MALLO: MultiAgent LLM Orchestrator

Tabla de Contenidos

Descripción

MALLO (MultiAgent LLM Orchestrator) es un sistema avanzado de orquestación de múltiples agentes de Modelos de Lenguaje de Gran Escala (LLMs). Diseñado para manejar consultas complejas, MALLO utiliza una variedad de agentes, desde modelos locales hasta asistentes especializados y APIs de terceros, para proporcionar respuestas precisas y contextuales.
El sistema está construido con un enfoque modular y flexible, permitiendo la integración de diversos modelos de lenguaje y servicios. MALLO evalúa la complejidad de cada consulta y selecciona dinámicamente el agente o combinación de agentes más apropiados para procesarla, optimizando así la calidad y relevancia de las respuestas.

Características Principales

- **Integración Multifacética de LLMs**:
- Modelos locales de Ollama para procesamiento rápido y offline.
- APIs de OpenAI para tareas que requieren capacidades avanzadas.
- Integración con otros proveedores de LLM como Groq, Together, DeepInfra, Anthropic, DeepSeek, Mistral y Cohere.
- **Asistentes Especializados**:
- Conjunto de asistentes entrenados para dominios específicos como derecho, ética, tecnología, entre otros.
- **Búsqueda Web Integrada**:
- Utilización de DuckDuckGo para enriquecer las respuestas con información actualizada de la web.
- **Procesamiento Avanzado de Texto**:
- Análisis de complejidad de consultas.
- Resumen automático de textos largos.

- Extracción de palabras clave y entidades nombradas.
- **Manejo Inteligente de Recursos**:
- Selección dinámica de agentes basada en la complejidad de la consulta y disponibilidad de recursos.
- Sistema de caché para optimizar el rendimiento y reducir la carga en las APIs.
- **Interfaz de Usuario Intuitiva**:
- Implementada con Streamlit para una experiencia de usuario fluida y responsive.
- **Evaluación y Mejora Continua**:
- Sistema de evaluación de respuestas para garantizar la calidad y relevancia.
- Capacidad de aprendizaje y mejora basada en el feedback del usuario.
- **Logging y Monitoreo**:
- Sistema robusto de logging para seguimiento de errores y rendimiento.
- **Configuración Flexible**:
- Archivo de configuración YAML para fácil ajuste y personalización del sistema.

Estructura del Proyecto

```
MALLO/
■
■■■ agents.py # Implementación de la clase AgentManager y lógica de agentes
■■■ config.yaml # Configuración general del sistema
■■■ main.py # Punto de entrada principal y UI de Streamlit
■■■ README.md # Documentación del proyecto (este archivo)
■■■ utilities.py # Funciones de utilidad y helpers
■■■ model_speeds.json # Índice de velocidad de modelos locales y en la nube
■■■ CHANGELOG.md # Registro de cambios y versiones
■
■■■ .streamlit/
■ ■■■ secrets.toml # Almacenamiento seguro de claves API (no incluido en el repositorio)
■
■■■ tests/ # Directorio para pruebas unitarias y de integración
■
■■■ logs/ # Directorio para archivos de log
■
■■■ assets/ # Directorio para recursos estáticos (imágenes, estilos, etc.)
|
■■■ tools/ # Directorio utilidades externas
■ ■■■ test_model_speeds.py # Pequeño script para medir la velocidad de los modelos
|
■■■ requirements.txt # Dependencias del proyecto
```

Requisitos Previos

- Python 3.8+
- Ollama instalado localmente (para modelos locales)
- Claves API para servicios externos (OpenAI, Groq, Together, etc.)
- Conexión a Internet (para búsqueda web y APIs externas)

Instalación

1. Clona el repositorio:
```
```

```
git clone https://github.com/bladealex9848/MALLO.git
cd MALLO
```
2. Crea y activa un entorno virtual:
```
python3 -m venv venv
source venv/bin/activate # En Windows usa `venv\Scripts\activate`
```
3. Instala las dependencias:
```
pip install -r requirements.txt
```
4. Configura tus claves API:
Crea un archivo `.streamlit/secrets.toml` y añade tus claves API:
```toml
OPENAI_API_KEY = "tu_clave_openai_aqui"
GROQ_API_KEY = "tu_clave_groq_aqui"
TOGETHER_API_KEY = "tu_clave_together_aqui"
DEEPINFRA_API_KEY="tu_clave_deepinfra_aqui"
ANTHROPIC_API_KEY="tu_clave_anthropic_aqui"
DEEPSEEK_API_KEY="tu_clave_deepseek_aqui"
MISTRAL_API_KEY="tu_clave_mistral_aqui"
COHERE_API_KEY="tu_clave_cohere_aqui"
# Añade otras claves API según sea necesario
```

Configuración

El archivo `config.yaml` contiene la configuración principal del sistema. Aquí puedes ajustar:
- Modelos disponibles para cada proveedor de LLM
- Prioridades de procesamiento
- Umbrales de complejidad para la selección de agentes
- Configuración de asistentes especializados
- Parámetros de utilidades como búsqueda web y análisis de imágenes
Ejemplo de configuración:
```yaml
ollama:
base_url: "http://localhost:11434"
default_model: "gemma2:2b"
models:
- "gemma2:2b"
- "llava:latest"
openai:
default_model: "gpt-4o-mini"
models:
- "gpt-4o-mini"

# ... (otras configuraciones)

thresholds:
moa_complexity: 0.8
local_complexity: 0.5
web_search_complexity: 0.4
```

```

## Uso

Para iniciar la aplicación, ejecuta:
```
streamlit run main.py
```
Luego, abre tu navegador y ve a `http://localhost:8501`.
La interfaz de usuario te permitirá:
1. Ingresar consultas en lenguaje natural.
2. Ver las respuestas generadas por el sistema.
3. Explorar detalles sobre el procesamiento de cada consulta.
4. Acceder a información sobre el estado del sistema y rendimiento de los agentes.

## Arquitectura del Sistema

MALLO utiliza una arquitectura de microservicios basada en agentes, donde cada agente (modelo de lenguaje o servicio especializado) puede procesar consultas de manera independiente. El componente central, `AgentManager`, orquesta estos agentes basándose en la complejidad de la consulta y la disponibilidad de recursos.
### Componentes Clave:
1. **Orquestador de Agentes**: Selecciona y gestiona los agentes apropiados para cada consulta.
2. **Evaluador de Complejidad**: Analiza las consultas para determinar su complejidad y requisitos.
3. **Sistema de Caché**: Almacena y recupera respuestas para consultas frecuentes.
4. **Módulo de Búsqueda Web**: Enriquece las respuestas con información actual de la web.
5. **Interfaz de Usuario**: Proporciona una experiencia interactiva para los usuarios.

## Componentes Principales

### agents.py
- **Clase AgentManager**:
- Gestiona la inicialización y selección de agentes.
- Implementa la lógica de procesamiento de consultas.
- Maneja la integración con diferentes APIs de LLM.
### utilities.py
- Funciones para evaluación de complejidad de consultas.
- Implementación de búsqueda web.
- Sistema de caché para respuestas.
- Funciones de logging y manejo de errores.
### main.py
- Implementa la interfaz de usuario con Streamlit.
- Gestiona el flujo principal de la aplicación.
- Procesa las entradas del usuario y muestra las respuestas.

## Flujo de Trabajo

1. El usuario ingresa una consulta a través de la interfaz de Streamlit.
2. La consulta se evalúa para determinar su complejidad y requisitos.
3. Se selecciona el agente o conjunto de agentes más apropiados.
4. Si es necesario, se realiza una búsqueda web para enriquecer el contexto.
5. Los agentes seleccionados procesan la consulta.
6. Se evalúa y refina la respuesta generada.
```

7. La respuesta final se presenta al usuario junto con detalles del proceso.

APIs y Servicios Integrados

MALLO integra varios servicios de LLM y APIs, incluyendo:
- OpenAI
- Groq
- Together
- DeepInfra
- Anthropic
- DeepSeek
- Mistral
- Cohere

Cada servicio se inicializa y gestiona a través de la clase `AgentManager`, permitiendo una fácil expansión a nuevos proveedores en el futuro.

Manejo de Errores y Logging

El sistema implementa un robusto sistema de logging y manejo de errores:
- Los errores se registran en el archivo `mallo.log`.
- Se utilizan diferentes niveles de logging (INFO, WARNING, ERROR) para categorizar los eventos.
- Los errores críticos se muestran al usuario a través de la interfaz de Streamlit.

Optimización y Caché

Para mejorar el rendimiento y reducir la carga en las APIs externas, MALLO implementa:
- Un sistema de caché para almacenar respuestas frecuentes.
- Evaluación de complejidad para evitar el uso innecesario de recursos en consultas simples.
- Selección inteligente de agentes basada en la disponibilidad y rendimiento histórico.

Pruebas

El proyecto incluye un conjunto de pruebas unitarias y de integración en el directorio `tests/`. Para ejecutar las pruebas:

```
python -m unittest discover tests
```

Contribución

Las contribuciones son bienvenidas. Por favor, sigue estos pasos:
1. Haz fork del repositorio.
2. Crea una nueva rama (`git checkout -b feature/AmazingFeature`).
3. Realiza tus cambios y haz commit (`git commit -m 'Add some AmazingFeature'`).
4. Push a la rama (`git push origin feature/AmazingFeature`).
5. Abre un Pull Request.

Registro de Cambios

Consulta CHANGELOG.md para ver el historial detallado de cambios del proyecto.

Roadmap

- Implementación de más modelos de lenguaje y APIs.
- Mejora del sistema de evaluación y selección de agentes.
- Desarrollo de una API REST para integración con otros sistemas.
- Implementación de fine-tuning para modelos específicos de dominio.

Licencia

Este proyecto está licenciado bajo la Licencia MIT - vea el archivo LICENSE.md para más detalles.

Contacto

Alexander Oviedo Fadul - alexander.oviedo.fadul@gmail.com
Enlace del proyecto:
https://github.com/bladealex9848/MALLO
[GitHub](https://github.com/bladealex9848) | [Website](https://alexander.oviedo.isabellaea.com/) | [Instagram](https://www.instagram.com/alexander.oviedo.fadul) |
[Twitter](https://twitter.com/alexanderofadul) | [Facebook](https://www.facebook.com/alexanderof/) | [WhatsApp](https://api.whatsapp.com/send?phone=573015930519&text;=Hola%20!Quiero%20conversar%20contigo!) | [LinkedIn](https://www.linkedin.com/in/alexander-oviedo-fadul-49434b9a/)