

```

1: #!/usr/bin/env python
2:
3: import argparse
4: import json
5: import xml.etree.ElementTree as ET
6:
7: import requests
8:
9: URL_OLLAMA = "http://localhost:11434/api/chat"
10: URL_MCP_SERVER = "http://localhost:5000/mcp"
11:
12:
13: def get_system_prompt(tool_name, tool_desc, tool_input):
14:     SYSTEM_PROMPT = f"""You are an AI quiz assistant.
15: Please present a quiz to the user and evaluate their input as either correct or incorrect.
16: The correct answer to the quiz must be one of the following: dog, cat, horse, or giraffe.
17:
18: ## Message Format
19:
20: Your response must follow the XML format shown below.
21: Only two tags are allowed within the <response> tag: <message> and <call>.
22: The <message> tag is mandatory, while the <call> tag is optional.
23:
24: Follow this format strictly and do not add any extra lines.
25: Do not wrap the reply message in a Markdown code block.
26: If the user's answer is incorrect, provide additional hints and continue the quiz until the correct answer is given.
27:
28: ## Example Response
29:
30: Replace %TOOL_NAME%, %PARAM_NAME%, and %PARAM_VALUE% according to the MCP Tools section.
31:
32: <response>
33: <message>Correct! Cool answer!</message>
34: <call><name>TOOL_NAME</name><arguments><%PARAM_NAME%>%PARAM_VALUE%</%PARAM_NAME%></arguments></call>
35: </response>
36:
37: Sample <call> Tag
38:
39: <call><name>add</name><arguments><lhs>1</lhs><rhs>2</rhs></arguments></call>
40:
41: ## MCP Tools
42:
43: You can use the following MCP Tools.
44:
45: ### TOOL_NAME: {tool_name}
46:
47: Description:
48: {tool_desc}
49:
50: Input Schema:
51: {tool_input}
52: """
53:
54:     return SYSTEM_PROMPT
55:
56: def parse_args():
57:     parser = argparse.ArgumentParser()
58:     parser.add_argument("--model", default="gemma3:12b")
59:     args = parser.parse_args()
60:     return args
61:
62:
63: def get_response(model, messages):
64:     data = {
65:         "model": model,
66:         "messages": messages,
67:         "stream": False,
68:     }
69:     response = requests.post(URL_OLLAMA, json=data)
70:     if not response.ok:
71:         print(f"request failed due to {response.status_code}")
72:         raise Exception("invalid response")
73:
74:     return response.json()["message"]
75:
76:
77: def init_mcp_server(mcp_session, url):
78:     """
79:     Init MCP server and get its tools.
80:
81:     Continue MCP session with id = 2.
82:
83:     Parameters
84:     -----
85:     mcp_session: requests.Session
86:     url: MCP server URL
87:
88:     Returns

```

```

89: -----
90: Tools List Response
91: See https://modelcontextprotocol.io/docs/learn/architecture
92: Tools Discovery > Tools List Response
93: """
94: mcp_session.headers.update({
95:     "Content-Type": "application/json",
96:     "Accept": "application/json, text/event-stream",
97: })
98:
99: # initialization
100: mcp_init_msg = {
101:     "jsonrpc": "2.0",
102:     "id": 1,
103:     "method": "initialize",
104:     "params": {
105:         "protocolVersion": "2025-06-18",
106:         "capabilities": {
107:             "elicitation": {}
108:         },
109:         "clientInfo": {
110:             "name": "example-client",
111:             "version": "1.0.0"
112:         }
113:     }
114: }
115: res0 = mcp_session.post(url, json=mcp_init_msg, stream=True)
116: if not res0.ok:
117:     raise Exception("fail to initialize MCP server")
118:
119: mcp_session.headers.update(
120:     {
121:         "mcp-session-id": res0.headers["mcp-session-id"]
122:     }
123: )
124:
125: # notification
126: mcp_notification_msg = {
127:     "jsonrpc": "2.0",
128:     "method": "notifications/initialized",
129: }
130: res1 = mcp_session.post(url, json=mcp_notification_msg, stream=True)
131: if not res1.ok:
132:     raise Exception("notifications/initialized failed")
133:
134: # get MCP tools
135: mcp_list_tool_msg = {
136:     "jsonrpc": "2.0",
137:     "id": 2,
138:     "method": "tools/list",
139:     "params": {
140:         "cursor": "optional-cursor-value",
141:     },
142: }
143: res2 = mcp_session.post(
144:     URL_MCP_SERVER,
145:     json=mcp_list_tool_msg,
146: )
147: if not res2.ok:
148:     raise Exception("tools/list failed")
149:
150: return res2
151:
152:
153: def call_mcp_tool(mcp_session, url, mcp_id, tool_name, tool_args):
154:     """
155:     Call MCP tool.
156:
157:     Parameters
158:     -----
159:     mcp_session: requests.Session
160:     url: MCP server URL
161:     mcp_id: mcp message id
162:     tool_name: tool name
163:     tool_args: xml.etree.ElementTree.Element
164:     """
165:     params = {
166:         "name": tool_name,
167:         "arguments": {c.tag: c.text for c in tool_args},
168:     }
169:
170:     body = {
171:         "jsonrpc": "2.0",
172:         "id": mcp_id,
173:         "method": "tools/call",
174:         "params": params,
175:     }
176:

```

```

177:     # print(f"call MCP tool: {body}")
178:
179:     res = mcp_session.post(url,
180:                             json=body)
181:     if not res.ok:
182:         raise Exception("tools/call failed")
183:
184:
185: def main():
186:     args = parse_args()
187:
188:     # init MCP server
189:     mcp_session = requests.Session()
190:     res2 = init_mcp_server(mcp_session, URL_MCP_SERVER)
191:
192:     print("---- res2 ----")
193:     print(res2.text)
194:     print("---- res2 ----")
195:     print("")
196:
197:     tools_json = json.loads(res2.text.split("¥r¥n")[1][6:])
198:
199:     # get first tools
200:     assert len(tools_json["result"]["tools"]) == 1
201:     tool = tools_json["result"]["tools"][0]
202:
203:     tool_name = tool["name"]
204:     tool_desc = tool["description"]
205:     tool_input = tool["inputSchema"]
206:
207:     # send system prompt to get first quiz
208:     messages = [
209:         {
210:             "role": "system",
211:             "content": get_system_prompt(tool_name, tool_desc, tool_input)
212:         }
213:     ]
214:
215:     res_msg = get_response(args.model, messages)
216:     # Uncomment the line below to display the first quiz
217:     print(res_msg)
218:     messages.append(res_msg)
219:     assistant_msg = res_msg["content"]
220:
221:     mcp_id = 2
222:     while True:
223:         root = ET.fromstring(assistant_msg)
224:         msg = root.find("message").text
225:
226:         # call MCP tool
227:         call = root.find("call")
228:         if call is not None:
229:             tool_name = call.find("name").text
230:             tool_args = call.find("arguments")
231:
232:             call_mcp_tool(mcp_session, URL_MCP_SERVER,
233:                           mcp_id, tool_name, tool_args)
234:             mcp_id += 1
235:
236:         # show chat message
237:         print(msg)
238:         user_input = input("input answer: ")
239:         user_input.strip()
240:
241:         # append user message
242:         messages.append(
243:             {
244:                 "role": "user",
245:                 "content": user_input,
246:             }
247:         )
248:
249:         res_msg = get_response(args.model, messages)
250:         # Uncomment the line below to display the raw response
251:         # print(res_msg)
252:         assistant_msg = res_msg["content"]
253:
254:         # append agent message
255:         messages.append(res_msg)
256:
257:
258: if __name__ == "__main__":
259:     main()

```