Quiz4

November 7, 2026

1 Take-home Quiz: The ReAct Fact Checker

Use the model `gemini-3.8-flash` through the GenerateContent or Interactions API to build an application that reads an input text document, e.g., `data/document.txt`, identifies all the *incorrect claims* that are made in the document, and proposes a correction for each. The deliverable captured in the notebook are as follows:

- The description of your **architecture**.
- The **implementation** containing the ReAct loop.
- The **traces** of running your code on the example `data/document.txt`.

The ReAct loop should run until all incorrect claims have been identified and fixed, or until a maximum number $N = 50$ of LLM calls have been done. As tools, your application may use only the Gemini's built-in Google Search and Code Execution tools. Local file proocessing such as ordinary file I/O or JSON parsing are fine. All Gemini model calls must go through the API. You must understand and be able to explain your implementation.

Submit this completed notebook with its saved outputs. Keep all your work in the following notebook sections:

1. **Architecture** cell: Explain claim extraction, state/observation handling, iteration, stopping criteria, failure-handling, and anything else that is important in your approach.
2. **Implementation** cell: This is your working Python code of the ReAct loop.
3. **Trace** cell: The output of running the ReAct loop should capture every model request and response, with call IDs, model/configuration, prompts, all API-exposed reasoning/thought-summary text, each tool invocation and result, and the final answer.
    - If code is generated and executed, show the executable code and its output; for search tool invocations, include the generated queries, grounding chunks/supports and metadata.
    - For searched claims, show direct links to supporting primary-source pages when available, rather than a homepage or secondary summary.
4. **Report** cell: create a final summary report that quotes each claim that was flagged as false, propose a correction, and link it to a specific step/call ID from the trace above. Include supporting evidence and direct source links where applicable. Mark unresolved claims uncertain.

Open this notebook by running Jupyter Lab or VS Code with a Python 3.10+ kernel, in the folder `quiz4/` so that `data/` and `requirements.txt` are available. Keep the data subfolder in the same

folder as this notebook.

Run the setup cells, write code in the TODO sections, then run the fact checker ReAct loop and extract the final report. The starter code intentionally raises `NotImplementedError` until you implement it.

You are free to reuse code provided in examples or assignments covered in class of the assignments implemented so far, as linked from the course website.

1.1 Notebook Setup

Import necessary packages, set training data cutoff from the Gemini 3.8 model card, read the Gemini API key from your environment file, and initialize the Gemini client.

If additional Python packages are needed for local file processing or display functions, import them here.

[ ]:
```
%pip install -r requirements.txt
```

[ ]:
```
from __future__ import annotations

from datetime import datetime, timezone
from pathlib import Path
from typing import Any
from IPython.display import JSON, display

import json

import os
import sys

from google import genai
from google.genai import types

if sys.version_info < (3, 10):
    raise RuntimeError("Select a Python 3.10+ notebook kernel.")


GEMINI_MODEL = os.getenv("GEMINI_MODEL", "gemini-3.8-flash")
GEMINI_CUTOFF = "2026-03"  # Official month-level precision; not an exact day.

print(f"Model: {GEMINI_MODEL}; documented cutoff: {GEMINI_CUTOFF}")

from google import genai
from dotenv import load_dotenv, find_dotenv

# Read the local .env file, containing the Gemini secret API key.
_ = load_dotenv(find_dotenv())
```

```
# Create the Gemini client.
gemini_client = genai.Client(api_key = os.environ["GEMINI_API_KEY"])
```

1.2 Load input document

Read the example document. Note that this is just an example, your code should be general so that it works on any document provided here.

[ ]:
```
DATA_PATH = Path("data/document.txt")

if not DATA_PATH.is_file():
    raise FileNotFoundError("Start the notebook from insided the quiz4 folder,␣
 ↪with data/document.txt present.")

document = DATA_PATH.read_text(encoding = "utf-8")
print(document)
```

1.3 Architecture of the Fact Checker

Explain claim extraction, state/observations, iteration, stopping case, budget, and one failure-handling choice.

TODO: describe below in this cell the architecture of your ReAct loop. Make sure that your description is well structured and nicely formatted

1.4 Implementation of the Fact Checker ReAct loop

Complete the ReAct loop implementation in the cell below.

[ ]:
```
def run_fact_checker(document: str, trace: TraceLogger):
    """
    - Implement the complete ReACt loop here.
    - Running this code should output everything specified in the Trace section␣
 ↪of the requirements.
    - Return a results object that contains all the necessary information␣
 ↪needed for extracting a summary report of the findings.
    """

    # TODO: YOUR CODE HERE
```

```
    raise NotImplementedError("Implement your architecture and ReAct loop here.
↪")
```

1.5 Trace the Fact Checker

After implementing the ReAct loop above, run the Fact Checker in the cell below. The Cell output should show the output of running the ReAct loop, as specified in te requirements at the beginning of this noteboook.

[ ]:
```
react_results = run_fact_checker(document, trace)
```

1.6 Report summary of the Fact Checker findings

The output of this cell should show a summary of the findings that quotes each flagged claim, proposes its correction, and identifyies the corresponding trace step/call IDs, with supporting evidence and direct source links where applicable. Distinguish uncertain claims from verified errors.

[ ]:
```
def create_summary(results: str):
    """
    Extract a summary of the findings from the result object returned by the␣
↪ReAct loop.
    """

    # TODO: YOUR CODE HERE

    raise NotImplementedError("Implement your Summary report of the findings␣
↪here.")
```

```
summary_report = create_summary(react_results)

print(summary_report)
```

1.7 Submission check

Submit the completed `.ipynb` and `.pdf` with the architecture, implementation, full trace output, and final-report output saved inside it, as captured from one run. No separate generated output files are required.