Project Report: Final

AI Agent for Rubik's Cube Knowledge and Optimal Solving

Assisted by: Gemini 2.5 Pro (AI assistant)
Date: December 2, 2025
Name: Brandon Lewis

1. Project Objective (Final)

The final objective of this project was to develop a resilient and efficient AI agent capable of handling all user interactions within the Rubik's Cube domain. The final agent architecture achieves two primary functions:

1. **Efficient Knowledge Retrieval (The Teacher):** The agent identifies, downloads, and persists instructional content (specifically from YouTube transcripts) into a local database. It then uses this local store for fast, reliable answering of qualitative questions ("How-to" steps, algorithm names, conceptual understanding) before resorting to external search.
2. **Optimal Computational Solving (The Solver):** The agent utilizes a high-performance external C-based implementation of the **Kociemba Two-Phase Algorithm** to instantly generate the near-optimal solution for any valid, scrambled 3x3 cube state.

2. Component 1: Knowledge Retrieval and Persistence

The agent's ability to learn and answer conceptual questions is handled by a three-tiered system: Local Retrieval, Focused Search, and Database Persistence.

Tool A: LocalTranscriptSearchTool (New)

- **Purpose:** Provides immediate, offline access to previously indexed instructional content.
- **Mechanism:** Queries the local cube_db.json (TinyDB) by keyword. This tool ensures the agent "remembers" information and avoids redundant external calls.
- **Output:** Returns snippets of saved transcripts related to the user's query (e.g., "white cross," "F2L").

Tool B: YoutubeTranscriptSearchTool

- **Purpose:** To find and ingest new instructional content when local knowledge is insufficient.
- **Mechanism:** Searches YouTube for videos, fetches the full transcript text, and saves it to the database for future use. Includes checks to prevent duplicate storage.
- **Output:** Returns the newly acquired transcript text for the agent to analyze and use in its immediate response.

Data Persistence: The Knowledge Base

All video transcripts are automatically stored in a local TinyDB database (cube_db.json). This knowledge base ensures efficient accumulation:

Data Field	Purpose
video_id	Unique key used for duplication checks and retrieval.
title, channel, url	Metadata for citation and context.
transcript	The full, parsed text content of the video.

3. Component 2: Computational Solving (Optimal Kociemba Algorithm)

The simplified approach now relies solely on a high-performance optimal solver, eliminating the complexity of a Python-managed, step-by-step method.

Tool C: 54 Character rubiks cube state string formatter

- **Purpose:** To provide a single, 54 character long, correctly formatted string of a rubiks cube state in the format (URFDLB) U for UP, R for RIGHT, F for FRONT, D for DOWN, L for LEFT, B for BACK. It has the option to convert to Color format (BGOYWR) Blue, Green, Orange, Yellow, White, Red. The way the cube sits does not matter, it will read the center of each face in order to determine its position for conversion.
- **Input:** A 54-character state string (e.g., URFDLB or BGOYWR) where each character represents a sticker color (B, G, O, Y, W, R).
- **Output:** A single string of moves in positional format (UUUUUUUUURRRRRRRRRFFFFFFFFFDDDDDDDDDLLLLLLLLLBBBBBBBBB).

Tool D: Optimal_Kociemba_Solver

- **Purpose:** To provide a single, complete, and near-optimal solution in the shortest number of moves.
- **Mechanism:** This tool acts as a Python wrapper around a compiled external C executable (based on Herbert's Kociemba Two-Phase Algorithm).
  - It sends the 54-character scrambled state and the hardcoded solved state to the binary via a subprocess
  - The C binary handles the computationally intensive search.
- **Input:** A 54-character state string (e.g.,URFDLB)
- **Output:** A single string of moves in standard notation (e.g., U' R2 F B2 L D' R...).

4. PEAS Framework Analysis (Final)

The agent's operational parameters are refined based on the final toolset:

- **Performance Measure:**
  - **Solver: Correctness (absolute necessity)**, **Optimality (minimum number of moves)**, and execution **Speed**. A successful outcome is a valid, short move sequence that solves the cube.
  - **Researcher:** The **relevance**, **accuracy**, and **clarity** of the information retrieved and synthesized from both the local database and new transcripts.
- **Environment:**
  - **Internal:** The local filesystem/database (cube_db.json). (Fully Observable, Static, Discrete).
  - **External:** The public internet (YouTube and Search). (Partially Observable, Stochastic, Dynamic).
- **Actuators (Agent's "Hands"):**
  - **Optimal_Kociemba_Solver:** Executes the subprocess call to the C binary, generating the solution moves.
  - **Text Response:** The primary actuator for communication, delivering summarized research or the final solution.
  - **cube_db.add_transcript():** Acts on the internal environment by writing new data to the database.
- **Sensors (Agent's "Eyes"):**
  - **User Input:** Receives the initial prompt (question or state string).
  - **LocalTranscriptSearchTool:** Senses local memory for existing answers.
  - **YoutubeTranscriptSearchTool:** Senses the external web/API for new video content.
  - **External C Solver Output:** Senses the result (the solution string) generated by the high-performance computational core.

5. Agent Architecture, Reasoning, and Data Flow

The agent operates as a **Tool-Calling Agent** built on the Gemini model, employing a strict, hierarchical decision process designed to prioritize speed and local knowledge.

Architectural Components

1. **Large Language Model (LLM):** The core reasoning engine (Gemini). It interprets user intent, formulates a plan, selects the correct tool, and synthesizes the final response.
2. **Tool Orchestrator:** The intermediary layer (smolagents framework) that manages tool availability and execution.
3. **Tool Set:** The three specialized actuators/sensors: LocalTranscriptSearchTool, YoutubeTranscriptSearchTool, and Optimal_Kociemba_Solver.
4. **Persistent Knowledge Base:** The cube_db.json TinyDB file.

Decision Process (Reasoning Logic)

The LLM follows a prioritized, rule-based reasoning chain to determine its course of action for any incoming query:

Step	Condition	Action	Justification
1 (Solve Check)	User input is a valid 54-character cube state string.	Execute **Optimal_Kociemba_Solver**.	Highest priority, deterministic task. Fastest route to solution.
2 (Local Check)	User asks a conceptual question (e.g., "What is F2L?").	Execute **LocalTranscriptSearchTool** with keywords.	Prioritize speed and resource conservation. Avoids external API calls.
3 (External Search)	LocalTranscriptSearchTool returns insufficient or zero results.	Execute **YoutubeTranscriptSearchTool** with the original query.	Acquire new knowledge for the current session and persistent storage.
4 (Final Synthesis)	A tool has returned its output (solution string or transcript text).	The LLM processes the output, formats it, and generates the final, conversational response.	Converts raw tool data into human-readable advice.

Data Flow Diagram (Conceptual)

The data flow is a loop initiated by user input and resolved by tool execution:

1. **User Input** --> **LLM (Reasoning)**
2. **LLM** --> **Tool Orchestrator** (Chooses Tool)
3. **Local Knowledge Flow (If Tool A):**
   - Tool A reads from **cube_db.json** --> Tool A Output --> **LLM** (Synthesize)
4. **External Search Flow (If Tool B):**
   - Tool B searches **YouTube API** --> Fetches Transcript --> Tool B writes to **cube_db.json** --> Tool B Output --> **LLM** (Synthesize)
5. **Solving Flow (If Tool C):**
   - Tool C executes **External C Solver** --> External Solver Output --> **LLM** (Synthesize)
6. **LLM** --> **Final Text Response** --> **User**