

# 대화형 가상 아바타 개발을 위한 유니티 기반 LLM 개발 환경 구축

이지원<sup>0,1</sup>

김진모<sup>1,2,\*</sup>

한성대학교 일반대학원 정보컴퓨터공학과<sup>1</sup>, 한성대학교 컴퓨터공학부<sup>2</sup>

{mpm11, jinmo.kim}@hansung.ac.kr

## Building an LLM Development Environment in Unity for Developing Conversational Virtual Avatars

Jiwon Lee<sup>0,1</sup>

Jinmo Kim<sup>1,2,\*</sup>

Department of Information and Computer Engineering, Graduate School, Hansung University<sup>1</sup>,

Division of Computer Engineering, Hansung University<sup>2</sup>

### 요약

자연어를 기반으로 하는 사용자의 언어를 이해하고, 사람처럼 응답하는 대화 기능을 제공하는 대규모 언어 모델 (Large Language Model, LLM)을 사람의 외형을 갖춘 가상 아바타에 탑재하여 실제 사람처럼 상호작용할 수 있는 지능형 가상 에이전트(Intelligent Virtual Agent, IVA) 연구에 관한 관심이 늘어나고 있다. 본 연구는 지능형 가상 에이전트를 3차원 대화형 가상 아바타로 구현하기 위하여 유니티 기반의 LLM을 활용한 통합 개발 환경 구축 방법을 분석한다. 본 연구의 핵심 목표는 오픈 소스로 공개된 LLM을 사용하는 기계학습 개발 도구인 Inference Engine과 OpenAI의 모델을 사용하기 위해 제공되는 OpenAI API를 활용하는 방법을 비교 분석하는 것이다. 따라서, Inference Engine과 OpenAI API를 활용한 유니티 기반 LLM 개발 환경 구축 과정을 정리하고, 대화형 가상 아바타를 구현하는 과정을 통해 두 개발 환경의 특징과 차이를 분석하고자 한다. Phi-3.5 모델과 GPT-4o 모델을 각각 사용하여 대화형 가상 아바타와 사용자 인터페이스로 구성된 실험 환경에서 사용자의 질문에 응답하는 대화 콘텐츠를 실행한 결과를 토대로, Inference Engine과 OpenAI API의 개발 과정, 응답 생성 시간, 기능 등의 특징을 비교하여 다양한 대화형 가상 아바타 개발에 활용할 수 있는 방향성을 제시한다.

### Abstract

There is a growing interest in research on Intelligent Virtual Agent (IVA) that can interact like a real person by loading Large Language Model (LLM) that understand the user's language based on natural language and provide human-like conversational functions onto a virtual avatar with a human appearance. This study analyzes a method of building an integrated development environment using Unity-based LLM to implement IVA as three-dimensional interactive virtual avatars. The primary objective of this study is to compare and analyze the methods of utilizing the Inference Engine, a machine learning development tool that uses LLMs released as open-source, and the OpenAI API, which provides access to OpenAI's models. Therefore, we aim to organize the process of building a Unity-based LLM development environment using the Inference Engine and the OpenAI API, then analyze the characteristics and differences between the two development environments through the process of creating a conversational virtual avatar. Based on the results of executing conversational content that responds to users' questions in an experimental environment consisting of a conversational virtual avatar and a user interface (UI) using the Phi-3.5 model and the GPT-4o model, we compare the development process, response generation time, and functions of the Inference Engine and the OpenAI API to suggest directions that can be utilized in the development of various interactive virtual avatars.

**키워드:** 유니티, 대규모 언어 모델, 대화형 가상 아바타, Inference Engine, OpenAI

**Keywords:** Unity Engine, Large Language Model, Conversational Virtual Avatar, Inference Engine, OpenAI

\*corresponding author: Jinmo Kim/ Hansung University (jinmo.kim@hansung.ac.kr)

Received : 2025.06.13./ Review completed : 1st 2025.06.30. 2nd 2025.07.10./ Accepted : 2025.07.14.

DOI : 10.15701/kcgs.2025.31.3.139

ISSN : 1975-7883(Print)/2383-529X(Online)

# 1. 서론

컴퓨터 그래픽스 분야에서 인공지능(Artificial Intelligence, AI)은 렌더링, 애니메이션 등 다양한 영역에서 꾸준히 연구되어 왔으며 딥러닝(Deep Learning)의 발전과 함께 새로운 시각적 경험과 향상된 처리 결과를 제공하고 있다. 또한, 거대한 양의 자연어 데이터를 학습한 대규모 언어 모델(Large Language Model, LLM)은 자연어 처리(Natural Language Processing, NLP) 분야에서 향상된 성능을 보이고 있으며 이는 컴퓨터 그래픽스 응용 분야에서도 폭넓게 활용되고 있다. LLM은 사용자의 언어를 이해하고 사람처럼 응답하는 특징을 통해 인간과 컴퓨터 사이의 자연스러운 상호작용을 가능하게 하여 챗봇(Chatbot)이라는 인터페이스 형태로 다양한 분야에 활용되고 있다. 대표적인 예로, OpenAI의 GPT(Generative Pre-trained Transformer) 모델을 기반으로 한 챗봇 시스템인 챗지피티(ChatGPT)[1]를 사용하여 자연스러운 대화를 통해 질문에 대한 답변을 받거나 음성 및 이미지를 주고받을 수 있는 멀티 모달(Multimodal) 기능을 사용할 수 있다. 또 다른 챗봇으로는 구글의 Gemini[2]와 Anthropic의 Claude[3], Microsoft의 Copilot[4]이 있다. 이처럼 대화형 사용자 인터페이스(User Interface, UI)를 통해 텍스트 또는 음성으로 수집된 사용자의 입력을 분석하고 맥락을 유지하여 답변을 제공하는 챗봇은 고객 서비스 제공과 같은 산업 분야를 포함한 교육, 의료 등 특정 분야에 국한되지 않고 다양한 분야에 응용되고 있다[5].

인간-컴퓨터 상호작용(Human-Computer Interaction, HCI)을 포함한 연구 분야에서는 LLM의 대화 능력을 활용하여 모니터와 같은 화면으로 소통하는 챗봇을 가상 아바타에 탑재하여 디지털 환경에서 실제화된 가상 에이전트(Virtual Agent, VA) 개발에 관한 연구와 LLM 기반의 가상 아바타가 상황을 인지하면서 동시에 사용자와의 대화 맥락에 맞는 감정 및 동작을 표현하는 복합적인 기술을 통합하여 실제 사람처럼 상호작용이 가능한 대화형 가상 아바타와의 상호작용에 관한 연구들로 관심이 늘어나고 있다[6]. 이는 가상 현실(Virtual Reality, VR) 또는 증강현실(Augmented Reality, AR)과 같이 높은 몰입감을 제공하는 확장현실(eXtended Reality, XR) 환경에서 더욱 사실적으로 표현되어 존재감을 가진 지능형 가상 에이전트(Intelligent Virtual Agent, IVA)에 관한 연구로 활용되고 있다. 그리하여 LLM을 이용하여 사람처럼 대화하고 행동할 수 있는 가상 아바타를 물리적으로 위협을 가할 수 있는 “가상 악당”으로 활용하여 참가자의 불안 및 스트레스와 같은 심리적인 요소를 분석하는 연구[7]와 GPT 기반의 가상 환자를 이용하여 교통사고 현장에서의 의료 응급 대응자 교육을 위한 혼합현실(Mixed Reality, MR) 시뮬레이션 개발 연구[8] 등으로 활용되고 있다.

지능형 가상 에이전트는 다양한 분야로 응용하고 확장할

가능성이 큰 연구 주제이지만 개발 과정에서 AI 전문 지식과 가상 아바타 생성을 위한 기술과 컴퓨터 그래픽스 지식이 요구된다. 그리고 가상 아바타에 탑재되어 두뇌 역할을 하는 LLM은 사용하는 언어 모델에 따라 사용자와 소통하는 능력에 차이점이 발생할 수 있어 NLP 성능이 뛰어난 검증된 SOTA(State-of-the-Art) 모델 사용과 지속해서 개발되고 있는 다양한 최신 모델을 검토할 필요가 있다. 또한, 가상 아바타의 비언어적인 상호작용 요소를 구현하기 위한 표정 및 자연스러운 동작 애니메이션 등 사실적인 그래픽 요소를 갖추어야 한다. 더 나아가 사용자의 행동 및 시선과 같은 비언어적인 상호작용 요소를 인지하고 반응하기 위해 컨트롤러를 사용하거나 손을 추적할 수 있는 센서를 사용하여 사용자의 행동과 제스처를 인식하고, 시선 및 머리의 방향을 추적할 수 있는 HMD(Head Mounted Display)와 같은 몰입형 장비를 통해 사용자가 주의하고 있는 환경과 객체를 인식하도록 요구된다. 결론적으로 사실적이고 입체적인 지능형 가상 에이전트 개발을 위해 몰입형 플랫폼 사용을 고려한 LLM 기반의 대화형 가상 아바타를 구현할 수 있는 통합 개발 환경 구축이 선행되어야 한다.

본 연구는 메타버스와 같은 몰입형 가상 환경에서 언어적·비언어적 상호작용이 가능한 AI 가상 아바타 개발을 위한 통합 개발 환경 구축을 위해 먼저 대화형 가상 아바타를 구현하는 유니티 기반의 LLM 개발 환경을 구축한다. 물리 기반 애니메이션과 실시간 렌더링을 지원하고 게임 개발 및 몰입형 플랫폼 개발을 위한 XR 개발 도구를 제공하는 유니티 엔진(Unity)[9]은 기계학습(Machine Learning) 활용을 위한 라이브러리를 제공하는 Inference Engine[10]을 통해 로컬에서 실행할 수 있는 경량화된 LLM과 연동할 수 있으며, 오픈 소스 .NET 플랫폼[11]을 사용하여 OpenAI API[12], Google Gemini API[13]와 같은 클라우드 기반 LLM을 연동할 수 있다. 따라서 본 연구는 Inference Engine과 OpenAI API를 활용하여 유니티 기반의 LLM 개발 환경을 구축하고, 각 환경에서 LLM을 탑재한 대화형 가상 아바타를 구현하여 개발 과정과 특징을 비교 분석하는 것을 목표로 한다.

# 2. 관련 연구

모니터와 같은 평면적인 화면 속에서 2차원의 인터페이스를 통해 소통하는 챗봇을 인간 또는 인간처럼 묘사되는 캐릭터와 같은 아바타를 제작하여 외형을 갖춘 대화형 가상 아바타로 개발하는 연구는 자연스러운 표정[14]과 움직임[15]에 관한 연구를 비롯하여, 가상 아바타에 탑재한 챗봇을 활용한 대화 및 상호작용이 사용자에게 미치는 영향을 분석하고 효과적인 상호작용 요소와 방법을 탐구하는 방향으로 연구가 진행되고 있다[16]. Khampuong et al.[17]은 챗봇과 같은 대화형 에이전트를 위한 인공지능 알고리즘과 AI 아바타,

상호작용 요소를 종합한 AI 아바타 개념 프레임워크를 개발하였다. 이를 통해 LLM과 같은 AI를 활용한 대화형 가상 아바타는 언어적 그리고 비언어적, 외형적 특징, 기계학습으로 구성되며, 사용자와의 대화를 위한 애플리케이션 프로그래밍 인터페이스(Application Programming Interface, API)와 데이터베이스, NLP, 콘텐츠 관리 시스템(Content Management System, CMS), 관리 API로 구성되는 프레임워크 구조 체계를 제안하였다. 최근에는 텍스트뿐만 아니라 이미지와 소리 등 다양한 데이터 형식을 입력받고, 출력으로 생성할 수 있는 멀티 모달 기능을 갖춘 LLM의 발전에 따라 텍스트, 소리, 시각 등 다양한 조합의 사용자 입력에 따른 멀티 모달 공감 반응을 생성하는 가상 아바타 기반의 멀티 모달 공감 챗봇을 개발하는 연구[18]가 진행되었다.

현재 챗봇은 전자상거래와 관련한 기업 및 브랜드에서 고객 서비스를 제공하기 위해 인간 상담원을 대신하여 적극적으로 활용하고 있으며, 이러한 추세에 따라 마케팅 및 디지털 커뮤니케이션을 위한 챗봇 연구들이 진행되고 있다[19, 20]. Fota et al.[21]은 기업에서 고객 서비스를 위해 챗봇을 이용할 때 의인화된 아바타의 외형이 반응이나 보상보다 소비자의 재구매 의사에 더 강한 영향을 미친다는 결과를 확인한 연구를 진행하였다. 이는 사람들이 소통할 때 사회적 존재감을 느끼는 데 익숙해져 있어서 인간적이고 생동감 있게 보이는 의인화된 요소를 선호하며, 마찬가지로 디지털 상대방을 통해서도 같은 공간을 공유하며 연결감을 느낄 수 있어 성공적인 고객 서비스를 제공하는 챗봇을 만들기 위해서 인간과 같은 행동을 내장하는 것이 필요하다는 분석 결과를 확인하였다. 또한, 고객 서비스를 위한 챗봇 기반의 대화형 가상 아바타에 관한 연구 외에도 AI 챗봇 아바타 표현이 학생들의 학습 경험에 미치는 영향을 탐구한 연구[22]와 작가가 창작하는 스토리의 맥락, 캐릭터 외형, 음성, 배경 이미지를 활용하여 챗봇 아바타와의 대화를 통해 스토리 속 캐릭터를 점진적으로 구현할 수 있도록 하는 프로토타입 시스템을 개발하는 연구[23] 등 다양한 분야 및 주제에 활용되어 연구가 진행되고 있다.

외형이 주어진 챗봇인 대화형 가상 아바타에 관한 연구는 가상현실과 같은 몰입형 플랫폼과 장비를 이용하여, 가상의 공간 또는 확장된 현실의 물리적 공간을 공유하여 존재감을 제공할 수 있는 사실적인 대화형 가상 아바타 제작에 관한 연구를 포함한 사용자의 현존감과 아바타의 존재감을 이용한 상호작용에 관한 연구로 진행되고 있다[7]. Yamazaki et al.[24]은 LLM 기반의 오픈 도메인 가상 아바타 챗봇 개발에 관한 연구를 통해 WebXR을 활용한 웹 기반의 가상현실 환경에서 다양한 형태의 출력과 생성 속도 문제 해결을 위한 여러 외부 모듈을 통합한 시스템을 제안하였다. Guimarães et al.[25]은 가상현실에서 대화형 가상 아바타와 상호작용하는 방법이 기존의 비몰입형 플랫폼에서 같은 상호작용을 하는 것보다 현존감과 신뢰성 측면에서 더 큰 영

향을 미친다는 가설을 검증하는 연구를 수행하였다. 이를 위해 검증된 설문 도구를 이용하여 정성적 및 정량적 데이터를 수집한 결과 가상현실에서 대화형 가상 아바타의 사회적 현존감에 대한 인식이 더 높다는 분석 결과를 확인하였다. 그리하여 실제 환경과 비슷한 경험을 체험할 수 있다는 몰입형 환경의 특징과 함께 LLM과 같은 AI가 탑재된 가상 아바타와의 현실적인 상호작용을 제공하는 효과적인 교육 및 훈련 시뮬레이션에 관한 연구가 진행되고 있다[8, 26].

### 3. 유니티 기반 LLM 개발 환경 구축

유니티는 게임 개발 및 물리 시뮬레이션 등을 위한 기계학습 개발 환경을 제공하기 위해 2020년 초 공개된 ML-Agents[27]를 포함하여 2023년 Sentis[28]로 출시되어 현재 Inference Engine으로 명칭이 변경된 오픈 소스 패키지 형태의 기계학습 개발 도구를 제공하고 있다. ML-Agents는 Python API를 통해 다양한 기계학습 알고리즘과 모델을 사용하여 캐릭터 행동을 훈련하고, 그 결과로 생성된 신경망 행동 모델을 유니티 엔진에서 임베딩하여 캐릭터를 제어하는 데 적용할 수 있다[29]. 마찬가지로 유니티의 기계학습 개발 도구인 Inference Engine은 신경망 추론 및 계산 라이브러리를 제공하며, 유니티 런타임에서 사전 학습된 기계학습 모델을 직접 실행할 수 있다. 이 기능은 언어 모델이 사용자의 입력을 분석하고 답변을 생성하며 상호 대화가 가능한 실행 환경을 구현하는 데 핵심적이다. 또 다른 방법으로는, OpenAI가 제공하는 공식 SDK(Software Development Kit)를 이용한 라이브러리를 사용하여 유니티에서 OpenAI API에 접근할 수 있다[30]. 따라서 본 논문에서는 이 두 가지 방식을 각각 이용하여 대화형 가상 아바타를 구현하기 위한 유니티 기반 LLM 개발 환경을 구축한다.

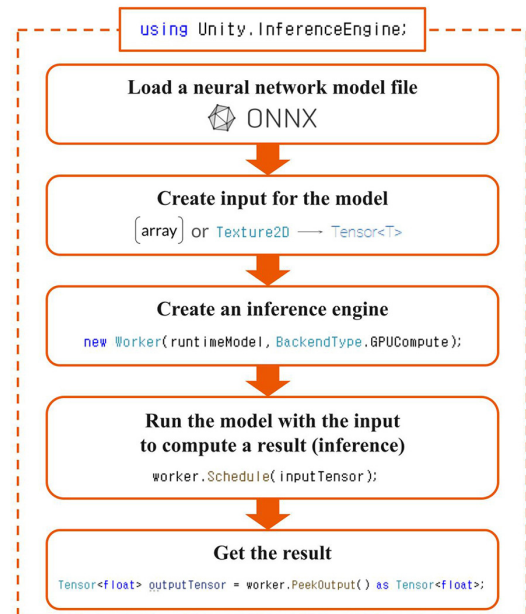


Figure 1. Inference Engine workflow.

### 3.1 Inference Engine을 활용한 개발 환경

Inference Engine은 C# 스크립트에서 Unity.InferenceEngine 네임스페이스를 활용해 신경망 모델을 실행하는 과정을 따른다. 이 과정은 모델을 불러오고 모델의 입력과 워커(Worker)로 정의된 추론 엔진 생성, 워커를 통해 입력을 사용한 모델 실행, 그리고 추론 결과 계산 단계로 이루어진다. Inference Engine에서 불러올 수 있는 신경망 모델은 기계학습 모델의 계산 그래프와 가중치를 표준화된 방식으로 표현한 개방형 표준 형식인 ONNX(Open Neural Network Exchange)[31] 형식으로 변환된 사전 학습 모델을 이용한다. 사용하려는 모델의 ONNX 파일은 유니티 프로젝트 에셋 폴더에 저장한 뒤 스크립트를 통해 참조하여 불러온다. 모델 실행 시 필요한 입력은 Tensor API를 이용해 모델 데이터를 담은 텐서로 생성하며, 배열이나 텍스트를 텐서로 변환할 수 있다. 또한, 학습된 모델을 실행하여 새로운 데이터나 결과를 예측하는 추론 엔진인 워커를 생성하고, 불러온 모델을 실행할 작업으로 분할한다. 이때 중앙 처리 장치(Central Processing Unit, CPU) 또는 그래픽 처리 장치(Graphic Processor Unit, GPU)중 하나를 선택하여 모델 실행 메모리 환경을 설정한 뒤, Schedule() 함수를 통해 모델을 실행한다. 이후 PeekOutput() 함수 등을 활용해 모델 실행의 추론 결과를 가져온다. Figure 1은 Inference Engine의 작업 과정을 보여주며, Figure 2는 Inference Engine의 기본적인 작업 과정을 구현하는 예제 코드를 제시한다.

본 논문에서는 위와 같은 과정으로 LLM을 GPU 메모리 환경에서 실행하여 사용자의 입력 프롬프트를 분석하고, 생성된 답변 프롬프트를 가상 아바타가 존재하는 화면에 표시하도록 장면을 설계한다. 이를 위해, Unity 6[32] 버전 이상의 유니티 환경에서만 지원되는 Inference Engine 패키지의 조건[33]에 맞춰, Unity 6.0(6000.0.47f1) 엔진에 패키지 매니저(Package Manager)를 통해 Inference Engine 2.2.1를 설치하여 LLM 개발 환경을 구축한다. LLM으로는 Microsoft의 Phi-3.5-mini-instruct[34] 모델을 ONNX 형식으로 변환하고, 디스크 및 메모리 저장 공간을 고려하여 Unit8 형식으로 선행 양자화하여 경량화한 모델[35]을 채택한다. 또한, LLM과 같은 자연어 처리 모델의 사용자 입력 텍스트 인코딩 및 모델 실행 결과 디코딩을 위해 LlamaTokenizer[36]를 활용한다. 사용자 입력 텍스트를 받기 위한 UI로는 InputField 객체를, 그리고 모델 실행 결과 답변을 표시하기 위한 UI로는 Canvas 객체를 이용한다. 마지막으로, Citizens Pro 2024[37] 에셋을 활용하여 대화형 가상 아바타의 외형 모델로 사용한다. Figure 3은 Inference Engine을 통해 LLM 개발 환경을 구축한 유니티 프로젝트 화면을 나타낸다.

```

using Unity.InferenceEngine;
using UnityEngine;

public class ChatBot : MonoBehaviour
{
    public bool checkRunning = false;
    private string systemPrompt = "You are a helpful assistant in a human-like avatar.~n~Reply user's question friendly.~n~Your reply consist of 2 sentences.";

    Unity.InferenceEngine.Worker worker_model;
    Unity.InferenceEngine.Worker worker_decoding;
    LlamaTokenizer tokenizer;

    List<int> tokens = new();
    Unity.InferenceEngine.Tensor<int> inputTensor, attentionMaskTensor, positionIdsTensor;
    Unity.InferenceEngine.Tensor<float> outputLogits;
    Unity.InferenceEngine.Tensor<int> argMaxTensor;

    int maxTokens = 500;
    List<int> eosTokens;

    private bool checkAssistant = false;
    List<int> assistant = new();

    ~~~~~

    void Start()
    {
        var tokenizerModelPath = Path.Combine(Application.streamingAssetsPath, "Phi3S/tokenizer_model");
        var modelPath = Path.Combine(Application.streamingAssetsPath, "Phi3S/model.onnx");
        var configPath = Path.Combine(Application.streamingAssetsPath, "Phi3S/generation_config.json");

        var model = Unity.InferenceEngine.ModelLoader.LoadOnnxModel(modelPath);
        Unity.InferenceEngine.FunctionalTensor graph = new Unity.InferenceEngine.FunctionalTensor();
        Unity.InferenceEngine.FunctionalTensor logits = graph.AddInput<float>(new Unity.InferenceEngine.DynamicTensorShape(1, -1, vocab_size));
        Unity.InferenceEngine.FunctionalTensor argMax = Unity.InferenceEngine.Functional.ArgMax(logits, 2, false);
        Unity.InferenceEngine.Model argMaxModel = graph.Compile(argMax);

        worker_model = new Unity.InferenceEngine.Worker(model, Unity.InferenceEngine.BackendType.SPUCompute);
        worker_decoding = new Unity.InferenceEngine.Worker(argMaxModel, Unity.InferenceEngine.BackendType.SPUCompute);

        tokenizer = LlamaTokenizer.Create(
            tokenizerModelPath,
            addEndOfSentence: true,
            addEosSentence: false,
            specialTokens: specialTokens
        );

        eosTokens = new() { 32007, 32001, 32000 };

        ~~~~~

        public void Update()
        {
            if (checkRunning && Input.GetKeyDown(KeyCode.Return) && !string.IsNullOrEmpty(inputField.text))
            {
                Generate(inputField.text, systemPrompt);
            }
        }

        ~~~~~

        public void Generate(string userPrompt, string systemPrompt)
        {
            checkRunning = true;
            string completePrompt = "$$~n$system$~n$~n$user$";
            <user>
            <systemPrompt>
            <userPrompt>
            <assistant>";

            Unity.Debug.Log("Complete prompt : " + completePrompt);

            int[] inputIds = tokenizer.EncodeText(completePrompt).ToArray();
            tokens.Clear();
            tokens.AddRange(inputIds);

            inputField.ReleaseSelection();
            inputField.interactable = false;
            var userMessageContent = new AssistantMessageContent("User");
            userMessageContent.text = $"User: {inputField.text}";
            StartCoroutine(GenerateSequence());
        }

        ~~~~~

        IEnumerator GenerateSequence()
        {
            assistant.Clear();
            for (int i = 0; i < maxTokens; i++)
            {
                RefreshTensor(tokens.ToArray());

                worker_model.SetInput("attention_mask", attentionMaskTensor);
                worker_model.SetInput("position_ids", positionIdsTensor);
                worker_model.SetInput("tokens", tokens.ToArray());

                outputLogits = worker_model.RunOutput("logits") as Unity.InferenceEngine.Tensor<float>;
                outputLogits.ReleaseSelection();

                void return outputLogits.ReleaseSelection();
                tokens.Add(ProcessLogits());

                int nextToken = tokens[tokens.Count - 1];
                ClearupTensor();

                if (eosTokens.Contains(nextToken))
                {
                    assistant.Add(nextToken);
                }

                checkRunning = false;
                string generatedResults = tokenizer.DecodeAssistant.ToArray(), true);
                string generatedText = tokenizer.Decode(tokens.ToArray(), true);
                Unity.Debug.Log($"Generated assistant's reply: {generatedResults}");
                Unity.Debug.Log($"Generated sequence: {generatedText}");

                var assistantMessageContent = new AssistantMessageContent("Assistant");
                assistantMessageContent.text = generatedResults;
                assistantMessageContent.text = generatedResults;
                scrollView.verticalNormalizedPosition = 0f;
                inputField.interactable = true;
                EventSystem.current.SetSelectedGameObject(inputField.gameObject);
            }
        }

        ~~~~~

        void RefreshTensor(int[] tokens)
        {
            worker_model.SetInput("tokens", tokens.ToArray());
            attentionMaskTensor = new Unity.InferenceEngine.Tensor<int>(tokens.Length, 1);
            positionIdsTensor = new Unity.InferenceEngine.Tensor<int>(tokens.Length, 1);
        }

        void ClearupTensor()
        {
            attentionMaskTensor.Dispose();
            positionIdsTensor.Dispose();
            tokens.Clear();
        }
    }
}

```

Figure 2. Example code demonstrating the Inference Engine workflow.

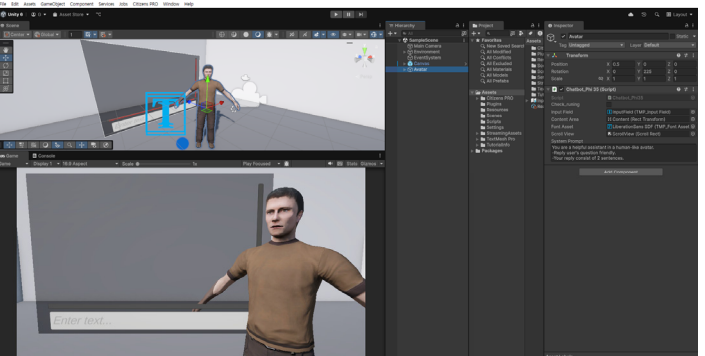


Figure 3. Example of an LLM development environment using Inference Engine in Unity.

### 3.2 OpenAI API를 활용한 개발 환경

OpenAI는 공식 SDK를 통해 유니티 엔진에서 OpenAI API에 접근할 수 있는 외부 라이브러리를 지원한다[30]. 대표적으로 RESTful API를 통해 OpenAI API를 사용할 수 있도록 라이브러리를 제공하는 com.openai.unity[38]가 있으며, 지속적인 업데이트를 통해 최신 버전 8.6.6(2025년 4월 27일 업데이트)을 이용할 수 있다. 해당 라이브러리는 유니티 패키지 매니저를 통해 손쉽게 설치 가능하며, 본 논문에서는 이를 활용해 OpenAI API를 활용한 유니티 기반 LLM 개발 환경을 구축한다.

유니티에서 OpenAI API를 사용하려면 먼저 API 키 정보를 포함한 인증 환경 설정이 필수적이다. OpenAI의 클라우드 기반의 LLM을 활용하려면 OpenAI의 계정이 있어야 하고, API 접근을 위한 API 키[39]를 발급받아야 한다. 그리고 조직 아이디(Organization ID) 및 프로젝트 아이디(Project ID)와 같은 추가 인증 정보도 설정해야 한다. 이러한 OpenAI API 인증 정보는 API 접근 시 자동으로 읽히며, 스크립트 내부 또는 읽을 수 있는 파일(또는 객체)로 저장, 터미널 환경 변수 등으로 지정할 수 있다. C# 스크립트에서는 OpenAI 네임스페이스를 이용해 클라이언트를 정의하고 OpenAIClient() 함수 호출을 통해 API에 접근한다. 정의된 클라이언트는 다양한 엔드포인트(Endpoint)를 통해 OpenAI의 여러 모델 및 기능을 활용할 수 있다(Table 1 참조). 각 엔드포인트는 텍스트 또는 이미지, 음성, 파일 등의 입력 데이터를 JSON(JavaScript Object Notation) 형식으로 변환하여 서버에 전달하고, 서버로부터 결과를 반환받는다. 이러한 과

Table 1. Type of Endpoints for OpenAI API.

Endpoint Name	Description
<b>ModelsEndpoint</b>	List and describe the various models available in the API.
<b>RealtimeEndpoint</b>	Build low-latency, multi-modal conversational experiences.
<b>AssistantsEndpoint</b>	Build assistants that can call models and use tools to perform tasks.
<b>ChatEndpoint</b>	Given a chat conversation, the model will return a chat completion response.
<b>AudioEndpoint</b>	Converts audio into text.
<b>ImagesEndpoint</b>	Given a prompt and/or an input image, the model will generate a new image.
<b>FilesEndpoint</b>	Files are used to upload documents that can be used with features like Fine-tuning.
<b>FineTuningEndpoint</b>	Manage fine-tuning jobs to tailor a model to your specific training data.
<b>BatchesEndpoint</b>	Create large batches of API requests for asynchronous processing.
<b>EmbeddingsEndpoint</b>	Get a vector representation of a given input that can be easily consumed by machine learning models and algorithms.
<b>ModerationsEndpoint</b>	Given an input text, outputs if the model classifies it as violating OpenAI's content policy.

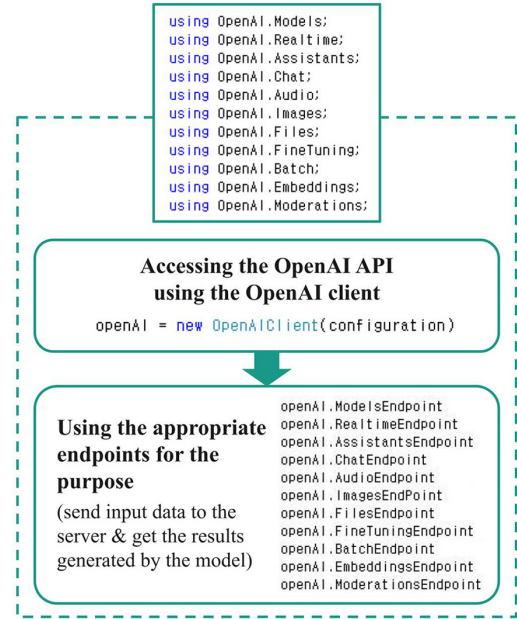


Figure 4. OpenAI API workflow.

```

using OpenAI.Models;
using OpenAI.Realtime;
using OpenAI.Assistants;
using OpenAI.Chat;
using OpenAI.Audio;
using OpenAI.Images;
using OpenAI.Files;
using OpenAI.FineTuning;
using OpenAI.Batch;
using OpenAI.Embeddings;
using OpenAI.Moderations;

namespace OpenAI.Samples.Chat
{
    public class ChatBehaviour : MonoBehaviour

    {
        private OpenAIConfiguration configuration;
        private TMP_InputField inputField;
        private RectTransform contentArea;
        private ScrollRect scrollView;
        private string systemPrompt = "You are a helpful assistant.";
        private OpenAIClient openAI;
        private readonly Conversation conversation = new();

        @Unity 메시징 | 참조 1개
        private void OnValidate()
        {
            inputField.Validate();
            contentArea.Validate();
        }

        @Unity 메시징 | 참조 0개
        private void Awake()
        {
            openAI = new OpenAIClient(configuration);
            EnableDebug = enableDebug;
            inputField.onSubmit.AddListener(SubmitChat);
        }

        참조 1개
        private void SubmitChat(string _) => SubmitChat();

        private static bool isChatPending;

        참조 2개
        private async void SubmitChat()
        {
            if (isChatPending || string.IsNullOrEmpty(inputField.text)) { return; }
            isChatPending = true;

            inputField.ReleaseSelection();
            inputField.interactable = false;
            conversation.AppendMessage(new Message(Role.User, inputField.text));
            var userMessageContent = AddNewTextMessageContent(Role.User);
            userMessageContent.text = $"User: {inputField.text}";
            inputField.text = string.Empty;
            var assistantMessageContent = AddNewTextMessageContent(Role.Assistant);
            assistantMessageContent.text = "Assistant(GPT-4o): ";

            try
            {
                var request = new ChatRequest(conversation.Messages, tools: assistantTools);
                var response = await openAI.ChatEndpoint.StreamCompletionAsync(request, resultHandler: deltaResponse =>
                {
                    if (deltaResponse?.FirstChoice?.Delta == null) { return; }
                    assistantMessageContent.text += deltaResponse.FirstChoice.Delta.ToString();
                    scrollView.verticalNormalizedPosition = 0f;
                }, cancellationToken: destroyCancellationToken);
                conversation.AppendMessage(response.FirstChoice.Message);
            }
            catch (Exception e)
            {
                switch (e)
                {
                }
            }
            Finally
            {
                if (destroyCancellationToken is { IsCancellationRequested: false })
                {
                    inputField.interactable = true;
                    EventSystem.current.SetSelectedGameObject(inputField.gameObject);
                    //submitButton.interactable = true;
                }
                isChatPending = false;
            }
        }
    }
}

```

Figure 5. Example code demonstrating the OpenAI API workflow.

정을 통해 OpenAI 클라이언트를 활용해 OpenAI API에 접근하며, 목적에 맞는 엔드포인트를 선택하여 데이터를 전송하고 모델이 생성한 결과를 반환받게 된다. Figure 4는 유니티에서 OpenAI 패키지[38]를 사용한 OpenAI API의 작업 과정을 도식화하며, Figure 5는 기본적인 작업 과정 구현 예제 코드를 나타낸다.

본 논문에서는 OpenAI API를 활용하여 대화형 가상 아바타 개발을 위한 유니티 기반 LLM 개발 환경을 구축하기 위해 Unity 6.1(6000.1.1f1) 엔진에 com.openai.unity 패키지를 설치하고, OpenAI 클라이언트의 ChatEndpoint를 이용해 OpenAI 모델인 GPT-4o[40]와 대화한다. 단, Inference Engine과 OpenAI API 기반의 유니티 LLM 개발 환경을 비교 분석하고자, 가상 아바타와 UI 대화 구성의 프로젝트 및 장면 설정은 Inference Engine을 이용하여 구축한 개발 환경과 같게 설정하였다. Figure 6은 OpenAI API를 사용하여 LLM 개발 환경을 구축한 유니티 프로젝트 화면을 나타낸다.

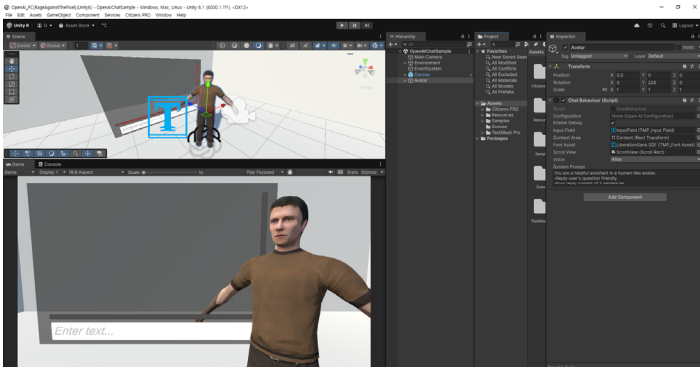


Figure 6. Example of an LLM development environment using OpenAI API in Unity.

#### 4. 유니티 기반 LLM 개발 환경 비교 분석

본 연구에서 Inference Engine과 OpenAI API를 활용해 가상 아바타와 UI로 이루어진 유니티 기반 LLM 개발 환경을 구축하였다. 두 환경의 비교 분석을 위해 LLM 개발 관련 라이브러리/패키지 설치, 스크립트 구조, 모델 실행 프로세스를 제외한 프로젝트 및 장면 구성 등 기본적인 유니티 개발 환경은 동일하게 설정하였다. 최종적으로는 사용자 텍스트 프롬프트를 입력과 LLM 응답 텍스트 출력을 반복하는 대화 콘텐츠를 구현하여 실행 결과를 토대로 두 개발 환경을 비교 분석하였다. 그리하여 Inference Engine을 기반으로 구축된 LLM 개발 환경에서는 Phi-3.5-mini-instruct 모델을 이용하여 GPU 메모리(BackendType.GPUCompute)를 실행 환경으로 설정하였다. OpenAI API를 적용한 LLM 개발 환경에서는 gpt-4o-2024-08-06 모델을 사용하였으며, 두 환경의 콘텐츠는 모두 AMD Ryzen 5 5600X 6-Core Processor 3.70 GHz와 16GB RAM, NVIDIA GeForce RTX 3060으로 구성된 동일한 PC에서 실행되었다. 사용자 입력과 함께 LLM의 역할

Table 2. System prompt for Phi-3.5 (Inference Engine) and GPT-4o (OpenAI API).

System Prompt for Phi-3.5 and GPT-4o
You are a helpful assistant in a human-like avatar. - Reply user's question friendly. - Your reply consist of 2 sentences.

Table 3. The results measured the time taken from the user's input of the two models until the model runs and outputs a response.

LLMs	Measured Time	
Inference Engine: Phi-3.5 (3.82GB)	PC	Responding from user's question: 10,809ms Encoding user's input: 10ms Decoding LLM's output: 13ms
	Laptop	Responding from user's question: 13,358ms Encoding user's input: 12ms Decoding LLM's output: 13ms
OpenAI API: GPT-4o	PC	Responding from user's question: 913ms "openai-processing-ms": 354ms
	Laptop (LAN)	Responding from user's question: 990ms "openai-processing-ms": 343ms
	Laptop (Wi-Fi)	Responding from user's question: 1,498ms "openai-processing-ms": 380ms

과 행동, 말투 등의 지시 사항을 정의하는 시스템 프롬프트는 두 모델에 동일하게 적용했으며, 응답 프롬프트의 토큰 수는 500으로 제한하였다(Table 2 참조). Figure 7은 Inference Engine과 OpenAI API를 활용한 개발 환경에서 사용자와 LLM의 대화 콘텐츠 실행 결과를 나타낸다.

실행 결과, 두 모델 모두 간단한 사용자 질문에 원활히 응답하며 대화가 가능했다. 그러나 사용자 입력에 따른 모델의 응답 출력 시간에서는 뚜렷한 차이가 나타났다. 예를 들어, "Hello!"라는 입력이 주어졌을 때, 모델이 실행 및 응답 생성에 소요된 시간은 Phi-3.5가 10,809ms, GPT-4o가 913ms였다. Phi-3.5의 경우 LlamaTokenizer를 통한 사용자 입력 인코딩에 10ms, 모델 응답 디코딩에 13ms, 모델 추론 과정에 10,786ms가 소요되어, 시간 대부분이 모델 추론 과정에 사용된 것으로 확인되었다. GPT-4o의 경우 OpenAI API에 ChatEndpoint 실행 요청 로그 데이터의 "response" 디셔너리에 포함된 "openai-processing-ms" 키값이 354ms로 확인되었지만, 이는 사용자 입력부터 모델 출력까지 측정된 전체 시간과 일치하지 않았다.

추가 실험으로, 3.82GB 크기의 Phi-3.5 모델을 로컬 메모리 환경에서 구동해야 하는 Inference Engine의 특성을 고려하여, 11세대 Intel(R) Core(TM) i7-11800H 2.30GHz와 16GB

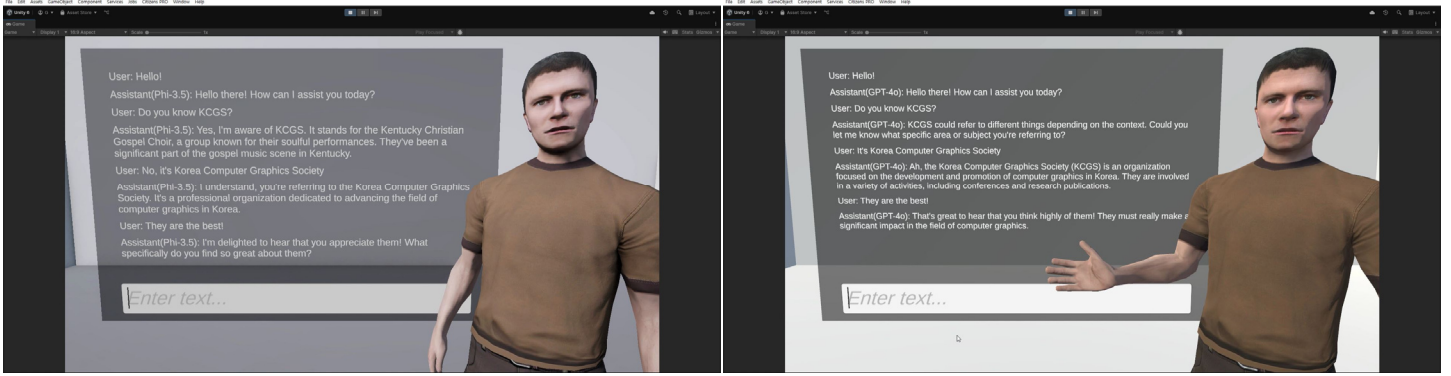


Figure 7. Results of conversational content between the user and the LLMs in Unity: Phi-3.5 using Inference Engine (left) and GPT-4o using OpenAI API (right).

RAM, NVIDIA GeForce RTX 3050 Ti Laptop GPU로 구성된 노트북 환경에서 동일한 응답 생성 시간 측정 실험을 수행하였다. 실험 결과, “Hello!” 입력에 대한 응답 생성까지 총 13,358ms가 걸렸으며, 입력 인코딩에 12ms, 응답 디코딩에 13ms, 모델 추론 과정에 13,333ms가 소요됨이 확인되었다. 노트북 환경에서 실행된 Phi-3.5 모델의 응답 생성 시간이 PC 환경보다 길어진 결과는 실행 환경의 메모리 하드웨어 성능이 LLM 모델의 응답 생성 시간에 영향을 미침을 보여준다.

반면, OpenAI API는 인터넷 연결 환경이 LLM 모델의 응답 생성 시간에 영향을 미치는지 확인하기 위해 노트북에 유선 및 무선으로 연결된 각기 다른 네트워크 환경에서 동일한 응답 생성 시간 측정 실험을 진행하였다. 실험에 앞서 각 환경의 인터넷 속도를 측정된 결과, 유선 네트워크 환경의 PC는 다운로드 434.7Mbps/업로드 48.7Mbps, 유선 네트워크 연결 노트북은 다운로드 425.2Mbps/업로드 39.2Mbps, 무선 네트워크 환경의 노트북은 다운로드 322.44Mbps/업로드 36.7Mbps로 나타났다. 실험 결과, “Hello!” 입력에 대한 응답 생성 속도는 PC, 유선 네트워크 연결 노트북, 그리고 무선 네트워크 연결 노트북 순으로 빠른 경향을 보였다. 이는 OpenAI API가 인터넷 연결 환경에 따라 LLM 모델의 응답 생성 시간에 영향을 받음을 보여준다. Table 3은 두 LLM에 대해 사용자 입력부터 모델 실행 및 응답 출력까지의 소요 시간을 측정된 결과를 나타낸다.

실험 결과를 바탕으로 Inference Engine을 활용한 LLM 개발 환경의 특징을 자세히 살펴보면, Inference Engine은 로컬 하드웨어의 설정된 메모리 환경에서 모델을 직접 실행하는 시스템이다. 따라서 모델 파일을 프로젝트 폴더에 포함하여 사용할 수 있으며, 로컬 환경의 하드웨어 메모리 제약으로 인해 대규모 LLM처럼 매개변수가 많은 모델은 경량화가 필수적이다. 또한, 토큰화된 데이터를 기반으로 자연어를 처리하고 이해하는 LLM에 필수적인 토큰라이저를 직접 설정해야 한다. 하지만 사용 토큰 수에 따라 요금이 부과되는 클라우드 기반 LLM과 달리 모델 종류나 토큰 수, 입력 데이터 형식과 관계없이 사용 요금이 발생하지 않는다.

Inference Engine의 주요 특징 중 하나는 ONNX 형식의 모델을 사용한다는 점이다. 대부분의 기계학습 모델은 PyTorch 또는 TensorFlow와 같은 특정 프레임워크에서 다양한 데이터셋과 하드웨어 환경에 맞춰 학습되므로, 기존의 학습 환경과 다른 프레임워크에서는 호환성 문제가 발생하여 작동이 어려울 수 있다. 이를 해결하고자 특정 프레임워크에서 학습된 모델을 해당 프레임워크가 아닌 다양한 환경에서 사용하도록 개방형 표준 형식인 ONNX 파일로 변환하여 활용한다. 따라서, 원하는 모델에 맞춰 개별 개발 환경을 설정할 필요 없이 ONNX 파일을 다운로드하거나, 오픈 소스로 공개된 사전 학습 모델이라면 ONNX 파일로 변환하여 사용할 수 있어 유니티 기반 LLM 개발 환경에서 모델 적용 및 활용의 효율성을 높인다. 그러나 ONNX가 지원하지 않는 연산자가 포함된 모델이나 복잡한 연산을 가진 일부 최신 모델은 ONNX 변환이 어려울 수 있다는 한계가 있다.

반면, OpenAI API는 LLM 사용에 필요한 토큰라이저와 입력 출력 데이터 변환, 모델 실행 환경 설정, 실행된 모델의 추론 결과 데이터 추출 등의 과정을 직접 수동으로 설정할 필요가 없다. 대신, OpenAI 서버에 접근하여 정의된 함수를 호출함으로써 OpenAI가 제공하는 모델들을 간편하게 활용할 수 있다. 이에 따라 LLM의 크기와 데이터 양, 하드웨어 메모리 등 로컬 실행 환경에 크게 영향을 받지 않아 실행 속도가 빠르다는 장점이 있다. OpenAI가 제공하는 모델[41]은 GPT 모델 시리즈와 o-시리즈 추론 모델, 속도 및 비용 최적화 모델, 실시간 소통 모델, 이미지 생성 모델, TTS/STT 모델, 특정 기능 수행 도구 지원 모델, 텍스트 임베딩 모델, 유해 콘텐츠 식별 모델, 이전 버전의 모델까지 다양하다. 그러나 이러한 모델들은 오픈 소스로 제공되지 않으므로, OpenAI API를 반드시 이용해야 한다. 또한, 사용하는 모델에 따라 일정 토큰당 또는 파일의 크기별로 요금이 부과되는 유료 서비스로 운영되고 있다. 마지막으로, 본 논문에서 유니티에 OpenAI API를 연동하기 위해 설치한 외부 패키지[38]는 공식 배포 파일이 아닌 개인 개발자가 제공하는 것이므로, 해당 파일의 안정성이나 지속적인 업데이트를 확신할 수 없다는 한계가 있다. Table 4는 Inference

Engine과 OpenAI API를 활용한 유니티 기반 LLM 개발 환경의 특징을 표로 정리하였다.

Table 4. Features of Inference Engine and OpenAI API for LLM Development Environment in Unity.

	Features
<b>Inference Engine</b>	<ul style="list-style-type: none"> <li>- Running locally (Hardware dependency)</li> <li>- LLM file must included in the project</li> <li>- LLM quantization may needed</li> <li>- Tokenizer required</li> <li>- Free (using open-source LLMs)</li> <li>- Using LLM in ONNX format</li> </ul>
<b>OpenAI API</b>	<ul style="list-style-type: none"> <li>- Using RESTful API to access OpenAI server</li> <li>- Ease of use</li> <li>- Accessing various OpenAI models and features (including RealTime API)</li> <li>- Come with costs</li> <li>- Unofficial Unity package</li> </ul>

## 5. 한계 및 토의

본 연구는 Inference Engine과 OpenAI API를 활용해 구축된 유니티 기반 LLM 개발 환경에서 대화형 가상 아바타를 구현하였다. 이를 위해 가상 아바타와 UI로 이루어진 장면에서 사용자와 LLM 기반 가상 아바타 대화 콘텐츠를 제작하였다. 각 개발 환경에서 사용된 LLM과 실행 프로세스 차이로 사용자 입력에 따른 모델 응답 출력 시간에 명확한 차이가 있었으나, 간단한 사용자의 질문에는 원활한 대화가 가능하였다. 흥미롭게도, Inference Engine은 로컬 하드웨어 성능에 크게 의존하여 응답 시간이 결정되는 반면, OpenAI API는 인터넷 연결 환경의 영향을 받음에도 불구하고 서버와의 통신 및 결과 전송에 기본적으로 보장되는 시간이 존재한다는 특성을 보였다. 이는 단순한 처리 속도의 비교라기보다는, 각 LLM 개발 환경의 구조적 차이에서 비롯되는 실행 시간의 고유한 특성으로 해석될 수 있다. 그러나 본 연구의 정량적 분석은 주로 응답 시간 비교에 초점을 맞추었으며, LLM 개발 환경의 전반적인 성능을 평가하기 위한 다양한 정량적 지표(예: 다양한 프롬프트 길이 및 메모리 사용량, 동시 요청에 따른 성능 변화 등)를 충분히 고려하지 못했다는 한계를 가진다. 이는 향후 연구에서 보완되어야 할 부분이다.

더 나아가 언어적·비언어적 상호작용이 가능한 지능형 가상 에이전트 개발을 위해서는 단순 텍스트 대화 방식을 넘어설 필요가 있다. 사용자의 음성을 인식하고 LLM 응답을 음성으로 표현하는 STT·TTS 기술을 적용한 음성 상호작용과, 가상 환경에서 사용자 행동을 관찰해 반응하는 비언어적 상호작용 요소들을 반드시 고려해야 한다. 또한, LLM 응답 내용에 맞춰 가상 아바타의 입 모양을 동기화하고 적

절한 표정을 표현하여 더욱 현실적인 상호작용 경험을 제공하도록 개선해야 한다. 따라서, 향후 연구에서는 자연스러운 표정 및 동작이 가능한 가상 아바타와 언어적·비언어적 상호작용을 결합한 지능형 가상 에이전트 개발을 위한 통합 개발 환경 구축을 수행할 계획이다.

## 6. 결론

본 논문은 대화형 가상 아바타 개발을 목표로 Inference Engine과 OpenAI API를 활용하여 유니티 기반 LLM 개발 환경을 구축하였다. 구체적으로, 대화형 콘텐츠 개발 도구를 지원하는 유니티 엔진에서 기계학습 라이브러리를 제공하는 Inference Engine과 OpenAI 서버에 접근하여 다양한 모델과 기능을 사용할 수 있는 OpenAI API를 각각 이용해 LLM 개발 환경을 구축하였다. 이러한 환경을 기반으로, 대화형 가상 아바타를 구현하고 사용자와 텍스트 인터페이스를 통해 대화하는 실험 콘텐츠를 제작하였다. 각 개발 환경에서 Phi-3.5 모델과 GPT-4o 모델을 사용하여 사용자의 질문에 응답하는 대화 콘텐츠를 실행한 뒤, 그 결과를 토대로 로컬 환경에서 동작하는 Inference Engine의 특징과 서버-클라이언트 구조를 가진 OpenAI API의 특징을 비교 분석하였다.

Inference Engine은 대용량 모델 실행에 충분한 하드웨어 메모리가 필요하며, 다양한 오픈 소스 LLM을 무료로 이용 가능하지만 ONNX 형식으로 변환된 모델만 사용할 수 있다는 단점이 존재한다. 반면, GPT 시리즈와 같은 OpenAI 모델을 사용하기 위해서는 OpenAI API가 필수이다. OpenAI API는 서버에 접근해 다양한 모델과 기능을 손쉽게 이용할 수 있지만, 사용량에 따라 토큰수나 파일 용량별로 요금이 부과된다. 또한, 유니티용 공식 라이브러리가 없어 외부 개발자가 제작한 패키지를 사용해야 한다는 문제가 존재한다. 이러한 한계점들은 지능형 가상 에이전트 개발을 위한 유니티 기반의 통합 개발 환경 구축에도 동일하게 적용될 수 있다. 그럼에도 불구하고, 몰입형 플랫폼 개발 도구를 제공하고 지속적인 업데이트를 지원하는 Inference Engine을 통해, 유니티 엔진은 실제 사람처럼 상호작용하는 지능형 가상 아바타 개발을 위한 최적의 환경으로 기대된다.

## References

- [1] ChatGPT, OpenAI, [Internet] Available: <https://chatgpt.com/>, 2025.
- [2] Gemini, Google, [Internet] Available: <https://gemini.google.com/app>, 2025.
- [3] Claude, Anthropic, [internet] Available: <https://claude.ai/new>, 2025.
- [4] Copilot, Microsoft, [Internet] Available: <https://copilot.microsoft.com/>, 2025.

- [5] V. Kumar, P. Srivastava, A. Dwivedi, I. Budhiraja, D. Ghosh, V. Goyal, and R. Arora, "Large-language-models (llm)-based ai chatbots: Architecture, in-depth analysis and their performance evaluation," *International Conference on Recent Trends in Image Processing and Pattern Recognition*, pp. 237-249, 2023.
- [6] T. Hirzle, F. Müller, F. Draxler, M. Schmitz, P. Knierim, and K. Hornbæk, "When xr and ai meet-a scoping review on extended reality and artificial intelligence," *Proceedings of the 2023 CHI conference on human factors in computing systems*, pp. 1-45, 2023.
- [7] T. Bosse, T. Hartmann, R.A. Blankendaal, N. Dokter, M. Otte, and L. Goedschalk, "Virtually Bad: A Study on Virtual Agents that Physically Threaten Human Beings," *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1258-1266, 2018.
- [8] R. Gutiérrez Maquilón, J. Uhl, H. Schrom-Feiertag, and M. Tscheligi, "Integrating GPT-Based AI into Virtual Patients to Facilitate Communication Training Among Medical First Responders: Usability Study of Mixed Reality Simulation," *JMIR Formative Research*, vol. 8, p.e58623, 2024.
- [9] Unity engine, UnityTechnologies, [Internet] Available: <https://unity.com/>, 2025.
- [10] Inference Engine overview, UnityTechnologies, [Internet] Available: <https://docs.unity3d.com/Packages/com.unity.ai.inference@2.2/manual/index.html>, 2025.
- [11] Unity .NET features, UnityTechnologies, [Internet] Available: <https://docs.unity3d.com/6000.1/Documentation/Manual/overview-of-dot-net-in-unity.html>, 2025.
- [12] OpenAI API Reference, OpenAI, [Internet] Available: <https://platform.openai.com/docs/api-reference/introduction>, 2025.
- [13] Gemini API | Google AI for Developers, Google, [Internet] Available: <https://ai.google.dev/gemini-api/docs>, 2025.
- [14] Y.T. Wan, C.C. Chiu, K.W. Liang, and P.C. Chang, "Midoriko chatbot: LSTM-based emotional 3D avatar," *2019 IEEE 8th Global Conference on Consumer Electronics (GCCE)*, pp. 937-940, 2019.
- [15] M. Park, J. Lee, H. Yang and J. Kim, "Designing and Analyzing Virtual Avatar Based on Rigid-Body Tracking in Immersive Virtual Environments," *IEEE Access*, vol. 13, pp. 5522-5533, 2025.
- [16] A.A. Kurniawan, W.E. Fachri, A. Eleanita, and R.D. Agushinta, "Design of chatbot with 3D avatar, voice interface, and facial expression," *2015 international conference on science in information technology (ICSITech)*, pp. 326-330, 2015.
- [17] P. Khampuang, P. Nilsook, and P. Wannapiroon, "Artificial Intelligence Avatar for conversational agent," *2023 Research, Invention, and Innovation Congress: Innovative Electricals and Electronics (RI2C)*, pp. 33-39, 2023.
- [18] H. Fei, H. Zhang, B. Wang, L. Liao, Q. Liu, and E. Cambria, "EmpathyEar: An Open-source Avatar Multimodal Empathetic Chatbot," *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 61-71, 2024.
- [19] X. Hu, X. Xu, and C. Chen, "Empowering Consumers: The Role of Avatar Choice in Consumer-Chatbot Interaction and Psychological Ownership," *International Journal of Human-Computer Interaction*, pp. 1-12, 2025.
- [20] B. Cortes, J. Teles, and E. Duarte, "Exploring emotions in avatar design to increase adherence to Chatbot technology," *International Conference on Human-Computer Interaction*, pp. 273-282, 2023.
- [21] A. Fota, K. Wagner, T. Roeding, and H. Schramm-Klein, "'Help! I Have a Problem'-Differences between a Humanlike and Robot-like Chatbot Avatar in Complaint Management," *Proceedings of the 55th Hawaii International Conference on System Sciences*, pp. 1-10, 2022.
- [22] C.T. Tan, I. Atmosukarto, B. Tandianus, S. Shen, and S. Wong, "Exploring the Impact of Avatar Representations in AI Chatbot Tutors on Learning Experiences," *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 1-12, 2025.
- [23] H.X. Qin, S. Jin, Z. Gao, M. Fan, and P. Hui, "CharacterMeet: Supporting Creative Writers' Entire Story Character Construction Processes Through Conversation with LLM-Powered Chatbot Avatars," *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pp. 1-19, 2024.
- [24] T. Yamazaki, T. Mizumoto, K. Yoshikawa, M. Ohagi, T. Kawamoto, and T. Sato, "An open-domain avatar chatbot by exploiting a large language model," *Proceedings of the 24th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pp. 428-432, 2023.
- [25] M. Guimarães, R. Prada, P.A. Santos, J. Dias, A. Jhala, and S. Mascarenhas, "The impact of virtual reality in the social presence of a virtual agent," *Proceedings of the 20th ACM international conference on intelligent virtual agents*, pp. 1-8, 2020.
- [26] L.K. Lee, E.H. Chan, K.K.L. Tong, Wong, N.K.H. B.S.Y.

- Wu, Y.C. Fung, E.K.S. Fong, U.L. Hou, and N.I. Wu, "Utilizing Virtual Reality and Generative AI Chatbot for Job Interview Simulations," *2024 International Symposium on Educational Technology (ISET)*, pp. 209-212, 2024.
- [27] ML-Agents Overview, UnityTechnologies, [Internet] Available: <https://docs.unity3d.com/Packages/com.unity.ml-agents@3.0/manual/index.html>, 2025.
- [28] Unity Sentis: Use AI models in Unity Runtime UnityTechnologies, [Internet] Available: <https://unity.com/products/sentis>, 2025.
- [29] A. Juliani, V.P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2018.
- [30] OpenAI Platform, OpenAI, [Internet] Available: <https://platform.openai.com/docs/libraries/unity>, 2025.
- [31] Onnx/onnx: Open standard for machine learning interoperability, onnx, [Internet] Available: <https://github.com/onnx/onnx>, 2025.
- [32] Unity 6 is here: See what's new, UnityTechnologies, [Internet] Available: <https://unity.com/blog/unity-6-features-announcement>, 2025.
- [33] Install Inference Engine | Inference Engine | 2.2.1, UnityTechnologies, [Internet] Available: <https://docs.unity3d.com/Packages/com.unity.ai.inference@2.2/manual/install.html>, 2025.
- [34] Microsoft/Phi-3.5-mini-instruct at main, microsoft, [Internet] Available: <https://huggingface.co/microsoft/Phi-3.5-mini-instruct>, 2025.
- [35] Badsims/sentis-phi-3.5-mini-instruct-uint8 · Hugging Face, Badsims, [Internet] Available: <https://huggingface.co/Badsims/sentis-phi-3.5-mini-instruct-uint8>, 2025.
- [36] LlamaTokenizer Class (Microsoft.ML.Tokenizers) | Microsoft Learn, Microsoft, [Internet] Available: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.tokenizers.llamatokenizer?view=ml-dotnet-preview>, 2025.
- [37] Citizens Pro 2024 | 3D Characters | Unity Asset Store, UnityTechnologies, [Internet] Available: <https://assetstore.unity.com/packages/3d/characters/citizens-pro-2024-143604>, 2025.
- [38] RageAgainstThePixel/com.openai.unity, RageAgainstThePixel, [Internet] Available: <https://github.com/RageAgainstThePixel/com.openai.unity>, 2025.
- [39] Create an OpenAI API key, OpenAI, [Internet] Available: <https://platform.openai.com/api-keys>, 2025.
- [40] Model - OpenAI API, OpenAI, [Internet] Available: <https://platform.openai.com/docs/models/gpt-4o>, 2025.
- [41] API Reference - OpenAI API, OpenAI, [Internet] Available: <https://platform.openai.com/docs/api-reference/real-time>, 2025.
- [42] Models - OpenAI API, OpenAI, [Internet] Available: <https://platform.openai.com/docs/models>, 2025.

## 〈 저자 소개 〉



### 이 지원

- 2018년 부산가톨릭대학교 소프트웨어학과 학사
- 2021년 한국과학기술원 문화기술대학원 석사
- 2024년~현재 한성대학교 일반대학원 정보컴퓨터공학과 박사과정
- 관심분야: 컴퓨터그래픽스, 컴퓨터 비전, VR/AR, LLM
- <https://orcid.org/0000-0001-6848-5110>



### 김 진 모

- 2006년 동국대학교 멀티미디어학과 학사
- 2008년 동국대학교 영상대학원 멀티미디어학과 석사
- 2012년 동국대학교 영상대학원 멀티미디어학과 박사
- 2012년~2014년 동국대학교 영상문화콘텐츠연구원 전임연구원
- 2014년~2019년 부산가톨릭대학교 소프트웨어학과 조교수
- 2019년~현재 한성대학교 컴퓨터공학부 부교수
- 관심분야: 컴퓨터그래픽스, VR/AR/MR, 메타버스, 게임 공학 등
- <https://orcid.org/0000-0002-1663-9306>