The Journal of Open Source Software

LLMOrchestrator: A Multi-Model LLM Orchestration Framework for Reducing Bias and Iterative Reasoning

Trisanth Srinivasan [1]

1 Cyrion Labs

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

LLMOrchestrator is a Python package for studying Large Language Models by orchestrating multiple models in defined roles—generation, verification, and refinement. It integrates API-based and local models (e.g., OpenAI, Hugging Face Transformers) under a unified workflow. Researchers can manage prompts, execute iterative reasoning loops, and monitor performance metrics like processing times and quality scores to uncover biases and emergent behaviors. The framework simplifies complex experiments through prompt templates, parallel execution, and caching, enabling reproducible, transparent analysis for AI safety, multi-agent simulations, and model benchmarking.

Statement of Need

Research on LLMs extends beyond single prompts; it demands diverse perspectives, rigorous validation, and multi-step reasoning. **LLMOrchestrator** offers a flexible environment supporting custom workflows, systematic verification, and comparative studies, helping researchers navigate experimental complexity with reproducibility and efficiency.

Core Capabilities

- **Model Abstraction:** `OpenAIModel` and `LocalModel` classes wrap API calls or local inference.
- **Custom Generators/Verifiers:** Plug in Python-based logic or LLMs for generation and evaluation.
- **Iterative Refinement:** Controlled loops guided by `max_iterations` and `max_verifications` parameters.
- **Prompt Management:** `PromptTemplate` for systematic prompt variation.
- **Performance Monitoring:** `ValidationMetrics` track token counts, times, and verifier scores.
- **Parallel Execution & Caching:** `execute_parallel` and `OutputCache` accelerate development and large-scale experiments.

Architecture Overview

Controller

The `Controller` orchestrates experiments. Instantiate with generator, verifier, and iteration settings:

```
from LLMOrchestrator.controller import Controller
controller = Controller(
    generator=generator_model,
    verifier=verifier_logic,
    max_iterations=5,
    max_verifications=3,
    monitoring_enabled=True
)
```

Models

Use `OpenAIModel(model_name)` for API calls or `LocalModel(model_name, device)` for local inference. Mix and match for diverse roles.

Generator & Verifier

Define generation and evaluation logic. Example verifier checks required keywords:

```
def keyword_check(text, prompt=None):
    keywords = ['research','llm','framework']
    passed = all(kw in text.lower() for kw in keywords)
    return passed, ('{"score":1.0}' if passed else '{"score":0.0}')
keyword_verifier = Verifier(custom_verifier=keyword_check)
```

Iteration Loop

`Controller.execute()` runs generation and verification steps, refining or retrying based on outcomes until limits are reached.

Experimentation & Analysis

Metrics Collection

Enable `monitoring_enabled` to gather `ValidationMetrics`. Retrieve via `controller.get_validation_m` or `controller.get_performance_report()`.

Parallel Experiments

Run multiple prompts concurrently:

```
prompts = ['Explain AI safety.', 'Study bias reduction.']
results = controller.execute_parallel(prompts, max_workers=4)
```

Caching

Use `OutputCache` during development to avoid redundant model calls.

Conclusions

LLMOrchestrator offers a modular, reproducible framework for orchestrating multiple LLMs, iterative reasoning, and comprehensive performance tracking. It facilitates diverse, validated insights across models, empowering research in AI safety, multi-agent simulations, and reliability studies.

Acknowledgements

We thank the developers of `openai`, `transformers`, `torch`, and `python-dotenv`.

References