too easy*. In this case, the intention is not clearly defined, but a sentiment analysis module tries to understand it and transmit it as input to the assistant to then take appropriate action. Another use case that is required from an industry perspective is to capture automatically detected defects or problems. For example, if a visual artifact or incorrect physics simulation is displayed in the current state of a game and the user vocalizes this, a quick capture of memory dumps and screenshots can be grabbed and sent to the developers as an open issue.

Input. The player can enter a command for the assistant via voice or text chat, Figure 1. In the case of a voice message, it is first converted into text using the OpenAI Whisper-tiny speech-to-text model (Radford et al., 2022). The GameAssistant then uses both the PlayerState information (e.g. the user's current situation as represented and exported by the game) and the data provided by the game (e.g. the current context the user is in, available environments, static information such as manuals or user guides, etc.) to analyze and answer the user's questions. The extracted data is used for the LLM within the GameAssistant component to provide a more targeted response, either in the form of a message or an action or a change in game state. Section 5 provides details on the implementation of these interactions and components. To increase the generality of the interaction between the GameAssistant, the user's state (*PlayerState*) and the game data, the framework keeps the communication as decoupled as possible by using interfaces.

Base Language Model. One of the main criteria in selecting a base model for further fine-tuning was the required computing resources and latency in responding to requests, taking into account the available end-user hardware. With these considerations in mind, we opted for Phi-3 (Abdin et al., 2024), a model with 3.8B (billion) parameters. In particular, we opted for the version with token windows of up to 4096 tokens instead of the 128K version, as it fulfils the goals of the assistance function while being cost-efficient. A detailed comparison to motivate the choice is presented in Section 6.

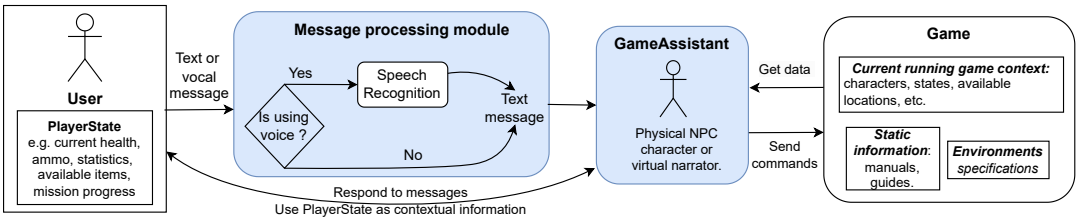

Figure 1: For an overview of the interaction between the components when the user, who is in a certain state during a game, sends a message either verbally or via text. The first step in the pipeline is to convert this message into a text representation and then pass it on to the GameAssistant framework, which uses NLP techniques to extract information from both the game and the user's state and then react or perform actions depending on the situation.

5 ARCHITECTURE AND IMPLEMENTATION

The architecture and components are designed to use a plugin pattern. This allows the integration of higher capacity models or other fine-tuning methods, as well as the ability to enable or disable different functionalities and components depending on the requirements of specific use cases. This approach corresponds to a principle of software engineering known as separation of concerns, which we want to take into account when developing the framework. Figure 2 illustrates the interaction between the system and its components during a user conversation. The LangChain[3] APIs are used to establish connections between the component interfaces, monitor the conversation and manage the message flow.

5.1 Conversation Handling

At each time t, a conversation history H_t is kept so that the assistant is able to respond in the context of previously requested messages. When a new user message Q is recognized, representing a request such as a question, an action request or a response from the user, the first step is to ask the LLM model to create a standalone request that contains both the history and the new user's request aggregated in a single prompt. In the figure, a concrete example is given by the element Q_c. In this step, the ability of LLMs to summarize long texts is used. If the input text is longer than the 4096 token limit, only the last part of the conversation is kept within this limit. Our experiments have shown that in a conversational environment, it is beneficial to store a limited number of the last conversational messages and forget what the context of the discussion was for more than the last 10 messages. To obtain Q_c, a summary request is made to LLM via a prompt template as specified in Listing 1.

[3]https://www.langchain.com/

```
[INST]Rephrase the following conversation and subsequent
      request so that it is a stand-alone question, in
     its original language.
Conversation:
{conversation_H_var}
Follow-up request:
{request_Q_var}
Stand-alone request:
[/INST]
```

Listing 1: The prompt template used to generate a standalone query Q_s from the ongoing discussion H and the new user's query Q. The variables enclosed in curly brackets are used to populate the prompt with specific instances. label

5.2 Retrieval Augmented Generation (RAG)

To summarize and extract content from the game and the user's current state, the framework uses Retrieval Augmented Generation (RAG). Figure 3 shows the detailed architecture of the RAG component, which was first shown in Figure 2. The data potentially queried at runtime is divided into two parts:

- A static part - it represents data that does not change during a game depending on user actions, e.g. manuals, instructions, environment and item descriptions, etc.
- A dynamic part - data that is constantly changing, in our case represented by the PlayerState and the current context of the game.

The static part can be represented in any text format (PDF, Word documents, etc.), Youtube video transcripts (a commonly used resource in the field of game development), formatted data sets (in CSV, SQL, JSON files, etc.). These are processed offline with the Langchain API, i.e. not during the runtime of a game, by splitting them into text sections, embedding them and then indexing them in a vector database.

For the dynamic part, which is constantly changing, a general interface is needed that allows the developers of a game to share the information for the

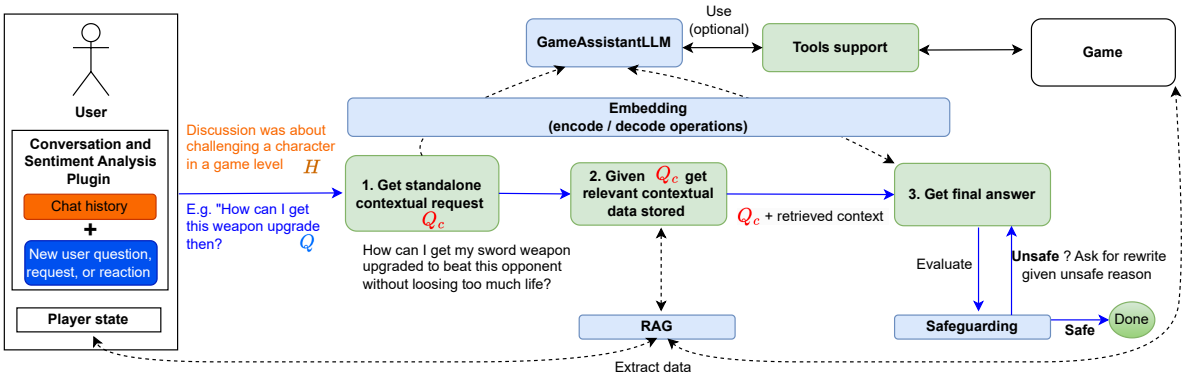

Figure 2: The conversational chat system implemented in the proposed method and its associated components. The flow is represented by the blue arrows. The three main steps (the green boxes) are used to aggregate the conversation flow, retrieve internally relevant stored data, and then obtain the final response after the safeguard component approves the results. The optional step is used by the LLM to interact with the registered tools on the deployed platform.

RAG. The reason for this is that each game has a different context of information (e.g., in a shooter game it could be information about weapons and locations, while in a football game it could be about statistics of the current game, the morale and energy status of the players, etc.). To represent the information from a game in a generic form, the framework uses the Jinja[4] template format, inspired by the way LLMs use the same method to adapt different types of datasets to text representation in the process of training or fine-tuning (Rozière et al., 2024), (Touvron et al., 2023), (Gallotta et al., 2024).

The conversion of the text representation into a vector database for embedding is done using the Faiss (Douze et al., 2024), (Johnson et al., 2019) which is a state-of-the-art indexing and retrieval method widely used in the construction of real-time assistants based on LLMs. The vector database is divided into two parts, as the static one, $VectorStore_{static}$, does not change at all during the runtime of a game, while the dynamic one, $VectorStore_{dynamic}$, is constantly changing and updated at each $VStore_{rate}$.

For a new query Q_c, both stores are checked, and the best K results based on the similarity metric are returned by Faiss. However, a threshold T_{RAG} is used for this similarity to filter the results and have finer control over the quality of the output results. The extracted information is used to help the LLM give a better quality response and avoid hallucinations as much as possible. For each query/prompt, the information is fed into the template variable $CONTEXT\_VAR$, as shown in Listing 2, line 16.

[4]https://jinja.palletsprojects.com/en/3.0.x

5.3 Tools and Interaction with the Backend Systems

The framework uses the ReACT agents (Yao et al., 2023) from the Langchain API, with fine-tuning of the LLM as described in Section 5.5. This allows our project to integrate state-of-the-art methods that handle the interaction between the user, the LLM and the system processes (in our case a game) in a decoupled way. The concepts are known in the literature under the terms *Agents*, and *Tools*, or *Functions*, (Yao et al., 2023), (Schick, 2023), (Patil et al., 2023). In short, ReACT is an AI agent based only on the language model that goes through a cycle of Thought-Action-Observation in which it tries to decompose the problem into easier-to-solve sub-problems. At the end, when it has enough knowledge, it will respond. An example from the game prototype is shown in Listing 2. Any developer can reuse the same template for the prompt and add their own customized set of functions in the variable $TOOLS\_VAR$, line 15. The available functions can be defined in a JSON format, line 1. When the user (or the ReACT agent itself) makes a query, line 19, the descriptions of the tools are intuitively correlated with the query by the LLM based on its training. If a strong similarity is detected, an attempt is made to extract the parameters and return the function call in order to obtain either an internal response or a final response for the user. To achieve this goal, the Phi-3 base model had to be fine-tuned, as shown in Figure 4.

```
1  TOOLS_VAR=
2  {....
3    {'name': 'show_path_to_destination',
4      'description': 'When the user requests to go to a
         certain position or location in the game, extract
         it and use as parameter to the function',
5      'parameter': {
6        'type': 'object',
7        'properties': {
```

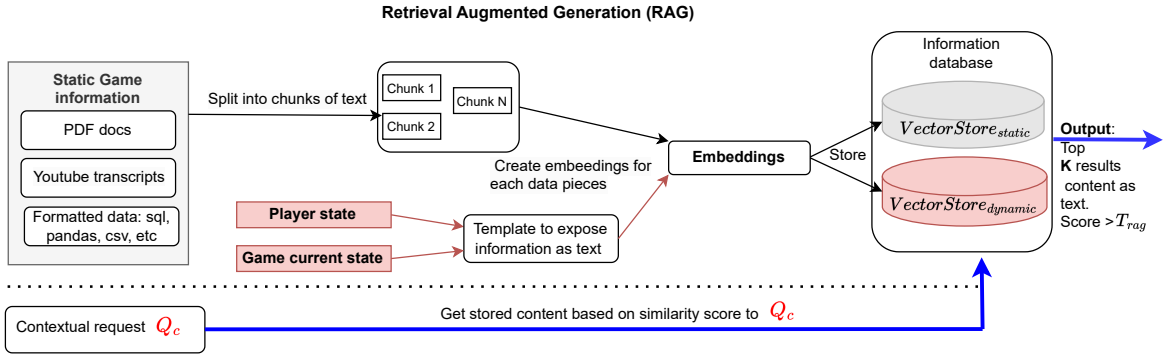

Figure 3: The pipeline for creating the RAG support for both the static data of the game (gray blocks and black arrows flowing), as well as for the dynamic state (red colored flow). The upper part represents the indexing process. The lower part represents the flow of the database query when a new query arrives.

```
        'location': {
          'type': 'string',
          'description': 'The position, location, area,
      or object name to go',
          'required': True}}}}
  .....
}
<s><|system|>
You are a helpful assistant in a video with access to
      the following functions: {TOOLS_VAR}. Use them
      only when necessary.
You can also use the following information in your
      answer: {CONTEXT_VAR}.

<|user|>
Can you show me the path to the weapons shop where I can buy the Sword with level 10?

<|end|>
<|assistant|>
Thought: To solve this query, I first need to find the
      location of the level 10 sword. I will use the
      search_manual tool.
Action: search_manual
Action Input: {'input': 'sword level 10'}
Observation: Sword level 10 is powerful...[omitted text
      ]... it can be found in Weapon Shop Mountain for
      10 coins and for free under the rock near the
      waterfall.
Thought: I have the answer now, it's in Weapon Shop
      Mountain. I will call a function that shows the
      path.
Action: show_path_to_destination
Action Input: {'location': 'Weapon Shop Mountain'}
Response: Here it is.<|end|>
```

Listing 2: An example where a user needs to find a path to a specific item and the response of the GameAssistantLLM in ReACT style. Highlighted are the The descriptions of the tools are defined in the TOOLS_VAR variable (for space reasons, only the path finding tool is shown). The relevant information in the current state is extracted with RAG and fed into the CONTEXT_VAR variable. In this example, the ReACT agent tries to find the answer in the internal manual by calling the *search_manual* function, and when it has found the answer, it displays the final answer with *show_path_to_destination*. In the game, this displays a travel path in 3D to a specific location, as in Figure 5.

5.4 Safeguarding Interactions

This system component, shown in Figure 2, serves two main purposes:

(a) Ensuring dialogue compliance: It ensures that interactions between users and chatbots adhere to various rules, such as gender neutrality and politeness. A comprehensive classification of these rules can be found in the work of (Inan et al., 2023).

(b) Secure interaction with user requests and tools: The component guarantees secure interactions when users make requests or use tools (e.g. when writing or executing code). If an unsafe response or request is detected, prompt engineering instructs the model to change the original output accordingly. By integrating these two aspects, the language learning model (LLM) can usually generate a safe response in one or more attempts.

In our approach, we first investigated the integration of Llama Guard (Inan et al., 2023), an LLM security model developed specifically for this use case. Llama Guard also provides a taxonomy of security threats. When prompted, the model first determines whether the response is secure or not. If it is deemed unsafe, it provides details about the specific unsafe element. Considering the computational overhead of loading a 7B model, our approach also includes packages from the Natural Language Toolkit (NLTK) (Bauer et al., 2020). These packages use traditional NLP techniques to account for different security classifications in prompts. There is a compromise between the two solutions: The Llama Guard provides more precise results without requiring a local taxonomy of elements, while the NLTK packages are fast and memory efficient on the request side.

5.5 Datasets Construction and fine-tuning

The full flow of the processes that collect the datasets and fine-tune the Phi-3 model on them is shown in Figure 4. The purpose is to fine-tune the model to the three categories of use cases mentioned in Section 4.

There are two types of datasets that are required for fine-tuning the base model:

- Data sets for the category Question Answering (from manuals, guides or video transcripts).
- Records for (explicit or implicit) action requests related to the state of the game.

The former is obtained directly from the static data content indexed by the RAG process, Figure 3. We denote this with $\mathcal{D}_{static}$.

The second area requires more attention in order to avoid wasting resources. With this in mind, we first start with a set of human annotated datasets (authors and collaborators) of possible questions, $Q_{initial}$, as shown in Listing 3. This set contains 15 question templates, where the variables between curly brackets are placeholders that must be filled, e.g. the variable *SETOFOBJECTS* in line 1, while the square brackets are optional context variables for different situations. In the example, there are three types of contexts: a) a general one, line 1, b) a situation where complex logic is required to proceed and the user might need help or hints, line 2, and c) and object information in specific situations, line 3. There are templates where a context is optional, like a), and those where the LLM cannot understand the question without a previous context, like cases b) and c). These contexts could intuitively correspond to the content retrieved by RAG in Listing 2, line 16. One idea to further automate this collection process is to use RAG to annotate more examples.

```
1  Q_initial={
2  Can you show me the path to the {SETOFOBJECTS}? [
       CONTEXT_G]
3  How can I solve this puzzle, can you show me?{CONTEXT_P}
4  What weapon should I get to defeat this opponent? {
       CONTEXT_O}
5  ...
6  }
7  TeacherPrompt:
8  You must create {N} variants for each of these:
9  Q_{initial}
10 Instructions:
11 You can vary the language and/or replace the variables
       in curly braces and square brackets with the
       following:
12 SETOFOBJECTS={Teleporter, Weapons Shop,...}
13 CONTEXT_P={Player is in the waterfall, near the elevator
       ,...}
14 CONTEXT_O={Guardian near the temple,...}
15 The variables in curly brackets must be filled with one
       of the specified options, the others are optional.
        Do not use any other contexts or options outside
       the specified text.
```

Listing 3: Examples from the initial dataset Q_{init} and the prompt used to vary the content with the teacher model GPT4.

The GPT4 model is considered as a teacher, and we collect a dataset for instruction tuning using the *TeacherPrompt* and the method of (Peng et al., 2023). We obtain a dataset Q_aug where the model was asked to create 30 variations for each template, resulting in a total of 450 questions. Since GPT4 is trained on function calls, we query it to generate function calls using the tool description and Q_aug as the user prompt, as shown in Listing 2. Of these, 314 were actually useful after being evaluated by the game. We refer to this set as $\mathcal{D}_{aug}$. Finally, the dataset $\mathcal{D} = Q_{initial} \cup Q_{aug}$ is used for fine-tuning the model using standard command fine-tuning and cross-entropy loss (Hui and Belkin, 2020).

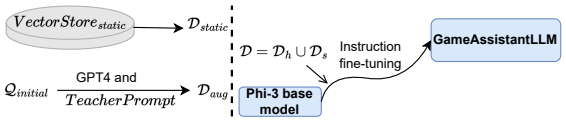

Figure 4: The pipeline for extracting the datasets and fine-tuning the Phi-3 (3.8B) base model using both static and possibly contextual actions requests.

6 EVALUATION

6.1 Setup

Demo Application. The decision to use a plugin on the UE5 engine was made to ensure a clear separation of responsibilities and minimal dependencies between the application and the proposed framework. During development and testing, Flask https://flask.palletsprojects.com/en/2.2.x was used to quickly switch between different models and parts used for training or inference, etc. Throughout the evaluation phase, the models were deployed locally using UE5's Neural Network Engine (NNE) plugin. This model was able to run the two models using the ONNX[5] format. Using this decoupled format allows the models to run on other similar solutions, e.g., Unity[6]. A snapshot from the game can be seen in Figure. 5.

Choosing the Foundation LLM Version. In this study, we evaluated several small models in the range of 2B-7B, including Llama2-7B (Touvron et al., 2023), Gemma 2B (Team et al., 2024) and Phi-3 (3.8B) (Abdin et al., 2024). The decision on which LLM to choose as the basis for further fine-tuning was mainly constrained by the resources consumed and the inference speed. With the currently available hardware, we only need to consider models that can fulfil the runtime expectations on the end-user hardware. Since most games are already pushing GPUs to the limit with their rendering systems, another limitation is to only use CPUs in comparison. Therefore, we evaluate the memory requirements and words per second generated by the LLMs on an entry-level AMD Ryzen 7700 CPU that is limited to using only 4

[5]https://onnx.ai/

[6]https://unity.com/products/sentis

Figure 5: A snapshot from the demo application in which the user has an ongoing conversation with the assistant (NPC). The user asks which item he needs to defend an enemy and where he can find it. After the conversation, the assistant suggests the item needed and shows a path to its location (the red arrows in the picture).

cores for inference. The results are shown in Table 1. Although the literature generally evaluates the metric *tokens per second* instead of words per second, we believe that this is a more natural choice for comparison in the current use case.

Table 1: Inference speed measured in words per second for an AMD Ryzen 7 CPU limited to 4 cores and memory requirements (RAM, not GPU memory). 8-bit quantization was used for all models.

Model Metric	Llama2B-7B	Gemma2B	Phi-3
Words/s	1.65	7.9	3.61
RAM	8.5GB	2.7GB	4.6GB

Voice-to-Text is processed with OpenAI Whisper-tiny(Radford et al., 2022) with dynamic quantization. On average, the inference time on the same CPU was 1.6 seconds for a short message of 9-10 seconds. However, for a voice prompt, this time is always added to the overall latency of the assistant's response, as it must finish the input before querying the LLM.

Fine Tuning Parameters. The fine-tuning process was performed over 10 epochs with LoRA (Low-Rank Adaptation) (Hu et al., 2022). All layers were fine-tuned with rank $r = 16$, $\alpha = 32$, $dropout = 0.05$. The default quantization parameters with $bfloat-16$ were used to maximize the training runtime. Nucleus sampling with $p = 0.95$ and $temperature = 0.8$ was used for decoding. However, during test time, the temperature was set to zero to obtain deterministic results that are predictable and repeatable, a decision inspired by previous research (Siddiq et al., 2022), (Alshahwan et al., 2024). The AdamW optimizer (Loshchilov and Hutter, 2019) is used, with $\beta 1 = 0.9$ and $\beta 2 = 0.95$, with a learning rate $\gamma = 3 \times 10^{-3}$, and a decay rate of 0.01. In the first training phase, a warm-up of 200 steps was used. Each batch had a size of 32, with 4 gradient accumulation steps and different context window sizes within the boundaries of the base model.

6.2 Quantitative Evaluation

In this perspective, it is of interest to see how the fine-tuned version of the *GameAssistantLLM* model performs compared to the base version Phi-3. The BLEU metric is used for evaluation similar to other related work ((Touvron et al., 2023)). The score ranges from 0 to 100, with a higher score indicating a stronger match between the generated response and the reference. The evaluation is based on samples from the data set $\mathcal{D}$ and the averaging of the results over 100 trials. The averaged results show a score of 79% for the fine-tuned model and 34% for the base model. Looking at the different results, it is concluded that the results of the base model were determined by the general knowledge that an LLM may possess and that it can hallucinate around topics even if it does not know the answer, e.g. telling the user to use different roads and modes of transportation when asked to point the way to different objects.

Other parameters.

6.3 Qualitative Evaluation

The experiments and statistical results for the qualitative evaluation of our framework and demo come from a group of 46 people (volunteers from the quality assurance departments of our industry partners and students from University of Bucharest) who played the demo for two hours and tried to navigate through the application by asking the NPCs questions on various topics. During our qualitative review, we focused on three research topics.

RQ1. Do the application or NPCs give correct answers, i.e. do they seem to understand the questions or requests asked and answer in the right context?

To measure this, each of the 46 participants played the demo for two hours and were instructed to send between 80 and 100 messages to the application/NPCs, using almost identical amounts of voice and text input. After each response from the application, they were asked whether the NPCs or the application had responded correctly. Table 2 shows participants' averaged feedback for both types of input, categorized by message. The results show that users are generally satisfied with the feedback,

with lower ratings for voice input as expected, as an additional layer is required to convert voice to text and performance naturally degrades along the way. The scores are expectedly lower for requests where actions need to be performed through function calls and inference. One way to improve this is to create a much larger dataset for fine-tuning. Also, the implicit actions are more difficult to understand as the intent expressed by the user sometimes does not match the understanding of the language model.

Table 2: Qualitative evaluation of the correctness of feedback depending on the type of input and the message category.

Input type	**Category**		
	Question answering	Explicit actions requests	Implicit actions (sentiment analysis)
Text	91%	87%	74%
Voice	88%	85%	71%

RQ2. Do users believe that speaking to the assistant is generally beneficial? (e.g. a better understanding of the physics of the simulation environment, removal of obstacles, help with difficult tasks or actions, etc.).

To assess this, each participant was given an answer template at the end of the test to help them answer this question. Overall, 39 out of 46 participants felt that the conversations supported them during the demo, while 7 wished they could discover the application and procedures themselves. The following key findings emerge from the user feedback: *explicit calls to action was the most useful* and helped them to cope with difficult parts of the demo. On the other hand, they did not use the support for implicit actions as they did not find it very useful and felt rather forced to use it. However, we think that improving its quality (Table 2) might change this opinion in the future. Another interesting fact is that all respondents wanted the physical form of an assistant, i.e. the use of an NPC in our demo case.

RQ3. Is the method suitable for use in real time? The choice of models was carefully evaluated with knowledge of the general user requirements, and the results were discussed in Section 6.1. On the same hardware, we evaluated the following two metrics by averaging over all user requests:

- The average latency to the first response from the assistant was ~ 1 second for text input and $\sim 2.5s$ for voice input.
- Response throughput: ~ 3.5 words per second, after the first word response.

Participants were asked to rate how they felt about the latency of the assistant's response. 41 out of 46 participants received both text and voice feedback from the assistant, while the remaining 5 participants preferred to use voice input and found it too slow. After evaluating the post-process surveys, we are not sure whether users were actually disappointed by a mixture of: a) the additional delay time in processing voice ($\sim 1.5s$) and b) the performance degradation in the quality of responses when voice input is used as an intermediate step before conversion to text.

7 CONCLUSION AND FUTURE WORK

The paper presented a framework for integrating the capabilities of large language models to create assistants during the runtime of a game. In our demo, the assistant could take the form of a physical 3D character (NPC) or a virtual background character (Narrative). Different models and architectures were evaluated and methods were selected by combining the general user requirements in terms of computational power requirements, latency and quality of results. Improvement methods such as synthetic dataset generation using a teacher LLM model, fine-tuning, retrieval-augmented generation (RAG) and conversational security were also discussed. The evaluation was both quantitative and qualitative, i.e. based on feedback from real users through surveys and interaction. Although the methods were implemented as an open-source Unreal Engine plugin and applied to a game demo, we believe that the scope could be extended to other real-time simulation applications. As future work, we plan to collaborate more with industry to apply the framework in larger projects and more use cases, and potentially evaluate the methods on a large scale.

ACKNOWLEDGMENTS

This research was supported by European Union's Horizon Europe research and innovation programme under grant agreement no. 101070455, project DYNABIC. We also thank our game development industry partners from Amber, Ubisoft, and Electronic Arts for their feedback.

REFERENCES

Abdin, M. I. et al. (2024). Phi-3 technical report: A highly capable language model locally on your phone. Technical Report MSR-TR-2024-12, Microsoft.

Allouch, M., Azaria, A., and Azoulay, R. (2021). Conversational agents: Goals, technologies, vision and challenges. *Sensors*, 21(24).

Alshahwan, N. et al. (2024). Automated unit test improvement using large language models at meta. *32nd ACM Symposium on the Foundations of Software Engineering (FSE 24)*.

Bauer, T. et al. (2020). # metoomaastricht: Building a chatbot to assist survivors of sexual harassment. In *Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019*, pages 503–521. Springer.

Cascella, M. et al. (2024). The breakthrough of large language models release for medical applications: 1-year timeline and perspectives. *Journal of Medical Systems*, 48(22).

Cox, S. R. and Ooi, W. T. (2024). Conversational interactions with npcs in llm-driven gaming: Guidelines from a content analysis of player feedback. In *Chatbot Research and Design*, pages 167–184, Cham. Springer Nature Switzerland.

Douze, M. et al. (2024). The faiss library. *arXiv preprint arXiv:2401.08281, https://github.com/facebookresearch/faiss*.

Gallotta, R. et al. (2024). Large language models and games: A survey and roadmap. *arXiv preprint arXiv:2402.18659*. Submitted on 28 Feb 2024.

Guan, Y., Wang, D., Chu, Z., Wang, S., Ni, F., Song, R., Li, L., Gu, J., and Zhuang, C. (2023). Intelligent virtual assistants with llm-based process automation. In *https://arxiv.org/abs/2312.06677*.

Hu, E. J. et al. (2022). LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Huber, S. E. et al. (2024). Leveraging the potential of large language models in education through playful and game-based learning. *Educational Psychology Review*, 36(25):1–17.

Hui, L. and Belkin, M. (2020). Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks. *arXiv preprint arXiv:2006.07322*.

Inan, H., Upasani, et al. (2023). Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674*.

Isaza-Giraldo, A. et al. (2024). Prompt-gaming: A pilot study on llm-evaluating agent in a meaningful energy game. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '24)*, page 12. ACM.

Johnson, J., Douze, M., and Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Li, R. et al. (2023). Starcoder: may the source be with you! In *https://arxiv.org/abs/2305.06161*.

Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019*.

Marvin, G., Hellen, N., Jjingo, D., and Nakatumba-Nabende, J. (2024). Prompt engineering in large language models. In Jacob, I. J., Piramuthu, S., and Falkowski-Gilski, P., editors, *Data Intelligence and Cognitive Informatics*, pages 387–402, Singapore. Springer Nature Singapore.

Munday, P. (2017). Duolingo. gamified learning through translation. *Journal of Spanish Language Teaching*, 4(2):194–198.

OpenAI et al. (2024). Gpt-4 technical report. In *https://arxiv.org/abs/2303.08774*.

Paduraru, C., Cernat, M., and Stefanescu, A. (2023). Conversational agents for simulation applications and video games. In *Proceedings of the 18th International Conference on Software Technologies, ICSOFT 2023*, pages 27–36. SCITEPRESS.

Patil, S. G. et al. (2023). Gorilla: Large language model connected with massive apis. *CoRR*, abs/2305.15334.

Peng, B. et al. (2023). Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*.

Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., and Sutskever, I. (2022). Robust speech recognition via large-scale weak supervision.

Rozière, B. et al. (2024). Code llama: Open foundation models for code. In *https://arxiv.org/abs/2307.09288*.

Schick, T. a. (2023). Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36.

Siddiq, M. L. et al. (2022). An empirical study of code smells in transformer-based code generation techniques. In *2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*.

Team, G. et al. (2024). Gemma: Open models based on gemini research and technology. In *https://arxiv.org/abs/2403.08295*.

Touvron, H. et al. (2023). Llama 2: Open foundation and fine-tuned chat models. In *https://arxiv.org/abs/2307.09288*.

Vaswani, A. et al. (2023). Attention is all you need.

Wankhade, M., Rao, A., and Kulkarni, C. (2022). A survey on sentiment analysis methods, applications, and challenges. *Artificial Intelligence Review*, 55:1–50.

Yao, S. et al. (2023). React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Yunanto, A. A. et al. (2019). English education game using non-player character based on natural language processing. *Procedia Computer Science*, 161:502–508. The Fifth Information Systems International Conference, 23-24 July 2019, Surabaya, Indonesia.

Zhao, P. et al. (2024). Retrieval-augmented generation for ai-generated content: A survey.