Multi-Step Reasoning Agent with Self-Checking

Objective

Build a small “reasoning agent” that can solve structured problems in multiple steps, check its own work, and only show the final answer to the user. You may use any LLM API you like (OpenAI, Anthropic, Gemini etc.) and any programming language.

1. Problem Domain

Your agent will solve **word problems** in this style:

- Math / logic / constraints examples:
  - “If a train leaves at 14:30 and arrives at 18:05, how long is the journey?”
  - “Alice has 3 red apples and twice as many green apples as red. How many apples does she have in total?”
  - “A meeting needs 60 minutes. There are free slots: 09:00–09:30, 09:45–10:30, 11:00–12:00. Which slots can fit the meeting?”

The input is a **plain text question**.
The output should be a **JSON object** with this schema:

```
{
  "answer": "<final short answer, user-facing>",
  "status": "success" | "failed",
  "reasoning_visible_to_user": "<short explanation, but no raw chain-of-thought
logs>",
  "metadata": {
    "plan": "<model's internal plan, can be abbreviated>",
    "checks": [
      {
        "check_name": "<string>",
        "passed": true,
        "details": "<string>"
      }
    ],
    "retries": <integer>
  }
}
```

- The **user** should only really need the answer and reasoning_visible_to_user.
- plan, checks, and retries are mainly for debugging/evaluation.

2. Agent Requirements

Your agent must implement at least **three phases** internally:

1. **Planner**
    - Reads the user question.
    - Produces a concise **step-by-step plan** (in text or structured form).
    - Example: parse → extract quantities → compute → check units → format answer.
2. **Executor**
    - Follows the plan to produce an **intermediate solution**, including intermediate numbers / deductions.
    - Can call the LLM again and/or use code (e.g., Python) for arithmetic or validation.
3. **Verifier**
    - Re-checks the intermediate solution.
    - At least one of:
        - Re-solve independently and compare.
        - Validate constraints (e.g., times, totals, non-negative counts).
        - Look for inconsistencies in the explanation.

If verification **fails**, the agent should:

- Either **retry** (plan+execute) a limited number of times, or
- Mark the status as "failed" and explain why.

Important: the **full chain-of-thought** (long reasoning) should *not* be shown directly to the user. It can appear in metadata or logs.

3. Interface

Implement a simple interface of your choice:

- **CLI**: python agent.py then type questions.
- OR a simple **HTTP endpoint** (e.g., /solve) that accepts {"question": "..."} and returns the JSON.
- OR a minimal **notebook function** solve(question: str) -> dict.

We care about the **agent loop and logic**, not fancy UI.

4. Technical Requirements

- Use any language you're comfortable with.
- Use any LLM provider. You may:
  - Call real APIs, **or**
  - Mock/stub the calls if needed (but the design should be realistic and swappable with a real LLM).
- Structure your code so that:
  - Prompts are not hard-coded everywhere.
  - The planner, executor, and verifier are reasonably separated (functions/classes/modules).

5. Prompting Requirements

You must write **separate prompts** (they can be in the same file) for:

1. Planner prompt
   - E.g., "Given a question, output a numbered plan with steps…"
2. Executor prompt (if LLM is used to execute)
   - E.g., "Given a plan and question, follow steps exactly and show intermediate calculations…"
3. Verifier prompt
   - E.g., "Given the question and a proposed solution, check if it's consistent and either approve or explain issues."

For each prompt, try to:
- Specify **format** of the model's output (lists, JSON, etc.).
- Give at least **2–3 examples** in the prompt or as tests in your code.

6. Evaluation & Test Cases

Include a small **test suite** (can be simple code assertions or a script) with at least:
- 5–10 "easy" questions (basic arithmetic / time differences).
- 3–5 "tricky" questions where naive reasoning might fail, for example:
  - Ambiguous numbers.
  - Multi-step calculations.
  - Edge cases around times or boundaries.

For each test, log:
- The question.
- The final JSON.
- Whether the verifier passed.
- If the agent retried.

7. What to Submit

Please provide:

1. **Code** (git repo link)
    - With a short README.md explaining:
        - How to run it.
        - Where prompts live.
        - Any assumptions.

2. **Prompt documentation** (this can be in the README)
    - Why you designed the prompts this way.
    - What you tried that didn't work well.
    - What you would improve with more time.

3. **Example run logs**
    - Copy/paste or files showing 5–10 example questions and outputs.