WiDS Project Report: Foundations to Agentic AI

Neural Networks, Transformers, LLMs, and Multi-Agent Systems

Jyotirya Agrawal
Electrical Engineering, IIT Bombay

January 2026

Abstract

This report documents the work completed during the WiDS AI program, spanning foundational deep learning concepts to modern agentic AI systems. Over five weeks, the project progressed from neural network fundamentals and PyTorch implementations to transformer architectures, large language model (LLM) deployment, retrieval-augmented generation, and multi-agent systems using LangGraph and Google ADK. Practical assignments reinforced theoretical understanding through hands-on implementation using Hugging Face, LangChain, LangGraph, and open-source LLM tooling. The culmination of this work is a set of functional agentic AI systems demonstrating routing, collaboration, and stateful reasoning.

Contents

1 Introduction

The rapid evolution of artificial intelligence has been driven by advances in neural networks, transformer architectures, and large language models. The WiDS project was structured to provide a strong conceptual foundation while simultaneously emphasizing practical implementation. The program followed a progressive learning path: beginning with classical neural networks, advancing through transformers and attention mechanisms, and culminating in modern agentic AI systems capable of reasoning, routing, and collaboration.

This report provides a detailed account of the learning objectives, implementations, assignments, and outcomes achieved during the five-week program.

2 Weeks 1–2: Foundations of Deep Learning

2.1 Neural Networks: Motivation and Intuition

Neural networks form the backbone of modern deep learning systems. Inspired by biological neurons, they consist of layers of weighted connections followed by nonlinear activation functions. Through training on data, these networks learn representations that enable tasks such as image classification, speech recognition, and language modeling.

Conceptual understanding was developed using visual and mathematical explanations, focusing on how networks approximate complex functions through composition of simple operations.

2.2 Learning Mechanism: Backpropagation

A key focus was understanding how neural networks learn. Training involves:

- Forward propagation to compute predictions
- Loss computation to measure error
- Backpropagation to compute gradients
- Gradient-based optimization to update weights

This demystified learning as a sequence of differentiable mathematical operations rather than a black-box process.

2.3 PyTorch Implementations

Practical deep learning skills were developed using PyTorch:

- **Multilayer Perceptron (MLP)** for basic supervised learning
- **Convolutional Neural Networks (CNNs)** for image classification
- **Recurrent Neural Networks (LSTM)** for sequential data

These implementations emphasized training loops, loss functions, optimizers, and evaluation workflows.

3 Transformers and Attention Mechanisms

3.1 Motivation for Transformers

Traditional RNN-based architectures suffer from sequential bottlenecks and vanishing gradients. Transformers address these limitations by replacing recurrence with attention mechanisms, allowing parallel computation and long-range dependency modeling.

3.2 Self-Attention

Self-attention enables a model to weigh different parts of the input when computing representations. Each token attends to every other token, producing context-aware embeddings. This mechanism is central to modern LLMs such as GPT.

A basic self-attention layer was implemented to understand query, key, and value projections mathematically and computationally.

3.3 Encoder-Decoder Architecture

The encoder-decoder paradigm was studied to understand sequence-to-sequence modeling. Emphasis was placed on decoder-only architectures (e.g., GPT), which power most modern conversational AI systems.

4 Week 3: Hugging Face and LLM Deployment

4.1 Hugging Face Pipelines

Hugging Face Transformers were used to rapidly prototype NLP tasks using pipelines:

- Text summarization
- Text generation
- Sentiment analysis

These abstractions enabled focus on task logic rather than low-level model details.

4.2 Running LLMs Locally

Open-source LLM deployment was explored using:

- Ollama
- Groq-backed inference
- Models such as LLaMA, Mistral, and Phi

This provided insight into performance, prompt engineering, and system-level constraints.

4.3 Retrieval-Augmented Generation

RAG pipelines were studied to enhance factual accuracy by combining LLMs with external knowledge sources, laying groundwork for agent-based systems.

5 Assignment 1: Hugging Face Pipelines

5.1 Summarization Pipeline

A Python script was implemented using `pipeline("summarization")` to compress a long paragraph while controlling output length via `min_length` and `max_length`. The script compared original and summarized text lengths to evaluate compression effectiveness.

5.2 Text Generation

A text generation pipeline was built to generate multiple continuations from a single prompt. This demonstrated sampling variability and creativity in generative models.

5.3 Sentiment Analysis

A batch sentiment analysis pipeline processed multiple reviews simultaneously, outputting predicted labels and confidence scores.

These tasks reinforced understanding of pipeline abstraction and batch inference.

6 Week 4: LangGraph and Agentic AI

6.1 Introduction to Agentic AI

Agentic AI extends LLMs with autonomy, state, and decision-making. Agents can plan, route tasks, and collaborate with other agents rather than responding statically.

6.2 LangGraph Framework

LangGraph was used to construct explicit computation graphs with:

- Stateful execution
- Conditional routing
- Multi-agent coordination

This framework provided fine-grained control over agent behavior and execution flow.

7 Assignment 2: Agent-Based Systems

7.1 Single-Agent QA System

A LangGraph application was built with a single Hugging Face-backed LLM node. The system maintained conversational state and was tested across coding, mathematics, and general knowledge queries.

7.2 Two-Agent Pipeline

A two-agent architecture was implemented:

- Question Analyzer: rewrites and clarifies user input
- Answer Generator: produces the final response

This separation improved answer quality and modularity.

7.3 Routing-Based Multi-Agent System

A router node was added to dynamically select between:

- Python-specialized agent
- General-purpose Q&A agent

This demonstrated conditional execution and specialization.

8 Week 5: Google ADK and Agent-to-Agent Communication

8.1 Google Agent Development Kit

Google ADK concepts were studied to understand standardized agent construction, lifecycle management, and scalable deployment.

8.2 Agent-to-Agent Communication

Agent-to-Agent (A2A) communication enables agents to delegate tasks, share intermediate results, and collaborate effectively. This represents a shift from monolithic models to distributed intelligent systems.

9 Project Repository

All implementations, assignments, and experiments are available in the following GitHub repository:

https://github.com/Jyotirya/Agentic_AI.git

10 Conclusion

The WiDS project provided a comprehensive journey from foundational neural networks to modern agentic AI systems. By combining theoretical understanding with hands-on implementation, the project developed practical skills in deep learning, LLM deployment, and multi-agent orchestration. The progression from simple models to collaborative agents highlights the evolving paradigm of AI systems and prepares the groundwork for advanced research and real-world applications.