PROJECT REPORT: Deep-100 Based Autonomous Code Analysis and Repair Agent (Code Doctor)

1. Introduction and Project Objective

The objective of this project is to transform Large Language Models (LLMs) from passive structures that merely generate text into **autonomous agents** capable of executing code, correcting errors, and retrieving information from external sources (RAG). The **Deep-100** model (based on DeepSeek/Qwen2.5) was utilized as the base model, and the **ReAct (Reasoning + Acting)** architecture was implemented.

The goal is to develop a system that analyzes faulty or incomplete user code requests, executes them in a secure virtual environment (Sandbox), and fixes errors through **Self-Correction**.

2. Architectural Design and Methodology

The project is built upon a hybrid architecture consisting of three main components:

A. Orchestration (The Brain)

Model: The Deep-100 model, optimized using the Unsloth library, was employed.
Loop: The system operates within a **Thought -> Action -> Observation** loop.
Memory: To prevent the model from losing context, conversation history is managed dynamically using a "Sliding Window" method.

B. Tools

Two fundamental tools were developed to enable the agent to interact with the external world:

`python_executor_tool`: Executes generated Python code within an isolated `subprocess`. For security, a `timeout=5s` constraint was enforced, and regex filters were added to block dangerous libraries (e.g., `sys`, `os`, `input`).

`rag_retrieval_tool`: Documents were vectorized using `SentenceTransformers (all-MiniLM-L6-v2)`, and semantic search was integrated to answer theoretical questions.

C. Security Layer

Before direct execution, the code generated by the model passes through a **"Smart Regex"** filter. Even if the Markdown formatting is missing, code blocks are detected; however, blocks requiring interaction such as `sys.argv` or `input()` are automatically blocked.

3. Experimental Scenarios and Analysis (Trace Outputs)

To test the success and limitations of the project, the following edge-case scenarios were applied:

Scenario 1: Training Bias and Constraint Management

Situation: When asked to write a simple Fibonacci code, the model persistently attempted to use `sys.argv` (command-line arguments) due to habits acquired from its training data (scripts).
Intervention: System security blocked this code. The behavior was corrected via negative prompting by the user ("Do not use sys, hardcode values").
Result: The model understood the constraints, simplified the code, and executed it successfully.

(Insert screenshot of the log showing the sys error and subsequent correction here)

Scenario 2: Performance Optimization and Time-Out Test

Situation: The model was asked to write a "Recursive Fibonacci 100" code.
Observation: The model selected the Naive Recursion method without using `lru_cache` (Memoization). Since the complexity of this algorithm is $O(2^n)$, the process could not complete in a reasonable time.
Result: The 5-second timeout mechanism in the python_executor was triggered, protecting the system from an infinite loop. This demonstrated that the agent's optimization capability requires guidance.

(Insert screenshot of the "Timeout" error here)

Scenario 3: Stateless Execution Challenge

Situation: The agent defined a function in one step and attempted to call that function in the subsequent step.
Error: A `NameError` was received because the Python execution environment is reset (Stateless) at every step.
Solution: The agent was instructed to use the previous result (hardcoded value), overcoming the issue.

4. Challenges and Solutions

5. Conclusion

The **Code Doctor** developed in this project has gone beyond being a simple code generator. Its ability to execute written code (**Action**), read and fix errors when they occur (**Self-Correction**), and operate within security boundaries is a successful implementation of the modern ReAct Agent architecture. In particular, thanks to RAG integration, a hybrid structure possessing both theoretical knowledge and practical application capabilities has been achieved.