Autonomous Learning Agent

1. Overview

The **Autonomous Learning Agent** is an AI-driven learning system built with **LangGraph**, **LangChain**, **Streamlit**, and **LangSmith**. It guides a learner through structured checkpoints, dynamically gathers learning context, evaluates understanding, applies the Feynman technique on failure, and tracks progress across sessions.

The system is designed to be: - **Interactive** (Streamlit frontend) - **Adaptive** (retry + Feynman teaching loop) - **Observable** (LangSmith tracing) - **State-driven** (LangGraph workflow)

---

2. Key Features

- 📚**Checkpoint-based learning** (topic, objectives, success criteria)
- 🔍**Automatic context gathering** (web search or user notes)
- **Assessment generation** (LLM-generated questions)
- 🔗**Evaluation & scoring** per checkpoint
- 🔦**Retry on failure** using the **Feynman Technique**
- 🎱**Manual progression** to the next checkpoint on pass
- 🕐**Automatic reset** to the first checkpoint after completion
- 📄**PDF upload support** for learner notes
- 🧹**Persistent progress tracking** (JSON-based)
- 🧪**LangSmith tracing** for every node, route, and LLM call

---

3. High-Level Architecture

```
User (Browser)
   │
   ▼
Streamlit UI (app_ex.py)
   │
   ▼
LangGraph Workflow (graph_workflow.py)
   │
   ├── start_checkpoint
   ├── gather_context
   ├── evalution_context
   ├── process_context
   ├── question_generation
   ├── evaluate_answer
```

```
    ├── detect_gap
    └── feynman_teaching
    │
    ▼
LangChain + LLMs (NVIDIA / DeepSeek / Mistral)
    │
    ▼
LangSmith (Tracing & Observability)
```

---

4. Project Structure

```
Learning_Agent_Ai/
│
├── app_ex.py                # Streamlit frontend (main entry point)
├── graph_workflow.py        # LangGraph workflow definition
├── nodes.py                 # All LangGraph node functions
├── routing.py               # Conditional routing logic
├── state.py                 # LearningState schema (TypedDict)
│
├── checkpoint_1.py           # Checkpoint definitions
├── checkpoint_class_1.py     # Checkpoint type
│
├── contextProcessor.py       # Text chunking & vector store logic
├── gathercontext.py          # Web search context gathering
│
├── llm_model.py              # LLM configuration (ChatNVIDIA)
├── promts.py                 # Prompt templates & parsers
├── structureOut.py           # Pydantic output schemas
│
├── ui_upload_view.py         # PDF upload UI
├── ui_pdf_loader.py          # PDF text extraction
├── ui_progress_store.py      # Progress persistence (JSON)
│
├── progress.json             # Saved learner progress
└── README.md                 # (Optional) external documentation
```

---

5. Learning Workflow (End-to-End)

Step 1: Start Checkpoint

- User clicks **Start / Continue Learning**

- Initial `LearningState` is created

Step 2: Context Gathering

- Uses **user notes** if provided
- Otherwise performs **web search** based on topic & objectives

Step 3: Context Validation

- LLM evaluates relevance of gathered context
- Retries gathering if relevance is low (up to max iterations)

Step 4: Question Generation

- Context is chunked
- Vector similarity search selects relevant chunks
- LLM generates **3 assessment questions**

Step 5: Answer Evaluation

- Learner submits answers
- LLM scores each answer (0–100)

Step 6: Routing

- **Pass (≥70%)** → checkpoint completed
- **Fail (<70%)** → gap detection → Feynman teaching → retry

Step 7: Completion Handling

- On last checkpoint completion:
- Progress resets automatically
- Learning restarts from **first checkpoint**

---

6. State Management (`LearningState`)

Key fields used across the workflow:

- `checkpoint`: Current checkpoint metadata
- `user_Notes`: Learner notes (text or PDF)
- `gether_context`: Gathered learning content
- `questions`: Generated assessment questions
- `answers`: Learner answers
- `score_percentage`: Per-question scores
- `passed`: Boolean pass/fail flag
- `gaps` / `gaps_list`: Detected learning gaps
- `feynman_explanation`: Simplified explanation

Important Rule:

> Any key accessed by a node must be initialized in the initial state.

---

7. Frontend Behavior (Streamlit)

- Displays current checkpoint details
- Allows PDF upload and note entry
- Shows generated questions dynamically
- After submission:
- Displays score and result
- **PASS:** shows "Go to Next Checkpoint" button
- **FAIL:** shows Feynman explanation + "Retry Same Checkpoint" button
- Automatically resets to checkpoint 1 after final completion

---

8. LangSmith Observability

LangSmith is enabled to trace: - Every LangGraph node - Routing decisions - LLM prompts and responses - Streamlit-triggered workflow runs

Environment Variables

```
LANGCHAIN_TRACING_V2=true
LANGCHAIN_API_KEY=<your_key>
LANGCHAIN_PROJECT=Autonomous-Learning-Agent
```

This provides full visibility into: - Errors - Latency - Prompt quality - State transitions

---

9. How to Run the Project

Install dependencies

```
pip install streamlit langgraph langchain langsmith
```

Run the app

```
streamlit run Learning_Agent_Ai/app_ex.py
```

---

10. Future Enhancements

- 📊Analytics dashboard (strengths & weaknesses)
- Multi-user authentication
- Adaptive difficulty per learner
- 📄Learning report export (PDF)
- 🧪Unit tests for nodes & routing

---

11. Summary

The Autonomous Learning Agent is a **production-ready, observable, adaptive learning system**. It demonstrates how LangGraph, LLMs, and Streamlit can be combined to create intelligent tutoring workflows with full transparency and control.

End of Documentation