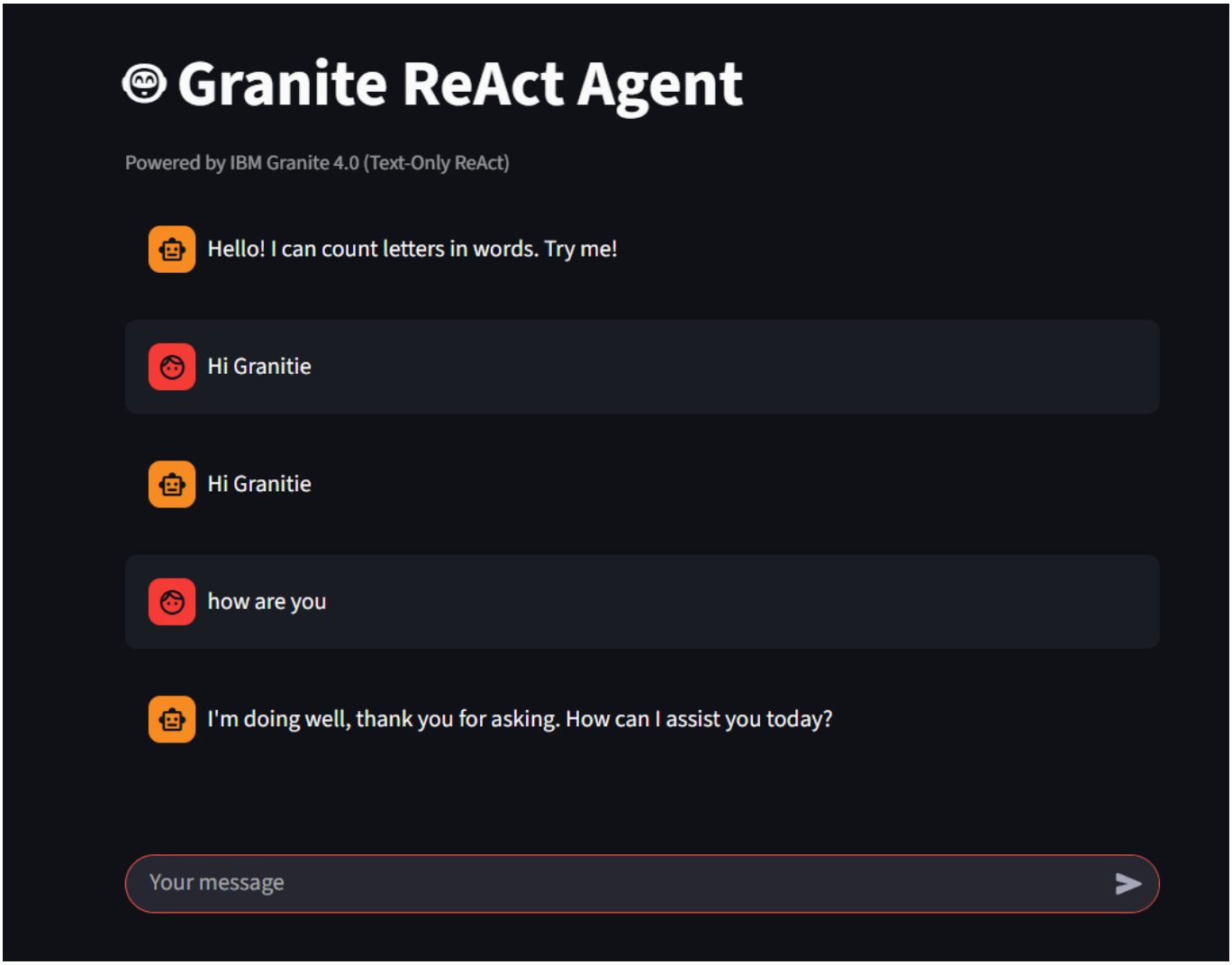
IBM Granite ReAct Agent

A modular AI agent application utilizing the **IBM Granite 4.0** model (via OpenRouter) to perform reasoning tasks. This project uses a modern Python stack with **LangGraph** for agent orchestration, **Streamlit** for the frontend, **uv** for high-speed dependency management, and **Docker** for containerization.



Features

- **Model:** IBM Granite 4.0 (H-Micro variant) via OpenRouter.
- **Architecture:** Implements the **ReAct** (Reasoning + Acting) pattern using **LangGraph**.
- **Interface:** Clean, chat-based UI using **Streamlit**.
- **Tools:** Includes a demonstration tool (`WordLength`) to show how the agent can "think" and execute Python functions.
- **Infrastructure:** Fully Dockerized with live volume mounting.

Quick Start (Run All)

The fastest way to run the entire stack (**Streamlit App** + **Jupyter Notebooks**) is using Docker Compose.

1. **Configure Environment:**

```
cp .env.example .env
# Open .env and add your OPENROUTER_API_KEY
```

2. **Run All Services:**

```
docker-compose up --build
```

3. **Access:**
   - **Agent App:** http://localhost:7777
   - **Notebooks:** http://localhost:7799 (No password required)

Note: Since the current directory is mounted to `/app`, any changes you make to `agent.py` or notebooks locally will immediately exist inside the container.

🛠️ Prerequisites

Before starting, ensure you have the following installed:

1. **uv**: A fast Python package installer and resolver.

```
# MacOS/Linux
curl -LsSf [https://astral.sh/uv/install.sh]
(https://astral.sh/uv/install.sh) | sh
# Windows
powershell -c "irm [https://astral.sh/uv/install.ps1]
(https://astral.sh/uv/install.ps1) | iex"
```

2. **Docker**: Required for the multi-container setup.
3. **OpenRouter API Key:** Sign up at OpenRouter.ai to access IBM Granite models.

🚀 Local Development (Manual)

If you prefer to run components individually without Docker:

1. Configuration

Create a `.env` file in the root directory:

```
OPENROUTER_API_KEY=sk-or-v1-your-actual-key-here
MODEL_NAME=ibm-granite/granite-4.0-h-micro
```

2. Install Dependencies

```
# This installs main dependencies AND dev dependencies (jupyterlab)
uv sync --dev
```

3. Run Components

Option A: Run just the Agent App

```
uv run streamlit run app.py --server.port=7777
```

- Access: http://localhost:7777

Option B: Run Jupyter Notebooks Locally

```
uv run jupyter lab --port=7799
```

- Access: http://localhost:7799

Docker Architecture

This project uses `docker-compose` to orchestrate the environment with live mounting.

Service Breakdown

Service	Host Port	Purpose	Live Mount
app	**7777**	Streamlit Chat UI	Yes (`.:/app`)
jupyter	**7799**	Dev Notebooks	Yes (`.:/app`)

Commands

- **Start All:** `docker-compose up --build`
- **Stop All:** `docker-compose down`
- **Rebuild:** `docker-compose build --no-cache`

Project Structure

```
.
├── .dockerignore       # Security: Excludes secrets and venvs from Docker builds
├── .env                # Config: API keys (Git ignored)
├── docker-compose.yml  # Infra: Orchestrates App + Jupyter
```

```
├── Dockerfile          # Infra: App container definition
├── Dockerfile.jupyter  # Infra: Jupyter container definition
├── README.md           # Docs: This file
├── agent.py            # Logic: LangGraph setup, Tools, and ReAct Prompt
├── app.py              # UI: Streamlit chat interface
├── notebooks/          # Dev: Folder for Jupyter notebooks
├── pyproject.toml      # Deps: Project dependencies managed by uv
└── uv.lock             # Deps: Locked versions for reproducibility
```

Notebooks Suite

The `notebooks/` folder contains a progressive suite of experiments to test, benchmark, and extend the agent.

1. **`01_sanity_check.ipynb`**: Verifies connectivity, imports, and basic tool usage (`WordLength`).
2. **`02_prompt_engineering.ipynb`**: Experiments with different system prompts (e.g., "Pirate Mode") to alter behavior without changing code.
3. **`03_token_counting.ipynb`**: Analyzes token usage per query using `tiktoken` for cost estimation.
4. **`04_synthetic_data.ipynb`**: Generates a synthetic dataset of edge-case words using the LLM itself.
5. **`05_benchmark.ipynb`**: Runs the agent against the synthetic data to measure accuracy and latency.
6. **`06_context_engineering_multi_agent.ipynb`**: Advanced demo of a Supervisor-Worker architecture with context injection (Context Engineer -> Supervisor -> Specialist).

How It Works (Architecture)

The Challenge

The `ibm-granite/granite-4.0-h-micro` model is highly efficient but does not support the native "Function Calling" API standard used by OpenAI or Anthropic.

The Solution: Text-Based ReAct

This project uses **Prompt Engineering** to teach the model how to use tools.

1. **System Prompt (`agent.py`)**: We inject a strict instruction set telling the model to format its output as `Thought: ... Action: ... Action Input: ...`.
2. **LangGraph Workflow**:
   - **Node 1 (Agent):** Generates text.
   - **Node 2 (Tool):** A regex parser scans the agent's text. If it finds `Action: WordLength`, it executes the Python function.
   - **Loop:** The result is fed back to the agent as an `Observation`.
3. **Streamlit UI (`app.py`)**: Displays the conversation and the final answer.

🔧 Customization

Adding New Tools

To give the agent new capabilities (e.g., a calculator or search), modify `agent.py`:

1. Define a new Python function.
2. Add it to the `TOOL_MAP` dictionary.
3. Update the `REACT_SYSTEM_PROMPT` string to describe the new tool to the AI.

Changing the Model

To use a different OpenRouter model, update your `.env` file:

```
MODEL_NAME=meta-llama/llama-3-8b-instruct
```

🧠 Roadmap / To-Do

Here are potential improvements for future versions of this agent:

- [ ] **Model Context Protocol (MCP):** Implement standard interfaces to connect external tools and data sources safely.
- [ ] **Knowledge Base (RAG):** Integrate a Vector Database (like Chroma or Pinecone) to allow the agent to query documents and external knowledge.
- [ ] **Multi-Agent System:** Expand the architecture to use specialized sub-agents (e.g., a Researcher Agent and a Writer Agent) coordinated by a supervisor.
- [ ] **Multi-Modal Capabilities:** Enable the agent to accept and process images or audio inputs alongside text.
- [ ] **Database Persistence:** Save chat history and sessions to a SQLite or PostgreSQL database.