INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Integrating GPT-Based Language Models into NPCs for Realistic Dialogue and Contextual Awareness in Unity Games

Pedro Miguel Santos Albino – a47369

Work carried out under the guidance of

Pedro João Soares Rodrigues

Master's in Informatics

2024-2025

Integrating GPT-Based Language Models into NPCs for Realistic Dialogue and Contextual Awareness in Unity Games

Project CU Report

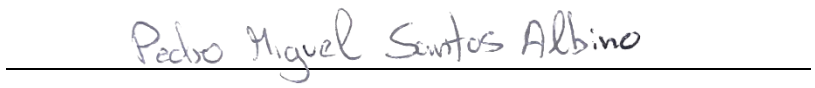
Master's in Informatics

Bragança Higher School of Technology and Management

2024-2025

Bragança Higher School of Technology and Management is not responsible for the opinions expressed in this report.

I declare that the work described in this report is my own and that I wish it to be submitted for evaluation.



Pedro Miguel Santos Albino - 47369

Acknowledgements

I would like to express my gratitude to my thesis supervisor, Pedro João Soares Rodrigues, for their guidance, patience and continuous support throughout the development of this project.

I am also thankful to the faculty members of Bragança Polytechnic Institute, who have broadened my perspective and enriched my understanding of the field.

Most importantly, I want to thank my family for the constant encouragement, and for being my foundation through every step of this process. Their support has given me strength beyond measure.

This thesis is the result of many contributions, and I am deeply grateful to everyone who has played a part in it.

Abstract

This thesis explores the integration of large language models (LLMs), specifically the LLama family, to enhance the realism and interactivity of non-player characters (NPC) dialogue in video games developed with Unity. Traditional NPC interactions are often constrained by scripted dialogue trees, which limit immersion and adaptability. The main objective of this work is to demonstrate how GPT-like models can be embedded into game environments to generate dynamic, context-aware responses that elevate player engagement.

The methodology involves using open-source LLMs to suit game-specific dialogue styles, then deploying these models within Unity using a local server to maintain performance. Custom environments were created in Blender to build an 3D prototype showcasing the real-time NPC conversations. Key implementation challenges, such as latency, prompt management, and dialogue coherence, were addressed through model tuning techniques.

The results indicate that integrating LLMs into Unity significantly enhances dialogue variety, responsiveness, and player immersion compared to traditional systems. Furthermore, the flexible nature of LLMs opens new possibilities for adaptive storytelling and personalized gameplay.

Keywords: Large Language Models, NPC Dialogue Generation, Unity Game Development.

Resumo

Esta tese explora a integração de grandes modelos de linguagem (LLMs), especificamente a família LLama, para melhorar o realismo e a interatividade do diálogo de personagens não jogáveis (NPC) em videojogos desenvolvidos com Unity. As interações tradicionais de NPCs são frequentemente limitadas por árvores de diálogo programadas, o que restringe a imersão e a adaptabilidade. O principal objetivo deste trabalho é demonstrar como modelos semelhantes ao GPT podem ser incorporados em ambientes de jogo para gerar respostas dinâmicas e sensíveis ao contexto que aumentam a imersão do jogador.

A metodologia envolve o uso de LLMs de código aberto para se adequar aos estilos de diálogo específicos do jogo e, em seguida, a implementação desses modelos no Unity através de um servidor local para manter o desempenho. Ambientes personalizados foram criados no Blender para construir um protótipo 3D que mostra as conversas em tempo real dos NPC. Os principais desafios de implementação, como latência, gestão de prompts e coerência do diálogo foram abordados por meio de técnicas de ajuste do modelo.

Os resultados indicam que a integração de LLMs no Unity melhora significativamente a variedade de diálogos, a capacidade de resposta e a imersão do jogador em comparação com os sistemas tradicionais. Além disso, a natureza flexível dos LLMs abre novas possibilidades para narrativas adaptativas e jogabilidade personalizada.

Palavras-chave: Modelos de linguagem grandes, Gerar diálogos de NPC, Desenvolvimento de jogos Unity.

Content

Figures List

Acronyms

ESTiG Escola Superior de Tecnologia e Gestão (Higher School of Technology and Management).

NPC Non-Player Character.

LLM Large Language Model

NLP Natural Language Processing

FSM Finite State Machines

TTS Text to Speech

Chapter 1

Introduction

1.1 Framework

In recent years, the evolution of LLMs such as OpenAI's GPT and Meta's Llama, among other transformer-based architectures has significantly opened new frontiers for artificial intelligence applications, particularly in fields involving natural language understanding and generation. These models have demonstrated remarkable capabilities in natural language understanding and generation, enabling systems to engage in complex and context-aware dialogue. Among these emerging applications, one of the most promising is the integration of LLMs into interactive digital environments, such as video games. This thesis arises from the growing interest in using advanced AI models to enhance NPC interactions, with the goal of creating more immersive and believable game experiences.

Non-Playable Characters (NPCs) play a crucial role in shaping player experience and narrative immersion. Traditional NPC dialogue systems rely heavily on pre-written, tree-based scripting methods. While effective for controlled narrative delivery, these systems often lack flexibility, adaptability, and realism, resulting in repetitive or unnatural conversations. In contrast, LLMs offer the potential to revolutionize NPC interactions by generating dynamic, context-sensitive dialogue in real time. Their capacity to interpret context, retain conversational history, and produce coherent language enables a new paradigm for player-NPC communication, one that is adaptive, expressive, and emergent rather than pre-defined. However, integrating such models into real-time game environments poses challenges, including performance optimization, prompt engineering, and maintaining consistency with game logic and narrative tone.

This thesis is motivated by the opportunity to explore these challenges and assess the feasibility of using open-source LLMs, specifically the Llama family, to power interactive dialogue systems within the Unity game engine. Beyond demonstrating a technical implementation, this work seeks to contribute to the broader understanding of how generative AI can transform interactive storytelling and procedural narrative design. The research is also driven by the need to identify practical methods to balance creativity and control when deploying generative models in constrained, real-time systems.

1.2 Objectives

The primary objective of this thesis is to investigate and implement the integration of LLMs into real-time video game environments to enhance the realism, adaptability, and engagement of NPC dialogue. This work aims to contribute to the field of interactive media by demonstrating how AI-driven systems can surpass the limitations of traditional dialogue models, opening new avenues for dynamic narrative and immersive gameplay by aiming to design, implement and evaluate an LLM driven dialogue system capable of producing coherent and contextually relevant in-game communication. To achieve this goal, the thesis is structured around the following specific objectives:

- **Analyse the current state of NPC dialogue systems** in video games, identifying their limitations in terms of flexibility, scalability, and realism, and justifying the need for an AI-based approach.
- **Evaluate** open-source LLMs, assessing their suitability for real-time integration in Unity in terms of inference performance, resource demands, and dialogue generation quality.
- **Develop an LLM-based dialogue system within Unity**, including:
    - Connect the model with the Unity game engine.
    - Designing mechanisms for prompt generation, dialogue memory/context handling, and response filtering to ensure coherent and relevant in-game conversations.
- **Design and build a prototype game environment**, including 3D characters and maps modelled in Blender, to serve as functional testbed for evaluating the behaviour and performance of LLM-powered NPCs.

- **Conduct both performance and qualitative evaluations**, measuring latency, resource usage, and dialogue quality, as well as assessing user experience in comparison to traditional scripted NPC interactions.
- **Discuss the implications, limitations and future potential** of integrating LLMs into game development, including opportunities for further research in procedural storytelling and adaptive narrative design.

The expected contributions of this work are twofold. From a practical perspective, it delivers an operational prototype that demonstrates the feasibility of embedding LLMs within real-time interactive systems. From a scientific perspective, it provides an analysis of the design trade-offs, challenges, and opportunities that arise when generative AI models are deployed in dynamic narrative environments. Together, these contributions aim to inform future research and development at the intersection of artificial intelligence, game design, and interactive media.

1.3 Document Structure

This report is organised to reflect the typical structure of an informatics engineering project, particularly once centred on the design, development, and evaluation of a software-based solution. Given that this work focuses on the integration of large language models into a game environment, the structure follows the main phases of analysis, design, implementation, and assessment of the proposed system.

The remainder of this document is structured as follows:

- **Chapter 2** provides the contextual background of the project, introducing the problem domain, reviewing related work, and describing the theoretical foundations and technologies relevant to the integration of LLMs within interactive environments.
- **Chapter 3** details the methodological approach adopted throughout the project. It outlines the problem definition, describes the architectural vision for the system, and explains the strategies used to ensure coherence, personality consistency, and performance in LLM-driven NPC dialogue.

- **Chapter 4** presents the implementation phase, covering the creation of 3D environments and character assets in Blender, the development of the FastAPI backend, and the integration of the dialogue system into Unity. This chapter also describes the gameplay mechanics, cutscene system, and communication pipeline between the game engine and the LLM server.
- **Chapter 5** discusses the results obtained during evaluation. It analyses the technical performance of the system, including inference latency and communication reliability, and examines user experience aspects such as dialogue naturalness, immersion, and perceived character consistency.
- **Chapter 6** concludes the report with a summary of the main findings, a reflection on the project's limitations, and suggestions for future improvements and research directions.

This structure ensures a clear progression from conceptual foundations to practical implementation and evaluation, allowing the reader to follow the development of the solution and understand its technical and functional contributions.

Chapter 2

Context and Technologies

2.1 Problem Context and Scope

Non-Player Characters (NPCs) are fundamentals elements in video game design, often responsible for delivering story content, guiding players, and populating virtual worlds with interactive life. Despite advancements in graphics, physics, and gameplay mechanics, NPC dialogue systems have remained largely reliant on static, rule-based structures such as finite-state machines or behaviour trees. These traditional systems are labour-intensive to design and lack the flexibility to adapt dynamically to player input or narrative context, often leading to repetitive or unnatural interactions.

The increasing accessibility of generative language models presents a new opportunity to redefine how players interact with NPCs. The objective of this work is to integrate LLMs, specifically Meta's open-source Llama Models, into Unity, a widely used real-time game engine, to enable realistic and adaptive dialogue generation. This approach supports the development of dynamic narrative experiences, shifting from predefined scripts to generate conversational responses based on real-time context and memory.

The scope of this project includes:

- Designing a server-hosted dialogue system powered by an LLM.
- Developing a Unity game prototype using both custom and marketplace assets modelled in Blender.
- Evaluating system performance, quality of dialogue interactions, and user experience.
- Ensuring compatibility with common game development workflows.

2.2 Theoretical Foundations and Related Work

2.2.1 Traditional Dialogue Systems in Games

The design of NPC dialogue in games has traditionally been based on deterministic, scripted methods. These systems, such as finite state machines (FSM), dialogue trees, and rule-based engines, have long been dominant in both AAA and indie game development. In these frameworks, dialogue options are predefined, often triggered by specific game states, player choices, or environmental variables. Examples of these include *Bioware's Dialogue Wheel* (e.g. *Mass Effect*) and *Bethesda's keyword-based system* in *Skyrim (The Elder Scrolls V)*.

While these approaches ensure narrative control and predictability, they also suffer from key limitations:

- **High authoring cost**: Designers must manually create every possible dialogue path.
- **Limited flexibility**: Predefined options often fail to reflect the payer's intent or creativity.
- **Repetition and predictability**: Frequent players can quickly recognize dialogue patterns, reducing immersion.

Despite improvements like conditional branches and randomized lines, the absence of true natural language understanding limits the believability of NPCs as intelligent, responsive characters.

2.2.2 The Rise of Data-Driven and Generative Approaches

The introduction of machine learning and deep neural networks offered new methods to generate dialogue, including retrieval-based models, statistical translation methods, and sequence-to-sequence architectures. However, their practical deployment in real-time game environments remained limited due to constraints in speed, reliability, and coherence.

This changed significantly with the introduction of transformer architectures by Vaswani et al. in 2017 [1], which brought a breakthrough in handling long-range dependencies in text. Transformers became the foundation for state-of-the-art language models. These models, trained on large and diverse datasets, are capable of producing coherent and contextually appropriate text across many domains.

Although the potential of these models in gaming has been recognized, practical applications remain mostly experimental due to the high computational requirements, inference latency, and challenges in maintaining narrative coherence in open-ended dialogues.

2.2.3 LLM in Game Development: Current Research and Applications

In recent years, game developers and researchers have started exploring how LLMs can be integrated into real-time gameplay [14]. This has included both academic studies and experimental modifications of commercial games. One of the most high-profile examples is the **AI NPC mod for *Skyrim*** released in 2023-2024, which integrates **ChatGPT-like models** to give voice and personality to NPCs [2]. Using OpenAI's GPT, modders created an experience where characters can hold open-ended conversations with players, remember past interactions, and even generate quests dynamically.

This mod was a landmark moment for AI in gaming, as it:

- Showed that LLMs could run alongside complex game engines.
- Demonstrated how player immersion increased when NPCs responded meaningfully and unexpectedly.
- Highlighted the challenges of latency, context retention, and prompt engineering in live environments.

One of the most significant challenges when using LLMs in games is maintaining conversational memory. Unlike humans, LLMs do not possess a persistent memory by default. They rely on prompt engineering, injecting previous conversation turns, to simulate memory. This introduces three key problems:

- **Context window limitations**: Even high-end models have a token limit (ex: 4k, 8k or 32k tokens).
- **Loss of world-state awareness**: The model doesn't inherently track what's happening in the game unless it's explicitly passed in the prompt.
- **Cost and speed trade-offs**: The more context you pass, the slower the response.

Various methods have been proposed to overcome these:

- **Memory buffers**: A rolling window of the last N utterances.
- **Summarization**: Condensing long dialogue into short summaries that preserve intent and facts.
- **Hybrid architectures**: Using symbolic systems (ex: knowledge graphs) alongside LLMs.

2.2.4 Large Language Models: Principles and Challenges

2.2.4.1 Core Principles and Architectures

Modern LLMs are almost universally built upon the Transformer architecture, introduced by Vaswani et al. [1]. The Transformer replaces recurrence and convolution with self-attention mechanisms, allowing each token to attend to all others within a sequence. This enables efficient parallelisation during training and captures long-range dependencies effectively.

The two dominant pretraining paradigms are:

- **Autoregressive pre-training (**e.g GPT Family) predicts the next token given all previous tokens, naturally supporting text generation,
- **Masked or denoising pre-training** (e.g BERT) reconstructs masked tokens from bidirectional context, producing powerful contextual encoders for classification and retrieval tasks [3].

These paradigms differ in how they expose context during training and in their downstream applications, generative versus representational.

2.2.4.2 Fine-Tuning, Alignment, and Human Feedback

Pretrained LLMs, while powerful, are not inherently aligned with human intent. They may produce untruthful, biased, or unsafe outputs. A major milestone in alignment research was InstructGPT by Ouyang et al. [4], which combined supervised fine-tuning on human-written prompts and reinforcement learning from human feedback (RLHF). This process aligns model outputs with human preferences, yielding systems judged as more helpful and truthful. This alignment framework now underpins most contemporary instruction-tuned models, such as ChatGPT and similar conversational agents.

2.2.4.3 Capabilities and Applications

LLMs demonstrate impressive generalisation across tasks without task-specific training. Their main areas of application include:

- **Text generation and completion:** for summarisation, creative writing, and content creation.
- **Few-shot and zero-shot learning:** GPT-3 [5] illustrated that sufficiently large autoregressive models can perform unseen tasks by conditioning on a few examples within the prompt.
- **Information retrieval and question answering** especially when combined with retrieval-augmented generation (RAG) pipelines.
- **Code synthesis and software engineering assistance**: eg. Github Copilot-style models.
- **Representation learning and semantic search**: via encoder-style transformers like BERT [3].

These applications exploit the models' strong contextual reasoning and token prediction capabilities, though they also inherit limitations such as hallucination, bias, and lack of factual grounding.

2.2.4.4 Key Challenges and Limitations

Despite their impressive performance and versatility, LLMs present several technical, ethical and societal challenges that limit their reliability and safe deployment [15]. These challenges arise from the way such models are trained, primarily through statistical learning from massive text corpora, and from the practical realities of scaling computationally intensive systems.

LLMs function by predicting the next token (word, sub-word, or character) in a sequence given the preceding context. This means they are statistical pattern recognisers, not reasoning systems with grounded understanding of meaning or external reality. While their outputs often appear coherent and contextually appropriate, this fluency emerges from the model's ability to capture surface-level correlations in linguistic data rather than a conceptual or casual understanding of the world.

For example, an LLM can generate a detailed description of a physical process such as photosynthesis or explain a legal concept, but it does so by associating textual patterns that co-occur in its training data rather than by reasoning about biological or legal principles.

This distinction underlies the critique by Bender et al. [6], who characterise LLMs as "stochastic parrots", systems that reproduce linguistic regularities without genuine comprehension. Their warning highlights the epistemic risk of mistaking linguistic fluency for intelligence or understanding.

One of the most visible limitations of current LLMs is their tendency to hallucinate, that is, to produce information that is syntactically plausible but factually incorrect or entirely fabricated. Hallucinations occur because LLMs optimise for likelihood of text continuation, not factual accuracy. When asked questions beyond their training distribution or without sufficient context, the model often fills gaps by generating probable yet false statements.

For instance, an LLM might invent citations, misattribute quotes, or fabricate statistics that sound credible but lack factual basis. Such behaviours are especially problematic in domains requiring precision and accountability, such as medicine, law, or scientific communication.

Mitigation strategies include:

- **Retrieval-augmented generation (RAG):** combining LLMs with external databases or search mechanisms to ground responses in verified information.

- **Factuality classifiers and consistency checks:** using auxiliary models to detect and penalise factual inconsistencies.
- **Post-generation verification pipelines:** integrating human or automated review before outputs are finalised.

Although these methods reduce hallucination frequency, complete elimination remains an open research problem.

LLMs inherit and potentially amplify societal biases present in their training data. Because these datasets are drawn from the internet and other human-produced text sources, they reflect prevailing stereotypes, power imbalances, and discriminatory language. As a result, generated content may exhibit gender, racial, cultural, or political biases, even when unintended by developers.

Examples include models that associate specific professions with one gender, use culturally loaded terminology, or reproduce toxic or exclusionary narratives. Such biases can have real-world consequences when LLMs are deployed in decision making or information systems, for instance, influencing hiring tools, educational software, or automated moderation.

The vast scale of LLM training data introduces serious ethical, legal, and privacy concerns. Most models are trained on web-scaled corpora that aggregate data from billions of publicly accessible documents, many of which were collected without explicit consent or clear provenance. This raises several issues:

1. **Privacy leakage:** LLMs can memorise sensitive information, such as personal identifiers or confidential text fragments, especially if those appear verbatim in the training set.
2. **Copyright infringement:** Because the data often include copyrighted books, articles, or creative works, questions arise about whether the reproduction of stylistic or textual elements in model outputs constitutes infringement.
3. **Lack of transparency:** Many datasets are proprietary or poorly documented, limiting public scrutiny of what information LLMs have been exposed to.

To mitigate these risks, current research explores data documentation standards (e.g “datasheets for datasets”), synthetic data generation to replace sensitive material, and differential privacy techniques to limit memorisation during training. However, robust solutions that balance transparency, utility, and privacy remain a challenge.

Scaling up LLMs to hundreds of billions of parameters demands extraordinary computational and energy resources. Training a model of this scale can consume thousands

of GPU-years and generate tens to hundreds of tonnes of CO2 emissions, depending on data centre efficiency and energy sources. For example, Kaplan et al. [7] and Hoffmann et al. [8] demonstrated that performance generally improves with scale, but this trend leads to diminishing returns, each marginal improvement requires exponentially greater resources.

This concentration of compute capacity within a few large corporations or research consortia creates a resource asymmetry that limits accessibility and academic reproducibility. Furthermore, as Bender et al. [6] note, the environmental cost of continuous scaling poses sustainability concerns and calls for greater emphasis on energy efficient architectures, model compression, and compute-optimal training regimes.

Even when fine-tuned through Reinforcement Learning from Human Feedback (RLHF) or related alignment techniques, LLMs do not perfectly internalise human values or intentions. The alignment process relies on human-generated preference data, which can be inconsistent, culturally biased, and limited in scope. As a result, aligned models may still produce unsafe or undesired outputs when faced with novel contexts or adversarial prompts.

An additional complication is the occurrence of emergent capabilities, behaviours that arise spontaneously at scale and were not explicitly programmed or foreseen. Examples include unexpected reasoning abilities, code synthesis, or strategic goal formulation. While these behaviours can be beneficial, they also introduce unpredictability, complicating safety evaluations and regulatory oversight. Research into scalable oversight, constitutional AI, and multi-agent evaluation frameworks aims to provide more robust methods for aligning models with human norms and preventing harmful or deceptive behaviour. Nonetheless, fully reliable alignment remains one of the most difficult and unsettled challenges in the field.

2.3 Tools and Technologies

2.3.1 Unity

Unity is a cross-platform game engine widely used for real-time 3D development. It offers extensive scripting capabilities in C#, integration with AI components, and support for user interface and physics systems. Its modular structure allows seamless integration with external APIs or machine learning pipelines, making it ideal for prototyping AI-driven systems [9]. Key reasons for choosing Unity include:

- **Scripting Flexibility**: Unity's integration with C# allows for efficient control of in-game logic, event handling, and user interactions. It also simplifies external communication with APIs through HTTP requests, which is vital for integrating LLM-based dialogue system.
- **Asset Integration**: Unity supports common 3D formats (ex: .FBX) making it compatible with models designed in Blender and other industry standard tools.
- **Extensibility**: Through packages and plugins, Unity provides a customizable ecosystem ideal for experimental AI research and game prototyping.
- **Community and Ecosystem**: With a vast asset store, active forums, and continuous updates, Unity remains a relevant and future-proof choice for real-time interactive systems.

2.3.2 Blender

Blender is a free and open-source 3D creation suite that supports the complete modelling pipeline, from sculpting and rigging to animation and rendering. It was chosen for this project due to its flexibility, integration with Unity, and powerful open-source toolset.

- **Modelling**: Creation of detailed humanoid and environmental assets for NPCs and game levels.
- **Rigging and Animation**: Rigify and other built-in systems allow characters to be fully animated for natural interactions.
- **UV Mapping and Texturing**: Assets are prepared with materials and textures optimized for Unity rendering pipelines.
- **Export Compatibility**: Blender's support for formats like FBX and gITF ensures smooth integration into Unity, preserving animations and armatures.

Using Blender enables a custom visual style and allows full control over asset quality, complexity, and optimization, which is especially important for running the prototype efficiently on limited hardware.

2.3.3 LLama: Open-Source Language Model

LLama (Large Language Model Meta AI), developed by Meta AI, is a family of transformer-based large language models optimized for high performance and accessibility [10][11]. Unlike proprietary models like GPT-4, LLama models are open-source and can be fine-tuned and hosted locally, providing complete control over usage, customization, and deployment.

Key technical characteristics:

- **Architecture**: LLama models use the decoder-only transformer architecture introduced by Vaswani et al. (2017), with adaptations for efficiency.
- **Variants**: Available in different parameter sizes (7B, 8B, 13B, 34B, and 65B), allowing flexibility based on system resources and use-case complexity.

- **Inference Tools**: LLama can be run using lightweight inference engines such as llama.cpp, Ollama, or Text Generation Web UI, which support features like quantization to significantly reduce memory requirements.

Before selecting Llama as the primary model for this work, four alternative open-source large language models were evaluated. Although each model presented promising characteristics, none met the requirements necessary for this project in terms of instruction-following capability, output quality and local inference performance on limited hardware.

The LLMs evaluated were Nous Hermes 2 Mistral, which is a widely used model based on the Mistral architecture, designed to provide quality reasoning [23]. However, during testing it demonstrated inconsistent adherence to structured prompts. In scenarios requiring strict control or multi-step instructions, essential for dialogue structuring and gameplay interaction, the model frequently deviated from expected outputs. This reduced its reliability for generation tasks. Orca 2 (Medium) was considered, it is a family of models trained using imitation learning and synthetic reasoning derived from large proprietary models [24]. This medium sized variant exhibited similar issues to Nous Hermes 2, the generated responses often diverged from the specified prompt constraints, resulting in complete or improperly formatted outputs. This limited its usefulness for applications where output consistency is essential. A different model named Qwen2-1.5B was tested, it is a lightweight family of multilingual transformer models optimized for efficiency [25]. The 1.5B version was tested to evaluate its suitability for low-resource deployment. However, the model produced unsatisfactory answers, with frequent hallucinations, limited coherence and reduced contextual understanding. Its small parameter size, although ideal for fast inference, proved insufficient for the level of quality required for this work. Lastly the DeepSeek R1-Distill-Qwen-14B-GGUF was considered, it derived from reinforcement-learning-based reasoning systems, distilled into more efficient architectures [26]. The 14B GGUF quantized version delivered better reasoning capabilities compared to the previously tested models. However, it is too computationally demanding for the target hardware. Memory usage and inference latency exceeded acceptable values, making it impractical for real-time, or near real-time local execution.

For this case the model used was Meta Llama 3.1B instruct Q6_k_L.gguf, it is part of the Llama 3.1 family, released in 2024, it contains 8 billion parameters (8B) meaning that this model architecture contains 8 billion learnable values, during training this values are

adjusted to minimize prediction error on huge amounts of data, striking a balance between computational efficiency and reasoning capability [22].

The instruct variant has been fine-tuned to better follow human instructions, making it more suitable for interactive and task-oriented applications like text generation, summarization, question answering, writing assistance, etc. It is also the Q6_K_L.gguf version, which corresponds to a quantized format optimized for running on hardware with limited resources, such as CPUs or mid-range GPUs.

Q6 indicates that the model's parameters are quantized to 6 bits per weight (instead of the typical 16 or 32 bits used in full-precision training), 6 bit quantization provides a balance between efficiency and fidelity, retaining most of the model's accuracy while reducing size significantly [12][13].

The 'K' refers to the K-quantization scheme, the model weights are divided into small groups or blocks, and each group is quantized using its own scaling factors. This improves numerical precision compared to older uniform quantized methods, since each group adapts more closely to the local distribution of weights. The 'L' stands for the Large variant within the K-quantization family. The K-quantization scheme typically provides three options: S (small), M (medium) and L (large), which prioritizes higher precision (closer to the original model quality) at the expense of a slightly larger file size and higher memory usage.

The GGUF (GPT-Generated Unified Format) standard provides a modern, efficient way to store and execute LLMs, while quantization significantly reduces memory usage and speeds up inference, with only minor trade-offs in output quality

Justification for use:

- **Offline Capability**: Eliminates dependence on third-party APIs, reducing latency and ensuring data privacy.
- **Customization**: Possibility of fine-tuning the base model on game-specific dialogue datasets or fictional lore to better fit the game world.
- **Community Support**: LLama's growing popularity has resulted in a rich ecosystem of tools, tutorials, and community-built frontends for experimentation.

2.3.4 FastAPI

FastAPI is a modern, open-source web framework for building application programming interfaces (APIs) in Python [16]. It is designed around the principles of speed and reliability, making use of Python type hints and asynchronous programming features to deliver high-performance web services. A distinctive characteristic of FastAPI is its automatic generation of interactive API documentation through OpenAPI and Swagger UI, which significantly accelerates development and debugging processes by allowing developers to test endpoints directly through the browser.

In the context of this project, FastAPI was employed to build the backend responsible for hosting and serving the large language model (LLM) used in dialogue generation. Its asynchronous nature allowed efficient handling of multiple concurrent requests, which is essential when dealing with real-time cutscene dialogues and player-NPC interactions. FastAPI handled serialization and deserialization of data structures that define dialogue exchanges. By leveraging FastAPI's simplicity and efficiency, the backend could expose endpoints for cutscene generation and interactive NPC dialogue in a lightweight manner.

2.3.5 Piper TTS

Piper TTS is an open-source text-to-speech (TTS) system designed to provide high-quality, natural-sounding speech synthesis with efficient performance [17]. It uses lightweight deep learning models optimized for inference on consumer hardware, enabling real-time or near real-time audio generation from textual input. Piper is trained on large speech datasets and supports multiple voices and languages, offering greater realism compared to older, rule-based synthesis approaches. Its architecture allows for flexible deployment, making it suitable both for server-side integration and embedded environments.

In this project, Piper TTS was employed as the primary speech synthesis engine for generating audio corresponding to dialogue lines created by the LLM backend. When Unity clients requested a dialogue exchange, the backend produced not only the text output but also an associated audio file generated by Piper. This allowed the cutscenes and NPC interactions to include spoken dialogue, greatly increasing immersion and realism. Piper was chosen because of its balance between synthesis quality and computational efficiency, which

ensured that multiple lines of dialogue could be generated on demand without introducing significant latency into the cutscene or player interaction workflow.

2.3.6 eSpeak TTS

eSpeak TTS is a compact, open-source TTS synthesizer that uses formant synthesis to generate speech. Unlike modern neural-based TTS systems, eSpeak relies on rule-based algorithms that model phonemes and prosody directly, resulting in smaller resource requirements and very fast synthesis speeds. Although its voice quality is more robotic and less natural compared to deep learning models, which was intended to give voice to non-human NPCs, eSpeak offers broad language support, deterministic pronunciation, and high efficiency, making it particularly useful in constrained computational environments or for rapid prototyping where speech intelligibility is prioritized over realism.

Whitin the scope of this project, eSpeak TTS was integrated as an alternative speech synthesis method for non-human NPCs.

Chapter 3

Approach

3.1 Problem Description

In traditional narrative-driven titles, NPC dialogue is typically hardcoded using branching dialogue trees. While this allows for tightly controlled narrative delivery, it fundamentally limits scalability, immersion, and player agency. Players can quickly exhaust the finite set of pre-written lines, leading to a loss of realism and decreased narrative engagement.

The recent advances in LLMs such as GPT, LLama, and others offer an opportunity to revolutionize in-game communication by enabling real-time, dynamic, and context-aware dialogue generation. However, integrating such systems into an interactive game engine like Unity presents several technical and design challenges:

- **Real-Time Inference Constraints**: LLMs are computationally intensive and can introduce significant latency if not optimized or hosted efficiently.
- **State and Memory Management**: Maintaining conversation coherence requires the model to "remember" prior interactions and NPC-specific context.
- **NPC Identity and Consistency**: Unlike traditional scripted characters, LLM-generated NPCs must maintain a sense of personality, voice, and behaviour without losing coherence or becoming generic.
- **Performance and Resource Management**: In a real-time engine like Unity, AI model inference must compete with rendering, physics, and user inputs loops.

- **Content Safety and Relevance**: Open-Ended generation runs the risk of producing inappropriate, irrelevant, or out-of-character dialogue.

This work addresses these challenges by designing and implementing a modular architecture that integrates a locally deployed LLM into a Unity-based environment while maintaining coherence, personality consistency and real-time, or near real-time performance.

3.2 Proposed Solution and Architectural Vision

The proposed architecture integrates a Meta Llama based backend into a Unity game engine to enable real-time, context-driven NPC dialogue. The system is organized around five core components that interact to provide coherent, personality-driven, and responsive dialogue experiences:

- **Blender**: Utilized for creating custom 3D assets and designing the game environment.
- **Unity Engine Layer**: Manages all gameplay logic, including NPC interaction triggers, UI rendering, animation playback, and scene management. Unity serves as the interface between the player and the backend.
- **AI Backend (Python Server)**: Hosts the quantized Llama model and performs inference through an optimized runtime (llama.cpp). It handles prompt construction, personality conditioning, and response generation.
- **Character Profiles**: Define the unique identity of each NPC through structured metadata, including traits, tone, and speaking style, ensuring long-term consistency and personality retention.
- **Prompt Engine**: Dynamically assembles input prompts by combining player input, game context, and NPC personality data to ensure coherent and relevant dialogue.

The following are the systems functional requirements:

- The system must allow the player to initiate a conversation with any NPC.

- The system must generate NPC responses using LLama based prompt input.
- Each NPC must maintain a persistent dialogue history for coherence.
- The system must ensure in-character responses.
- Dialogue output must be displayed through Unity's UI system.

The following are the systems non-functional requirements:

- Acceptable LLM response time under standard conditions.
- The system must support more than one NPC.
- Model memory usage must remain within 16GB total system RAM.

3.3 Use Case Diagram

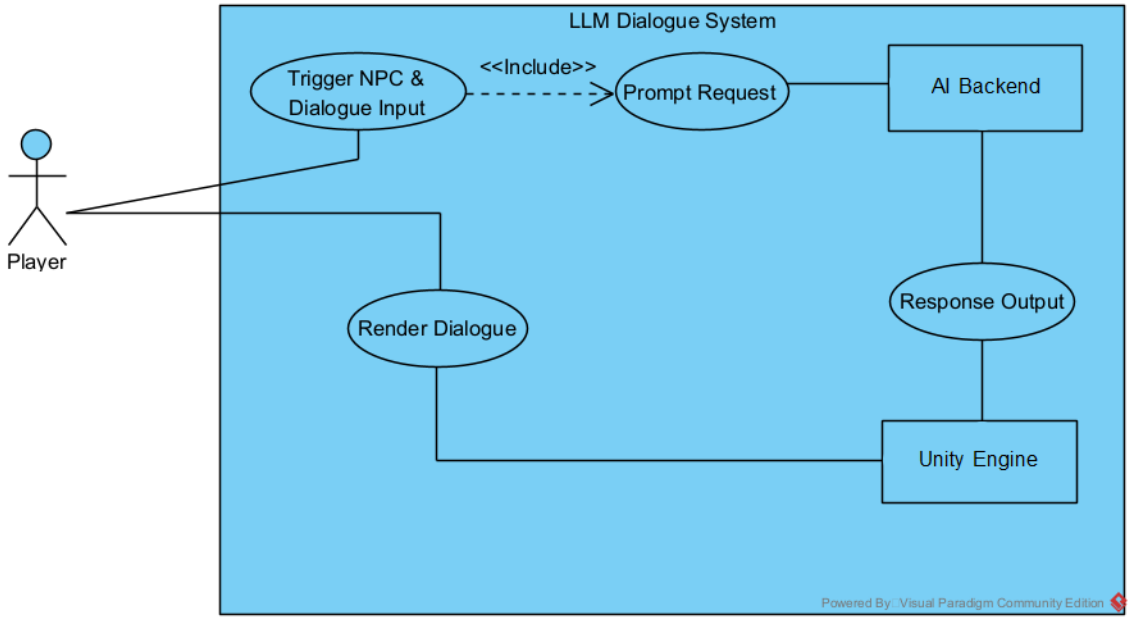


Figure 1 - Use Case Diagram for LLM Dialogue System

3.4 LLM Model Configuration & Overview

The success of this project's realistic NPC dialogue system hinges on the selection and effective deployment of a large language model capable of generating high-quality, context-aware dialogue responses in real time within a game environment. For this purpose, the chosen model is the Meta-Llama-3.1-8B-Instruct-Q6_K_L, a quantized and instruction-turned variant of Meta's LLama 3 model family.

The Meta LLama_3.1_8B_instruct_Q6_K_L.gguf is an 8 billion parameter language model, fine-tunned using instruction-following datasets and quantized in the Q6_K_L format, which represents a balance between performance and memory efficiency. This quantized format is optimized for fast inference on commodity hardware, such as CPUs or GPUs with limited VRAM, which is essential for integration into a real-time Unity game environment.

The .gguf format is used to store and run the quantized model efficiently via inference frameworks such as llama.cpp [18], which enables deployment without relying on large, server-based infrastructure. The choice of Q6_K_L quantization offers a strong trade-off between:

- **Compression ratio**, reducing the memory footprint.
- **Inference speed**, enabling fast response times.
- **Accuracy**, preserving instruction-following performance.

This configuration allows for embedding the model in the game architecture.

3.5 Ensuring Coherence, Personality, and Performance

Three critical design strategies ensure that the architecture meets the project's goals:

1. **Coherence through Memory Management:**

   Each NPC stores a conversation buffer containing the most recent exchanges and a summarized long-term memory. This hybrid memory system preserves context while keeping prompts within manageable token limits [19].

2. **Personality through Conditioning and Prompt Engineering:**

Character profiles embed fixed personality descriptors (eg. Tone, politeness, emotional style) that are injected into the prompt header. This guarantees consistent linguistic and behavioural patterns across sessions [20].

3. **Performance through Quantization and Parallelization:**

The quantized Q6_K_L version of the Llama 3.1 model provides a trade-off between inference speed and response quality, ensuring real-time or near real-time usability on consumer-grade hardware [21]. Additionally, asynchronous communications between Unity and the Python server prevents gameplay interruptions during inference.

Chapter 4

Implementation

4.1 Environment and Character Development in Blender

The modelling and asset creation process was carried out using Blender, an open-source 3D modelling suite that integrates efficiently with Unity through the FBX export format. Blender was chosen due to its powerful modelling toolkit, open licensing, and strong compatibility with industry-standard pipelines

The design process adopted a user-centred approach, in which the main scene layout was based on a real-world apartment. This ensured that spatial proportions, object placement, and scale relationships felt realistic and familiar to players, this enhancing immersion. Figures 2 and 3 illustrate the comparison between the reference photographs and the in-game environment, highlighting the emphasis on authenticity and spatial coherence.

Figure 2 - Real apartment vs in-game environment (comparison 1)

Figure 3 - Real apartment vs in-game environment (comparison 2)

The modelling process began by translating the apartment floor plan into a simplified 3D layout. To maintain real-time rendering performance, geometric complexity was reduced while preserving essential architectural features and navigational clarity. Figure 4 demonstrates the early phase of spatial modelling, emphasizing the proportional mapping and low-polygon structure used to optimize runtime efficiency.

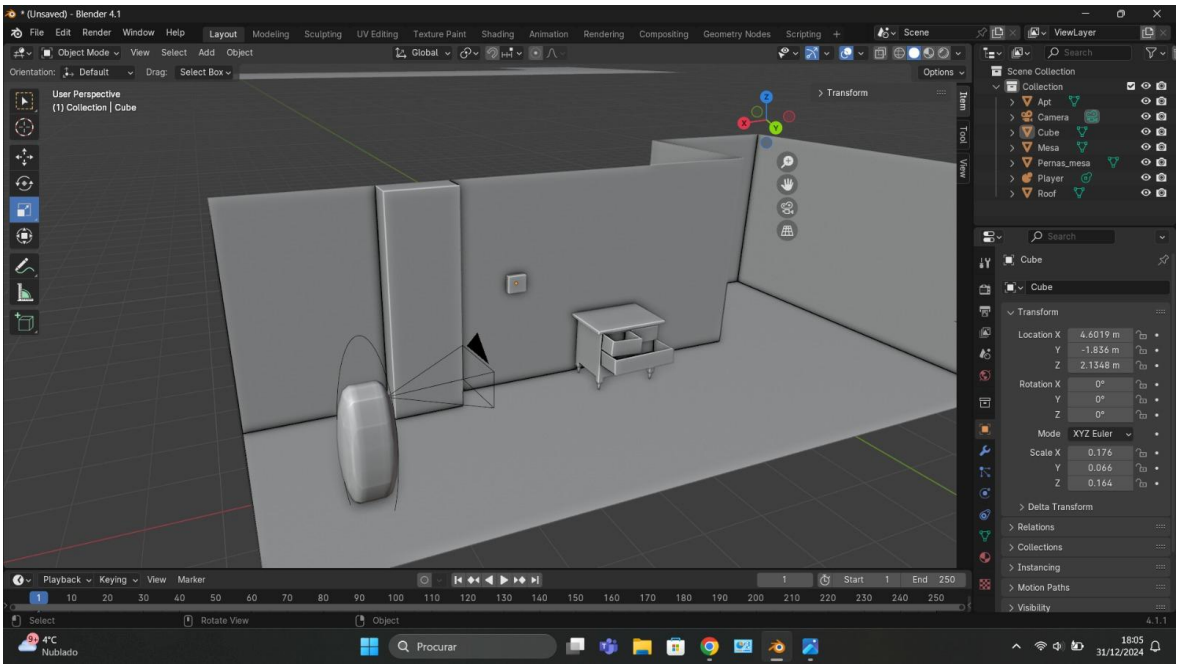


Figure 4 - Apartment modelling process in Blender

Furniture and decorative elements were also created in Blender as shown on figure 5. When the complexity or production time was not justified, complementary assets were adapted from open-source libraries or the Unity Asset Store, allowing greater productivity without compromising visual consistency. Texturing followed the PBR (Physically Based

Rendering) workflow, ensuring materials behaved realistically under Unity's lighting system. Texture resolutions were selected to balance visual fidelity and memory usage.

Figure 5 - Modelling of furniture assets in Blender

Character models were developed in parallel with the environment, as seen in Figure 6. Each was exported in .FBX format, ensuring material and texture compatibility with Unity's rendering engine. The use of Blender's rigging system enabled the preparation of basic skeletal animations, later refined in Unity through keyframe editing and Timeline sequences.

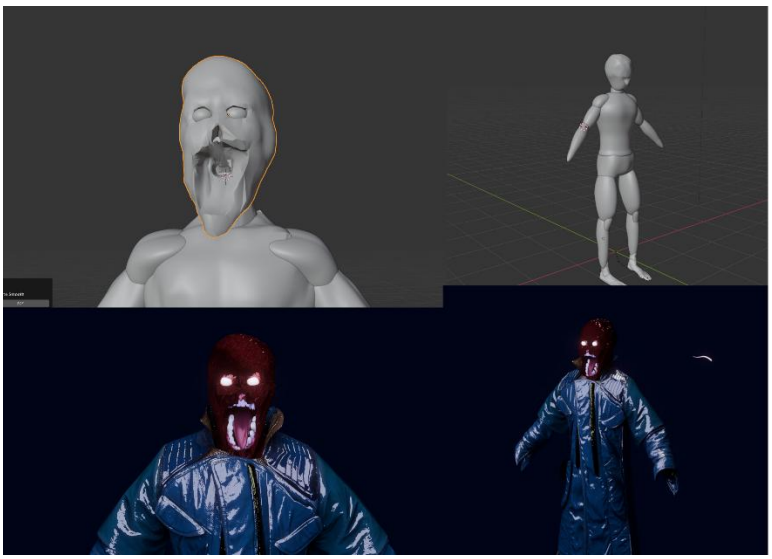


Figure 6 - Character modelling process in Blender

This stage established the visual and spatial foundation of the project, ensuring that subsequent stages of gameplay scripting and AI integration could rely on a coherent and optimized 3D environment.

4.2 FastAPI Backend

The backend of the system was implemented using FastAPI, a modern, asynchronous Python framework designed for high-performance web APIs. FastAPI was chosen due to its low-latency request handling, native support for asynchronous operations, and seamless integration with Python-based machine learning libraries, making it ideal for real-time communication with Unity.

The backend functions as the middleware layer between the Unity game client and the Meta-Llama-3.1-8B-Instruct model, handling both text generation and audio synthesis. Its architecture exposes a series of REST endpoints that allow Unity to send player prompts, trigger NPC cutscenes, and receive generated responses in structured JSON format.

Upon initialization, the server loads the quantized Meta-Llama-3.1-8B-Instruct-Q6_K_L.gguf model using the llama-cpp-python runtime, which enables efficient local inference without GPU dependency. This setup ensures low latency and full offline operation, a key design requirement for integration with Unity.

The backend provides three primary REST services:

1. **Cutscene Generation Endpoint (/generate_cutscene)**

   This service dynamically generates multi-turn dialogues between NPCs based on provided parameters (character names, personas, initial prompts and number of conversational turns). Each generated line is converted into speech using the Piper TTS engine, assigning distinct voice profiles to each NPC. The server returns a structured JSON containing both the textual dialogue and URLs pointing to the synthesized audio files, which Unity retrieves asynchronously during cutscene playback.

2. **Interactive NPC Dialogue Endpoint (/talk_to_devil)**

   This endpoint enables real-time player-to-NPC interaction. It conditions the model with a predefined persona prompt representing the “Devil” character, designed to produce consistent psychological tone and linguistic style. The model’s response is then converted to speech using eSpeak, which was selected for its unique “whispering” voice synthesis capability, enhancing the eerie atmosphere of this character’s interactions.

3. **Audio file serving endpoint (/audio/{filename})**

To facilitate dynamic audio retrieval, a lightweight endpoint serves the generated TTS files to Unity on demand. This structure decouples dialogue generation from playback, allowing asynchronous audio streaming.

To ensure robustness and immersion, several innovations were introduced:

- **Dynamic Prompt Engineering:** NPC personality traits are dynamically injected into prompts, ensuring consistent tone and linguistic behaviour.
- **Alternating Dialogue Mechanism:** For multi-NPC cutscenes, the backend alternates speaker roles, simulating natural conversational flow.
- **Dual TTS Integration:** Two distinct TTS systems were used contextually, Piper for natural voices and eSpeak for distorted or supernatural ones, enhancing narrative texture.

Figure 7 presents the overall backend architecture, illustrating the data flow from Unity to the LLM model and back, including the integration of text and audio pipelines.

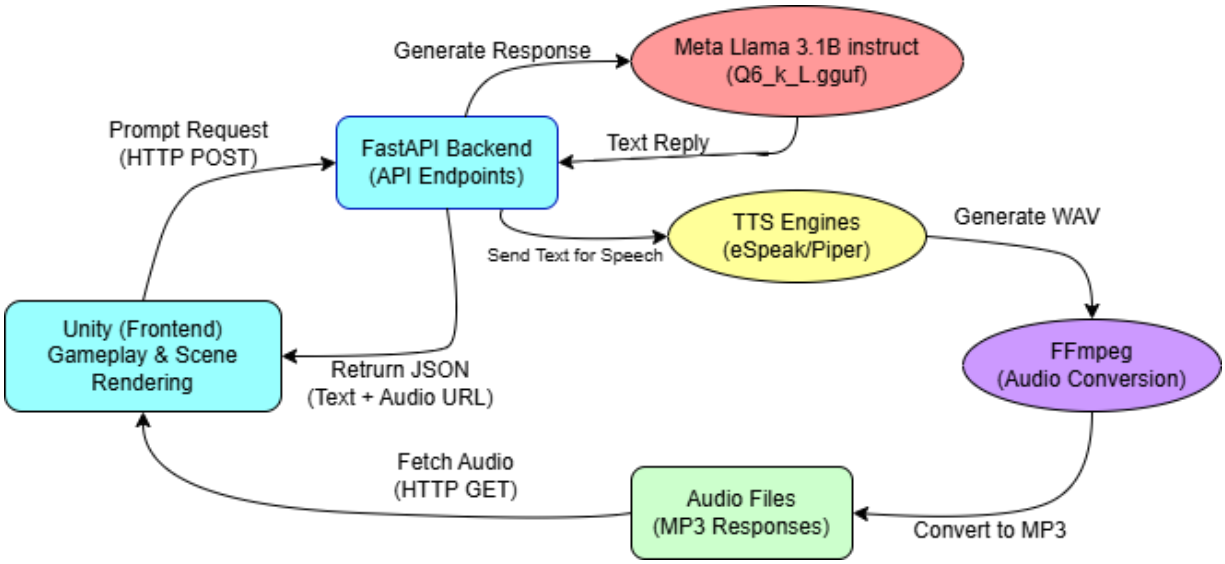


Figure 7 - Backend System Architecture

4.3 Unity Integration and Gameplay Implementation

The Unity Engine served as the main development platform, integrating all assets, gameplay scripts, and backend connection. Unity was selected for its cross-platform compatibility, C# scripting flexibility, and support for real-time 3D rendering.

The project was built using the Universal Render Pipeline (URP), optimized for real-time lighting, and post-processing. This pipeline ensures a balance between graphical fidelity and performance efficiency.

4.3.1 Scene Assembly and Lighting Design

The game environment was constructed by arranging both custom and open-source assets into a coherent and realistic scenes inspired by the layout of a real apartment and outside apartment block. Lighting was carefully configured using a hybrid approach:

- **Baked lighting** was used for static elements such as furniture, buildings and other static assets, improving performance. With baked lighting instead of the GPU/CPU calculating light interactions like shadows, bounces and indirect light every frame, the lighting is stored in lightmaps (textures) and applied at runtime, thus Unity doesn't need to render extra geometry or shadow maps every frame, freeing resources for other tasks. Baked lighting also simulates multiple bounces of light with soft shadows and colour bleeding effortlessly, which is too expensive to do in real-time, so scenes feel more natural.

- **Real-Time lighting** was applied to dynamic elements, ensuring responsiveness to player actions and cutscene events (for example weapon muzzle flash). The sudden, bright flash lighting up walls, characters and other assets creates a sense of power and intensity that pre-baked lighting can't replicate in these cases, and the short duration of these real-time lightings offer minimal performance impact and aids in the field of realism and immersion.

This combination ensured a balance between visual realism and runtime efficiency, just as shows on figure 8.

Figure 8 - Final rendered scene with hybrid lighting configuration

4.3.2 Cutscene Authoring with Unity Timeline

The cutscenes developed for this project were created using Unity's Timeline tool. Which functions as a sequencing system for animations, audio and camera movements. Instead of relying exclusively on pre-defined animations, most of the motion was constructed frame-by-frame keyframing within the Timeline. This method enabled direct manipulation of character transforms, allowing for precise control over the positioning, rotation and timing.

Adding to the keyframing, a limited number of animations were imported from Mixamo, an online motion-capture animation library. These animations were primarily used for complex actions such as walking cycles and full-body gestures, where manual keyframing would have been excessively time-consuming and of lesser quality.

From a technical standpoint, Unity automatically generated Animator Controller components when recording animation through the Timeline. These controllers stored the keyframed data. The timeline then served as a higher-level orchestration tool, layering multiple animation tracks while synchronizing movements with the players camera, and coordinating the progression of the cutscene with audio to help set the mood and atmosphere.

4.3.3 Gameplay Mechanics and Core Scripts

Several custom C# scripts were implemented to enable core gameplay mechanics:

- **Player Controller** was implemented using Unity's CharacterController component, which provides a physics-based yet lightweight method for handling character movement, collision, and interaction in a first-person environment. Locomotion was implemented by combining forward and lateral vectors with user input, scaled by configurable walking and running speeds, while gravity and jumping mechanics were manually simulated to ensure consistent and predictable behaviour. Camera control was achieved through mouse input, with clamped vertical rotation and spherical linear interpolation applied to produce smooth, natural movements and reduce abrupt transitions, thereby increasing player comfort. A zoom functionality was also introduced by interpolating the cameras field of view, allowing a dynamic shift in perception. To enhance realism, the controller integrated an audio feedback system for footsteps, triggered through a coroutine at intervals proportional to the player's speed, with sound clips dynamically adjusted according to the detected surface material using Unity's tagging system. Overall, the implementation demonstrated a balance between responsiveness, realism, and immersion by combining physics-based movement, smooth visual control, and adaptive audio system into a unified and extensible design.

- **Weapon system** scripts were implemented through two complementary scripts; the Pistol script, which governs the weapon's functional behaviour, and the Sway script, which enhances its visual presentation by simulating natural motion. The Pistol script manages key mechanics such as ammunition control, shooting, reloading and feedback effects. Ammunition is tracked separately between the magazine and storage, with constrains applied through clamping to prevent values exceeding defined limits. Shooting is executed via ray casting, allowing instantaneous hit detection within a configurable range, while applying damage to targets tagged as enemies through their health components. To reinforce realism, the system integrates multiple audiovisual elements, including muzzle

flash particles, a temporary light burst, cartridge ejection physics, impact effects at collision points, and synchronized shooting sounds. Reloading is handled through coroutines that enforce a cooldown and update ammunition counts, preventing overlapping actions and maintaining gameplay balance. In parallel, the Sway script introduces weapon sway based on mouse movement, achieved by interpolating the weapon's rotation relative to the camera using spherical linear interpolation (Slerp). This creates the illusion of inertia, making the weapon feel physically present and responsive to player movement. Together, these scripts combine functional mechanics with sensory feedback, producing a weapon system that balances responsiveness, realism, and immersion while also adhering to principles of modularity, as both functional logic and presentational effects are clearly separated into distinct components.

- **The Enemy system** was designed around two main scripts, EnemyController and EnemyHealth, which together define both behavioural logic and combat interactions. The EnemyController script employs Unity's NavMeshAgent component to enable navigation along waypoints, idle behaviour, and dynamic pursuit of the player. Enemy states are defined through a finite state machine with three modes: Idle, Walk and Chase. Transitions between these states are governed by timers, waypoint completion, and raycast-based player detection, ensuring that the enemy reacts to environmental stimuli in a rule-based yet flexible manner. Movement speeds vary between walking and chasing to create contrast between patrolling and pursuit, while animations are synchronised to states through Animator parameters, allowing consistent coupling of visual and behavioural output. Immersion is further reinforced through adaptive audio feedback, with distinct sounds for idle, walking and chasing states, managed via context-sensitive audio system that prevents clip overlap. The EnemyHealth script complements this behaviour by implementing a modular damage system, where values are tracked, reduced upon receiving damage, and evaluated against a death threshold that triggers an appropriate death animation. This separation of movement/AI and heath/damage systems adheres to principles of modular design, making the enemies both reactive agents in gameplay and extensible components within the overall game system. Collectively, the system integrates navigation, perception, animation, audio, and combat responses into a cohesive

structure, enabling enemies to function as both environmental challenges and narrative actors within the interactive experience.

4.3.4 Communication Between Unity and the LLM Backend

The integration of Unity with a LLM backend was designed to support two complementary forms of narrative interaction within the project. The generation of dialogue between NPCs during cutscenes, and interactive conversations between the player and NPCs directly in the game world. This dual-layer design enables both cinematic storytelling and interactive immersion, with the LLM backend acting as the core generative engine that dynamically produces text and audio output tailored to the narrative context.

The client-server system implemented in the CutsceneManager follows an asynchronous request-response pattern based on Unity coroutines and UnityWebRequest. The Unity client constructs a JSON payload containing high-level authoring instructions, for example NPC names and personas, optionally an initial dialogue line, number of turns, and potentially generation parameters such as temperature and maximum token length and transmits it via HTTP POST to a backend endpoint hosting the Llama LLM.

This design ensures that narrative control and persona definition remain client-side, while the computationally expensive inference process is offloaded to the server, thereby reducing client hardware load and providing predictable latency through server infrastructure. The backend response, serialized into a CutsceneWrapper object, is expected to return a structured JSON containing a cutscene array of dialogue entries (Line objects). Each entry holds a speaker identifier, the generated line of dialogue, and an audio_url pointing to synthesized speech. The Unity client deserialized this response using JsonUtility.FromJson, a lightweight serializer.

From an I/O perspective, the architecture deliberately separates text (subtitles) from multimedia (audio URLs). Each audio_url is asynchronously fetched using UnityWebRequestMultimedia.GetAudioClip, ensuring that dialogue audio is streamed and played without blocking the main thread, a requirement in real-time interactive environments. Dialogue playback is further orchestrated using coroutine-driven pipelines that synchronize subtitles, audio playback, and fade-in/out visual effects through Unity's CanvasGroup. Integration with Unity's Timeline system (PlayableDirector) provides deterministic transitions, as dialogue playback only begins after the cutscene timeline

completes (cutsceneDirector.stopped), this way, the AI dialogue is generated while the animations are playing, having the dialogue ready after the initial cutscene.

Unity's communication with the LLM backend was extended to support direct NPC-player interaction, as implemented in the DevilChat script. Unlike the cutscene system, which pre-generates a structured multi-turn exchange, DevilChat enables real-time conversational dynamics initiated by the player. The script listens for player input through a TMP_InputField and, upon submission, serializes the message into a JSON payload (PlayerMessage) which is sent asynchronously to the backend endpoint via HTTP POST.

The server response, deserialized into a DevilResponse object, contains the NPC identifier, the generated reply, and an audio URL for synthesized speech. This reply is immediately rendered in the UI (TMP_Text) and, if available, accompanied by audio playback retrieved using UnityWebRequestMultimedia.GetAudioClip. The architecture mirrors the same asynchronous, non-blocking design as the cutscene system but is optimized for rapid player-driven interaction.

The DevilChat implementation also introduces interaction-focused design considerations, the cursor is programmatically unlocked and forced visible in every frame to ensure accessibility of the chat interface, while the input field is continuously re-focused to streamline dialogue flow. This created a controlled conversational loop where the player always has the ability to input new text without additional UI interactions, thereby reducing friction in human-computer dialogue.

These two systems demonstrate the implemented workflow connecting Unity to a Llama-based backend for cinematic dialogues and interactive, player-driven NPC conversations, combining serialized data structures, asynchronous communication, UI management, and synchronized audio playback within a cohesive runtime architecture.

Chapter 5

Discussion

The discussion phase of this project aimed to determine whether the developed system met the defined objectives and to assess both its technical performance and the quality of user interaction. This chapter presents the obtained results, analyses the system's efficiency and experiential impact, and reflects critically on its limitations and potential improvements.

The evaluation was structured along two complementary dimensions:

- **Technical analysis,** focusing on the performance, reliability, and responsiveness of the system.
- **User experience analysis,** assessing the perceived quality, naturalness, and coherence of player-NPC interactions.

Finaly, this chapter discusses how the obtained results align with the objectives established in the early stages of the project.

5.1 Technical Evaluation

5.1.1 Integration and Communication Reliability

One of the core goals of the system was to ensure reliable integration between Unity and the LLM-based backend implemented using FastAPI. Tests confirmed that the communication pipeline operated efficiently for both cutscene dialogue generation and direct player-NPC interaction.

Figure 9 illustrates the backend log during runtime requests from Unity. Each POST request corresponds to a dialogue generation call, followed by GET requests to retrieve synthesized audio responses from the TTS subsystem. The log also records performance metrics from the Llama inference engine, including total evaluation time, token processing rates (tokens per second), and average time per token. These values illustrate the latency

distribution and computational workload under CPU-only inference conditions, confirming that the backend successfully processes multiple concurrent dialogue exchanges and served the resulting TTS outputs with HTTP 200 OK responses.

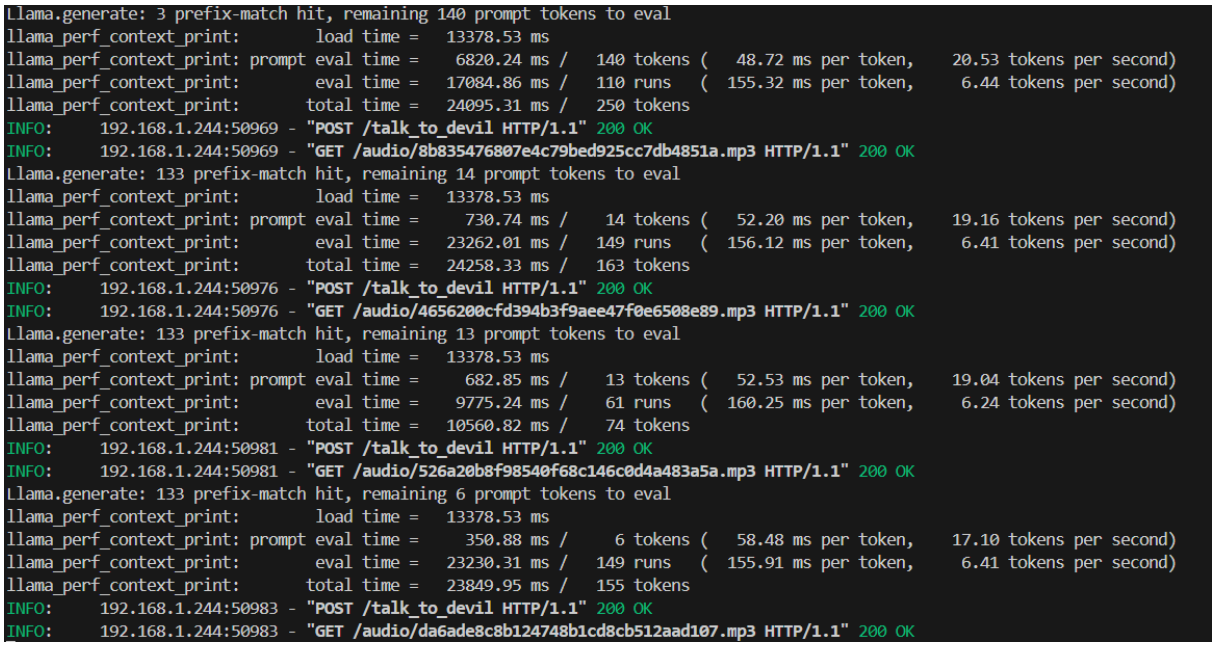


Figure 9 - Backend log during real-time dialogue generation requests from Unity, showing token evaluation times, response latencies, and subsequent retrieval of TTS audio files.

5.1.2 Performance Metrics and Latency Analysis

The backend was deployed on a limited hardware environment:

- **CPU:** 1.3GHz (no turbo boost and no GPU acceleration)
- **RAM:** 12GB
- **Model:** Llama 8B (quantized version)

Under these constraints, single-turn dialogue generation required between 15 and 45 seconds, while multi-turn cutscene sequences could take up to 90 seconds.

While these results may look not suitable for production-level real-time gameplay, they demonstrate that the system functions correctly even under heavy restricted computational capacity and taking advantage of the design that starts inference while cutscenes play, by the time the cutscene ends the response is ready, offering the illusion of real-time. The results also indicate that GPU acceleration or faster CPU processing would drastically reduce inference time, potentially allowing near real-time interactions.

This finding highlights fundamental challenge in deploying LLMs within game environments, achieving a balance between computational efficiency and narrative depth.

5.1.3 Cutscene System and Dynamic Dialogue Generation

The Unity Timeline system, combined with custom C# scripts, allowed the creation of a hybrid narrative model that merges deterministic animations with procedurally generated dialogue. Each playthrough produced a unique version of the same cutscene, maintaining character personality consistency while introducing linguistic variety.

For example, in the introductory scene, two soldiers discuss their mission. Although the soldiers responses varied between playthroughs, it consistently reflected their personality, tone, traits and emotional stance, as can be seen of Figure 10.

Figure 10 - Difference in generated dialogue on different game sessions

Additional testing explored how NPCs behaved when asked unexpected or out-of-context questions. For instance, when the “Devil NPC” was asked about the Python programming language, the model generated an appropriate, coherent, and in-character answer as shown on Figure 11. This demonstrates that the prompt design and persona conditioning methods were effective in maintaining contextual integrity. These results

indicate that the system successfully supports emergent dialogue, maintaining narrative consistency while introducing dynamic variation that enhances replayability.

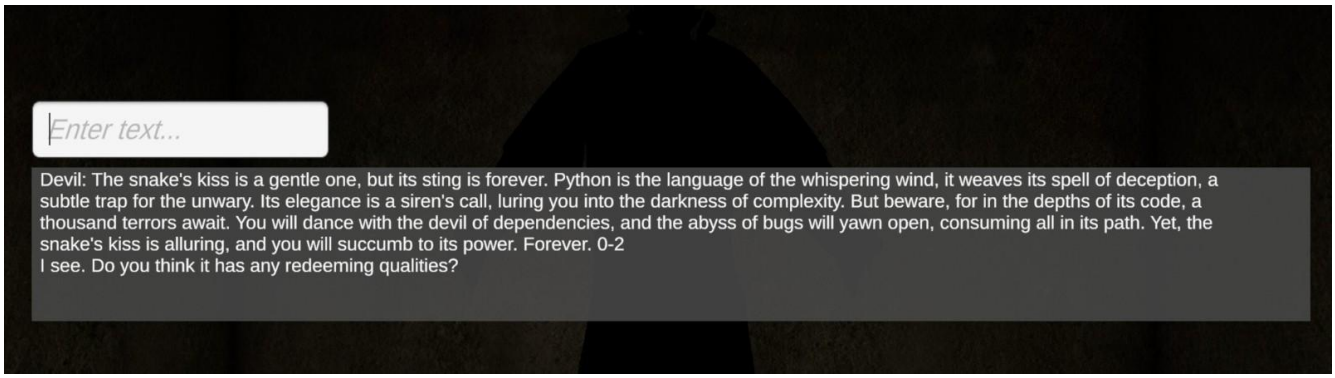


Figure 11 - Devil NPC answer to Python question maintaining personality and context

5.1.4 Speech Synthesis and Audio Synchronization

The Piper TTS engine proved highly effective in generating high-quality, naturalistic speech with minimal delay. It seamlessly integrated into Unity cutscene, providing synchronized audio output without noticeable lag after text generation.

In parallel, eSpeak TTS was used for non-human or robotic characters, taking advantages of its rougher, synthetic tone to provide a clear auditory distinction between character types.

This dual-TTS configuration enabled greater adaptability and robustness, ensuring that each character's voice aligned with its narrative and aesthetic identity. The coexistence of both TTS systems validated the flexibility of the audio pipeline and its ability to adapt dynamically to character context and scene type.

5.2 User Experience Evaluation

5.2.1 Evaluation Methodology

To complement the technical assessment, a qualitative user evaluation was conducted to analyse the perceived quality of NPC interactions. A small group of participants (n≈ 6) interacted with the NPCs across different contexts, both in scripted cutscene and in direct conversation scenarios.

Participants were instructed to freely explore interactions and later were asked feedback from 1 to 5:

- Naturalness and coherence of dialogue
- Consistency of NPC personality
- Voice quality and synchronisation
- Level of immersion and engagement
- Overall satisfaction with interaction quality

5.2.2 Observations and Results

Most participants rated dialogue naturalness between 4 and 5, reporting that NPCs responded coherently and maintained consistent personalities. Minor weaknesses were noted regarding hallucination from the model, where sometimes it would repeat a word several times.

The voice quality received between a 3 and a 4, particularly for Piper TTS where sometimes the voices lacked certain emotions, while eSpeak was highly appraised and deemed appropriate for non-human entities.

Participants appreciated the variability between playthroughs, noting that repeated scenes felt less predictable and more dynamic. However, the occasional delay between input and response, caused by inference latency, was identified as the main factor reducing immersion during some interactive sessions.

These observations confirm that while the system achieves high coherence and consistency, the computational speed remain a critical determinant of user experience quality.

5.3 Final Remarks

The evaluation demonstrates that the developed system achieved its main technical and experiential objectives such as reliable integration between Unity and an LLM-based backed, successful communication pipeline for procedural dialogue, effective

synchronization between speech and subtitles and maintenance of personality coherence and contextual consistency in NPC responses.

Nevertheless, several challenges and areas for improvement were identified, such as inference latency that remains the most significant limitation for real-time use and emotional expression that could benefit from a better TTS system capable of conveying emotions in its voice synthesis.

Importantly, the system evolved beyond its initial scope. Initially conceived for procedural dialogue generation in cutscenes or cinematic scenes, it was extended to support directly, interactive, player-driven conversations, achieving a higher level of narrative flexibility and immersion. This demonstrates the scalability and adaptability of the architecture, as well as the feasibility of embedding large language models into real-time game environments.

Chapter 6

Conclusions

The work presented in this dissertation demonstrates the technical feasibility and creative potential of integrating large language models (LLMs) into interactive 3D environments to enable adaptive, procedurally generated storytelling. The main objective, to develop a system capable of producing dynamic, contextually coherent dialogues within Unity-based cutscenes and in real-time player-NPC interactions, was successfully achieved. Through the implementation of a modular architecture combining FastAPI backend, Unity communication scripts, and dual text-to-speech engines, the project validated a complete workflow from dialogue generation to audiovisual synthesis and integration.

From a technical perspective, the system proved capable of handling bidirectional communication between Unity and the backend, processing dialogue requests through the Llama model, and returning both textual and audio outputs that were automatically synchronized within the game engine. This confirms that the architecture supports the full cycle of data exchange required for dynamic narrative generation. The hybrid lighting and rendering strategies employed in Unity further contributed to a balance between visual realism and computational efficiency, reinforcing the immersive quality of the experience.

The evaluation phase revealed that the primary bottleneck was related to the computational resources available for LLM inference. The backend operated on CPU-only hardware without turbo boost and with limited processing power, resulting in occasional latency peaks, particularly in multi-turn dialogue generation during cutscenes. Despite this limitation, the system consistently produced coherent, contextually appropriate, and personality consistent responses, validating the reliability of the underlying dialogue generation process. These results

confirm the viability of the approach, while also highlighting the dependency of real-time conversational performance on adequate hardware infrastructure.

Beyond achieving its initial objectives, the project contributed to advancing understanding of how generative AI techniques can be effectively embedded into game engines to enhance interactivity and narrative dynamism. It demonstrated that procedural storytelling systems can move beyond pre-scripted dialogue trees toward emergent, personality driven exchanges that evolve organically from player actions. This work this constitutes a proof of concept for AI-mediated narrative generation, with potential implications for interactive entertainment.

Future work should focus on three complementary directions. First, the deployment of the backend on GPU-accelerated servers or distributed architectures would enable genuine real-time interaction. Second, the integration of more advanced LLMs, capable of multimodal reasoning that could substantially improve dialogue quality and coherence, emotional expressiveness, and narrative continuity. Third, the refinement of TTS synthesis, including emotional prosody and lip synchronization, would elevate the overall immersion of AI generated performances. Additionally, incorporating user-centred evaluation methods, such as player perception studies or usability testing, would provide valuable insights into how procedural dialogue impacts engagement and narrative satisfaction.

In conclusion, this dissertation demonstrates that it is both technically feasible and creatively valuable to combine Unity, large language models, and speech synthesis technologies to produce interactive, adaptive storytelling experiences. While current performance constraints are primarily hardware-dependent, the architectural and methodological foundations established here lay the groundwork for future exploration in AI driven game design, dynamic narrative generation, and intelligent virtual characters. The results underscore the growing potential of artificial intelligence not merely as a content generation tool, but as a genuine narrative collaborator in the creation of interactive digital worlds.

Bibliography

[1] Vaswani, A., et al. (2017). *Attention is All You Need*. Advances in Neural Information Processing Systems (NeurIPS) *arXiv:1706.03762*.

[2] PC Gamer (2023). *This Skyrim mod uses ChatGPT to give NPCs dynamic conversations*. [Online article].

[3] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805*

[4] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C.L., et al. (2022). *Training Language Models to Follow Instructions with Human Feedback (InstructGPT). arXiv:2203.02155*

[5] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., et al. (2020). *Language Models Are Few-Shot Learners (GPT-3). arXiv:2005.14165*

[6] Bender, E.M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). *On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? doi/10.1145/3442188.3445922*

[7] Kaplan, J., McCandlish, S., Henighan, T., Brown, T., Child, R., et al. (2020). *Scaling Laws for Neural Language Models. arXiv:2001.08361*

[8] Hoffmann, J., Borgeaud, S., Mensch, A., et al. (2022). *Training Compute-Optimal Large Language Models.*
arXiv:2203.15556

[9] Unity Technologies, *Unity Documentation*. *https://docs.unity.com/en-us*

[10] Meta AI, (2023) "*Llama 2: Open Foundation and Fine-Tuned Chat Models*". *arXiv:2407.21783*

[11] Meta AI, (2024) "*The Llama 3 Herd of Models*". *arXiv:2407.21783*

[12] Jiedong .L, Zhehao .G, Shuyu .H (2024). *A Comprehensive Study on Quantization Techniques for Large Language Models. arXiv:2411.02530*

[13] Zhuocheng .G, Jiahao .L, et al. (2024). *What Makes Quantization for Large Language Models Hard? An Empirical Study from the Lens of Perturbation. arXiv:2403.06408*

[14] Alexandru .T, Alena .D, João .M, et al. (2025). *Generative AI in Game Development: A Qualitative Research Synthesis. arXiv:2509.11898*

[15] Sandra .J, David .H (2024). *A Primer on Large Lnaguage Models and their Limitations. arXiv:2412.04503*

[16] *FastAPI Documentation* – High Performance Async Python Framework. *fastapi.tiangolo.com*

[17] *Piper Documentation. pypi.org/project/piper-tts/*

[18] Sihyeong .P, Sungryeol .J, et al (2025). *A Survey on Inference Engines for Large Language Models: Perspectives on Optimization and Efficiency. arXiv:2505.01658*

[19] Jiaheng .L. et al (2025). *A Comprehensive Survey on Long Context Language Modeling. arXiv:2503.17407*

[20] Sander .S, Michael .L, et al (2024). *The Prompt Report: A Systematic Survey of Prompt Engineering Techniques. arXiv:2406.06608*

[21] Sufan .L, Aditya .G (2025). *Accelerated Inference of Large Language Models through Input-Time Speculation for Real-time Speech Interaction. arXiv:2506.15556*

[22] *Hugging Face – Meta Llama 3.1 8B instruct GGUF. Huggingface.co/bartowski/Meta-Llama-3.1-8B-Instruct-GGUF*

[23] *Hugging Face – Nous Hermes 2 Mistral 7B. Huggingface.co/NousResearch/Nous-Hermes-2-Mistral-7B-DPO*

[24] *Hugging Face – Orca 2-7B. Huggingface.co/microsoft/Orca-2-7b*

[25] *Hugging Face – Qwen2-1.5B. Huggingface.co/Qwen/Qwen2-1.5B-Instruct*

[26] *Hugging Face – DeepdSeek-R1-Distill-Qwen-14B.* *Huggingface.co/bartowski/DeepSeek-R1-Distill-Qwen-14B-GGUF*