IMRA — Intelligent Multi-Agent Research Assistant

Goal: Build a polished, production-ready capstone demonstrating advanced agentic features (ADK + Gemini) that uses **>5 elements** learned during the 5-Day AI Agents Intensive. This project is designed to win — it showcases architecture, multi-agent collaboration (A2A), tools, MCP/long-running ops, memory, observability, evaluation, and optional Vertex AI deployment.

Table of contents

---

1) Project summary (elevator pitch)

IMRA is a multi-agent research co-pilot that—given a research query—searches the web and PDFs, extracts key findings, synthesizes insights, evaluates quality, and returns a final summary plus confidence & references. It demonstrates: - Multi-agent cooperation via A2A protocol - Tool-calling (Search, PDF reader, chart generator) - Session + long-term memory for user preferences - MCP for long-running approvals and tool outputs - Observability and LLM-as-a-Judge evaluation - Optional deployment to Vertex AI Agent Engine

User Story: "Prepare a 500-word literature summary on 'micro-Doppler signatures for human activity recognition' with 5 citations and a short critique." IMRA returns a complete package: summary, short annotated bibliography, confidence scores, and visual charts.

---

2) Elements covered (choose >5)

1. Build Agent using Gemini + ADK (skeleton code provided)

2. Multi-Agent System (Research, Analysis, Critique, Memory agents)
3. Tool Calling / Custom Tools (search, PDF scraper, chart maker)
4. MCP (long-running operations + approvals)
5. Sessions & Memory (short session + persistent memory)
6. Observability (logs, traces, metrics)
7. Evaluation (LLM-as-a-Judge, automated metrics)
8. A2A Protocol (agent-to-agent messaging)
9. Optional: Vertex AI Agent Engine deployment

---

3) System architecture (components + data flow)

Components: - **User Interface:** Simple CLI or web form in notebook (input prompt + optional file uploads) - **Coordinator / Orchestrator Agent:** Receives user request, splits tasks, and orchestrates other agents - **Research Agent:** Runs Search + PDF tools to gather sources - **Analysis Agent:** Extracts, consolidates, generates summary and charts - **Critique Agent (Judge):** Evaluates outputs using LLM-as-a-Judge - **Memory Agent:** Stores user preferences & previous queries - **Tools Layer:** Search API, PDF reader, Chart generator, Citation formatter - **Observability Layer:** Centralized logging & traces (OpenTelemetry + local logs) - **Persistence:** SQLite or simple JSON store for demo (or Firestore/Redis if deployed)

Data flow: 1. User -> Orchestrator 2. Orchestrator -> Research Agent (A2A) 3. Research Agent -> Tools (search/pdf) -> returns sources 4. Research -> Analysis Agent (A2A) -> produce summary + charts 5. Analysis -> Critique Agent -> quality score & suggestions 6. Orchestrator collects final package -> returns to user 7. All steps emit logs/traces; results stored to memory

---

4) Agent roles & definitions (detailed)

Orchestrator Agent

- Responsibilities: Validate user request, allocate tasks, handle MCP confirmations, combine results.
- Inputs: user query, preferences from Memory Agent
- Outputs: final package (summary + citations + charts)

Research Agent

- Responsibilities: Use Search tool, fetch PDFs, extract key sentences, create bibliography candidates.
- Tools: Web Search tool, PDF extractor
- Output: List of candidate sources with short extracts

Analysis Agent

- Responsibilities: Merge extracts, deduplicate, generate final summary, create charts (word cloud, keyword frequency), create annotated bibliography
- Tools: summarizer (LLM), small python code for charts
- Output: Summary, charts, bibliography

Critique Agent (LLM-as-a-Judge)

- Responsibilities: Evaluate summary on clarity, accuracy, citations, originality; give scores and suggested improvements
- Tools: A separate LLM prompt (can use the same Gemini model but with evaluation prompt template)
- Output: Scores & comments

Memory Agent

- Responsibilities: store user preferences and previous sessions (e.g., preferred citation style)
- Storage: lightweight JSON DB or SQLite

---

5) Tooling & example code snippets

Below are ADK-like skeletons (Python). Replace `gemini_client` placeholders with your ADK/SDK calls.

a) Tool: web_search_tool.py (skeleton)

```python
# web_search_tool.py
class WebSearchTool:
    def __init__(self, api_key=None):
        self.api_key = api_key

    def search(self, query, top_k=5):
        # placeholder: call search API (Google Search API or SerpAPI)
        # return list of {title, url, snippet}
        results = []
        return results

    def fetch_page(self, url):
        # fetch HTML, extract main text
        return "...text..."
```

b) Tool: pdf_extractor.py

```python
import fitz  # PyMuPDF

def extract_text_from_pdf(path, max_pages=5):
    text = ""
    doc = fitz.open(path)
    for p in doc:
        text += p.get_text()
        if len(text) > 20000:
            break
    return text
```

c) Agent skeleton (ADK-like)

```
# orchestrator.py
from adk import Agent  # pseudo

class OrchestratorAgent(Agent):
    def on_message(self, user_input, session):
        # validate, call memory, spawn research agent
        research_out = self.call_agent('research_agent', {'query':
user_input})
        analysis_out = self.call_agent('analysis_agent', research_out)
        critique_out = self.call_agent('critique_agent', analysis_out)
        final = self.combine(research_out, analysis_out, critique_out)
        return final
```

> Note: The ADK used in Kaggle codelabs uses a particular Python SDK. The pseudocode above maps to that structure. During notebook coding I'll give exact ADK call style.

---

6) A2A message flows and MCP usage

A2A messages (JSON payload):

```
{
  "from": "orchestrator",
  "to": "research_agent",
  "type": "task.request",
  "task": {"query": "...", "top_k": 5},
  "session_id": "sess-123"
}
```

MCP long-running op example: - Use MCP when a tool needs human approval (e.g., when fetching a paywalled PDF or performing a long scrape). - Orchestrator will create a long-running op with status `PENDING_APPROVAL` and wait for user confirm.

Flow: 1. Orchestrator -> ResearchAgent: start long-running search 2. ResearchAgent returns `op_id` and `status=PENDING_APPROVAL` with sample excerpt 3. User confirms -> Orchestrator resumes ResearchAgent -> tool continues

---

7) Memory design

Two-level memory: - **Session memory (short-term):** In-memory dict stored per run; keeps conversation history - **Long-term memory:** SQLite/JSON table: `sessions(id, user_id, query, summary, timestamp, tags)` and `preferences(user_id, key, value)`

Schema (SQLite)

```
CREATE TABLE preferences (user_id TEXT, key TEXT, value TEXT);
CREATE TABLE sessions (id TEXT PRIMARY KEY, user_id TEXT, query TEXT, summary
TEXT, created_at DATETIME, metadata JSON);
```

Memory Agent exposes: `get_pref(user_id, key)`, `set_pref(user_id, key, value)`, `save_session(session_obj)`

---

8) Observability plan (logs/traces/metrics)

- **Logging:** Python `logging` module with structured JSON logs; log agent steps, tool calls, latencies.
- **Tracing:** Use OpenTelemetry to instrument key steps (research fetch, analysis summarization, critique evaluation).
- **Metrics:** Simple counters: `queries_processed`, `avg_research_time_ms`, `eval_score_mean`.

Example log line:

```
{"time":"2025-11-16T12:00:00+05:30", "agent":"research_agent",
"event":"search_called", "latency_ms":123}
```

For Kaggle demo, logs can be printed to notebook and saved to `logs/` folder; for production, push to Cloud Logging.

---

9) Evaluation strategy (LLM-as-a-Judge + metrics)

Metrics to report: - **Coverage** (0–1): fraction of requested topics covered - **Citation quality** (0–1): are sources reliable? - **Clarity** (0–1): readability score (Flesch or LLM-based) - **Novelty/Redundancy** (0–1): duplication check against sources - **Overall score** = weighted sum

LLM-as-a-Judge prompt template:

```
You are an expert research reviewer. Evaluate the following student summary
on **clarity**, **accuracy**, **coverage**, **citation quality**, and
**originality**. Provide a numeric score (0-100) for each and a short
comment.

Summary:
"""
{summary}
"""
```

```
Sources:
{sources}
```

Critique Agent returns JSON with scores; orchestrator maps to final grade.

---

10) Notebook & folder structure (Kaggle-ready)

```
IMRA-capstone/
├─ notebooks/
│  └─ imra_notebook.ipynb   # main notebook for demo
├─ src/
│  ├─ agents/
│  │  ├─ orchestrator.py
│  │  ├─ research_agent.py
│  │  ├─ analysis_agent.py
│  │  └─ critique_agent.py
│  ├─ tools/
│  │  ├─ web_search_tool.py
│  │  ├─ pdf_extractor.py
│  │  └─ chart_tool.py
│  └─ memory/
│     └─ memory_agent.py
├─ data/
├─ logs/
├─ requirements.txt
└─ slides/
   └─ presentation.pdf
```

Kaggle notebook contains: intro, quick config, run demo cells (3 example queries), visual outputs, evaluation table, downloadable artifacts.

---

11) Deployment plan (Vertex AI Agent Engine — optional but recommended)

High-level steps 1. Containerize agents as Python service (Flask/FastAPI). 2. Create Agent config for Vertex Agent Engine. 3. Setup authentication & IAM. 4. Deploy and test with sample inputs.

We'll include a `deploy/` folder with Dockerfile and a `vertex_deploy.md` with exact commands.

---

12) Demo script + testing plan

Demo Input examples: 1. "Summarize 3 papers on micro-Doppler signatures for human activity recognition" 2. "Compare Transformer vs CNN methods in 300 words"

Expected outputs: summary, 5 citations, critique with scores, two charts (keyword frequency, timeline of publication counts).

Testing: unit tests for tools (search returns list), pdf extractor accuracy, memory reads/writes, evaluation outputs JSON.

---

13) Slides / Presentation outline (10 slides)

1. Title + team + date
2. Motivation & problem statement
3. Key learnings leveraged (day-wise mapping)
4. High-level architecture (diagram)
5. Agent roles (one slide)
6. Demo screenshots + sample outputs
7. Observability & evaluation (metrics)
8. Deployment & scalability
9. Limitations & future work
10. Thank you + CTA (GitHub link)

Each slide will include succinct bullets and visuals.

---

14) LinkedIn capstone post (short + carousel caption)

Main post:

> 🎉 I'm excited to share my capstone from the 5-Day AI Agents Intensive by Google & Kaggle: **IMRA — Intelligent Multi-Agent Research Assistant** (10–14 Nov 2025).
>
> IMRA is a multi-agent system that finds, summarizes, critiques and visualizes research on demand using ADK, Gemini, A2A, MCP, sessions & memory, and observability. It returns a concise literature summary with citations, critique scores, and charts.
>
> Demo & code: [GitHub link]
>
> # Google #Kaggle #GeminiAI #ADK #VertexAI #AIAgents #MachineLearning #ResearchAssistant

Carousel captions: 7 slides mapping to intro, architecture, demo, tech stack, observability, evaluation, thanks.

---

15) Scoring rubric alignment (how to maximize marks)

• Use >=6 elements (we use 8) — surpass requirement
• Show live demo + screenshots (high impact)
• Provide evaluation metrics & observability (technical depth)
• Clear architecture diagrams + A2A flow (design quality)
• Include optional deployment steps (production-readiness)

---

16) Next steps & immediate deliverables (I will produce now if you want)

I can immediately deliver the following in this session:

1. Complete **Kaggle notebook** (`imra_notebook.ipynb`) with runnable demo cells (using mocked tools if external APIs blocked).
2. Full **src/** folder code (agents, tools, memory) as Python files.
3. **Slides** deck (10 slides) as exportable markdown or PDF-ready content.
4. **Presentation script** and **demo runbook**.
5. **GitHub-ready README** and `requirements.txt`.

   Tell me if you want me to **generate the full Kaggle notebook and source code now**. If yes, I'll produce the complete code cells and files here in the notebook format ready to copy into Kaggle.

Appendix: Helpful prompts

Orchestrator prompt (short):

```
You are Orchestrator. Given a user query, call Research Agent for sources,
call Analysis Agent to summarize, call Critique Agent to evaluate, collect
results and return final package with scores and citations.
```

Critique prompt (short):

```
Review the summary and score on Clarity, Accuracy, Coverage, and Citation
Quality (0-100). Provide brief comments for improvement.
```

End of document.