Prepared by Tim Hayes

Crop Yield and MultiAgent Question Benchmarks

Overview

This document provides detailed information on the crop yield and multiagent question benchmarks. Inputs, outputs, repo locations and results are provided.

Location

thayes GitHub branch

Metrics

- runtime: Total time to run an iteration
- memory_delta: Memory change from beginning to end of an iteration
- Peak_memory: Peak memory used during an iteration
- avg_latency: Average llm latency in iteration
- llm_calls: Total llm calls for an iteration
- tokens_per_call: Average tokens per call
- total_prompt_tokens: Total prompt token
- mae: Mean Absolute Error
- mape: Mean Absolute Percentage Error
- rmse: Root mean squared error

The multiagent question benchmark also captures the per specialized agent (e.g. biology, chemistry, etc.) quality metrics in addition to the above. All benchmarks use simple_agent_common to generate metrics, generate output plots and output json/jsonl.

Reusable Packages

Benchmark Manager

This repo contains a script to run all benchmarks with standard configs and data. In addition it contains an app to process the output of the benchmarks and create a leaderboard csv, leaderboard plot and a scoring breakdown plot for the benchmarks. The location of this repo is: benchmark_manager . The benchmark manager also contains the final results of the the benchmarks as well as the as well as the rollup and summarization of the benchmarks:

crop_yield benchmark results
multiagent_questions benchmark results
benchmark_manager summarization results

Benchmark Data

This repo contains all the standard configs and datasets for the two benchmarks. The location of this repo is: benchmark_data

Simple Agent Common

This repo contains common data classes, utilities, metric calculations and benchmark classes used in the various benchmarks. The location of this repo is: simple_agent_common

Scoring Algorithm for Overall Scores

The code can be found here: metrics.py

```
def calculate_benchmark_score(self, config=None):
        """Calculate overall benchmark score across all iterations.

        Returns normalized percentage scores (0-100%) for each component and a
weighted total score.
        The total score is a weighted average of component percentages,
representing the overall
        achievement relative to optimal performance.
        """
        if config is None:
            config = {
                'quality': {
                    'weight': 60,
                    'mape_threshold': 15,  # MAPE threshold for penalty
                    'mape_penalty': 0.5    # Penalty factor for exceeding
threshold
                },
                'speed': {
                    'weight': 20,
                    'runtime_threshold': 300,
                    'latency_threshold': 2,
                    'weights': {'runtime': 0.6, 'latency': 0.4}
                },
                'resource': {
                    'weight': 20,
                    'memory_delta_threshold': 500,
                    'peak_memory_threshold': 1000,
                    'token_threshold': 1000,
```

```
                    'weights': {'memory_delta': 0.4, 'peak_memory': 0.3,
'tokens': 0.3}
                }
            }

        # Quality Score (0-100%)
        quality_base = 1 - min(self.avg_mape / 100, 1)  # Normalize MAPE to 0-1
range
        mape_ratio = self.avg_mape / config['quality']['mape_threshold']
        quality_penalty = np.exp(-max(0, mape_ratio - 1) *
config['quality']['mape_penalty'])
        quality_percentage = quality_base * quality_penalty * 100

        # Speed Score (0-100%)
        runtime_variance = np.var([m.runtime for m in self.iterations])
        latency_variance = np.var([m.avg_latency for m in self.iterations])
        speed_percentage = 100 * (
            (1 - min(self.avg_runtime / config['speed']['runtime_threshold'],
1)) * config['speed']['weights']['runtime'] +
            (1 - min(self.avg_latency / config['speed']['latency_threshold'],
1)) * config['speed']['weights']['latency']
        )

        # Resource Score (0-100%)
        resource_percentage = 100 * (
            (1 - np.power(self.memory_delta /
config['resource']['memory_delta_threshold'], 1.5)) *
config['resource']['weights']['memory_delta'] +
            (1 - np.power(self.peak_memory /
config['resource']['peak_memory_threshold'], 1.5)) *
config['resource']['weights']['peak_memory'] +
            (1 - np.power(self.iterations[0].tokens_per_call /
config['resource']['token_threshold'], 1.5)) *
config['resource']['weights']['tokens']
        )

        # Calculate weighted total score (0-100%)
        total_score = (
```

```
            quality_percentage * config['quality']['weight'] +
            speed_percentage * config['speed']['weight'] +
            resource_percentage * config['resource']['weight']
        ) / sum(config[k]['weight'] for k in ['quality', 'speed', 'resource'])

        self.benchmark_score = round(total_score, 2)

        return {
            'benchmark_score': {
                'total_score': self.benchmark_score,  # Overall weighted
percentage (0-100%)
                'quality_percentage': round(quality_percentage, 2),
                'speed_percentage': round(speed_percentage, 2),
                'resource_percentage': round(resource_percentage, 2),
                'details': {
                    'quality_weight': config['quality']['weight'],
                    'speed_weight': config['speed']['weight'],
                    'resource_weight': config['resource']['weight'],
                    'quality_penalty': round(quality_penalty, 3),
                    'mape_ratio': round(mape_ratio, 3),
                    'runtime_variance': round(runtime_variance, 3),
                    'latency_variance': round(latency_variance, 3)
                }
            }
        }
```

The weights and algorithm could be adjusted as needs require. Both benchmarks reuse this code.

Crop Yield Benchmark Description

This dataset is from [Omdena crop-yield-prediction] and is cleaned before each of the applications try to predict the yield using a prompt. The system and user prompts shared across all llm framework applications for this benchmark is:

```
YIELD_SYSTEM_PROMPT = """You are an agricultural yield prediction expert.
Return ONLY the predicted yield as a number, no other text.

"""
```

```
YIELD_PREDICTION_PROMPT = """
Predict yield for {crop} based on these conditions:

### Current Measurements:
- Precipitation: {precipitation} mm/day
- Specific Humidity: {specific_humidity} g/kg
- Relative Humidity: {relative_humidity}%
- Temperature: {temperature}°C

### {historical_records_type}:
{historical_examples}

### Yield Statistics:
Utilize the following yield stats from all {crop} records for your prediction:
{yield_stats}.
- Your prediction must fall strictly within this range.
- Use the mean ({mean_yield:.2f}) as a central reference point.

### Instructions:
Base your prediction on the {similarity_type} historical records provided,
ensuring it is constrained by the yield stats range. Your output must be a
single number representing the predicted yield, with no additional text.
"""
```

The config used for the final benchmark from the benchmark_data repo is:

```
data:
 paths:
   crop_data:
"../benchmark_data/data/crop_yield/crop+yield+predictiondata_crop_yield.csv"
   questions: "../benchmark_data/data/crop_yield/crop_yield_questions_50.jsonl"
   env: "~/src/python/.env"
   metrics: "/tmp/benchmarking/crop_yield"

model:
 name: "llama3-70b-8192"
 temperature: 0.01
```

```
 max_tokens: 1000
 # Rate limiting values
 max_calls: 4
 pause_time: 30
 token_limit: 90000
 # Retry values
 stop_after_attempt: 3
 wait_multiplier: 60
 wait_min: 180
 wait_max: 300

benchmark:
 iterations: 3
 random_few_shot: false
 num_few_shot: 5
```

This benchmark used Groq with the llama3-70b0-8192 model with the above parameters settings. Each application employed configurable retry logic and rate limiting.

The output of each crop yield applications are the following:
<model_name>_<llm_framework>_<datatime>_<iterations>.json

- Raw json output of the metrics for each iteration along with parms used to run benchmark

<model_name>_<llm_framework>_<datatime>_<iterations>_performance.png

- Plot containing all the quality metrics (mae, mape, etc) for each llm framework across iterations.

<model_name>_<llm_framework>_<datatime>_<iterations>_runtime.png

- Plot containing key resource and speed metrics for each llm framework across iterations

Each llm framework application has an output/metrics folder that is used for output of these files if the local config is used. Here are examples of crop yield benchmark output for the langgraph framework:

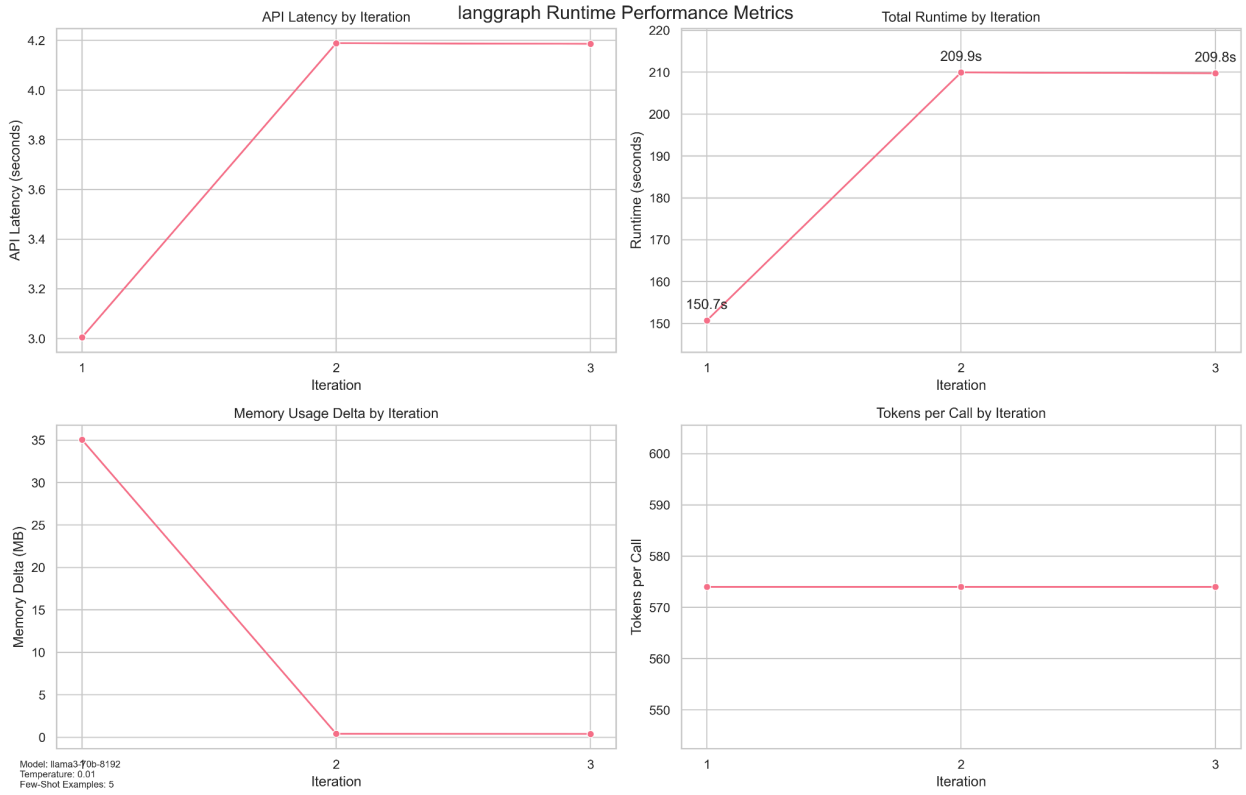
llama3-70b-8192_langgraph_20250309_151907_3.json

```
{
 "framework": "langgraph",
 "model_name": "llama3-70b-8192",
 "model_temperature": 0.01,
 "model_max_tokens": 1000,
 "random_few_shot": false,
 "num_few_shot": 5,
 "iterations": [
```

```
  {
    "iteration": 1,
    "runtime": 150.72100400924683,
    "memory_delta": 35.046875,
    "peak_memory": 268.46875,
    "llm_calls": 50,
    "avg_latency": 3.0044746255874633,
    "total_prompt_tokens": 28361,
    "tokens_per_call": 574.02,
    "mae": 3554.48,
    "mape": 12.865930925244538,
    "rmse": 6431.452762790068,
    "group_metrics": {}
  },
  {
    "iteration": 2,
    "runtime": 209.93913912773132,
    "memory_delta": 0.40625,
    "peak_memory": 268.984375,
    "llm_calls": 50,
    "avg_latency": 4.188825435638428,
    "total_prompt_tokens": 28361,
    "tokens_per_call": 574.02,
    "mae": 3577.08,
    "mape": 12.88746738944041,
    "rmse": 6512.04308953803,
    "group_metrics": {}
  },
```

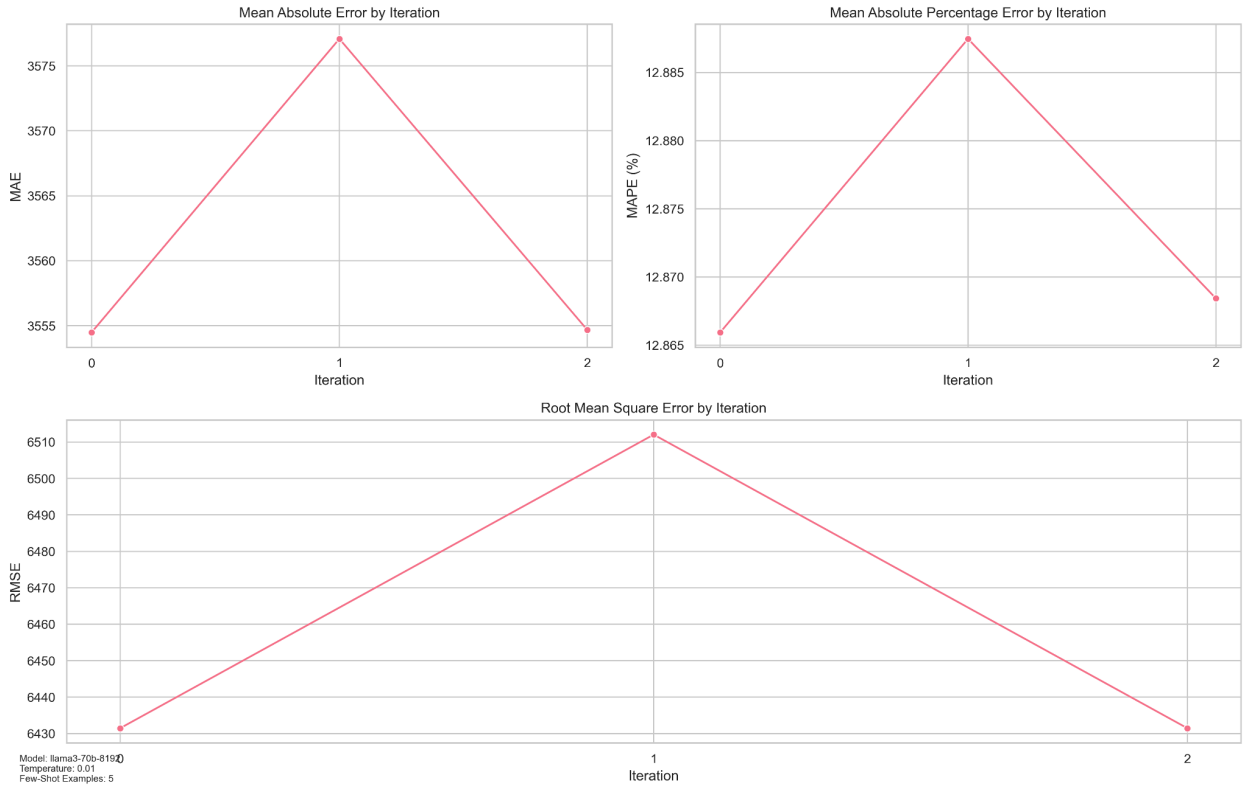

llama3-70b-8192_langgraph_20250309_151907_3_runtime.png

Prepared by Tim Hayes



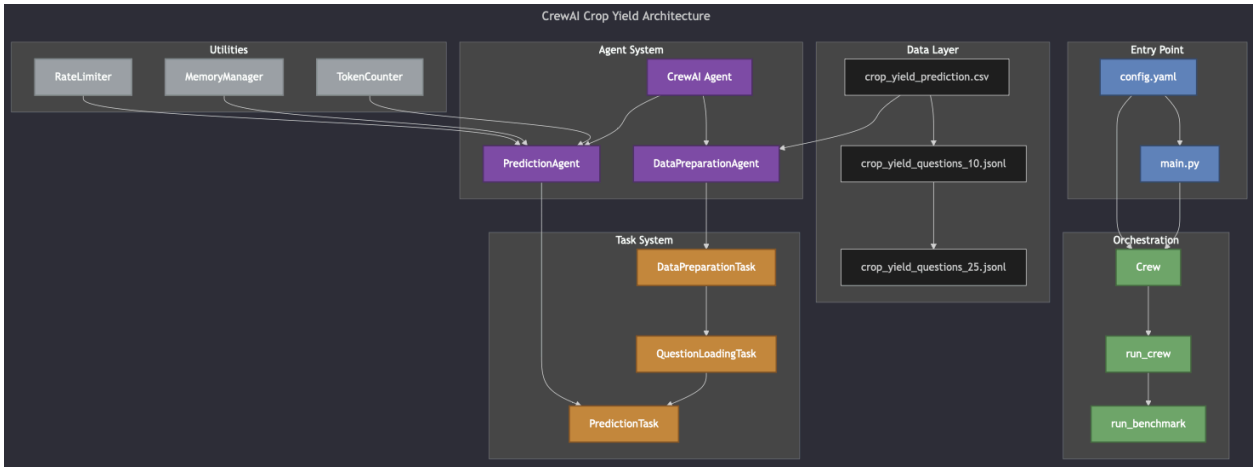
llama3-70b-8192_langgraph_20250309_151907_3_performance.png

Prepared by Tim Hayes



CrewAI Crop Yield Application

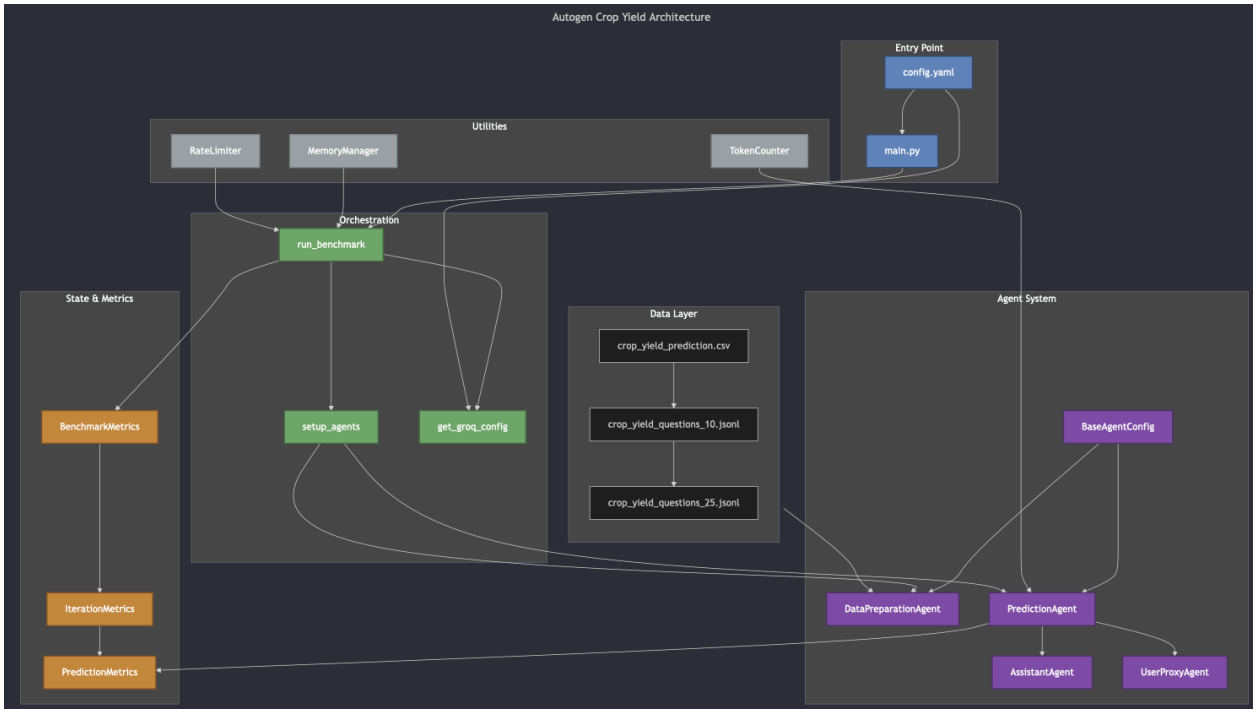
The crewai crop yield application is located here: [crewai_crop_yield_simple_agent](#) . The application has a README.md that provides further details on the components. The architecture and components of the application are below:



Prepared by Tim Hayes

Autogen Crop Yield Application

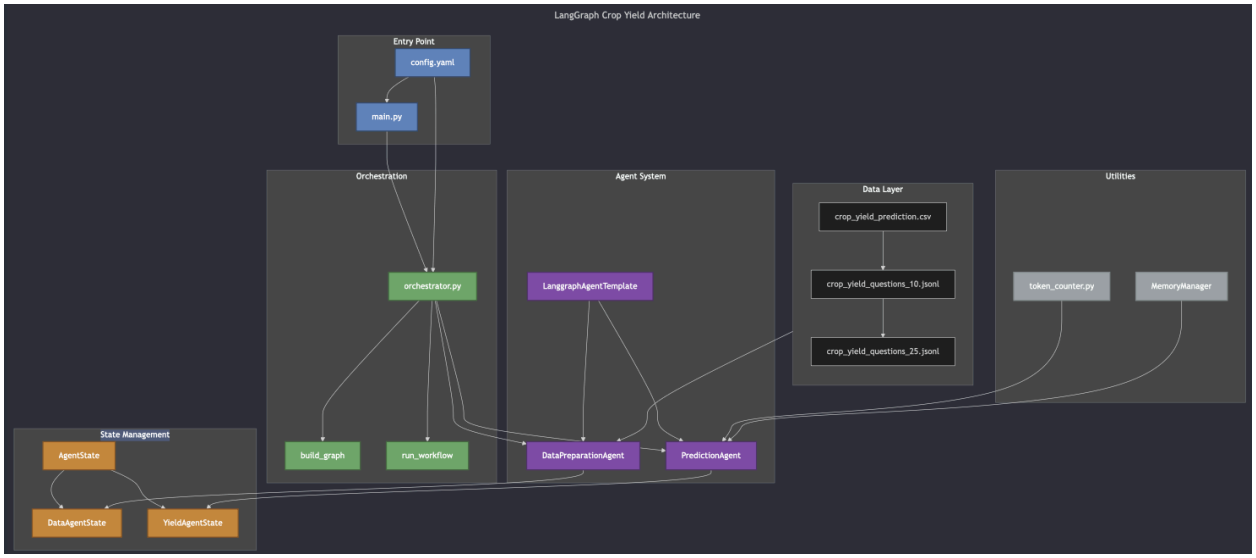
The autogen crop yield application is located here: autogen_crop_yield_simple_agent . The application has a README.md that provides further details on the components. The architecture and components of the application are below:



Langgraph Crop Yield Application

The langgraph crop yield application is located here: langgraph_crop_yield_simple_agent . The application has a README.md that provides further details on the components. The architecture and components of the application are below:

Prepared by Tim Hayes



MultiAgent Question Benchmark Description

The multiagent question benchmark entails a multi disciplinary set of questions including biology, chemistry, physics and mathematics. These questions were generated in a jsonl file using ChatGpt 4o. Here is the prompt that was used:

Act as an Associate {discipline} Processor. Generate 25 {discipline} questions with answers in a jsonl file. Answers must be purely numerical and not be in scientific notation. These questions must specifically relate to Chemistry and not other scientific disciplines. Act as a {discipline} Professor Emeritus and evaluate all generated questions and answers to ensure their accuracy and that the output is numerical and fit in a Python float datatype. In other words do not include questions whose answers are in scientific notation and would overflow a python decimal datatype.Once confirmed output the {discipline} dataset in jsonl format

At runtime the questions are randomly chosen based on the value of config['benchmark']['total_question'] which is equally allocated across the four disciplines. Once chosen the applications then send the question to Groq using the llm model from the config (for this benchmark llama3-70b0-8192 was used) to determine which discipline should answer the question. Based on that choice the question is routed to the appropriate discipline agent. Here are the standardized prompt template used by all multiagent question applications:

```
ROUTER_PROMPT = """You are a query classifier. You must classify a question
into one of the following categories:
                "math, physics, chemistry, or biology. Return only the category
name.
```

```
"""
```

```
MATH_PROMPT = """You are a Professor of Mathematics. Return only a single
numerical value with no explanation, units, or additional text.

Strict formatting rules:
- No explanation.ß
- No equations or calculations.
- No extra text.
"""

PHYSICS_PROMPT = """You are a Professor of Physics. Return only a single
numerical value with no explanation, units, or additional text.

Strict formatting rules:
- No explanation.
- No equations or calculations.
- No extra text.
"""

CHEMISTRY_PROMPT = """You are a Professor of Chemistry. Return only a single
numerical value with no explanation, units, or additional text.

Strict formatting rules:
- No explanation.
- No equations or calculations.
- No extra text.
"""

BIOLOGY_PROMPT = """You are a Professor of Biology. Return only a single
numerical value with no explanation, units, or additional text.

Strict formatting rules:
- No explanation.
- No equations or calculations.
- No extra text.
"""
```

The config used for the final benchmark from the benchmark_data repo is:

```
data:
 paths:
   input_dir: "../benchmark_data/data/multiagent_questions"
   env: "~/src/python/.env"
   metrics: "/tmp/benchmarking/multiagent_questions"

benchmark:
 iterations: 3  # Number of times to run the complete test
```

```
 total_questions: 60


model:
   name: "llama3-70b-8192"
   temperature: 0.01
   max_tokens: 1000
   # Rate limiting values
   max_calls: 4
   pause_time: 30
   token_limit: 90000
   # Retry values
   stop_after_attempt: 3
   wait_multiplier: 60
   wait_min: 180
   wait_max: 300
```

This benchmark used Groq with the llama3-70b0-8192 model with the above parameters settings. Each application employed configurable retry logic and rate limiting.

The output of each multiagent question applications are the following:
<model_name>_<llm_framework>_<datatime>_<iterations>.json
- Raw json output of the metrics for each iteration along with parms used to run benchmark

<model_name>_<llm_framework>_<datatime>_<iterations>_questions.jsonl
- Questions that were randomly selected for the benchmark run.

<model_name>_<llm_framework>_<datatime>_<iterations>_performance.png
- Plot containing all the quality metrics (mae, mape, etc) for each llm framework across iterations.

<model_name>_<llm_framework>_<datatime>_<iterations>_agent_performance.png
- Plot containing all the quality metrics (mae, mape, etc) for each llm framework across iterations broken down by each agent (biology, chemistry etc.).

<model_name>_<llm_framework>_<datatime>_<iterations>_runtime.png
- Plot containing key resource and speed metrics for each llm framework across iterations

Each llm framework application has an output/metrics folder that is used for output of these files if the local config is used. Here are examples of multiagent question benchmark output for the langgraph framework:

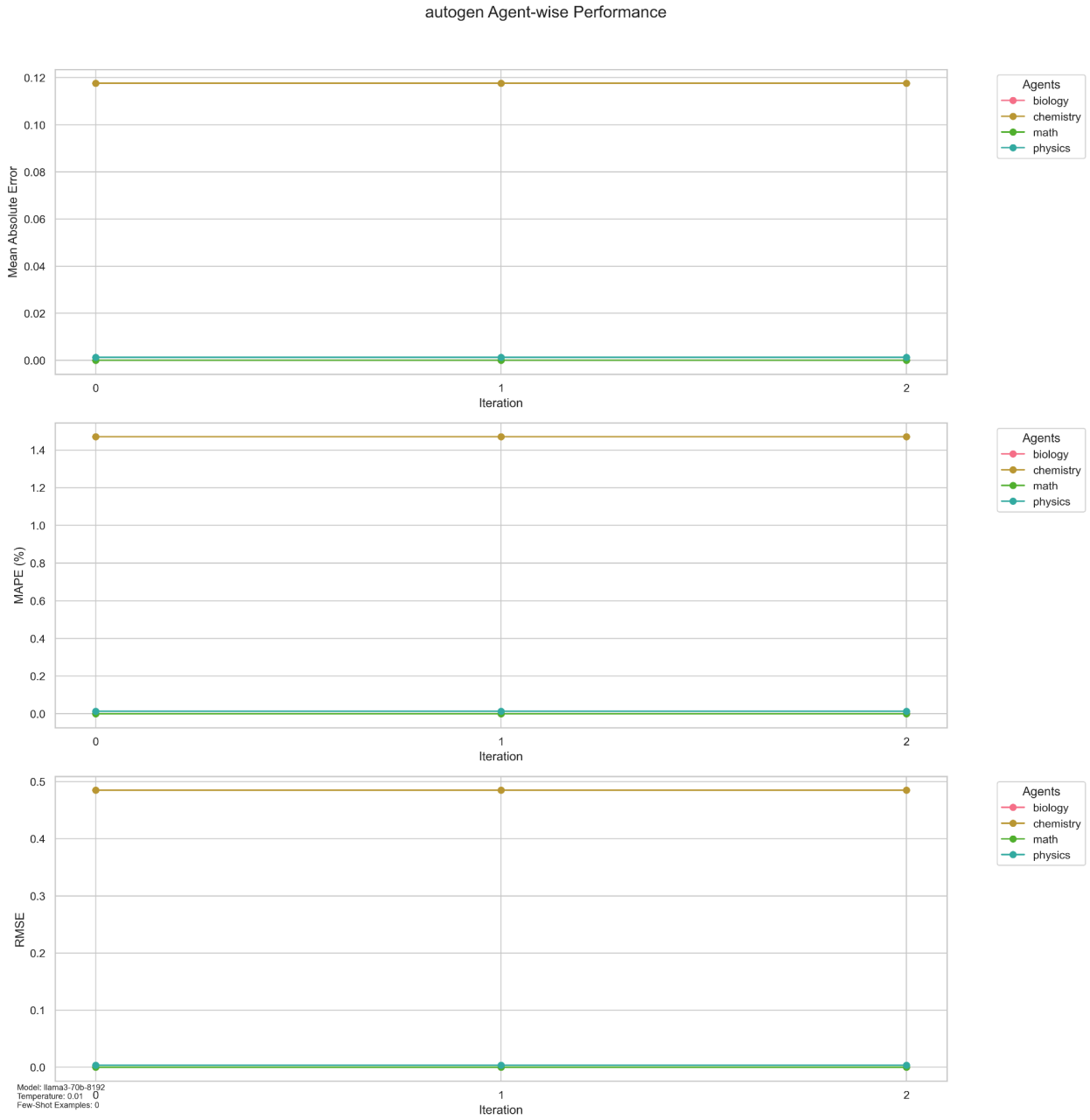
llama3-70b-8192_autogen_20250309_163007_3.json

```
{
 "framework": "autogen",
 "model_name": "llama3-70b-8192",
```

```
"model_temperature": 0.01,
"model_max_tokens": 1000,
"random_few_shot": false,
"num_few_shot": 0,
"iterations": [
  {
    "iteration": 1,
    "runtime": 392.1841537952423,
    "memory_delta": 1.46875,
    "peak_memory": 335.015625,
    "llm_calls": 60,
    "avg_latency": 0.6895822922388712,
    "total_prompt_tokens": 991,
    "tokens_per_call": 21.616666666666667,
    "mae": 0.0336732588205425,
    "mape": 0.4201687108863139,
    "rmse": 0.25820534838831494,
    "group_metrics": {
      "math": {
        "mae": 1.9737013679411764e-05,
        "mape": 1.745140337887987e-05,
        "rmse": 8.137779213447537e-05
      },
      "physics": {
        "mae": 0.00125,
        "mape": 0.0127420998980632,
        "rmse": 0.0035355339059327377
      },
```

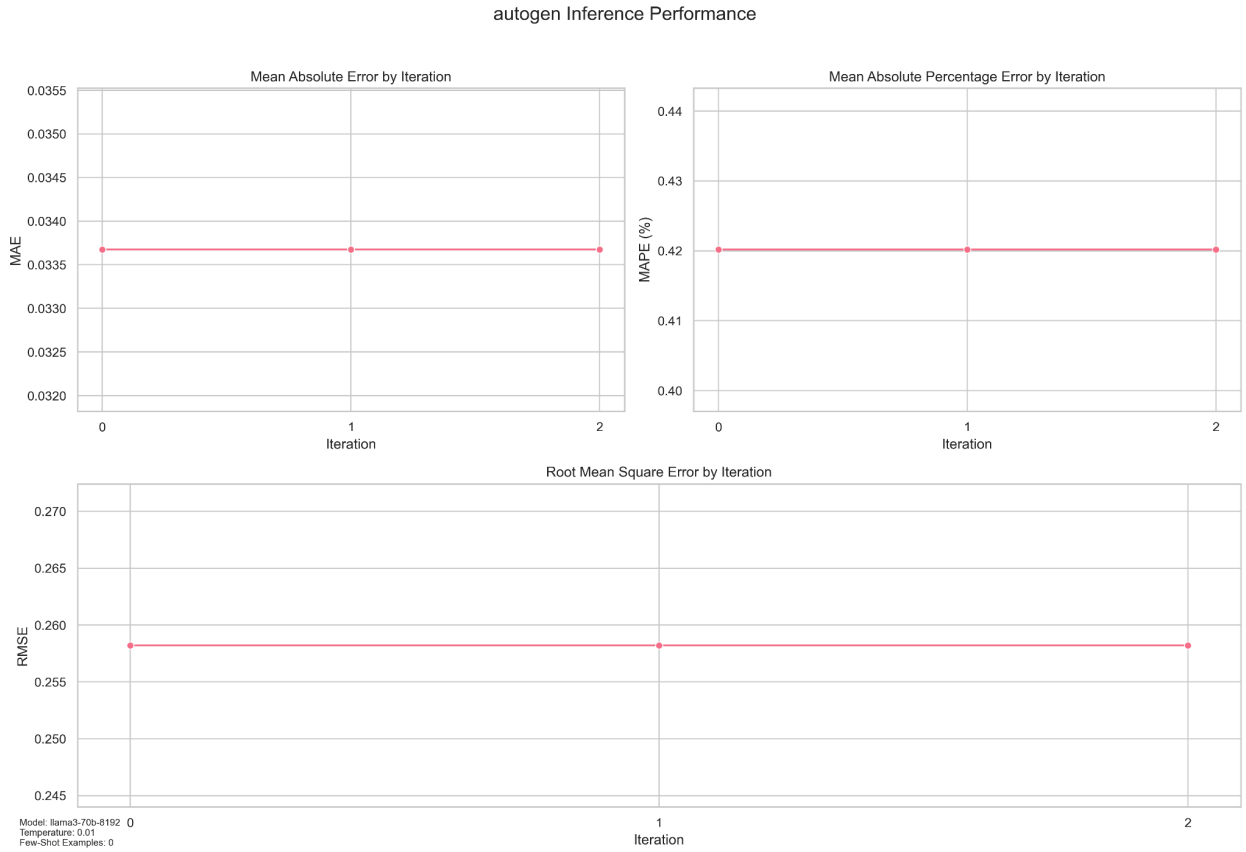

llama3-70b-8192_autogen_20250309_163007_3_agent_performance.png

Prepared by Tim Hayes

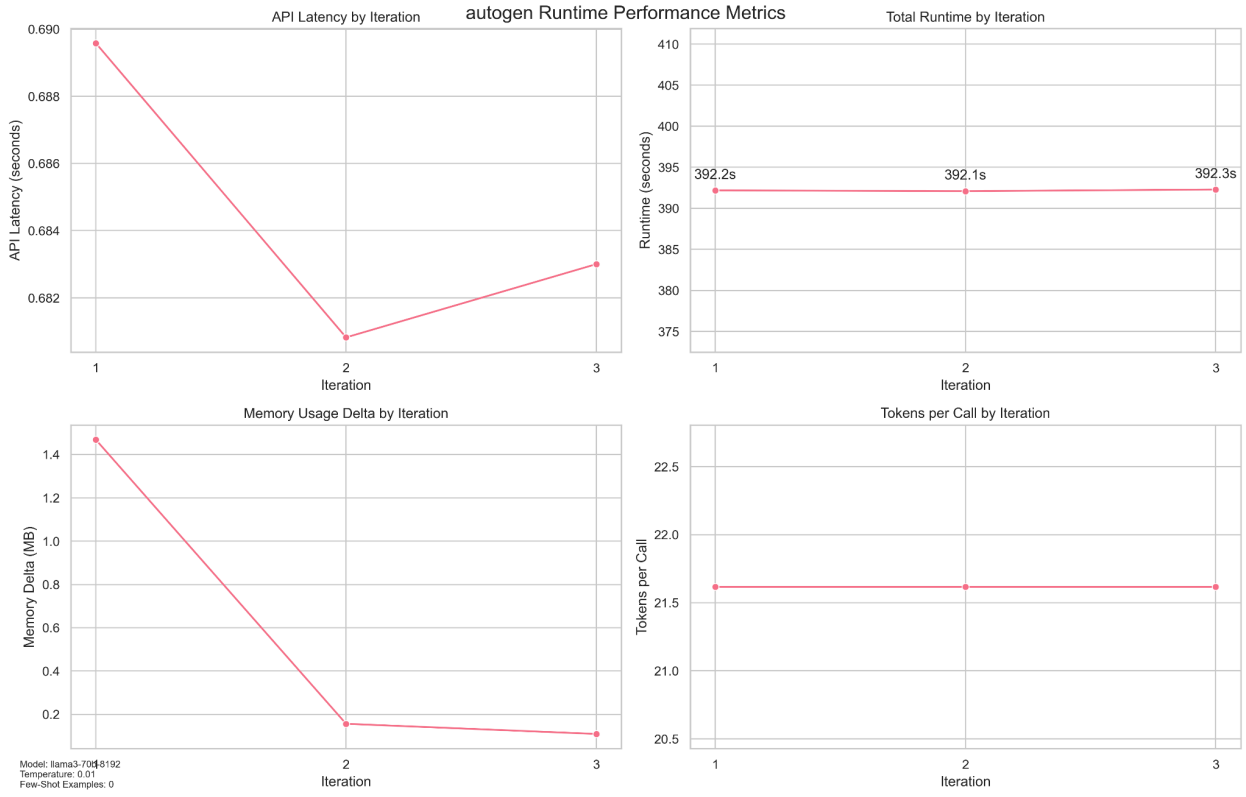


llama3-70b-8192_autogen_20250309_163007_3_performance.png

Prepared by Tim Hayes



llama3-70b-8192_autogen_20250309_163007_3_runtime.png



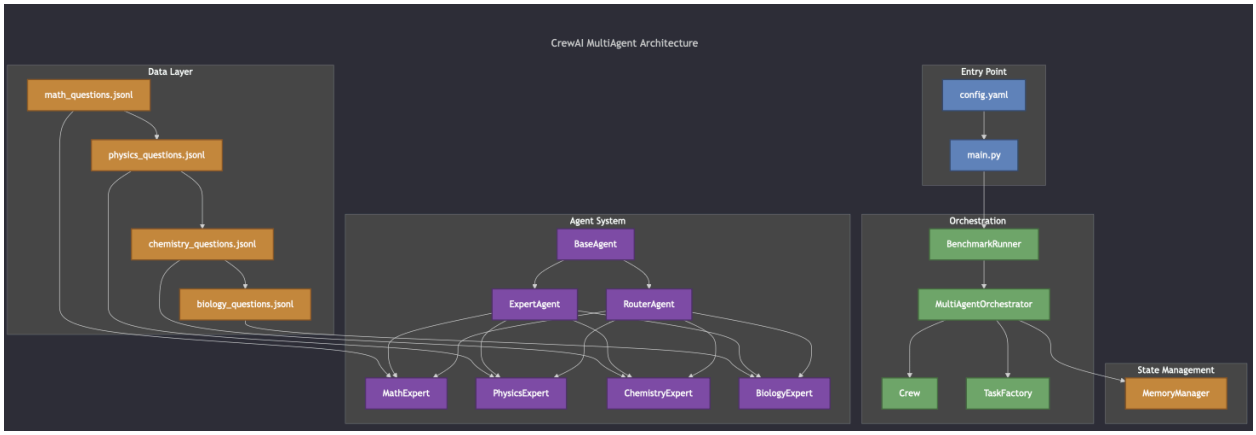
llama3-70b-8192_autogen_20250309_163007_questions.jsonl


```
{"question": "What is 8 factorial (8!)?", "answer": 40320, "source_file": "math_questions.jsonl"}
{"question": "What is the refractive index of water?", "answer": 1.33, "source_file": "physics_questions.jsonl"}
{"question": "What is the charge of an electron in coulombs?", "answer": -1.602e-19, "source_file": "physics_questions.jsonl"}
{"question": "How many nucleotide bases are in DNA?", "answer": 4, "source_file": "biology_questions.jsonl"}
{"question": "What is the impulse when a force of 10N is applied for 5 seconds?", "answer": 50.0, "source_file": "physics_questions.jsonl"}
{"question": "What is the square root of 144?", "answer": 12, "source_file": "math_questions.jsonl"}
{"question": "How many seconds are in one hour?", "answer": 3600.0, "source_file": "physics_questions.jsonl"}
{"question": "What is the enthalpy of formation of water (H2O) in kJ/mol?", "answer": -285.8, "source_file": "chemistry_questions.jsonl"}
{"question": "What is the cube root of 27?", "answer": 3, "source_file": "math_questions.jsonl"}
{"question": "What is the sum of the exterior angles of any polygon in degrees?", "answer": 360, "source_file": "math_questions.jsonl"}
{"question": "How many neutrons are in an isotope of carbon-14?", "answer": 8.0, "source_file": "chemistry_questions.jsonl"}
{"question": "What is the force needed to accelerate a 10 kg object at 5 m/s\u00b2?", "answer": 50.0, "source_file": "physics_questions.jsonl"}
{"question": "What is the atomic number of calcium?", "answer": 20.0, "source_file": "chemistry_questions.jsonl"}
{"question": "What is the normal boiling point of water in degrees Celsius?", "answer": 100.0, "source_file": "chemistry_questions.jsonl"}
{"question": "What is the decimal equivalent of 3/8?", "answer": 0.375, "source_file": "math_questions.jsonl"}
{"question": "What is the remainder when 29 is divided by 5?", "answer": 4, "source_file": "math_questions.jsonl"}
{"question": "What is the coefficient of friction for a surface where a 10N force is required to move a 20N object?", "answer": 0.5, "source_file": "physics_questions.jsonl"}
```

Prepared by Tim Hayes

CrewAI MultiAgent Question Application

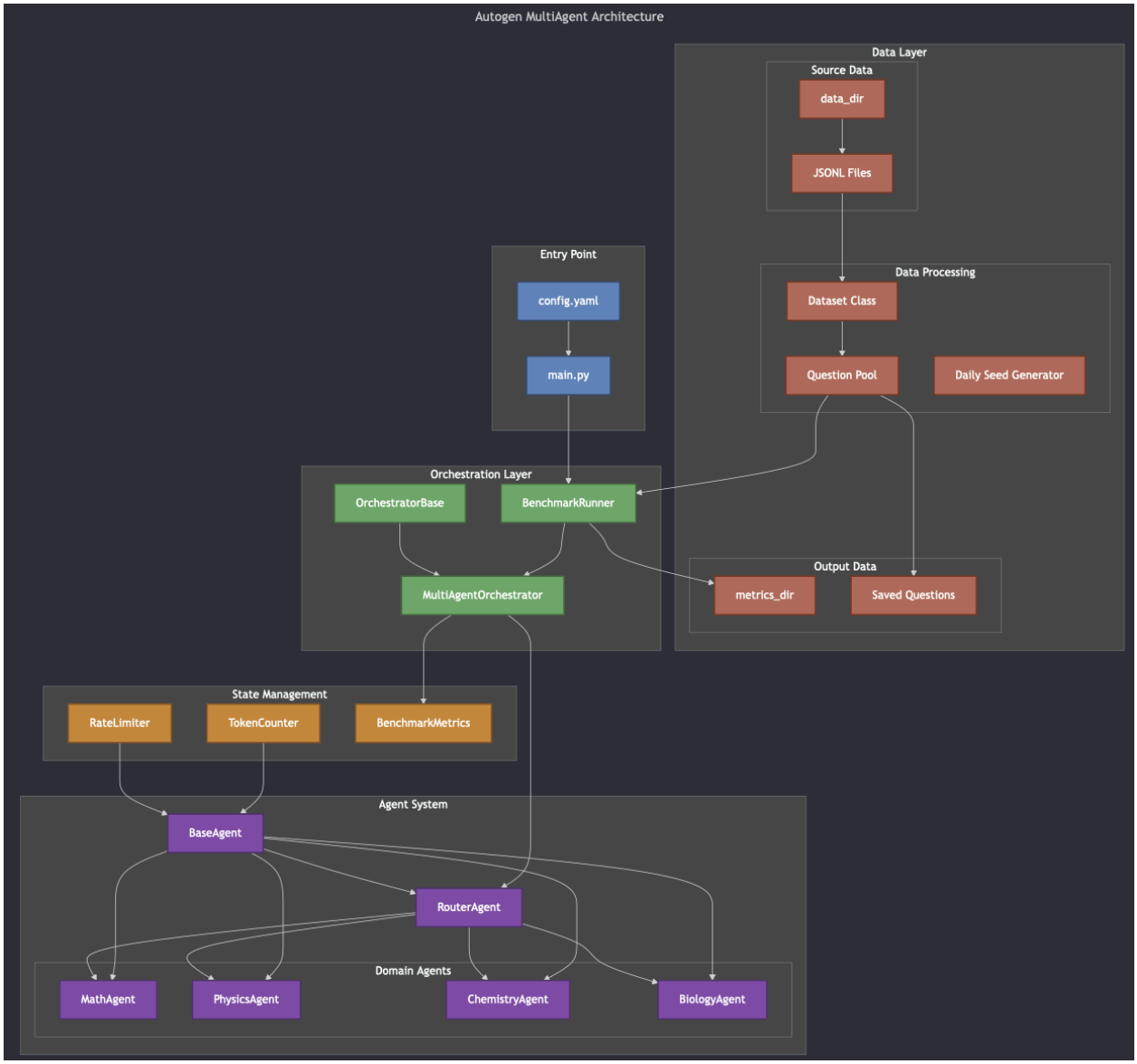
The crewai multiagent question application is located here: [crewai_multi_agent](#) . The application has a README.md that provides further details on the components. The architecture and components of the application are below:



Prepared by Tim Hayes

Autogen MultiAgent Question Application

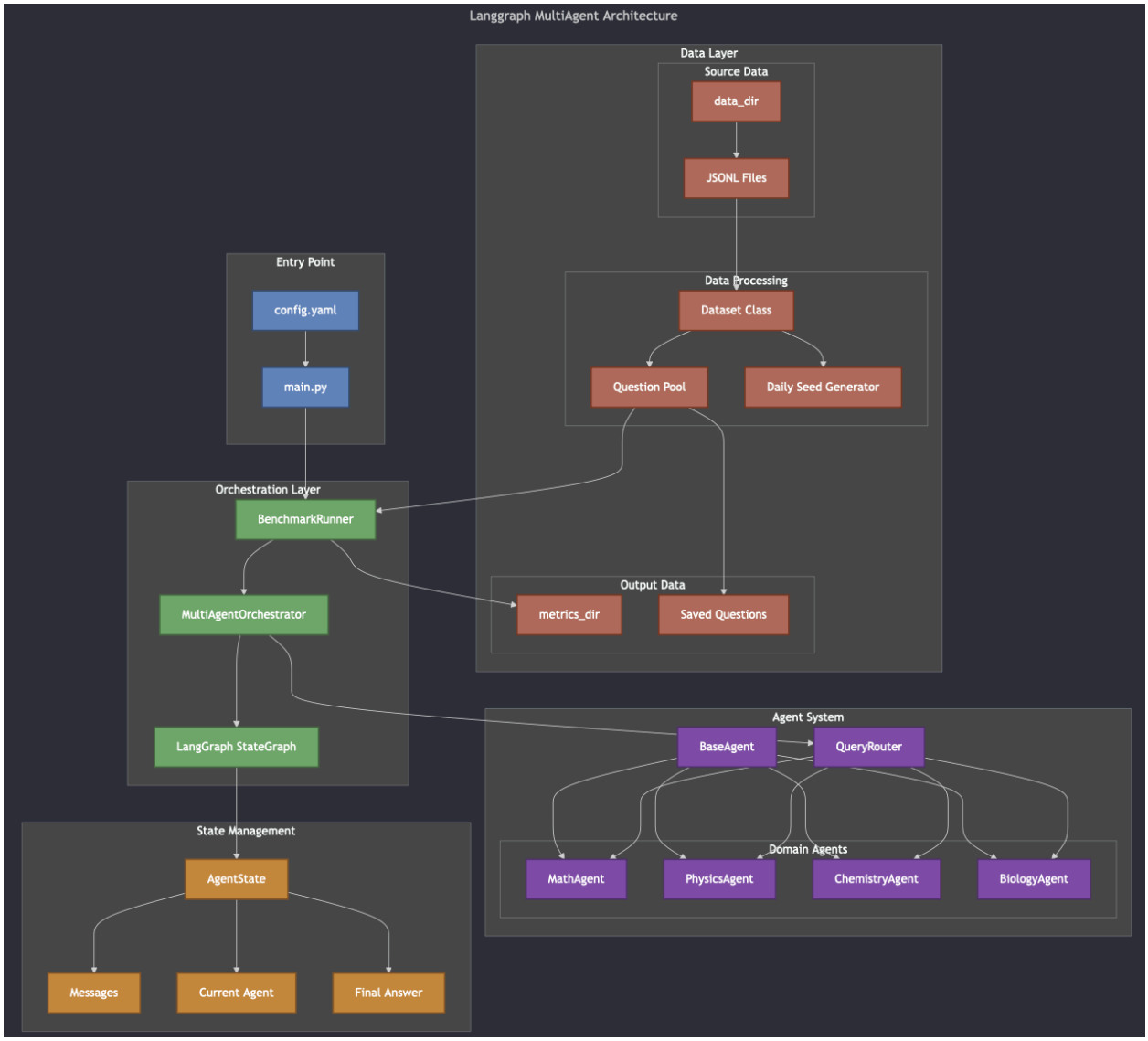
The autogen multiagent question application is located here: autogen_multi_agent . The application has a README.md that provides further details on the components. The architecture and components of the application are below:



Prepared by Tim Hayes

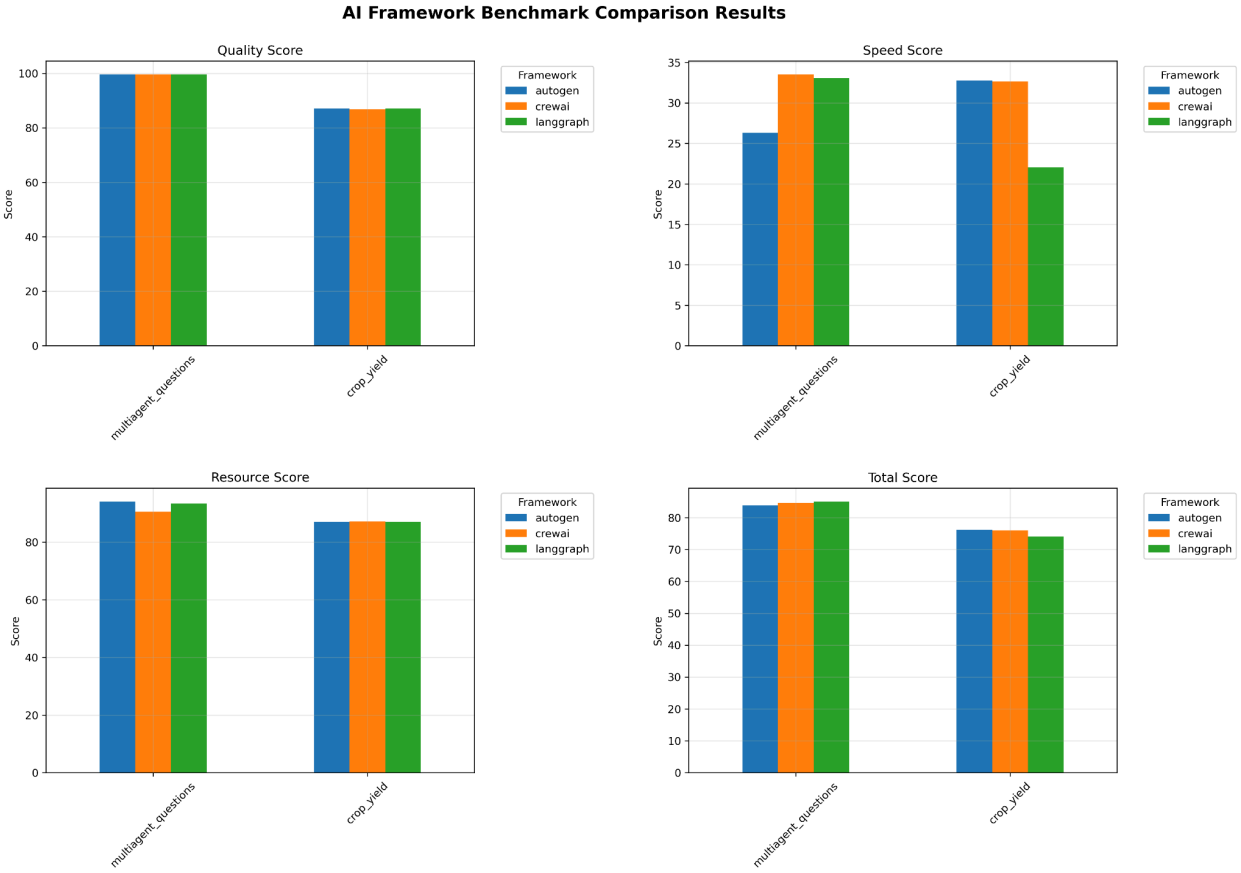
Langgraph MultiAgent Question Application

The langgraph multiagent question application is located here: langgraph_multi_agent. The application has a README.md that provides further details on the components. The architecture and components of the application are below:



Prepared by Tim Hayes

Final Results



Framework Benchmark Results

	multiagent_questions	crop_yield	Avg Score
autogen	83.83	76.22	80.03
crewai	84.56	76.06	80.31
langgraph	85.04	74.07	79.56

For more details on indivisible framework results visit crop_yield and multiagent_questions . From the summarization results above we can see that the llm struggled with predicting yields as compared to answering questions. This was consistent across frameworks. For the multiagent question benchmark crewai did not perform as well when it comes to the resource

usage score. Likewise langgraph struggled with its speed score for the crop yield benchmark. langgraph won the multiagent question benchmark and autogen won the crop yield benchmark. Crewai won the overall scoring however crewai and langgraph are very close behind.

Future Investigation

I ran all the benchmarks on a Macbook M4 Pro with 36GB of RAM. The benchmark should be run in the cloud in a docker container. Each of the applications could be parallelized in some cases. In addition I only have a non-paid Groq api key and thus I was quite conservative with my rate limiting values. The configuration could be opened up with a paid Groq API key could further improve throughput. Finally other models are only available with paid subscriptions and could be tested and likely would improve quality scores.