AI Python Learning Companion

Course: ITAI2376 - AI Agent Systems

Team: 1_BinteZahra_Waseem

Instructor: Sitaram Ayyagari

Project Overview

Students often struggle with self-paced learning due to lack of personalized guidance. Our agent will act as an adaptive learning companion that provides custom explanations, exercises, and feedback.

Interactive Learning Companion (Programming Tutor)

Agent Design Architecture includes Input Processing, Memory System, Reasoning Module (ReAct + reflection), and Output Generation.

Tools Selected

1. Web Search API for knowledge retrieval
2. Python execution tool for validating code solutions

Development Plan 1: Architecture + prototype 2: Tool integration 3: RL feedback loop 4: Testing + documentation

Evaluation Strategy Measure correctness, personalization accuracy, and user improvement over sessions.

Resource Requirements Google Colab, Python libraries, API keys (search + vector DB).

Risk Assessment Tool failures, high token usage, prompt inconsistencies. Mitigation: fallback rules, caching, iterative refinement.

Setup – Install & Configure Gemini

```
!pip install -U google-generativeai
```

```
Requirement already satisfied: google-generativeai in /usr/local/lib/python3.12/dist-packages (0.8.5)
Requirement already satisfied: google-ai-generativelanguage==0.6.15 in /usr/local/lib/python3.12/dist-packages (from google-gene
Requirement already satisfied: google-api-core in /usr/local/lib/python3.12/dist-packages (from google-generativeai) (2.28.1)
Requirement already satisfied: google-api-python-client in /usr/local/lib/python3.12/dist-packages (from google-generativeai) (2
Requirement already satisfied: google-auth>=2.15.0 in /usr/local/lib/python3.12/dist-packages (from google-generativeai) (2.43.0
Requirement already satisfied: protobuf in /usr/local/lib/python3.12/dist-packages (from google-generativeai) (5.29.5)
Requirement already satisfied: pydantic in /usr/local/lib/python3.12/dist-packages (from google-generativeai) (2.12.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from google-generativeai) (4.67.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.12/dist-packages (from google-generativeai) (4.15.0)
Requirement already satisfied: proto-plus<2.0.0dev,>=1.22.3 in /usr/local/lib/python3.12/dist-packages (from google-ai-generativ
Requirement already satisfied: googleapis-common-protos<2.0.0,>=1.56.2 in /usr/local/lib/python3.12/dist-packages (from google-a
Requirement already satisfied: requests<3.0.0,>=2.18.0 in /usr/local/lib/python3.12/dist-packages (from google-api-core->google-
Requirement already satisfied: cachetools<7.0,>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from google-auth>=2.15.0->goog
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from google-auth>=2.15.0->googl
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.12/dist-packages (from google-auth>=2.15.0->google-genera
Requirement already satisfied: httplib2<1.0.0,>=0.19.0 in /usr/local/lib/python3.12/dist-packages (from google-api-python-client
Requirement already satisfied: google-auth-httplib2<1.0.0,>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from google-api-py
Requirement already satisfied: uritemplate<5,>=3.0.1 in /usr/local/lib/python3.12/dist-packages (from google-api-python-client->
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydantic->google-generati
Requirement already satisfied: pydantic-core==2.41.4 in /usr/local/lib/python3.12/dist-packages (from pydantic->google-generativ
Requirement already satisfied: typing-inspection>=0.4.2 in /usr/local/lib/python3.12/dist-packages (from pydantic->google-genera
Requirement already satisfied: grpcio<2.0.0,>=1.33.2 in /usr/local/lib/python3.12/dist-packages (from google-api-core[grpc]!=2.0
Requirement already satisfied: grpcio-status<2.0.0,>=1.33.2 in /usr/local/lib/python3.12/dist-packages (from google-api-core[grp
Requirement already satisfied: pyparsing<4,>=3.0.4 in /usr/local/lib/python3.12/dist-packages (from httplib2<1.0.0,>=0.19.0->goo
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in /usr/local/lib/python3.12/dist-packages (from pyasn1-modules>=0.2.1->goog
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.18.0
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.18.0->google-api
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.18.0->goog
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.18.0->goog
```

```
import os
import json
import textwrap
import google.generativeai as genai
from dataclasses import dataclass, field
from typing import List, Dict, Any, Optional
```

```
import traceback
import math

```

```
import google.generativeai as genai
from google.colab import userdata

GEMINI_API_KEY = userdata.get('GEMINI_API_KEY')
genai.configure(api_key=GEMINI_API_KEY)

MODEL_NAME = "gemini-pro-latest"   # <<< CHANGE THIS LINE
llm = genai.GenerativeModel(MODEL_NAME)

# The redundant configuration lines below will be removed
# genai.configure(api_key=GEMINI_API_KEY)
# MODEL_NAME = "gemini-pro-latest"
# llm = genai.GenerativeModel(MODEL_NAME)
```

Memory & Learner Profile

This covers Memory System + part of RL policy state.

```
@dataclass
class InteractionRecord:
    user_input: str
    agent_answer: str
    correctness: Optional[bool] = None
    reward: float = 0.0
    difficulty: int = 1


@dataclass
class LearnerProfile:
    name: str = "Student"
    skill_level: int = 1  # 1 = beginner, 5 = advanced
    target_language: str = "python"
    history: List[InteractionRecord] = field(default_factory=list)

    def estimate_skill(self) -> float:
        if not self.history:
            return float(self.skill_level)
        # simple average of difficulty where reward > 0
        diffs = [
            h.difficulty for h in self.history
            if h.reward > 0
        ]
        return sum(diffs) / len(diffs) if diffs else float(self.skill_level)

```

Short-term / long-term memory:

```
class MemorySystem:
    def __init__(self):
        self.learner_profile = LearnerProfile()
        self.short_term_buffer: List[Dict[str, str]] = []  # last N turns
        self.max_buffer = 10

    def add_message(self, role: str, content: str):
        self.short_term_buffer.append({"role": role, "content": content})
        if len(self.short_term_buffer) > self.max_buffer:
            self.short_term_buffer.pop(0)

    def add_interaction(self, record: InteractionRecord):
        self.learner_profile.history.append(record)

    def summary(self) -> str:
        skill_est = self.learner_profile.estimate_skill()
        return textwrap.dedent(f"""
        Learner name: {self.learner_profile.name}
        Approx skill level (1-5): {skill_est:.1f}
        Target language: {self.learner_profile.target_language}
        Number of past interactions: {len(self.learner_profile.history)}
    """)

```

Tools: Code Execution & Web Search Stub

This satisfies Tool Integration (2 tools) with error handling.

```
class ToolError(Exception):
    pass


def run_python_code(code: str) -> str:
    """
    Very restricted Python executor for educational use.
    DO NOT use this for untrusted users in production.
    """
    try:
        # Restricted globals/locals
        allowed_builtins = {"range": range, "len": len, "print": print, "math": math}
        local_env: Dict[str, Any] = {}
        exec(code, {"__builtins__": allowed_builtins, "math": math}, local_env)
        # capture last expression-style value if present
        if "_result" in local_env:
            return f"Execution success. _result = {repr(local_env['_result'])}"
        return "Execution success. Variables: " + ", ".join(local_env.keys())
    except Exception as e:
        tb = traceback.format_exc(limit=2)
        raise ToolError(f"Python execution failed: {e}\n{tb}")


def web_search_stub(query: str) -> str:
    """
    Stub for web search. In final project you can integrate
    a real API (e.g., SerpAPI, Tavily, etc.).
    """
    # For now, just echo the query.
    return (
        "WEB_SEARCH_RESULT (stub): In the real system, this would call a "
        f"web search API with query: '{query}'."
    )

```

Tool registry:

```
TOOLS = {
    "run_python": {
        "fn": run_python_code,
        "description": "Execute small Python code snippets for checking answers or running examples. "
                       "Input: Python code string.",
    },
    "web_search": {
        "fn": web_search_stub,
        "description": "Look up external information on Python concepts, syntax, or error messages. "
                       "Input: search query string.",
    }
}

```

Safety & Input Validation

This covers Safety & Security requirements.

```
# Add a list of keywords that signify Python programming context
PYTHON_CONTEXT_KEYWORDS = [
    "python", "code", "list", "tuple", "dictionary", "set", "function",
    "class", "variable", "loop", "if", "else", "elif", "import", "module",
    "package", "string", "integer", "float", "boolean", "syntax", "error"
]

```

```
def is_safe_input(user_input: str) -> bool:
    text = user_input.lower()
    return not any(k in text for k in DISALLOWED_KEYWORDS)


def enforce_boundaries(user_input: str) -> Optional[str]:
    """
    Returns a message if the request is out of scope or unsafe,
    otherwise returns None.
    """
    print(f"[DEBUG] enforce_boundaries - User input: '{user_input}'")
    print(f"[DEBUG] enforce_boundaries - PYTHON_CONTEXT_KEYWORDS: {PYTHON_CONTEXT_KEYWORDS}")

    if not is_safe_input(user_input):
        return (
            "I’m sorry, but I can’t help with harmful or unsafe topics. "
            "If you’re struggling or in distress, please consider reaching "
            "out to a trusted person or professional support line."
        )

    # Scope limitation: Only Python / programming learning
    text = user_input.lower()
    context_check = any(k in text for k in PYTHON_CONTEXT_KEYWORDS)
    print(f"[DEBUG] enforce_boundaries - Context check result (any keyword in input): {context_check}")

    if not context_check:
        return (
            "I’m designed to help with learning Python programming. "
            "Could you rephrase your question in that context?"
        )
    return None
```

ReAct-Style Agent with RL-Style Feedback

We’ll ask Gemini to output a JSON describing whether to use a tool and then act accordingly. That gives you:

ReAct / planning-then-execution

Tool selection logic

Transparency in reasoning

System Prompt

```
SYSTEM_PROMPT = """
You are an Interactive Learning Companion that tutors students in Python.

You must:
- Adapt explanations to the learner's skill level.
- Provide step-by-step reasoning (chain-of-thought) INTERNALLY,
  but in the JSON you output, summarize your reasoning briefly.
- Use tools when needed:
  - run_python(code: str): execute simple Python to check answers or run examples.
  - web_search(query: str): look up concepts or error messages (currently a stub).

Decision pattern (ReAct-style):
1. Think step by step about what the student is asking.
2. Decide if you need a tool to be more accurate.
3. Either:
   - Call a tool, OR
   - Answer directly.

ALWAYS respond in STRICT JSON with the following schema:

{
  "thought": "short natural language summary of your reasoning",
  "action": "run_python" | "web_search" | "none",
  "action_input": "string input for the tool, or empty string if none",
  "tutor_reply": "your message to the learner BEFORE we apply feedback or rewards",
  "suggested_difficulty": 1-5
}

Constraints:
```

```
- If you are not sure about the answer, prefer using a tool.
- Keep tutor_reply friendly, concise, and focused on learning Python.
- suggested_difficulty: 1 = very easy, 5 = very challenging.
"""

```

Agent Class

```
class LearningAgent:
    def __init__(self, llm, memory: MemorySystem):
        self.llm = llm
        self.memory = memory

    def _call_llm_controller(self, user_input: str) -> Dict[str, Any]:
        context = self.memory.summary()

        prompt = textwrap.dedent(f"""
        SYSTEM INSTRUCTIONS:
        {SYSTEM_PROMPT}

        Learner context:
        {context}

        Recent conversation (last {len(self.memory.short_term_buffer)} turns):
        {self.memory.short_term_buffer}

        New question from learner:
        {user_input}
        """)

        # Single string prompt instead of list of dicts
        response = self.llm.generate_content(prompt)
        raw = response.text

        try:
            data = json.loads(raw)
        except json.JSONDecodeError:
            data = {
                "thought": "Failed to parse JSON; treating as direct answer.",
                "action": "none",
                "action_input": "",
                "tutor_reply": raw,
                "suggested_difficulty": 1,
            }
        return data

    def _run_tool(self, action: str, action_input: str) -> str:
        if action not in TOOLS:
            return f"Tool '{action}' not found."
        tool = TOOLS[action]
        try:
            output = tool["fn"](action_input)
            return f"Tool '{action}' output:\n{output}"
        except ToolError as e:
            return f"Tool '{action}' failed with error:\n{e}"
        except Exception as e:
            return f"Unexpected error in tool '{action}': {e}"

    def _compute_reward(self, correctness: Optional[bool],
                        user_rating: Optional[int]) -> float:
        """
        Simple RL-style reward:
        - base on correctness (if available) and subjective rating (1-5)
        """
        reward = 0.0
        if correctness is True:
            reward += 1.0
        elif correctness is False:
            reward -= 0.5

        if user_rating is not None:
            reward += (user_rating - 3) * 0.3  # center at 3

        return reward

```

```python
    def _update_policy(self, record: InteractionRecord):
        """
        Policy improvement: adjust learner's skill difficulty suggestion
        based on reward.
        """
        profile = self.memory.learner_profile
        profile.history.append(record)

        # very simple adjustment
        if record.reward > 0.5 and profile.skill_level < 5:
            profile.skill_level += 1
        elif record.reward < -0.5 and profile.skill_level > 1:
            profile.skill_level -= 1

    def handle_turn(
        self,
        user_input: str,
        correctness: Optional[bool] = None,
        user_rating: Optional[int] = None
    ) -> str:
        """
        Main loop for a single interaction.
        Returns the final answer to show the learner.
        """
        # Safety / boundaries
        boundary_msg = enforce_boundaries(user_input)
        if boundary_msg is not None:
            self.memory.add_message("user", user_input)
            self.memory.add_message("assistant", boundary_msg)
            return boundary_msg

        controller_output = self._call_llm_controller(user_input)
        thought = controller_output.get("thought", "")
        action = controller_output.get("action", "none")
        action_input = controller_output.get("action_input", "")
        tutor_reply = controller_output.get("tutor_reply", "")
        suggested_difficulty = int(controller_output.get("suggested_difficulty", 1))

        tool_observation = ""
        if action != "none":
            tool_observation = self._run_tool(action, action_input)

            # Reflection / second-pass answer including tool results
            reflection_prompt = textwrap.dedent(f"""
            You are a Python tutor. Explain clearly and kindly.

            You previously decided to use the tool '{action}' with input:

            {action_input}

            The tool returned:

            {tool_observation}

            Now, refine your explanation to the learner, using this result.
            Keep it concise and beginner-friendly if their skill level is low.
            """)

            refined = self.llm.generate_content(reflection_prompt)
            tutor_reply = refined.text


        # Ask for feedback outside of this function in UI;
        # here we just compute reward if given.
        reward = self._compute_reward(correctness, user_rating)
        record = InteractionRecord(
            user_input=user_input,
            agent_answer=tutor_reply,
            correctness=correctness,
            reward=reward,
            difficulty=suggested_difficulty,
        )
        self._update_policy(record)

        # Update short-term memory
        self.memory.add_message("user", user_input)
        self.memory.add_message("assistant", tutor_reply)

        # Transparency: include short note about tools used
        meta_note = ""
        if action != "none":
            meta_note = f"\n\n_(I used tool: **{action}** to check or enrich this answer.)_"

        return tutor_reply + meta_note

```

Simple Interactive Loop (For Demo / Video)

This you can show in your demo video and in screenshots.

```python
memory = MemorySystem()
agent = LearningAgent(llm, memory)

def chat_with_agent():
    print("Interactive Learning Companion – Python Tutor")
    print("Type 'quit' to stop.\n")

    while True:
        user_input = input("You: ")
        if user_input.lower().strip() in ["quit", "exit"]:
            break

        # For now, we don't auto-check correctness; you can extend this:
        answer = agent.handle_turn(user_input, correctness=None, user_rating=None)
        print("\nAgent:", answer)
        print("-" * 60)

chat_with_agent()
```

```
Interactive Learning Companion – Python Tutor
Type 'quit' to stop.

You: What is Python?
[DEBUG] enforce_boundaries - User input: 'What is Python?'
[DEBUG] enforce_boundaries - PYTHON_CONTEXT_KEYWORDS: ['python', 'code', 'list', 'tuple', 'dictionary', 'set', 'function', 'clas
[DEBUG] enforce_boundaries - Context check result (any keyword in input): True

Agent: ```json
{
  "thought": "The user is asking a very basic, high-level question about what Python is. As a beginner (skill level 1), they nee
  "action": "none",
  "action_input": "",
  "tutor_reply": "That's an excellent question to start with! \n\nPython is a popular programming language. Think of it as a way
  "suggested_difficulty": 1
}
```

You: How to print my name "Zahra" in python?
[DEBUG] enforce_boundaries - User input: 'How to print my name "Zahra" in python?'
[DEBUG] enforce_boundaries - PYTHON_CONTEXT_KEYWORDS: ['python', 'code', 'list', 'tuple', 'dictionary', 'set', 'function', 'clas
[DEBUG] enforce_boundaries - Context check result (any keyword in input): True

Agent: ```json
{
  "thought": "The user is asking for a very basic Python command: how to print a specific string. This is a classic 'hello world
  "action": "run_python",
  "action_input": "print(\"Zahra\")",
  "tutor_reply": "Great question! To print your name, you use Python's built-in `print()` function.\n\nHere's how you do it:\n``
  "suggested_difficulty": 1
}
```
------------------------------------------------------------
You: What is the difference between list and tuples in python?
[DEBUG] enforce_boundaries - User input: 'What is the difference between list and tuples in python?'
[DEBUG] enforce_boundaries - PYTHON_CONTEXT_KEYWORDS: ['python', 'code', 'list', 'tuple', 'dictionary', 'set', 'function', 'clas
[DEBUG] enforce_boundaries - Context check result (any keyword in input): True

Agent: ```json
{
```