**Project Documentation: AI Agents Inference Benchmarking**

**Overview**

The project "OmdenaKnowledge\_AIAgentsInferenceBenchmarking" is designed to benchmark AI agent inference performance using CrewAI, LangGraph, and AutoGen with Groq models. It evaluates the efficiency of text generation tasks based on various input queries, measuring factors such as latency, memory usage, and token consumption.

**Objectives**

* To generate structured and engaging paragraphs from input keywords.
* To benchmark the performance of AI models by analyzing latency, memory usage, and token utilization.
* To store and evaluate the quality of AI-generated text using an automated rating mechanism.
* To compare multiple frameworks (CrewAI, LangGraph, AutoGen) for inference efficiency.
* To store benchmarking results in a structured CSV format.

**Key Components**

**1. Text Generation Pipeline**

The text\_generation function in crewai\_benchmark.py, langgraph\_benchmark.py , autogen\_benchmark.py orchestrates the benchmarking process:

* Loads configuration settings.
* Initializes AI models using Groq.
* Executes text generation tasks using CrewAI, LangGraph, and AutoGen.
* Captures and analyzes performance metrics, including memory and latency.
* Rates the quality of generated text.
* Stores the benchmarking results in a CSV file.

**2. Framework-Specific Implementations**

**CrewAI Implementation**

* Uses agent-based collaboration to generate and refine text.
* agents.py: Defines AI agents with roles and objectives.
* tasks.py: Specifies content generation tasks with constraints.
* Implements structured task execution workflows.

**LangGraph Implementation**

* Employs graph-based execution for text generation.
* Defines nodes representing various processing steps.
* Uses graph.py to model AI workflows and dependencies.

**AutoGen Implementation**

* Implements an agent-based workflow with AutoGen.
* Uses UserProxyAgent and AssistantAgent for task execution.
* Handles interactions between agents to generate optimized text.

**3. Configuration Management**

* config.yaml: Stores AI model parameters, prompts, benchmarking keywords, and CSV configuration.
* config\_loader.py: Loads and processes configuration settings, integrating API keys from environment variables.

**4. Benchmarking and Evaluation**

* common\_functions.py:
  + save\_results\_to\_csv: Saves benchmarking results with summary statistics.
  + rate\_paragraph: Rates generated text using a Groq-based AI model.
* Metrics considered:
  + Latency per keyword.
  + Memory usage (peak and delta).
  + Token consumption (input, output, and total).
  + AI-generated paragraph rating.
* Comparative analysis of CrewAI, LangGraph, and AutoGen performance.

**5. Setup and Dependencies**

* setup.py: Defines the project package structure and dependencies.
* Requires Python 3.12 or later.
* Utilizes AutoGen, Groq, tiktoken, pandas, CrewAI, and LangGraph libraries.

**Workflow Execution**

1. Load configuration settings from config.yaml.
2. Initialize AI models and encoding settings.
3. Execute text generation for predefined benchmark keywords using CrewAI, LangGraph, and AutoGen workflows.
4. Monitor performance metrics and rate generated outputs.
5. Store results in CSV format.
6. Display summarized benchmarking statistics.

**Expected Outcomes**

* A structured evaluation of AI model efficiency in text generation.
* A comparative analysis of latency, memory footprint, and token usage across CrewAI, LangGraph, and AutoGen.
* An automated quality rating system for generated content.
* A CSV-based benchmarking report for further analysis.

**Potential Enhancements**

* Integration of additional AI models for comparative benchmarking.
* Expansion of evaluation criteria (e.g., sentiment analysis, readability scores).
* Optimization of token usage and memory management for enhanced efficiency.
* Further refinement of task execution strategies across different frameworks.