

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 25, 2025

Contents

1	Abstract	4
2	Introduction	4
2.1	Background	4
2.2	Project Objectives	4
2.3	Project Scope	4
3	Task 1: Development Environment Setup	4
3.1	Python Environment Configuration	4
3.2	TensorFlow Platform Familiarization	5
3.3	Development Tools and Libraries	5
4	Task 2: LLM Platform Development	5
4.1	Open Source LLM Exploration	5
4.1.1	Literature Review on LLM Categories	5
4.1.2	Comparative Analysis	7
4.1.3	Recent Trends and Future Directions	8
4.2	Local LLM Platform Implementation	8
4.3	Comparison of Different Pretrained Models	11
5	Task 3: Dialogue System Development	11
5.1	Dialogue System Architecture	11
5.2	Natural Language Processing Components	12
5.3	Multi-modal Communication	12
5.4	Dialogue Management	12
6	Task 4: System Integration and Testing	12
6.1	Component Integration	12
6.2	System Testing	12
6.3	Performance Optimization	12

7 Task 5: LLM Evaluation	12
7.1 Evaluation Metrics	12
7.2 Evaluation Framework Implementation	12
7.3 Performance Analysis	13
7.4 Solution Implementation	13
7.5 Code Documentation	13
8 Results and Discussion	13
8.1 System Performance Results	13
8.2 Task Achievement Summary	13
8.3 Lessons Learned	13
9 Individual Contributions	13
9.1 Member 1: Niu Mu	13
9.2 Member 2: Wu Zining (A0294373W)	13
9.3 Member 3: Zhao Jinqiu	14
10 Conclusion	14
10.1 Project Objectives Achievement	14
10.2 Future Work	14
11 References	14
12 Appendix	14
12.1 Code Documentation	14
12.2 Configuration Files	14
12.3 User Manual	14

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

2.3 Project Scope

[Placeholder for project scope content]

3 Task 1: Development Environment Setup

3.1 Python Environment Configuration

[Placeholder for Python environment setup content]

3.2 TensorFlow Platform Familiarization

[Placeholder for TensorFlow familiarization content]

3.3 Development Tools and Libraries

[Placeholder for development tools content]

4 Task 2: LLM Platform Development

4.1 Open Source LLM Exploration

4.1.1 Literature Review on LLM Categories

Large Language Models (LLMs) can be categorized into three main architectural paradigms based on their transformer configurations: encoder-decoder, encoder-only, and decoder-only models. Each architecture has distinct characteristics, strengths, and applications in natural language processing tasks.

Encoder-Decoder Models Encoder-decoder models employ a dual-transformer architecture where the encoder processes input sequences to generate contextual representations, while the decoder generates output sequences based on these representations. This architecture is particularly effective for sequence-to-sequence tasks such as machine translation, text summarization, and question answering.

Key Characteristics:

- Bidirectional attention in the encoder captures context from both directions
- Unidirectional attention in the decoder enables autoregressive generation
- Explicit separation between understanding (encoding) and generation (decoding) phases

Representative Models:

- **T5 (Text-to-Text Transfer Transformer)**: Treats all NLP tasks as text-to-text problems, achieving state-of-the-art performance across diverse benchmarks
- **BART (Bidirectional and Auto-Regressive Transformers)**: Combines bidirectional encoder with autoregressive decoder, excelling in text generation and denoising tasks
- **mT5**: Multilingual extension of T5 supporting over 100 languages

Encoder-Only Models Encoder-only models utilize only the encoder component of the transformer architecture, employing bidirectional attention to process input sequences. These models excel at understanding and representation learning tasks rather than text generation.

Key Characteristics:

- Bidirectional attention mechanism captures full context
- Optimized for understanding tasks rather than generation
- Require task-specific heads for downstream applications
- Typically used for classification, named entity recognition, and feature extraction

Representative Models:

- **BERT (Bidirectional Encoder Representations from Transformers):** Pioneer in bidirectional language modeling, achieving breakthrough performance in NLU tasks
- **RoBERTa:** Optimized version of BERT with improved training procedures and longer training duration
- **DeBERTa:** Enhanced BERT with disentangled attention mechanism and enhanced mask decoder
- **ELECTRA:** More efficient pre-training using replaced token detection instead of masked language modeling

Decoder-Only Models Decoder-only models rely exclusively on the decoder component with causal (unidirectional) attention, making them highly effective for autoregressive text generation tasks. This architecture has become the dominant paradigm for modern conversational AI systems.

Key Characteristics:

- Unidirectional attention prevents information leakage during training
- Optimized for text generation and completion tasks
- Can be fine-tuned for various downstream tasks through instruction following
- Generally require larger model sizes to achieve competitive performance

Representative Models:

- **GPT (Generative Pre-trained Transformer) Series:** GPT-1, GPT-2, GPT-3, and GPT-4 represent the evolution of decoder-only models with increasing scale and capabilities
- **LLaMA (Large Language Model Meta AI):** Efficient decoder-only model achieving competitive performance with smaller parameter counts
- **GLM (General Language Model):** Chinese-developed model combining autoregressive and autoencoding approaches
- **PaLM (Pathways Language Model):** Google's large-scale decoder-only model with 540B parameters

4.1.2 Comparative Analysis

Table 1: Comparison of LLM Architecture Types

Aspect	Encoder-Decoder	Encoder-Only	Decoder-Only
Primary Use	Seq2Seq tasks	Understanding tasks	Generation tasks
Attention Mechanism	Bidirectional + Causal	Bidirectional	Causal
Training Efficiency	Medium	High	Low (for large models)
Inference Speed	Medium	Fast	Slow (for large models)
Task Flexibility	High	Medium	High
Parameter Efficiency	Medium	High	Low
Representative Models	T5, BART	BERT, RoBERTa	GPT, LLaMA

Performance Trade-offs:

- **Encoder-Decoder Models:** Offer balanced performance for both understanding and generation tasks, but require more computational resources due to dual architecture
- **Encoder-Only Models:** Excel at understanding tasks with high efficiency, but limited generation capabilities
- **Decoder-Only Models:** Superior generation quality and conversational abilities, but require significant computational resources for training and inference

Application Scenarios:

- **Encoder-Decoder:** Machine translation, text summarization, question answering systems
- **Encoder-Only:** Sentiment analysis, named entity recognition, text classification, feature extraction

- **Decoder-Only:** Conversational AI, creative writing, code generation, instruction following

4.1.3 Recent Trends and Future Directions

Recent developments in LLM architectures show several emerging trends:

- **Scale Integration:** Modern models increasingly combine multiple architectural paradigms (e.g., encoder-decoder with decoder-only components)
- **Efficiency Optimization:** Focus on reducing computational requirements while maintaining performance through techniques like knowledge distillation and model compression
- **Multimodal Integration:** Extension of decoder-only models to handle multiple modalities (text, vision, audio)
- **Specialized Architectures:** Development of task-specific architectures optimized for particular domains or applications

This comprehensive understanding of different LLM architectures provides the foundation for selecting appropriate models for specific applications in dialogue systems and local LLM platforms.

4.2 Local LLM Platform Implementation

This subsection details the design and implementation of our local Large Language Model (LLM) platform developed in Task 2. The platform enables fully offline, multi-turn dialogue with an optimized quantized Llama 3.2 3B Instruct model using the llama-cpp-python backend, supporting streaming token-level generation, context management, and persistent conversation storage.

System Architecture

The platform is organized into two core layers: (1) **LLMPlatform** (model runtime + generation control) and (2) **DialogueSystem** (session orchestration + persistence).

- **Model Runtime:** Wraps the quantized GGUF model (Llama-3.2-3B-Instruct-Q4_K_M.gguf) providing unified load / generate / stream interfaces.

- **Generation Engine:** Applies sampling controls (temperature, top-p, top-k, repeat penalty) and stop-sequence management for safe termination.
- **Context Builder:** Dynamically composes an instruction-style prompt with system, user, assistant roles; trims history (sliding window) to remain within `n_ctx`.
- **Streaming Layer:** Exposes a Python generator yielding incremental tokens for real-time UX.
- **Persistence Module:** Serializes conversations to JSON with timestamps and metadata (conversation ID).
- **Configuration Loader:** Merges external `config.json` with safe defaults; separates model, generation, dialogue, and hardware domains.

Configuration and Parameterization

Four configuration groups enable reproducibility and hardware-aware tuning (excerpt from `config.json`).

Table 2: Configuration Groups and Key Parameters

Group	Key Fields	Purpose
Model	<code>model_path</code> , <code>n_gpu_layers</code> , <code>n_ctx</code> , <code>n_threads</code>	Load quantized model; balance context length vs memory footprint.
Generation	<code>temperature</code> , <code>top_p</code> , <code>top_k</code> , <code>repeat_penalty</code> , <code>max_tokens</code>	Control diversity, prevent repetition, limit response budget.
Dialogue	<code>max_history</code> , <code>system_prompt</code> , <code>streaming</code> , <code>save_conversations</code>	Maintain conversational coherence and UX features.
Hardware	<code>gpu_enabled</code> , <code>max_gpu_layers</code> , <code>memory_fraction</code>	Allocate GPU layers and avoid memory over-subscription.

Inference Workflow

1. **Initialization:** Validate model presence; instantiate Llama with GPU offloading (`n_gpu_layers=35`).
2. **Input Capture:** User utterance appended to in-memory history (role-tagged JSON objects).

3. **Context Assembly:** Select last k exchanges ($j=6$) + system prompt into structured token template.
4. **Generation:** Call synchronous or streaming API; apply sampling constraints and stop tokens.
5. **Streaming (Optional):** UI prints incremental tokens; latency perceived as reduced.
6. **Post-processing:** Trim whitespace; append assistant reply to history.
7. **Persistence:** If enabled, serialize pair into conversation JSON (timestamped).

Streaming Mechanism

The streaming interface wraps the backend iterator, yielding *delta* fragments; the UI layer concatenates them to form the final assistant turn. This improves responsiveness for longer generations and mirrors modern production chat UX. A termination check monitors `finish_reason` in the final chunk.

Data Persistence and Reproducibility

Conversations are stored under `conversations/` using ISO8601 timestamps for auditing. Each file aggregates ordered message tuples preserving role, content, and creation time, enabling later evaluation or fine-tuning dataset curation.

Performance Considerations

We adopt Q4_K_M quantization to balance memory (`3GBGPUUsage`) and quality for a consumer-grade 16GB GPU. Streaming reduces perceived latency; selective history truncation prevents context overflow. CPU threads (`n_threads=8`) parallelize token probability computation for non-offloaded layers.

Reliability and Error Handling

Model load failures (missing file / incompatible quantization) are trapped with fallback messaging. Generation exceptions during streaming yield an inline error token without aborting the application. Conversation save errors are gracefully warned (non-fatal).

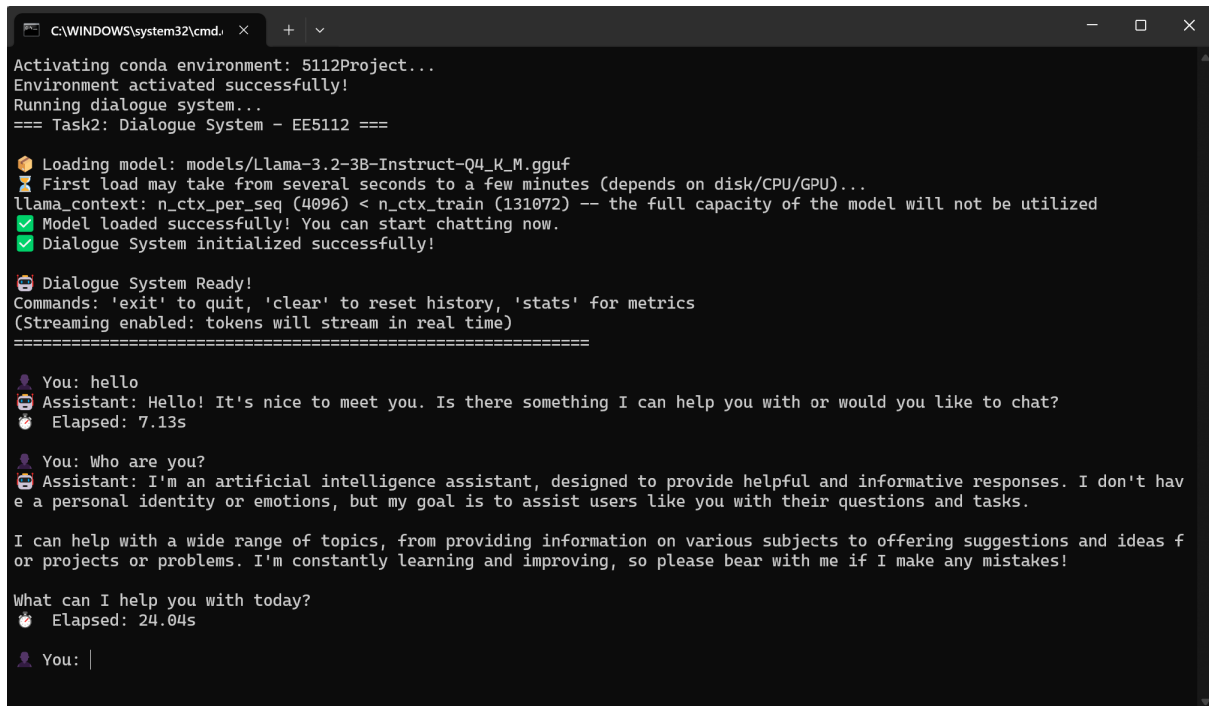
Strengths and Limitations

- **Strengths:** Offline privacy; modular layering; streaming UX; clean JSON audit trail; hardware-aware configuration.

- **Limitations:** Single-model runtime (no dynamic model pool); absence of advanced memory (vector retrieval); limited evaluation hooks in current phase.
- **Future Work:** Add retrieval-augmented generation, multi-model routing, automated quality metrics, and GUI integration.

User Interaction Example

Figure 1 shows a real dialogue example captured from the streaming session.



```

C:\WINDOWS\system32\cmd.exe
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 have 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 for 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: Example streaming dialogue with local Llama 3.2 3B model.

Overall, the implementation delivers a privacy-preserving, extensible inference substrate suitable for subsequent integration with higher-level dialogue management and evaluation modules in later tasks.

4.3 Comparison of Different Pretrained Models

5 Task 3: Dialogue System Development

5.1 Dialogue System Architecture

[Placeholder for dialogue system architecture content]

5.2 Natural Language Processing Components

[Placeholder for NLP components content]

5.3 Multi-modal Communication

[Placeholder for multi-modal communication content]

5.4 Dialogue Management

[Placeholder for dialogue management content]

6 Task 4: System Integration and Testing

6.1 Component Integration

[Placeholder for component integration content]

6.2 System Testing

[Placeholder for system testing content]

6.3 Performance Optimization

[Placeholder for performance optimization content]

7 Task 5: LLM Evaluation

7.1 Evaluation Metrics

[Placeholder for evaluation metrics content]

7.2 Evaluation Framework Implementation

[Placeholder for evaluation framework implementation content]

7.3 Performance Analysis

[Placeholder for performance analysis content]

7.4 Solution Implementation

[Placeholder for solution implementation content]

7.5 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

[Placeholder for references - Use proper citation format]

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]