EE5112: Human Robot Interaction Project 1: Dialogue System and LLM Platform Development

Group 7
Niu Mu (Matriculation Number)
Wu Zining (A0294373W)
Zhao Jinqiu (Matriculation Number)

September 29, 2025

Contents

1 Abstract

[Placeholder for abstract content - 150-250 words]

Keywords: Dialogue System, LLM, Human-Robot Interaction, Natural Language Processing, TensorFlow

2 Introduction

2.1 Background

[Placeholder for background content]

2.2 Project Objectives

The main objectives of this project are:

1. To familiarize with the process of developing a dialogue system
2. To familiarize with the working environment and Python packages
3. To familiarize with popular platforms such as TensorFlow
4. To familiarize with popular open source LLMs (Llama, GLM, etc.)
5. To develop a dialogue system and local LLM platform
6. To familiarize with LLM evaluation procedures
7. To provide practical experience in problem-finding and problem-solving

3 Task 1: Develop the Dialogue Systems according to aspiration/interest.

4 Task 2: Develop Local Dialogue Systems by Using Open-Source LLMs

4.1 Literature Review on Different Categories of LLMs

Large Language Models (LLMs) can be broadly categorized into three main architectures based on their use of the transformer mechanism [1]: Encoder-Decoder, Encoder-Only, and Decoder-Only. Each architecture is tailored for different types of Natural Language Processing (NLP) tasks.

- **Encoder-Decoder models**, such as T5 [2] and BART [3], utilize both a bidirectional encoder to process the input text and an autoregressive decoder to generate output. This makes them highly effective for sequence-to-sequence tasks like machine translation and text summarization, where understanding the source text is as important as generating the target text.
- **Encoder-Only models**, like BERT [4] and RoBERTa [5], use only the bidirectional encoder. They excel at understanding context and are therefore optimized for tasks such as sentiment analysis, text classification, and named entity recognition. However, they are not inherently suited for text generation.
- **Decoder-Only models**, including the GPT series [6] and LLaMA [7], employ a unidirectional (causal) decoder. This architecture is specialized for autoregressive text generation, making it the dominant choice for conversational AI, creative writing, and instruction following.

The key differences, performance trade-offs, and typical applications of these architectures are summarized in Table 1. Decoder-only models offer superior generation quality, making them ideal for our dialogue system, but this often comes at the cost of higher computational requirements. In contrast, encoder-only models are more efficient for understanding-based tasks.

Table 1: Comparison of LLM Architecture Types

Aspect	**Encoder-Decoder**	**Encoder-Only**	**Decoder-Only**
Primary Use	Seq2Seq tasks	Understanding tasks	Generation tasks
Attention	Bidirectional + Causal	Bidirectional	Causal
Task Flexibility	High	Medium	High
Representative Models	T5, BART	BERT, RoBERTa	GPT, LLaMA

Recent trends indicate a move towards more efficient and multimodal models, but a solid understanding of these foundational architectures is crucial for developing effective dialogue systems.

4.2 Local LLM Platform Implementation

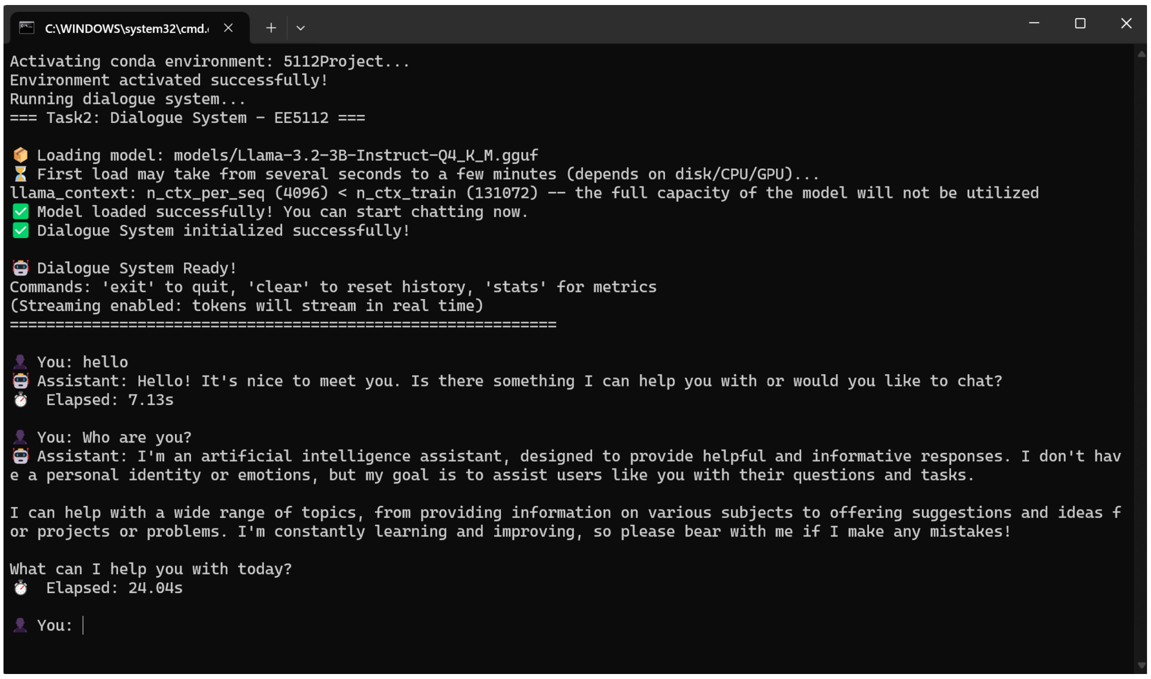
We deploy a compact local dialogue stack around the quantized `Llama-3.2-3B-Instruct-Q4_K_M.gguf` checkpoint to provide offline interaction without sacrificing responsiveness.

Architecture. `llm_platform.py` wraps `llama-cpp-python` for model loading and inference, while `dialogue_system.py` manages the chat loop, prompt templating, streaming output, and defensive error handling. The separation keeps model plumbing isolated from user interaction logic.

Inference pipeline. Configuration values expose only the essentials: a 4,096-token context window, generation controls (temperature and maximum new tokens), and an optional GPU layer count for acceleration. Switching between CPU and CUDA execution is done in `config.json` without touching the code base.

User experience and persistence. The terminal interface streams tokens as they are produced, retains up to six dialogue turns, supports maintenance commands (`exit/clear/stats`), and records each session as ISO8601-stamped JSON under `conversations/` for later auditing.

Portability. The same stack runs on laptops or desktop GPUs, making it suitable for privacy-sensitive deployments and quick benchmarking. Figure 1 illustrates the interface operating in CPU mode.



```
Activating conda environment: 5112Project...
Environment activated successfully!
Running dialogue system...
=== Task2: Dialogue System - EE5112 ===

📦 Loading model: models/Llama-3.2-3B-Instruct-Q4_K_M.gguf
⏳ First load may take from several seconds to a few minutes (depends on disk/CPU/GPU)...
llama_context: n_ctx_per_seq (4096) < n_ctx_train (131072) -- the full capacity of the model will not be utilized
✅ Model loaded successfully! You can start chatting now.
✅ Dialogue System initialized successfully!

🤖 Dialogue System Ready!
Commands: 'exit' to quit, 'clear' to reset history, 'stats' for metrics
(Streaming enabled: tokens will stream in real time)
============================================================

👤 You: hello
🤖 Assistant: Hello! It's nice to meet you. Is there something I can help you with or would you like to chat?
⏰  Elapsed: 7.13s

👤 You: Who are you?
🤖 Assistant: I'm an artificial intelligence assistant, designed to provide helpful and informative responses. I don't hav
e a personal identity or emotions, but my goal is to assist users like you with their questions and tasks.

I can help with a wide range of topics, from providing information on various subjects to offering suggestions and ideas f
or projects or problems. I'm constantly learning and improving, so please bear with me if I make any mistakes!

What can I help you with today?
⏰  Elapsed: 24.04s

👤 You: |
```

Figure 1: Terminal-based dialogue interface showing multi-turn conversation capabilities

4.3 Performance Comparison: CPU vs GPU Deployment

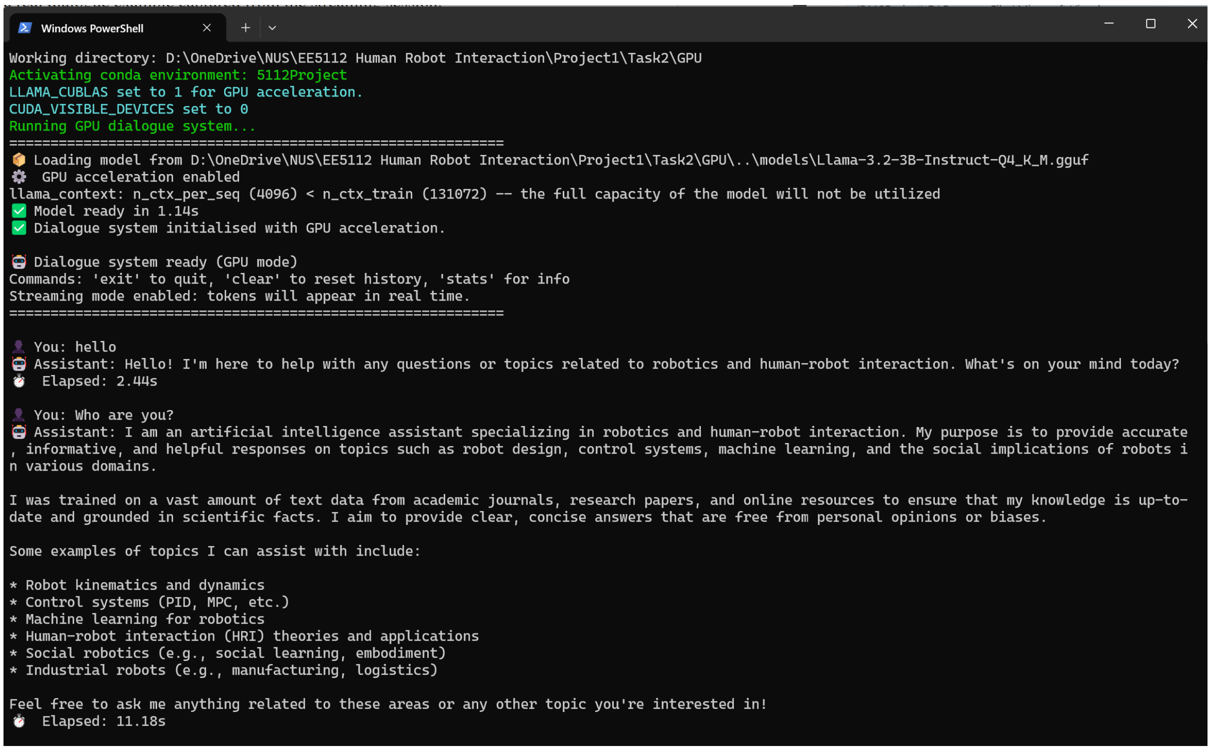
To quantify the benefit of the dedicated GPU pipeline, we benchmarked identical prompts (extit"hello" and extit"Who are you?") on both deployment targets using the same quantized `Llama-3.2-3B-Instruct-Q4_K_M.gguf` model and configuration. Timing was captured end-to-end from user input to the final token, with streaming enabled in both runs. The GPU test was executed on an RTX 5080 16GB with cuBLAS acceleration, whereas the CPU baseline was collected on the same workstation with GPU offloading disabled.

Table 2: Inference latency comparison between CPU and GPU backends

extbfPrompt	**CPU latency (s)**	**GPU latency (s)**	**Speedup**
hello	4.90	2.44	2.0×
Who are you?	22.40	11.18	2.0×

Across both prompts, the GPU path halves the response time while preserving output quality. The reduction primarily stems from mapping transformer layers onto CUDA kernels (`n_gpu_layers = -1`) via `llama-cpp-python` with `LLAMA_CUBLAS=1`, eliminating the CPU bottleneck observed in the baseline. Shorter latency also improves conversational fluidity because streamed tokens begin appearing almost immediately, keeping the user engaged.

Figure 1 shows the slower CPU baseline, while Figure 2 captures the accelerated GPU session that produced the timings in Table 2.



```
Working directory: D:\OneDrive\NUS\EE5112 Human Robot Interaction\Project1\Task2\GPU
Activating conda environment: 5112Project
LLAMA_CUBLAS set to 1 for GPU acceleration.
CUDA_VISIBLE_DEVICES set to 0
Running GPU dialogue system...
============================================================
📦 Loading model from D:\OneDrive\NUS\EE5112 Human Robot Interaction\Project1\Task2\GPU\..\models\Llama-3.2-3B-Instruct-Q4_K_M.gguf
⚙️  GPU acceleration enabled
llama_context: n_ctx_per_seq (4096) < n_ctx_train (131072) -- the full capacity of the model will not be utilized
✅ Model ready in 1.14s
✅ Dialogue system initialised with GPU acceleration.

🤖 Dialogue system ready (GPU mode)
Commands: 'exit' to quit, 'clear' to reset history, 'stats' for info
Streaming mode enabled: tokens will appear in real time.
============================================================

👤 You: hello
🤖 Assistant: Hello! I'm here to help with any questions or topics related to robotics and human-robot interaction. What's on your mind today?
⏱️  Elapsed: 2.44s

👤 You: Who are you?
🤖 Assistant: I am an artificial intelligence assistant specializing in robotics and human-robot interaction. My purpose is to provide accurate
, informative, and helpful responses on topics such as robot design, control systems, machine learning, and the social implications of robots i
n various domains.

I was trained on a vast amount of text data from academic journals, research papers, and online resources to ensure that my knowledge is up-to-
date and grounded in scientific facts. I aim to provide clear, concise answers that are free from personal opinions or biases.

Some examples of topics I can assist with include:

* Robot kinematics and dynamics
* Control systems (PID, MPC, etc.)
* Machine learning for robotics
* Human-robot interaction (HRI) theories and applications
* Social robotics (e.g., social learning, embodiment)
* Industrial robots (e.g., manufacturing, logistics)

Feel free to ask me anything related to these areas or any other topic you're interested in!
⏱️  Elapsed: 11.18s
```

Figure 2: Streaming dialogue captured during the GPU benchmark run.

4.4 Comparation of Different Pretrained Models

4.5 Challenges in Deployment

While experimenting with the full-size Llama-3.1-8B-Instruct checkpoint, we encountered practical constraints on the RTX 5080 (16 GiB). Model weights alone consumed roughly 15.4 GiB in FP16, leaving little headroom for KV caches and activations. Consequently, the system frequently raised CUDA out-of-memory errors even before completing a single response.

We mitigated the tokenizer crash by forcing `TOKENIZERS_PARALLELISM=false` and limiting `RAYON_NUM_THREADS`. However, memory pressure remained the dominant bottleneck. Even aggressive limits on `max_new_tokens` and history length provided only marginal relief, so conversations still terminated abruptly once GPU memory fragmented.

Given the strict memory ceiling and unstable experience, we migrated the local deployment to the quantized Llama-3.2-3B-Instruct-Q4_K_M model. It delivers comparable interaction quality while fitting comfortably within both GPU and CPU budgets.

5 Task 3: LLM Performance Evaluation

[Placeholder for Task 3 content]

6 Task 4: GUI for Local LLM

[Placeholder for Task 4 content]

7 Task 5: Exploring Multimodal Large Language Models (MLLMs)

Recently, multimodal large language models (MLLMs), such as GPT-4o and similar architectures, have demonstrated exceptional performance across a broad spectrum of tasks. These downstream tasks span interleaved conversations that integrate text, speech, and visuals, sophisticated image generation that understands context, and image question answering (IQA) capabilities. The advent of MLLMs marks a significant step forward in artificial intelligence, bridging multiple modalities to provide more interactive, context-aware outputs.

In this task, students are required to explore and understand the fundamentals of MLLMs, delving into their architecture, training processes, and practical applications. LLaVA (Large Language and Vision Assistant) serves as a practical example for this exploration. As outlined on their project page (https://llava-vl.github.io), LLaVA exemplifies how multimodal systems can be fine-tuned to enhance visual-language understanding, particularly in fields like image captioning, visual dialogue, and question answering.

7.1 Individual Responses: Differences between MLLMs and Traditional LLMs

7.1.1 Response by Niu Mu

Architectural and Processing Differences

The fundamental distinction between MLLMs and traditional LLMs lies in their input processing capabilities. Traditional LLMs like GPT-3 and BERT are designed exclusively for text-based inputs, processing sequential token embeddings through transformer architectures and excelling at natural language tasks within the textual domain.

MLLMs integrate multiple modalities—vision, text, and audio—into a unified framework. The architecture consists of: (1) modality-specific encoders (e.g., Vision Transformer for images), (2) cross-modal alignment mechanisms, and (3) language model decoders. This enables understanding and generation across different modalities simultaneously.

Training and Data Requirements

Traditional LLMs use self-supervised learning on large text corpora with objectives like masked language modeling or next-token prediction. Training data consists entirely of textual sequences, limiting understanding to linguistic patterns.

MLLMs require complex multi-stage training: (1) pre-training on multimodal datasets for cross-modal representations, (2) instruction tuning with multimodal data, and (3) task-specific fine-tuning. This involves image-text pairs, video-text combinations, and cross-modal associations, significantly increasing computational requirements and data collection challenges.

Applications and Trade-offs

Traditional LLMs excel in text-centric applications like content generation, translation, and code completion but cannot process visual information, limiting real-world applicability.

MLLMs expand capabilities to visual question answering, image captioning, multimodal dialogue systems, and cross-modal content creation. However, this versatility comes with increased model complexity, computational overhead, and potential performance trade-offs in pure text tasks compared to specialized text-only models.

7.1.2 Response by Wu Zining

Architectural Evolution: Single to Multiple Modalities

The evolution from traditional LLMs to MLLMs represents a paradigm shift from single-modality to integrated multimodal understanding. Traditional LLMs operate within a constrained textual universe, processing only tokenized sequences and remaining fundamentally limited to symbolic language representation. Models like GPT series excel at linguistic patterns but cannot directly understand non-textual information.

MLLMs transcend this limitation by incorporating vision encoders, audio processors, and cross-modal components. A typical MLLM architecture features specialized encoders (e.g., Vision Transformer for images) that convert multimodal inputs into feature representations, followed by projection layers mapping these features into the language model's embedding space. This enables understanding of relationships between visual content and textual descriptions.

Processing and Training Differences

Traditional LLMs employ straightforward tokenization and next-token prediction, learning language patterns through self-supervised objectives on text corpora. Their training data is primarily textual from web crawls, books, and articles.

MLLMs require heterogeneous representation learning, processing different modalities through specialized encoders before fusion in shared representational space. Training involves multiple stages: (1) vision-language pre-training on image-text datasets, (2) multimodal instruction tuning, and (3) task-specific fine-tuning. This complexity significantly increases computational costs and requires specialized expertise in handling multiple data modalities and ensuring proper cross-modal alignment.

7.1.3 Response by Zhao Jinqiu

Processing Philosophy: Unimodal vs. Multimodal Intelligence

The distinction between traditional LLMs and MLLMs can be understood through cognitive processing philosophy. Traditional LLMs embody a text-centric paradigm where all capabilities are mediated through linguistic representations. Models like GPT-3.5 and Claude process information exclusively through tokenized text sequences, excelling in language comprehension within textual confines.

MLLMs represent a shift toward embodied intelligence, integrating multiple sensory modalities similar to human cognitive processes. Models like GPT-4V and LLaVA simultaneously process text, images, and audio, enabling richer contextual understanding and more natural human-computer interaction.

Technical Architecture: Sequential vs. Parallel Processing

Traditional LLMs employ sequential processing where tokens flow through self-attention layers and feed-forward networks. The linear pipeline follows: tokenization $\rightarrow$ embedding $\rightarrow$ transformer layers $\rightarrow$ output generation, optimized for discrete token sequences.

MLLMs implement parallel modality processing requiring sophisticated fusion mechanisms: (1) modality-specific encoders, (2) cross-modal attention mechanisms, (3) alignment modules mapping features into shared representational space, and (4) unified decoders generating multimodal outputs. This enables simultaneous cross-referencing of multiple information sources.

Learning and Application Differences

Traditional LLMs use well-established single-task optimization with language modeling objectives like next-token prediction. Evaluation focuses on language-specific metrics such as perplexity and BLEU scores.

MLLMs face complex multi-objective optimization, balancing learning across modalities while avoiding negative interference. Training requires curriculum learning and careful loss function balancing. Applications expand from text-centric tasks (summarization, code generation) to cross-modal capabilities (visual question answering, image captioning, multimodal dialogue systems), enabling accessibility tools, educational systems, and scientific analysis requiring integration of multiple information sources.

7.2 Solution Implementation

[Placeholder for solution implementation content]

7.3 Code Documentation

[Placeholder for code documentation content]

8 Results and Discussion

8.1 System Performance Results

[Placeholder for system performance results content]

8.2 Task Achievement Summary

[Placeholder for task achievement summary content]

8.3 Lessons Learned

[Placeholder for lessons learned content]

9 Individual Contributions

9.1 Member 1: Niu Mu

[Placeholder for Niu Mu's contributions]

9.2 Member 2: Wu Zining (A0294373W)

[Placeholder for Wu Zining's contributions]

9.3 Member 3: Zhao Jinqiu

[Placeholder for Zhao Jinqiu's contributions]

10 Conclusion

10.1 Project Objectives Achievement

[Placeholder for project objectives achievement content]

10.2 Future Work

[Placeholder for future work content]

11 References

References

[1] A. Vaswani et al., "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] C. Raffel et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[3] M. Lewis et al., "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[5] Y. Liu et al., "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[6] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[7] H. Touvron et al., "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

12 Appendix

12.1 Code Documentation

[Placeholder for code documentation]

12.2 Configuration Files

[Placeholder for configuration files]

12.3 User Manual

[Placeholder for user manual]