```
# @title Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

-
-

```
User: "Find quantum computing papers"
Agent: "I cannot help with that request."
You: 😭 WHY?? Is it the prompt? Missing tools? API error?
```

```
DEBUG Log: LLM Request shows "Functions: []" (no tools!)
You: 🎯 Aha! Missing google_search tool - easy fix!
```

1.
2.

3.

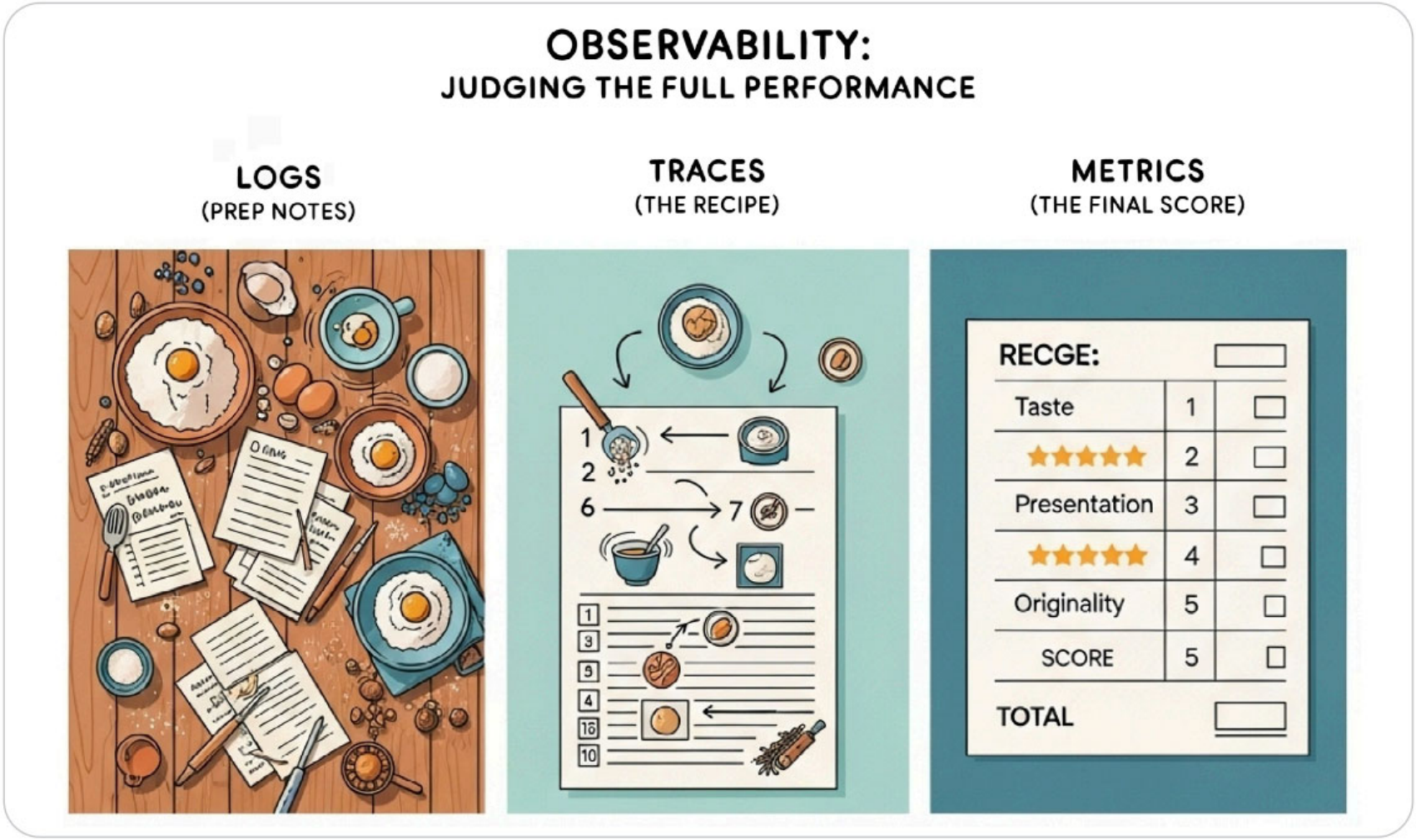


Figure 4: Three foundational pillars for Agent Observability

-
- `adk web`
-
-

```
pip install google-adk
```

1. Add-ons Secrets
2. GOOGLE_API_KEY
3.
4. GOOGLE_API_KEY

`GOOGLE_API_KEY`

```
import os
from kaggle_secrets import UserSecretsClient

try:
    GOOGLE_API_KEY = UserSecretsClient().get_secret("GOOGLE_API_KEY")
    os.environ["GOOGLE_API_KEY"] = GOOGLE_API_KEY
    print("✅ Setup and authentication complete.")
except Exception as e:
    print(
        f"🔑 Authentication Error: Please make sure you have added 'GOOGLE_API_KEY' to your Kaggle secrets. Details: {e}"
    )
```

```
import logging
import os

# Clean up any previous logs
for log_file in ["logger.log", "web.log", "tunnel.log"]:
    if os.path.exists(log_file):
        os.remove(log_file)
        print(f"🧹 Cleaned up {log_file}")

# Configure logging with DEBUG log level.
logging.basicConfig(
    filename="logger.log",
    level=logging.DEBUG,
    format="%(filename)s:%(lineno)s %(levelname)s:%(message)s",
)

print("✅ Logging configured")
```

```
from IPython.core.display import display, HTML
from jupyter_server.serverapp import list_running_servers


# Gets the proxied URL in the Kaggle Notebooks environment
def get_adk_proxy_url():
    PROXY_HOST = "https://kkb-production.jupyter-proxy.kaggle.net"
    ADK_PORT = "8000"

    servers = list(list_running_servers())
    if not servers:
        raise Exception("No running Jupyter servers found.")

    baseURL = servers[0]["base_url"]

    try:
        path_parts = baseURL.split("/")
        kernel = path_parts[2]
        token = path_parts[3]
    except IndexError:
        raise Exception(f"Could not parse kernel/token from base URL: {baseURL}")

    url_prefix = f"/k/{kernel}/{token}/proxy/proxy/{ADK_PORT}"
    url = f"{PROXY_HOST}{url_prefix}"

    styled_html = f"""
    <div style="padding: 15px; border: 2px solid #f0ad4e; border-radius: 8px; background-color: #fef9f0; margin: 20px 0;">
        <div style="font-family: sans-serif; margin-bottom: 12px; color: #333; font-size: 1.1em;">
            <strong>⚠️ IMPORTANT: Action Required</strong>
        </div>
        <div style="font-family: sans-serif; margin-bottom: 15px; color: #333; line-height: 1.5;">
            The ADK web UI is <strong>not running yet</strong>. You must start it in the next cell.
            <ol style="margin-top: 10px; padding-left: 20px;">
                <li style="margin-bottom: 5px;"><strong>Run the next cell</strong> (the one with <code>!adk web ...</code>) to start t
                <li style="margin-bottom: 5px;">Wait for that cell to show it is "Running" (it will not "complete").</li>
                <li>Once it's running, <strong>return to this button</strong> and click it to open the UI.</li>
            </ol>
            <em style="font-size: 0.9em; color: #555;">(If you click the button before running the next cell, you will get a 500 error
        </div>
        <a href='{url}' target='_blank' style="
            display: inline-block; background-color: #1a73e8; color: white; padding: 10px 20px;
            text-decoration: none; border-radius: 25px; font-family: sans-serif; font-weight: 500;
            box-shadow: 0 2px 5px rgba(0,0,0,0.2); transition: all 0.2s ease;">
            Open ADK Web UI (after running cell below) ↗
        </a>
    </div>
```

```
    """

    display(HTML(styled_html))

    return url_prefix


print("   Helper functions defined.")
```

`adk create`

```
!adk create research-agent --model gemini-2.5-flash-lite --api_key $GOOGLE_API_KEY
```

- `LlmAgent`
- `root_agent` `google_search_agent`
- `count_papers`

`count_papers`

```
%%writefile research-agent/agent.py

from google.adk.agents import LlmAgent
from google.adk.models.google_llm import Gemini
from google.adk.tools.agent_tool import AgentTool
from google.adk.tools.google_search_tool import google_search

from google.genai import types
from typing import List
```

```
retry_config = types.HttpRetryOptions(
    attempts=5,  # Maximum retry attempts
    exp_base=7,  # Delay multiplier
    initial_delay=1,
    http_status_codes=[429, 500, 503, 504],  # Retry on these HTTP errors
)

# ---- Intentionally pass incorrect datatype - `str` instead of `List[str]` ----
def count_papers(papers: str):
    """
    This function counts the number of papers in a list of strings.
    Args:
      papers: A list of strings, where each string is a research paper.
    Returns:
      The number of papers in the list.
    """
    return len(papers)


# Google Search agent
google_search_agent = LlmAgent(
    name="google_search_agent",
    model=Gemini(model="gemini-2.5-flash-lite", retry_options=retry_config),
    description="Searches for information using Google search",
    instruction="""Use the google_search tool to find information on the given topic. Return the raw search results.
    If the user asks for a list of papers, then give them the list of research papers you found and not the summary.""",
    tools=[google_search]
)


# Root agent
root_agent = LlmAgent(
    name="research_paper_finder_agent",
    model=Gemini(model="gemini-2.5-flash-lite", retry_options=retry_config),
    instruction="""Your task is to find research papers and count them.

    You MUST ALWAYS follow these steps:
    1) Find research papers on the user provided topic using the 'google_search_agent'.
    2) Then, pass the papers to 'count_papers' tool to count the number of papers returned.
    3) Return both the list of research papers and the total number of papers.
    """,
    tools=[AgentTool(agent=google_search_agent), count_papers]
)
```

`adk web --log_level DEBUG`

`--log_level DEBUG`

-
-
-

```
url_prefix = get_adk_proxy_url()
```

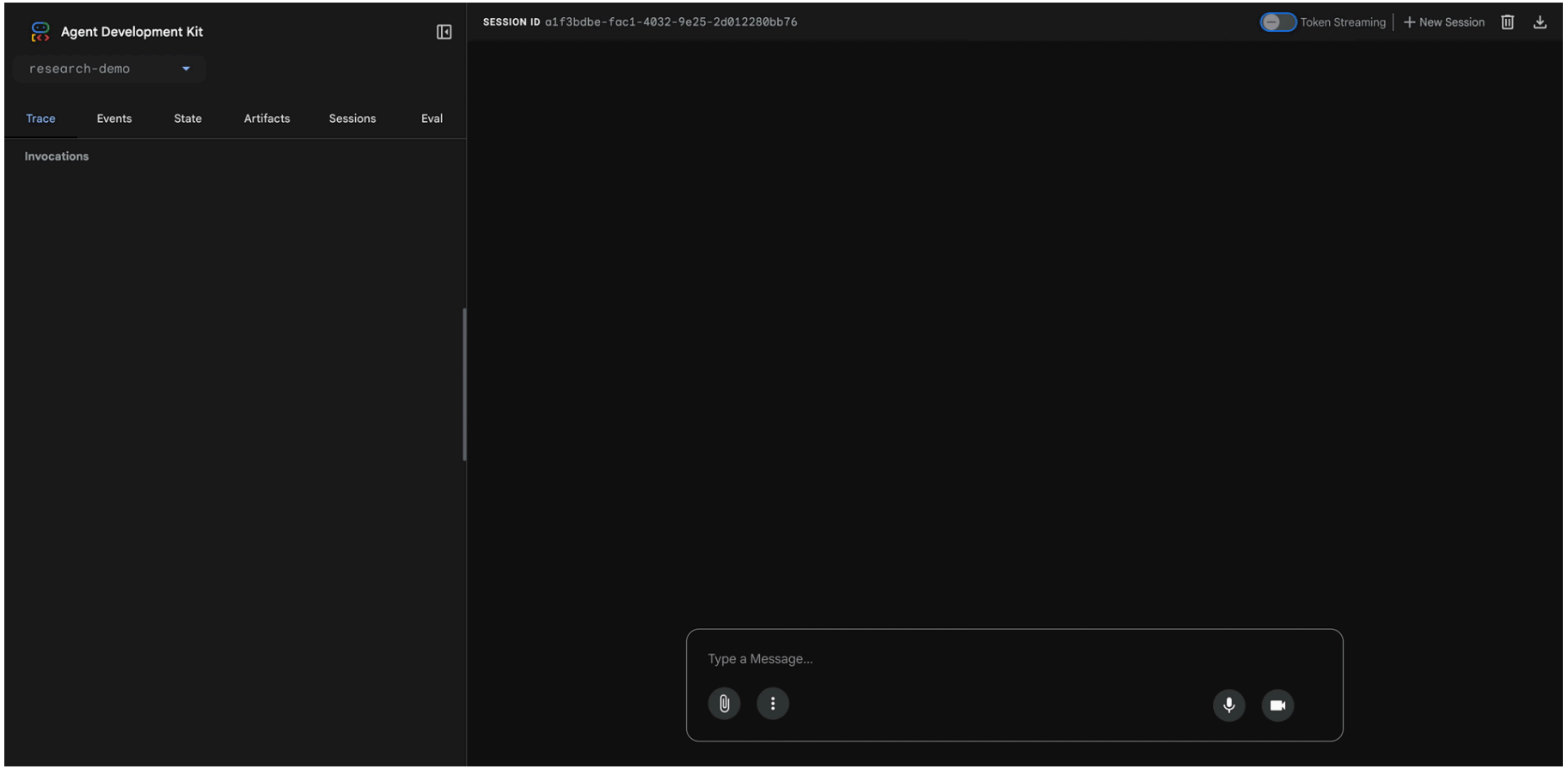
`--log_level`

```
!adk web --log_level DEBUG --url_prefix {url_prefix}
```

1.
2. `Find latest quantum computing papers`
3.

1.
2.
3.
4.
5. `execute_tool count_papers` `count_papers`
6.
7. `call_llm` `count_papers`

- `function_call` `papers`
- `root_agent` `papers`



`papers` `count_papers` `List[str]` `adk web`

Stop the ADK web UI

`adk web`

```
DEBUG - google_adk.models.google_llm - LLM Request: ...
DEBUG - google_adk.models.google_llm - LLM Response: ...
```

```
    # Check the DEBUG logs from the broken agent
    print("🔍 Examining web server logs for debugging clues...\n")
    !cat logger.log
```

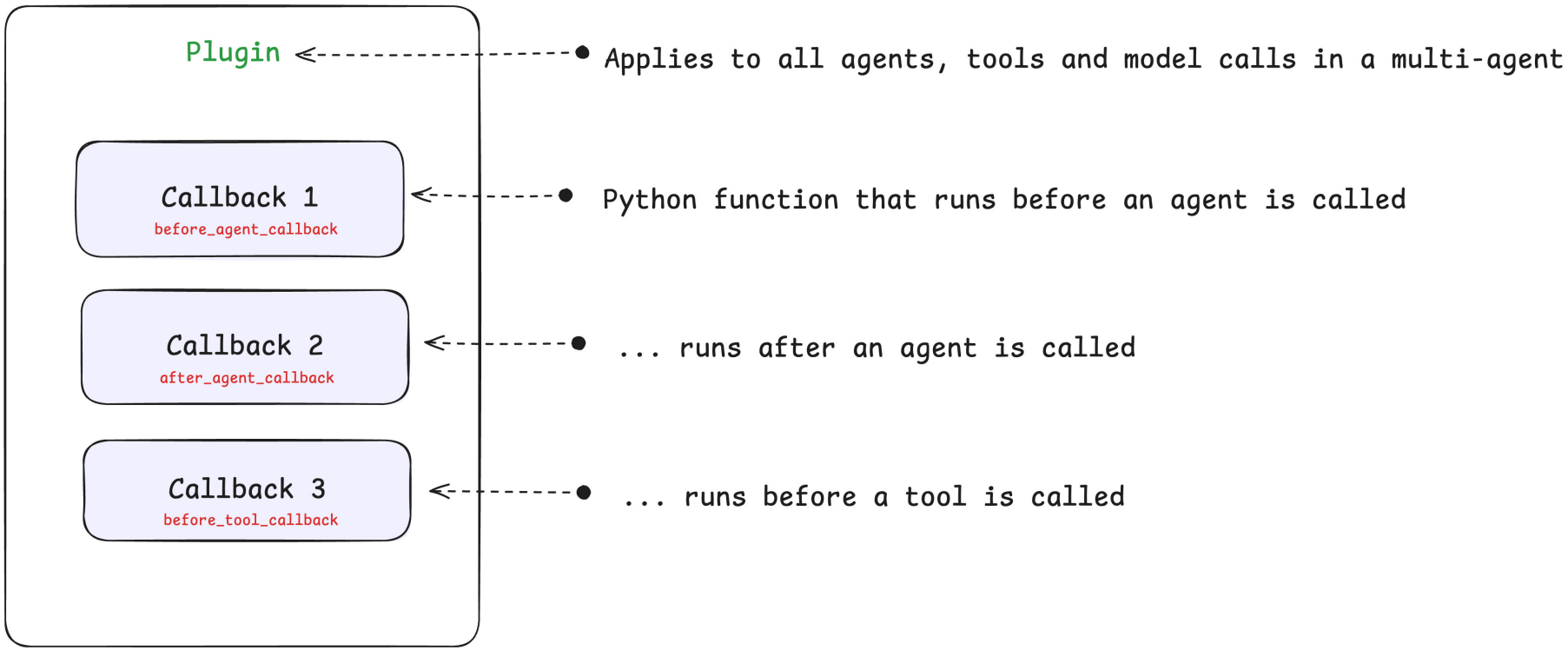
-
-
-
-

`symptom → logs → root cause → fix`

```
You: "Let me open the ADK web UI to check why the agent failed"
DevOps: "Um... this is a production server. No web UI access."
You: 😱 "How do I debug production issues?"
```

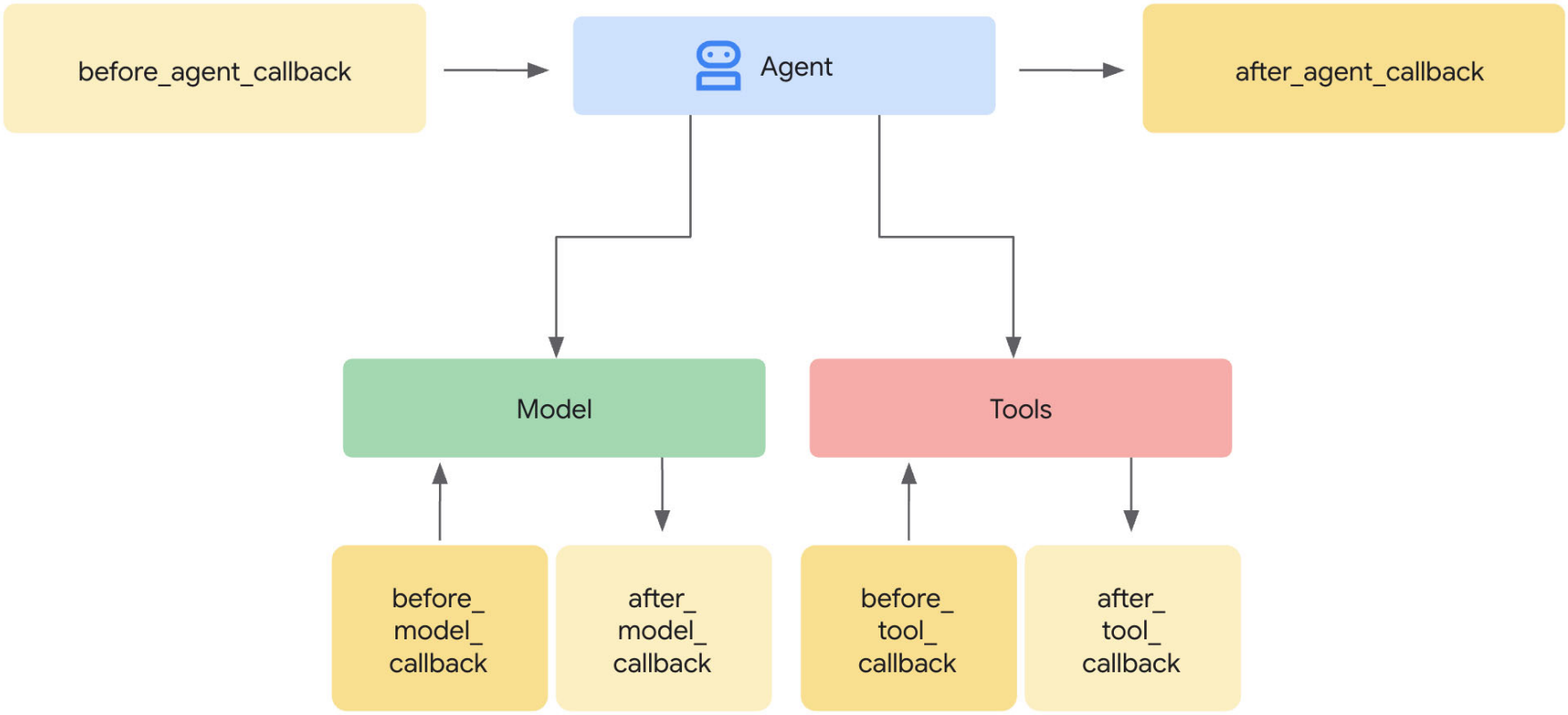
```
You: "The agent runs 1000 times per day in our pipeline"
Boss: "Which runs are slow? What's our success rate?"
```

```
You:    "I'd have to manually check the web UI 1000 times..."
```



Callbacks

-
-
-
-




```
    print("----- EXAMPLE PLUGIN - DOES NOTHING ----- ")

    import logging
    from google.adk.agents.base_agent import BaseAgent
    from google.adk.agents.callback_context import CallbackContext
```

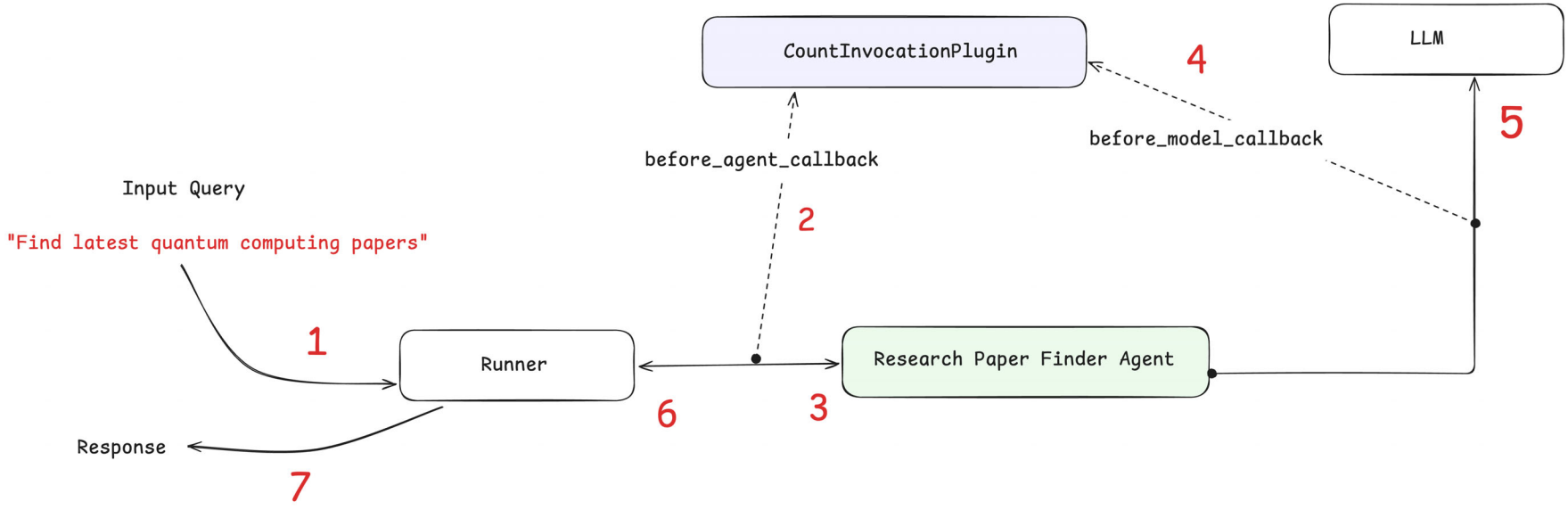
```
from google.adk.models.llm_request import LlmRequest
from google.adk.plugins.base_plugin import BasePlugin


# Applies to all agent and model calls
class CountInvocationPlugin(BasePlugin):
    """A custom plugin that counts agent and tool invocations."""

    def __init__(self) -> None:
        """Initialize the plugin with counters."""
        super().__init__(name="count_invocation")
        self.agent_count: int = 0
        self.tool_count: int = 0
        self.llm_request_count: int = 0

    # Callback 1: Runs before an agent is called. You can add any custom logic here.
    async def before_agent_callback(
        self, *, agent: BaseAgent, callback_context: CallbackContext
    ) -> None:
        """Count agent runs."""
        self.agent_count += 1
        logging.info(f"[Plugin] Agent run count: {self.agent_count}")

    # Callback 2: Runs before a model is called. You can add any custom logic here.
    async def before_model_callback(
        self, *, callback_context: CallbackContext, llm_request: LlmRequest
    ) -> None:
        """Count LLM requests."""
        self.llm_request_count += 1
        logging.info(f"[Plugin] LLM request count: {self.llm_request_count}")
```



The Runner calls the `CountInvocationPlugin` (2) before calling the agent since the plugin has a `before_agent_callback`.
Similarly, the plugin is also executed before an LLM call is made by the agent (4).

`LoggingPlugin`

standard

-
-
-
-
-

Agent definition

```
from google.adk.agents import LlmAgent
from google.adk.models.google_llm import Gemini
from google.adk.tools.agent_tool import AgentTool
from google.adk.tools.google_search_tool import google_search
```

```
from google.genai import types
from typing import List

retry_config = types.HttpRetryOptions(
    attempts=5,  # Maximum retry attempts
    exp_base=7,  # Delay multiplier
    initial_delay=1,
    http_status_codes=[429, 500, 503, 504],  # Retry on these HTTP errors
)


def count_papers(papers: List[str]):
    """
    This function counts the number of papers in a list of strings.
    Args:
      papers: A list of strings, where each string is a research paper.
    Returns:
      The number of papers in the list.
    """
    return len(papers)


# Google search agent
google_search_agent = LlmAgent(
    name="google_search_agent",
    model=Gemini(model="gemini-2.5-flash-lite", retry_options=retry_config),
    description="Searches for information using Google search",
    instruction="Use the google_search tool to find information on the given topic. Return the raw search results.",
    tools=[google_search],
)

# Root agent
research_agent_with_plugin = LlmAgent(
    name="research_paper_finder_agent",
    model=Gemini(model="gemini-2.5-flash-lite", retry_options=retry_config),
    instruction="""Your task is to find research papers and count them.

   You must follow these steps:
   1) Find research papers on the user provided topic using the 'google_search_agent'.
   2) Then, pass the papers to 'count_papers' tool to count the number of papers returned.
   3) Return both the list of research papers and the total number of papers.
   """,
    tools=[AgentTool(agent=google_search_agent), count_papers],
)

print("  Agent created")
```

`InMemoryRunner`

`LoggingPlugin`

1.
2. `InMemoryRunner`

```
from google.adk.runners import InMemoryRunner
from google.adk.plugins.logging_plugin import (
    LoggingPlugin,
)  # <---- 1. Import the Plugin
from google.genai import types
import asyncio

runner = InMemoryRunner(
    agent=research_agent_with_plugin,
    plugins=[
        LoggingPlugin()
    ],  # <---- 2. Add the plugin. Handles standard Observability logging across ALL agents
)

print("✅ Runner configured")
```

`run_debug`

```
print("🚀 Running agent with LoggingPlugin...")
print("📊 Watch the comprehensive logging output below:\n")
```