Agentic AI Orchestration System using Google Cloud

Objective

Design and implement an **Agentic AI system** that:

- Orchestrates prompts across multiple LLMs (Large Language Models)
- Selects the most accurate LLM output
- Supports **model-based prompt routing**
- Allows **seamless hot-swapping of LLMs** without downtime

Background

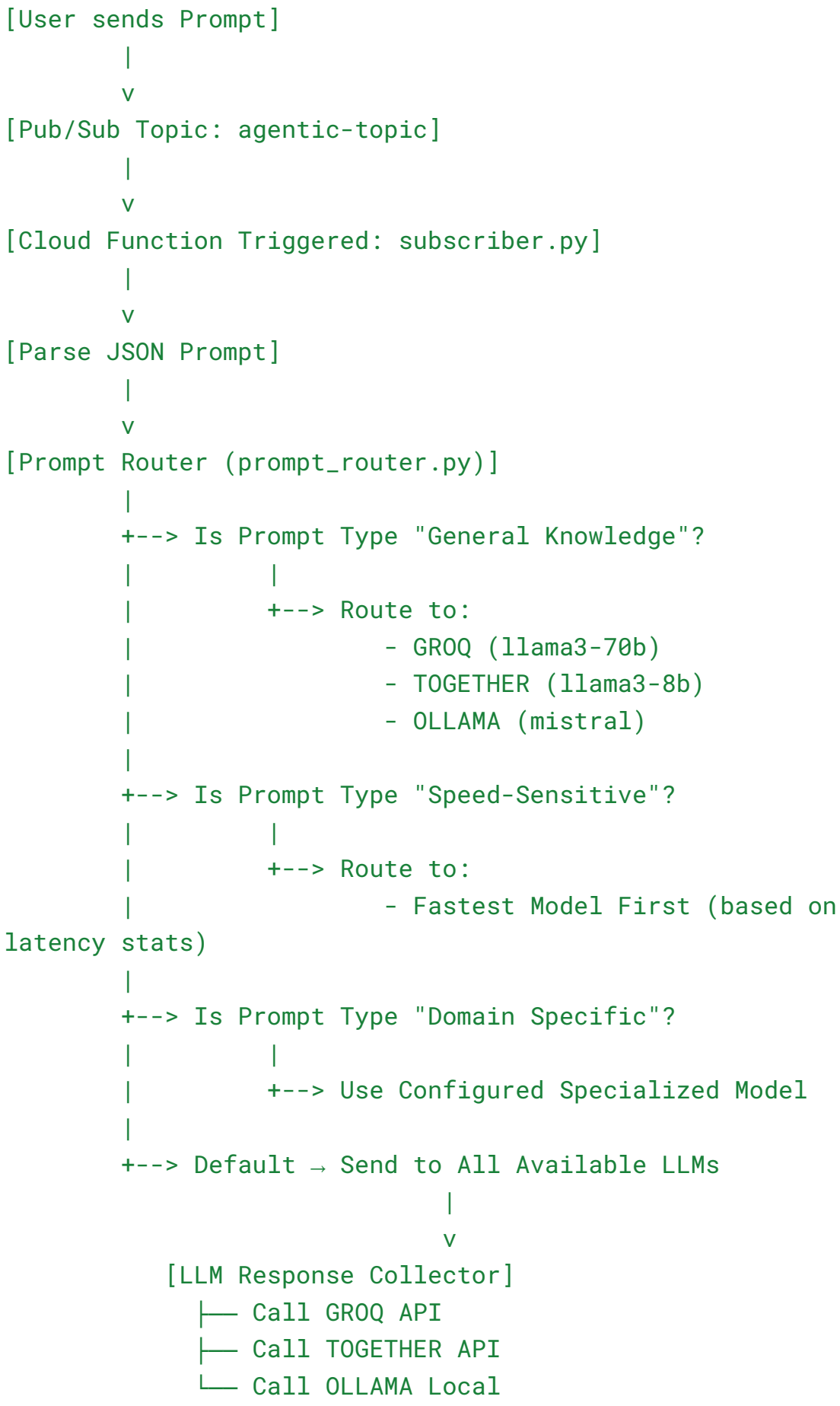
In real-world AI agent systems, using multiple LLMs (e.g., **Gemini, PaLM, Ollama, Groq, Together AI**) is common:

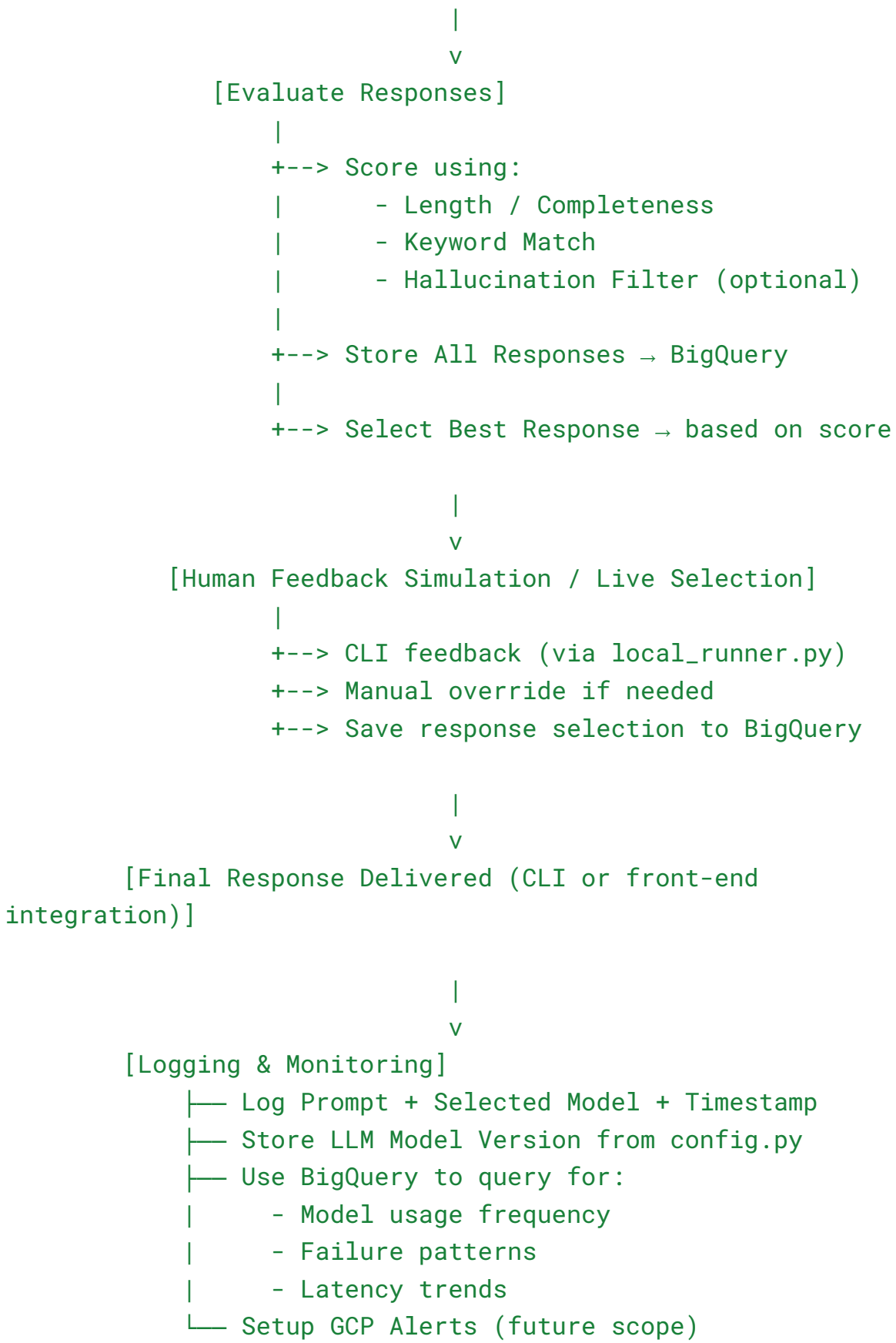
- For comparing outputs and quality
- For fallback and redundancy
- For domain-specific optimizations (e.g., technical answers from one, creative from another)

As models evolve, they are upgraded or replaced frequently. This architecture ensures **flexibility, modularity, and robustness**.

Architecture Design

High-Level Architecture Overview


[User sends Prompt]
|
v
[Pub/Sub Topic: agentic-topic]
|
v
[Cloud Function Triggered: subscriber.py]
|
v
[Parse JSON Prompt]
|
v
[Prompt Router (prompt_router.py)]
|
+--> Is Prompt Type "General Knowledge"?
| |
| +--> Route to:
| - GROQ (llama3-70b)
| - TOGETHER (llama3-8b)
| - OLLAMA (mistral)
|
+--> Is Prompt Type "Speed-Sensitive"?
| |
| +--> Route to:
| - Fastest Model First (based on
latency stats)
|
+--> Is Prompt Type "Domain Specific"?
| |
| +--> Use Configured Specialized Model
|
+--> Default → Send to All Available LLMs
|
v
[LLM Response Collector]
├── Call GROQ API
├── Call TOGETHER API
└── Call OLLAMA Local


[Evaluate Responses]
+--> Score using:
- Length / Completeness
- Keyword Match
- Hallucination Filter (optional)
+--> Store All Responses → BigQuery
+--> Select Best Response → based on score
[Human Feedback Simulation / Live Selection]
+--> CLI feedback (via local_runner.py)
+--> Manual override if needed
+--> Save response selection to BigQuery
[Final Response Delivered (CLI or front-end integration)]
[Logging & Monitoring]
├── Log Prompt + Selected Model + Timestamp
├── Store LLM Model Version from config.py
├── Use BigQuery to query for:
- Model usage frequency
- Failure patterns
- Latency trends
└── Setup GCP Alerts (future scope)

```
                        |
                        v
[Model Version Management]
    ├── config.py holds model version names
    ├── To hot-swap:
    |     - Update version in config
    |     - Re-deploy only affected parts
    └── New models tested via local_runner.py
```

Prompt Orchestration Workflow

1. **User** submits a prompt through CLI/Publisher.
2. **Pub/Sub Topic** receives the prompt and triggers `run_agents` Cloud Function.
3. The **Prompt Router** reads the config and routes the prompt to enabled LLMs (based on type, availability, latency).
4. All **LLM outputs** are collected.
5. **Response evaluation logic** selects the best answer using:
    - Length
    - Coherence
    - Confidence (if available)
    - Optional: human feedback (in local version)
6. Results are **logged into BigQuery**.

LLM Version Tracking & Hot-Swapping

Version Tracking Implementation

In `llms/config.py`:

```
def get_model_config():

    return {
        "groq": {"enabled": True, "model": "llama3-70b-8192",
"version": "v1.0"},
        "together": {"enabled": True, "model":
"meta-llama/Llama-3-8b-chat-hf", "version": "v1.0"},
        "ollama": {"enabled": True, "model": "mistral",
"version": "v1.1"}
    }
```

This ensures:

- Each model's version is logged along with the response
- Easy rollback and upgrade tracking

Hot-Swapping Strategy

When upgrading models:

- New model is added to `config.py` with `enabled: False`
- A **validation run** is performed manually/local
- On success, `enabled: True` is updated → traffic automatically routes to it
- Zero downtime and **no function redeploy** is required

Prompt Routing Logic

In `prompt_router.py`:

- You can implement **routing rules** such as:
    - If `type == "creative"`, prefer GROQ or Together
    - If `low latency` is required, skip OLLAMA (if local)
    - If model is marked disabled in config, skip it

```
def should_use_model(model_info, prompt_type):
    if not model_info["enabled"]:
        return False
    if prompt_type == "fast" and model_info["model"] ==
"ollama":
        return False
    return True
```

Logging, Monitoring & Evaluation

Logging in BigQuery

Logged Fields:

- Prompt
- Model Name + Version
- Response
- Latency (if available)
- Selection Outcome (best model)
- Timestamp

Monitoring

- View logs in BigQuery

- Track which models are frequently selected
- Use response lengths or user feedback to detect regressions

---

Human-in-the-loop Evaluation (Local Only)

In `local_runner.py`:

- User is shown all LLM responses in terminal
- User manually selects the best response
- This choice is also **logged** for training future evaluation logic

---

Bonus: Implementation Sketch

Pub/Sub Publisher:

bash
CopyEdit

```
gcloud pubsub topics publish agentic-topic \
--message="{\"prompt\": \"Explain agentic AI\"}"
```

Cloud Function (`main.py`):

- Reads from Pub/Sub
- Routes prompt via `prompt_router`
- Gets responses
- Stores all in BigQuery

BigQuery Table:

- `project.dataset.responses`
- Fields: prompt, response, model, version, time, is_best, user_rating (optional)

API/Service Selection

Component	Technology	Reason
Trigger Mechanism	Pub/Sub	Decouples publisher and agents
Orchestration Logic	Cloud Functions	Serverless, scalable, managed
Model Routing	Python + Config File	Modular and editable routing
Data Logging	BigQuery	Fast analytics and cheap storage
LLMs	Groq, Together, Ollama	Variety and fallback/resilience
Evaluation	Custom Python Logic	Flexible best-answer selection
Version Control	Config.py versions	Enables hot-swapping

Key Features Achieved

- Modular LLM config with versions
- Routing to multiple models
- Best answer selection
- Cloud-native (GCP Pub/Sub, Cloud Functions, BigQuery)
- Logging and Monitoring
- Human-in-the-loop support

- Hot-swapping with zero downtime
