Two-week take-home assignment (≈ 40 engineering hours).
Deliver a fully runnable, testable, Docker-packaged **Research LLM Agent (CLI only)** that demonstrates competence in:

1. **Prompt Engineering**
2. **Tool Orchestration (LangGraph, Web Search, etc.)**
3. **Engineering Delivery (tests, Docker, docs)**

Business Scenario

A user types a natural-language question in the terminal.
Your agent must automatically

> **Generate queries → Web-search → Reflect (decide if more search is needed) → *optional* second search → Synthesise answer**

and finally return an English answer with Markdown citations in pure JSON.

Minimum-Viable Product (MVP)

#	Feature	# Requirements
1	**LangGraph Pipeline**	Four nodes**: GenerateQueries → WebSearchTool → Reflect → Synthesize.** Support up to `MAX_ITER = 2` cycles.
2	**GenerateQueries**	Use an LLM to break the original question into **3-5 English search queries.** Return a JSON array.
3	**WebSearchTool**	Issue concurrent Bing Web Search (or Mock) calls for all queries; merge and de-duplicate the results into `docs`.
4	**Reflect (Simple)**	Prompt goal: decide if current `docs` are enough. If not, return 1-3 refined queries.

		Output format: `{ "need_more": true/false, "new_queries": ["..."] }`
5	**Synthesize**	Use the LLM to produce a concise English answer **not exceeding 80 words (≈ 400 characters).** End the text with Markdown references `[1][2]`… Return a JSON object: `{ "answer": "...", "citations": [...] }`
6	**One-Command Run**	`docker compose run --rm agent "<question>"` prints the final JSON.
7	**Unit Tests**	≥ 5 Pytest cases: ① happy path ② no result ③ HTTP 429 ④ timeout ⑤ two-round supplement.
8	**Documentation**	`README.md` with architecture diagram, run steps, extension ideas. Design doc ≤ 10 pages (PDF or MD).

Project Conventions

Item	Specification
Language	Python ≥ 3.9
LLM Provider	via `OPENAI_API_KEY` or `GEMINI_API_KEY` (or choose any free tier model as you wish)

Search API	`BING_API_KEY`; if absent, automatically fall back to `MockWebSearchTool`.
Run Method	`Dockerfile` + `compose.yaml`

Suggested Directory Layout

```
llm-research-agent/@
├─ src/agent/        # state / prompts / tools / nodes / graph
├─ tests/            # Pytest + mock data
├─ Dockerfile
├─ compose.yaml
└─ README.md
```

Bonus Items (choose any)

📡 SSE / WebSocket streaming

♻️ Redis LRU cache for query results

📈 OpenTelemetry traces + Prometheus metrics

💬 Minimal React/Vite front-end dialog

📑 OpenAI function-calling to constrain `Synthesize` output

🔗 Slot-Aware Reflect (details & examples below)

Slot-Aware Reflect (Advanced Bonus)

Purpose — increase verifiability and pinpoint missing facts.

Variant	Reflect Output Example	Explanation

Without slots (MVP)	`json { "need_more": true, "new_queries": ["Argentina goals 2022 final","France goals 2022 final"] }`	Agent simply decides whether more search is required.
With slots (bonus)	`json { "slots": ["winner"], "filled": ["winner"], "need_more": false, "new_queries": [] }`	Agent first lists mandatory *slots* (fields), checks which ones are filled by evidence. Missing or conflicting slots trigger `need_more=true` and targeted queries.

How to earn the slot bonus

1. Extend the Reflect prompt to:
    - list required slots,
    - match evidence sentences in `docs`,
    - judge completeness & consistency.
2. Return the structured JSON above.
3. Adapt your unit tests to assert that **all declared slots are filled** before `need_more=false`.

Submission Checklist

- Public or invited Git repository
- All tests pass: `pytest -q`
- `docker compose run --rm agent "<question>"` works offline (mock) and online
- Design document & architecture diagram

- (Optional) 3-minute screencast demo

Finish the MVP first, then pick any bonus items—including slot-aware Reflect—to raise your score.
Good luck and have fun building!

Example Run 1 — “Fact” question

bash

```
docker compose run --rm agent "Who won the 2022 FIFA World Cup?"
```

json

```
{
  "answer": "Argentina won the 2022 FIFA World Cup, beating France on penalties after a 3-3 draw in extra time.[1]",
  "citations": [
    {
      "id": 1,
      "title": "Argentina win World Cup 2022",
      "url": "https://www.fifa.com/worldcup/news/argentina-win"
    }
  ]
}
```

Example Run 2 – “Tech-comparison” question

bash

```
docker compose run --rm agent "Compare Kubernetes HPA and KEDA: key scaling features and use cases."
```

json

```
{
```

```
  "answer": "HPA scales Deployments by CPU/Memory metrics pulled
from the cluster, suiting steady, resource-bound web services. KEDA
adds 50+ event sources (Kafka, Prometheus, Azure Queue, etc.),
enabling scale by business events such as queue length—ideal for
bursty, event-driven workloads and batch jobs.[1][2]",
  "citations": [
    {
      "id": 1,
      "title": "Kubernetes HPA official docs",
      "url":
"https://kubernetes.io/docs/tasks/run-application/horizontal-pod-aut
oscale/"
    },
    {
      "id": 2,
      "title": "CNCF Blog: Introducing KEDA",
      "url": "https://www.cncf.io/blog/2024/03/03/introducing-keda/"
    }
  ]
}
```


*Both answers stay within the **≤ 80 English-word** limit and end with Markdown-style numeric citations, matching the assignment specification.*