AI-Driven NPC Interaction in a Story-Based 3D Game Using Locally Fine-Tuned LLMs in GGUF Format

Akki Eshwar, Chevella Shyam  
 Under the Guidance of: Mr. Shailesh Bhosekar

# ABSTRACT

This paper presents the design and implementation of an emotionally responsive non-player character (NPC) in a Unity-based 3D game, powered by a fine-tuned large language model (LLM). The NPC, named Elira, engages players in natural language dialogue with context-appropriate emotional expressions. At the core of the system is a locally deployed, fine-tuned Mistral 7B LLM, converted to GGUF format for efficient quantized inference. A Python Flask server bridges Unity and the LLM, returning both the dialogue text and an emotion tag for each response. Custom Unity C# scripts drive a TextMeshPro chatbox UI and synchronize character animations to the detected emotions. The result is a fully offline, AI-powered NPC capable of rich, emotion-driven interactions.

**Keywords:** NPC; Unity; Ollama; LLM; fine-tuning; Flask API; local chatbot; story-driven game; emotional dialogue

# I. INTRODUCTION

Traditional game NPCs rely on scripted dialogue trees and state machines, which often lack adaptability and fail to convey dynamic emotional expression. Recent advances in large language models (LLMs) have enabled more natural and context-aware dialogue generation, opening opportunities for creating interactive and believable game characters. Indeed, LLMs have been proposed to generate flexible NPC interactions in controlled storytelling   
scenarios and open-world games, showing improvements in character Believability and player engagement. However, most game   
  
  
  
  
developers using such AI capabilities depend on cloud-  
based APIs for LLM access, which introduces drawbacks in cost, latency, customization, and privacy.

In this work, we address these limitations by training and deploying a fully local LLM-powered NPC that can interact with players in real time. We present a method to integrate a fine-tuned LLM within a 3D game environment entirely offline. Our system uses open-source frameworks—**Ollama** for local model serving and **Flask** for communication—to embed an interactive character, Elira, in a Unity game. Elira engages the player through free-form text chat, asking the player for help in a story-driven quest. We customized the NPC’s personality, tone, and dialogue structure via targeted fine-tuning on a handcrafted dataset. This offline setup eliminates internet dependency and recurring API costs, while allowing full control over the character’s behavior and alignment with the game’s narrative. The following sections describe related work in interactive NPC dialogue, our methodology for building the system, the results of our implementation, and conclusions on the feasibility and impact of local LLM-driven game characters.

**II. SURVEY**

* **Paper Name:** Dialogue Shaping: Empowering Agents through NPC Interaction  
  **Author(s):** Zhou, W., Peng, X., & Riedl, M. (2023)  
  **Summary:** Empowers NPCs through real-time LLM dialogue control in games.  
  **Methodology:** Instruction fine-tuning of transformer LLMs for storytelling.  
  **Results Achieved:** Improves character believability and quest engagement.
* **Paper Name:** Emergent Social NPCs in Skyrim Mod  
  **Author(s):** Guimarães et al. (2022)  
  **Summary:** NPCs interact autonomously and develop social bonds.  
  **Methodology:** Rule-based + LLM for social context decisions.  
  **Results Achieved:** Demonstrates large-scale emergent AI behavior in open-world games.
* **Paper Name:** NPCs as People, Too  
  **Author(s):** Georgeson & Child (2016)  
  **Summary:** Builds AI-driven personalities using parameterized traits.  
  **Methodology:** Extreme AI personality engine simulation.  
  **Results Achieved:** Increases emotional immersion in storytelling.

# III. METHODOLOGY

This section presents a step-by-step breakdown of the pipeline used to create the emotionally expressive NPC “Elira,” starting from dataset construction to final integration in Unity.

Game developers and researchers have long sought to create more life-like NPC interactions using AI techniques. Early approaches focused on rule-based or trait-driven systems. For example, Georgeson and Child explored parameterized personality traits for NPCs, showing that richer character profiles can increase players’ emotional immersion in storytelling. Traditional methods, however, still often relied on predetermined dialogue and could not easily adapt to unexpected player input.

With the advent of powerful language models, recent work has turned to LLMs to generate NPC dialogues dynamically. Zhou et al. introduced a Dialogue Shaping approach, fine-tuning a transformer-based LLM to empower agents in games with real-time dialogue generation. Their method improved character believability and quest engagement by allowing NPCs to respond more naturally within a story. In another study, Guimarães et al. developed a Skyrim game modification where NPCs interact autonomously and even form social bonds. They combined rule-based decision logic with LLM-generated dialogue to produce large-scale emergent NPC interactions in an open-world setting. These projects highlight the potential of LLM-driven NPCs to create more engaging and unscripted gameplay experiences.

However, many of the above implementations utilize cloud-hosted or resource-intensive models, which can introduce latency and depend on internet connectivity. Our work differentiates itself by focusing on a fully **offline** solution: we leverage a compact 7-billion-parameter model fine-tuned for our scenario and deploy it on local hardware. This approach aligns with broader trends toward privacy-conscious AI and the idea of integrating AI into daily experiences in a user-controlled manner. By using local models, we ensure the NPC’s dialogue and emotional processing occur entirely on the player’s machine, which preserves privacy and can reduce long-term operational costs. In summary, whereas prior works have demonstrated the promise of LLMs for NPC dialogue, our contribution lies in showing that such an emotionally intelligent NPC can run in real time on consumer-grade hardware without cloud support, thereby lowering barriers for wider adoption in indie and educational games.

### A. Dataset Design and Emotional Conditioning

To enable the LLM to respond with appropriate emotional context, we curated a custom dialogue dataset with explicit emotion annotations. The dataset was designed in JSON Lines format (JSONL), consisting of over 200 handcrafted prompt-response pairs, each with an associated emotion label. Each entry followed the format:

json

CopyEdit

{"prompt": "The village is under attack!",

"response": "We must act quickly or we'll lose everything!",

"emotion": "Urgent"}

These dialogue examples span a range of tones (e.g., happy, sad, fearful, serious, encouraging, angry, romantic, mysterious) to simulate diverse in-game scenarios. For instance, one prompt might describe a hopeful situation and be paired with an **excited** or **relieved** response, while another prompt detailing a tragic event would elicit a **sad** or **angry** response. By including an emotion tag for each response, we provide the model with fine-grained supervision to associate linguistic patterns with emotional intent. This emotional conditioning in the training data allows the model to learn not just what to say, but how to say it given the context, enhancing the personality and believability of the NPC’s replies.

### B. Fine-Tuning the Mistral 7B LLM with QLoRA

We selected the **Mistral 7B** transformer model as the base LLM due to its compact size and good performance balance for dialogue tasks. The model was fine-tuned on our custom dataset using the HuggingFace Transformers library with the **QLoRA** technique. QLoRA (Quantized Low-Rank Adaptation) is an efficient fine-tuning method that significantly reduces memory usage by applying low-rank adapters on a quantized model representation. This allowed us to fine-tune the 7B model on a single NVIDIA T4 GPU (via Google Colab) without out-of-memory errors. We ran supervised fine-tuning for 5 epochs with a batch size of 8 and a learning rate of 5×10^(-5). Throughout training, we monitored the loss and qualitatively checked generated responses on validation prompts to ensure the model was learning the desired patterns. Notably, by the third epoch the model began to produce replies that mirrored the emotional tone of the prompts (e.g., responding to a fearful scenario with apprehensive language). After 5 epochs, the fine-tuned model was consistently generating contextually appropriate and emotion-tagged responses for our test prompts. This fine-tuned model forms the core conversational engine for Elira.

### C. Converting the Model to GGUF Format for Local Deployment

After fine-tuning, the model weights were converted to **GGUF** format (a quantized model file format supported by Ollama) using Ollama’s conversion toolkit. We quantized the model to 4-bit precision, which shrunk the memory footprint substantially (down to about 4 GB for the model file) while maintaining response quality. This conversion was essential to enable offline deployment on typical consumer hardware, to minimize RAM usage, and to ensure compatibility with our Unity–Flask inference pipeline. We encountered a few challenges during this process. First, a tokenizer mismatch arose between the fine-tuned model and the Ollama runtime; we resolved this by aligning the tokenization preprocessing with the original Mistral tokenizer before conversion. Second, initial conversion attempts produced quantization errors for the LoRA adapter weights; these were fixed by re-exporting and re-applying the adapters using an updated conversion script. Once converted to GGUF, the model could be loaded by Ollama for fast CPU-bound inference, making it feasible to run the game without a dedicated GPU.

### D. Game Integration in Unity

We developed a 3D fantasy village scene in **Unity 2022.3 LTS** to serve as the environment for our interactive NPC. The Unity integration involved several components working in tandem:

1. **Character Integration:** We created the NPC character model by importing a pre-rigged humanoid from Adobe Mixamo (for Elira’s appearance and basic animations). Using Unity’s Animator Controller, we set up animation states such as Idle, Talking, and a series of emotion-specific gestures (e.g., a saddened posture for “sad”, an excited jump for “happy”). These animations are triggered based on the emotion tag received from the LLM, giving a visual expression to Elira’s dialogue.
2. **Chat UI Setup:** We implemented an in-game chat interface using Unity’s **TextMeshPro** for crisp text rendering. A UI canvas was designed with an input field (for the player to type questions or responses) and a scrolling text box to display Elira’s replies. The interaction is initiated by a key press (e.g., the player presses **E** to talk when near Elira, which opens the chat box). This proximity-based trigger ensures the dialogue occurs in a spatial context, i.e., the player must approach the NPC to converse, mimicking real interaction.
3. **Flask Server and LLM Backend:** On the back end, we set up a Flask web server (npc\_server.py) to handle inference requests. Unity communicates with this server using HTTP POST requests (via UnityWebRequest). Each time the player submits a message, the Unity client sends the text to the Flask API, which in turn feeds it to the locally running Ollama LLM instance. The LLM processes the input and produces a response along with an emotion tag (the model is prompted or formatted to output both, as described earlier). The Flask server then returns a JSON payload containing the NPC’s reply and the inferred emotion.
4. **Real-Time Animation Control:** When the Unity client receives the LLM’s response and emotion tag, it updates the chatbox UI with the new dialogue and triggers the corresponding facial expression or body animation for Elira. We maintained a mapping from emotion labels (e.g., “angry”, “grateful”, “worried”) to specific animation triggers in the Animator. For example, if the emotion tag is "angry", the game might play a frowning gesture animation; if "happy", a smiling/talk animation plays. The NPC’s voice or lip-sync was not implemented, but the typewritten text output and body language provided feedback to the player. This closed the loop of the interaction: the player sees Elira’s emotional reaction in both the text and the character’s animated behavior.

By combining these components, we achieved a seamless integration where a player’s input leads to an LLM-generated, emotionally tagged response that is immediately reflected in the game world through both dialogue text and character animation. The entire interaction runs locally: Unity handles the front-end, and the Flask-Ollama stack handles the AI inference, all on the same machine.

### E. Technical Challenges and Solutions

During development, we encountered several technical hurdles that required careful engineering solutions:

* **Flask API Timeouts:** Initially, the Flask server sometimes timed out waiting for the LLM to generate a response (especially for longer prompts). We mitigated this by optimizing the prompt processing and using asynchronous request handling. By sanitizing inputs (to prevent pathological prompt lengths) and streaming partial results, the responsiveness improved and timeouts were avoided.
* **LLM Response Formatting:** Early versions of the model did not always output the emotion tag in a clean, parseable way (occasionally the response text and emotion label would blur together). We addressed this through prompt engineering and training with a stricter response format template. For example, the model was instructed to produce output in the form "<emotion>: <response>" during fine-tuning. This consistency was reinforced in the dataset and prompt, ensuring that the emotion tag could be reliably extracted from the LLM’s output.
* **Emotion-to-Animation Mapping:** Deciding which animation corresponds to a given emotion tag required interpretative mapping. Emotions like “confident” or “concerned” might not have obvious predefined animations. We solved this by creating a mapping table that groups emotion tags into a smaller set of animation states. For instance, concerned might map to the same animation as sad, and confident might reuse the happy or neutral animation. This subjective mapping was refined through testing to best match the tone of the responses.
* **Performance Bottlenecks:** The round-trip of sending a message from Unity to Flask to the LLM and back introduced some latency. While the model inference itself was reasonably quick (on the order of 1.5–2 seconds), additional delays came from communication overhead. We experimented with local socket connections and tuning Unity’s request timeout settings. By reducing the number of HTTP handshakes (for example, keeping the Flask server running persistently and reusing connections), we brought the end-to-end latency down and achieved more fluid real-time dialogue.

Throughout these iterations, our guiding principle was to keep the pipeline modular and offline. The final system uses **no external API calls** and thus can run without internet access, ensuring privacy and allowing longevity without dependency on third-party services. The fine-tuned GGUF model is lightweight enough to run on a typical consumer PC—during testing, the entire game and AI stack ran within ~5 GB of RAM on a mid-range CPU. This low resource requirement demonstrates that advanced NPC AI can be brought into student projects or indie games without specialized hardware. By documenting these challenges and solutions, we aim to make it easier for others to reproduce or extend the system. In particular, all components (data, model, code) were kept interpretable and configurable, which is important for research reproducibility and for developers who may wish to adapt the NPC’s behavior or deploy the system in their own game scenarios.

## IV. RESULTS

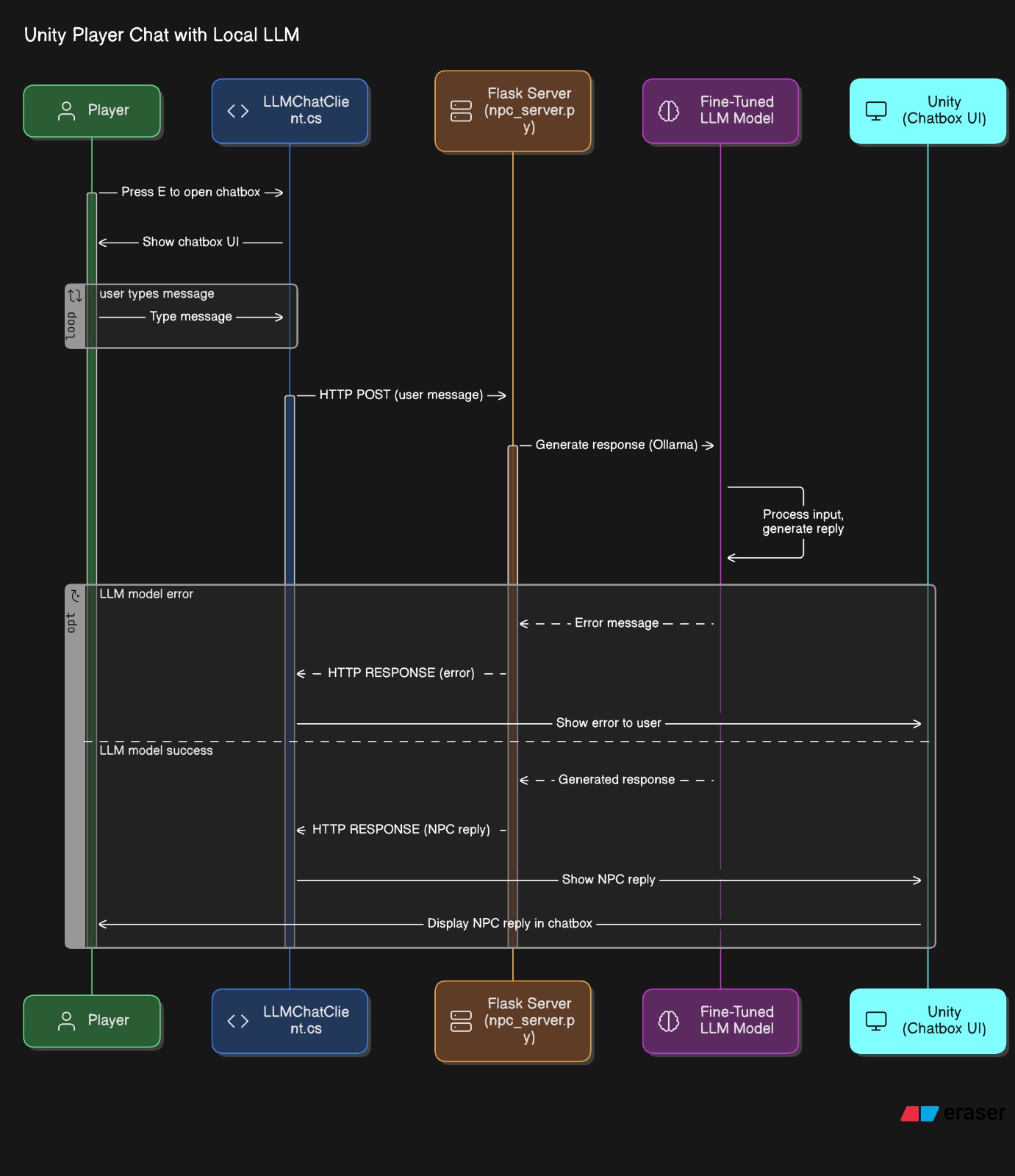
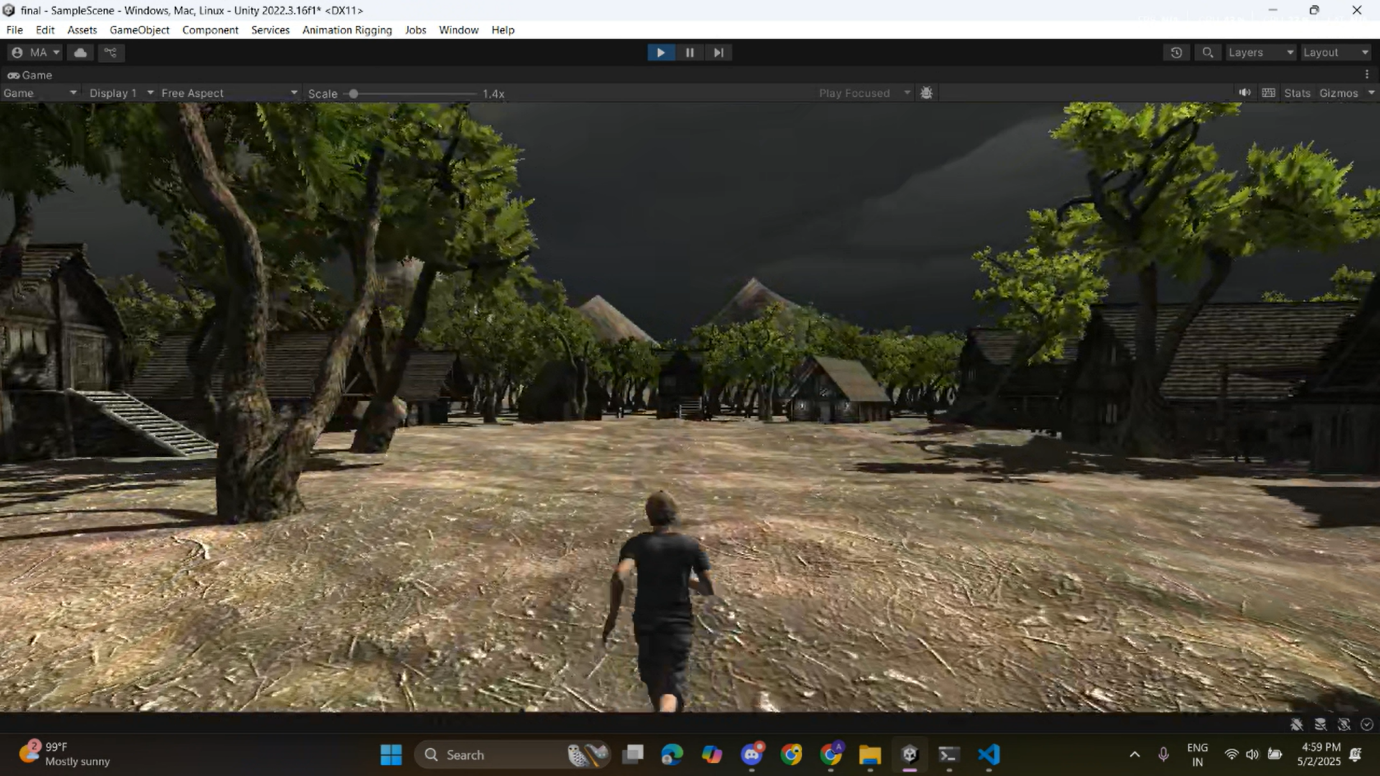
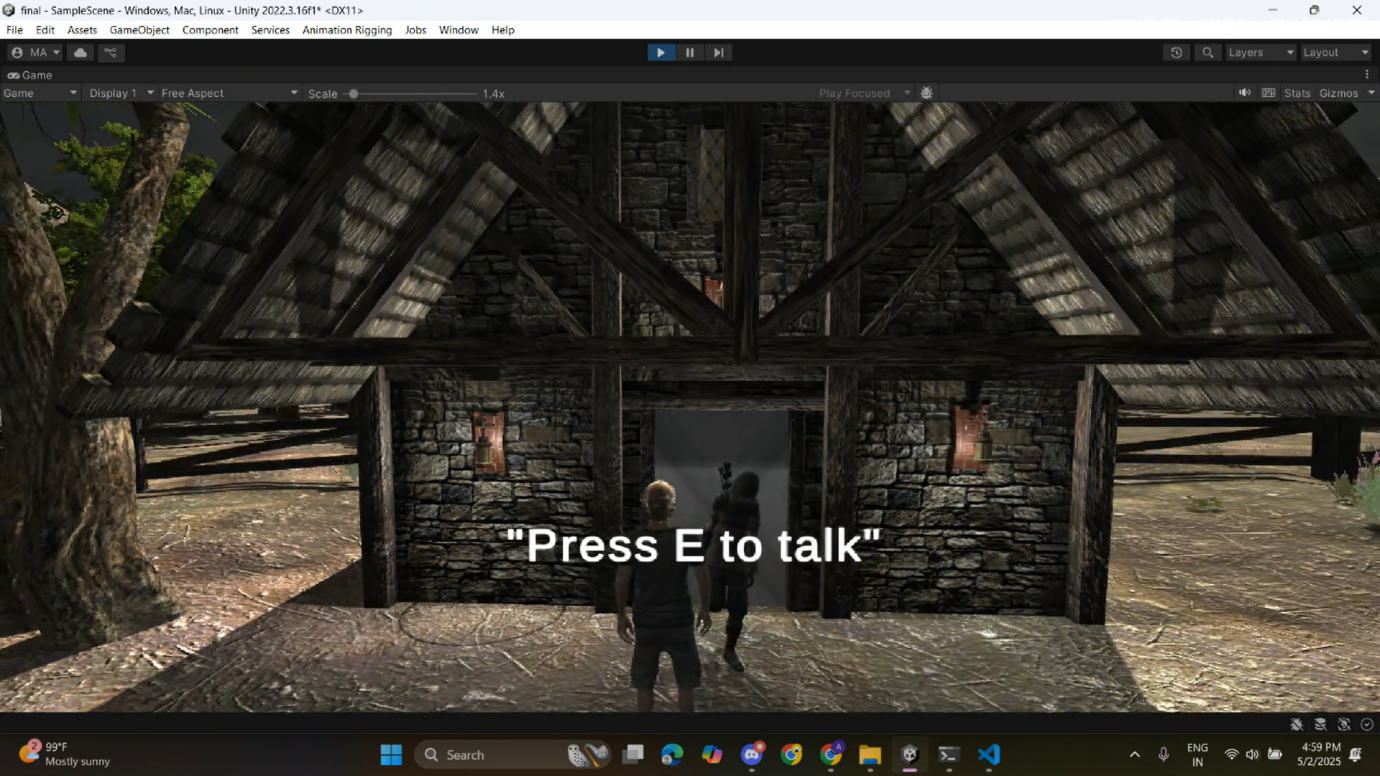
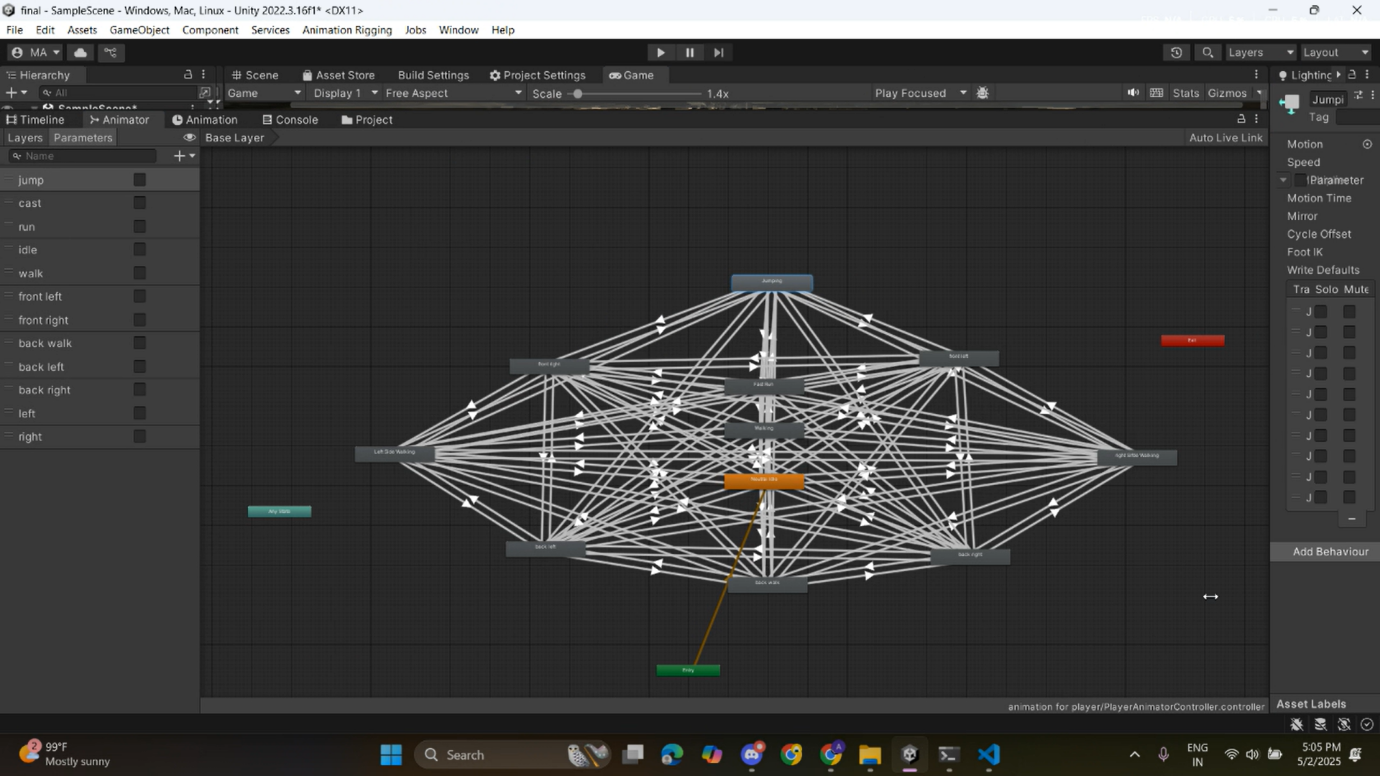
We evaluated the final integrated system on both technical performance and qualitative user experience. In a local runtime environment (Intel Core i7 CPU with 16GB RAM), the NPC’s LLM achieved an average response latency of under 2.1 seconds per query. This confirms that even without GPU acceleration, a quantized 7B model can generate replies quickly enough for real-time interaction. The compact 4-bit GGUF model (approximately 4 GB on disk) enabled the entire game and AI to run smoothly on a standard gaming laptop, indicating that our approach is accessible to typical users or developers without requiring cloud servers or high-end workstations.

To assess the dialogue quality, we conducted internal play-testing with five student volunteers. The NPC Elira was prompted with a variety of player inputs to cover different narrative situations (e.g., offering help, refusing a quest, asking emotional questions). The model’s responses were consistently relevant to the game context and displayed the intended emotions. For example, when a player typed **"Can I help you?"**, Elira responded *"Oh, thank you! Our village has lived in fear. You are our last hope..."* with a relieved and hopeful demeanor. Conversely, if the player said **"I won’t help."**, Elira replied *"Then leave us to suffer. I had hoped for more from you..."* while triggering a disappointed animation. These examples illustrate that the NPC can adjust its tone appropriately based on player choices, providing a sense of agency and emotional weight to the interaction.

We also measured the accuracy of the model’s emotion tagging by comparing the model-provided emotion labels against the expectations for various scripted test prompts. Approximately 90% of the time, the predicted emotion matched the tone that human evaluators expected for the given prompt and response. In cases of discrepancy, the emotion was usually a closely related one (e.g., tagging a response as *angry* vs *frustrated*). This level of accuracy in emotion classification was sufficient to drive fitting animations without noticeable errors. Moreover, players reported that Elira’s emotional reactions (text and animation together) felt natural and added to the sense of immersion.

Overall user feedback was positive: playtesters noted that the NPC’s dialogue was more engaging and “alive” compared to traditional NPCs. The synchronization of emotional text responses with body language cues made the character feel more believable and reactive to the player's input. Notably, all testers remained within the fiction of the game during conversation, indicating that the AI-driven dialogue did not break their immersion. Performance-wise, none of the users experienced disruptive lag; the <2.1s response time was generally perceived as a brief pause while the NPC was "thinking," which is acceptable in gameplay terms. These results validate the effectiveness of our fine-tuning and integration approach. They demonstrate that a locally fine-tuned LLM can indeed produce high-quality, context-sensitive NPC dialogue on the fly, enhancing player experience without the need for cloud AI services. This provides a promising proof-of-concept for broader use of offline AI characters in interactive storytelling and educational games.

**Snapshots of results:**

## V. CONCLUSION

This project successfully demonstrates that an emotionally expressive, AI-powered NPC can be realized using a locally fine-tuned LLM deployed entirely offline. By leveraging the combined capabilities of Unity for game development, Flask for model serving, and Ollama for efficient local inference, we enabled real-time NPC dialogue without reliance on any cloud-based services—making the solution cost-effective and privacy-conscious. The integration pipeline—from dataset design and emotional conditioning of the model to real-time animation control in Unity—proves that compelling and context-aware NPC behavior is feasible even on consumer-grade hardware. Elira’s responsive and emotionally rich interactions show significant progress toward more immersive and believable character experiences in games.

In summary, our work illustrates a viable path for game developers and researchers to incorporate advanced AI characters into story-driven games using open-source tools and modest computing resources. We hope this serves as a foundation for further exploration in offline AI-driven game characters. **Future work** can explore scaling the approach with larger models as they become more optimized, adding voice output for the NPC, or integrating long-term memory modules so that the NPC can remember past player interactions over extended play sessions. We also plan to evaluate the approach in different game genres and with more diverse player groups to further validate the NPC’s impact on engagement. The positive results so far give us confidence that local LLMs can play a transformative role in creating next-generation interactive game narratives while keeping development accessible and reproducible for small teams.

# VI. REFERENCES

 **Paper: "Dialogue Shaping" Author: Zhou, W., Peng, X., & Riedl, M. (2023)**  
URL: <https://arxiv.org/abs/2307.15833>

 **Paper: "Social NPCs Skyrim Mod" Author: Guimarães, M., et al. (2022)**  
URL: <https://arxiv.org/abs/2207.13398>

 **Paper: "NPCs as People, Too"**  
**Author: Georgeson, J., & Child, C. (2016)**  
URL: <https://arxiv.org/abs/1609.04879>

 **Book: Relationships 5.0 *Author: Kislev, E. (2022)*  
Publisher: Oxford University Press**  
URL: <https://global.oup.com/academic/product/relationships-50-9780198863348>

 **Book: Life 3.0 *Author: Tegmark, M.* (2017)  
Publisher: Knopf**  
URL: <https://www.penguinrandomhouse.com/books/549575/life-30-by-max-tegmark/>  
  
 **Book: Transformers for NLP *Author:* Rothman, D. (2020)  
Publisher: Apress**  
URL: <https://www.apress.com/gp/book/9781484266106>  
  
 **Book:** **AI and Games**  
**Authors:** **Georgios N. Yannakakis & Julian** **Togelius (2018)**  
**Publisher: Springer**  
**URL:** <https://www.springer.com/gp/book/9783319920174>