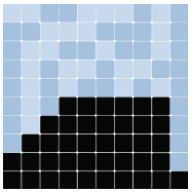
Mesa-Frames: Stats & Event Driven Data Collection with Streamed Storage

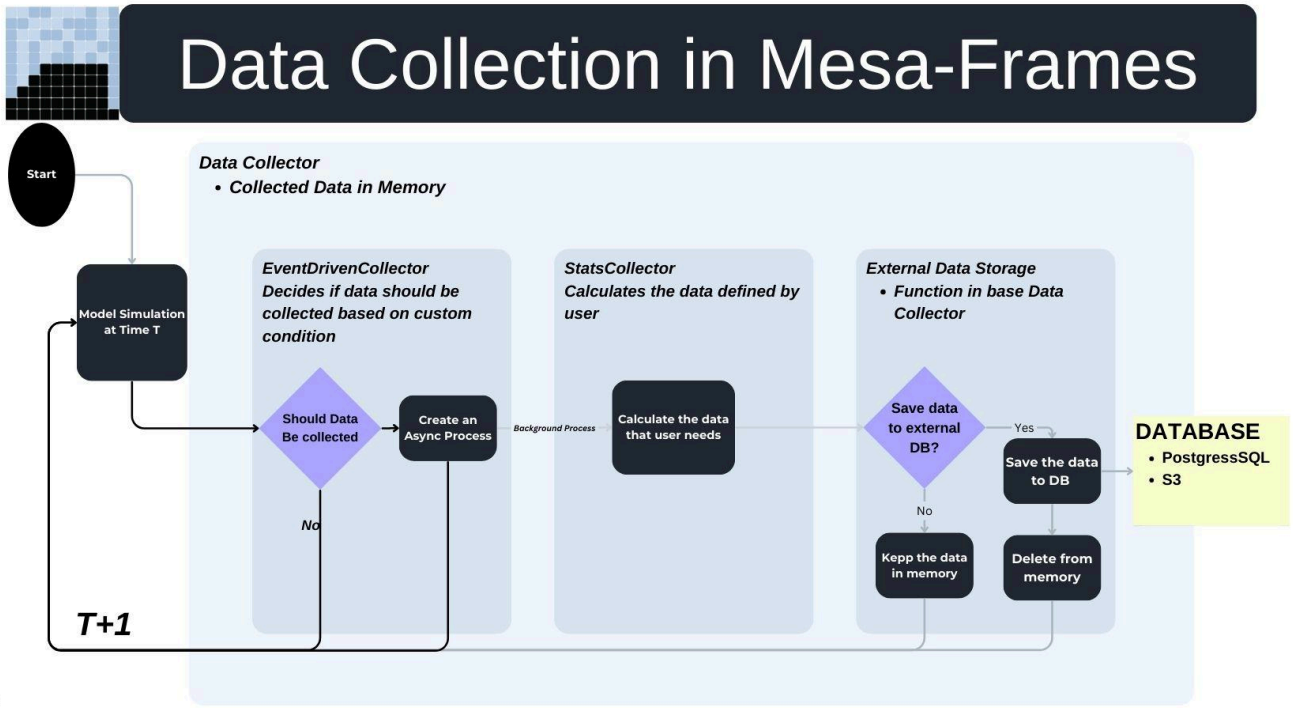


"I want to release a stable version of Mesa-Frames, enabling scalable agent-based simulations with millions of agents using efficient, vectorized operations."

Introduction

Abstract

This proposal outlines my plan to significantly enhance Mesa-Frames' data collection capabilities during Google Summer of Code 2025. The focus is on developing a flexible, efficient, and scalable framework tailored for advanced researchers working with large-scale agent-based simulations.



How StatsCollector EventDrivenCollector and ExternalStorage module will work together

The core enhancements include:

- **Stats Collection** – A lightweight module that allows users to specify exactly which statistics (e.g., mean, max, min, count etc) they want to collect, reducing memory usage and computational overhead.

- **Event Driven Collection** – A mechanism to record data only when predefined conditions are met, via predicate functions (e.g., lambda model: model.sheep_count < 10) or time-based triggers (every_n_steps=50) ensuring researchers capture only important insights while avoiding unnecessary storage.

- **External Data Storage Integration** – Direct support for storing collected data in PostgreSQL/S3 or other databases, minimizing memory footprint and enabling large-scale analysis.

These enhancements will make large-scale agent-based simulations feasible on standard hardware, unlocking new research possibilities while significantly reducing computational costs. Beyond these core deliverables, I will focus on performance enhancements by transitioning Polars operations to lazy execution and optimizing vectorization. Additionally, improving documentation and usability will ensure Mesa-Frames remains an intuitive and powerful tool for the research community.

By the end of GSoC, my aim is to transform Mesa-Frames into a production-ready tool for large-scale agent-based simulations, ensuring its adoption and long-term impact in the Mesa community and beyond.

Benefits to Community

This project introduces **Stats Collection**, **Event Driven Collection**, and **External Data Storage Integration** to enhance data collection, analysis, and storage efficiency in agent-based simulations.

- *Optimized Performance & Memory*
  - With Stats Collection, researchers can collect aggregated statistics (e.g., max wealth, count of surviving agents) rather than raw agent data.
  - Event Driven Collection ensures data is only stored when meaningful events occur, reducing unnecessary logging.
  - Result: Large-scale simulations run faster with significantly reduced memory consumption.
- *Flexible Data Collection*
  - Not all studies require full agent state tracking. Researchers often need only key statistics or event-triggered insights.
  - Stats Collection enables users to specify exactly what statistics they need (e.g., mean wealth, count of agents meeting a condition).
  - Event-driven collection allows tracking of critical model dynamics without excessive data logging.
  - Result: Researchers can customize their data collection strategies, ensuring efficient use of computational resources.
- *Scalability for Large Simulations*
  - External Data Storage Integration offloads collected data to PostgreSQL or cloud storage, eliminating memory constraints.
  - Result: Mesa-Frames can handle larger, more complex models than before.

Deliverables

During GSoC 2025, my primary goal is to enhance the data collection capabilities of Mesa-Frames to make it more efficient, flexible, and tailored to the needs of advanced researchers. The key deliverables will include:

> **Note:** The **Boltzmann Wealth example** is based on real data and a fully implemented model, whereas the **Wolf-Sheep example** uses synthetic data, as Mesa-Frames does not currently include an actual Wolf-Sheep model.

1. Stats Collection - Similar to model collector

- **Motivation**: Researchers often need only a few specific statistics rather than all available data. Collecting everything can use more memory and slow down computations unnecessarily. A lightweight statistics collection method allows researchers to choose exactly what to collect, making the process faster and more efficient.
- **Approach**: Works similarly to Mesa's model reporter, allowing users to define ***statistical aggregations*** (mean, max, min, sum, count of specific values or functions, etc.).

  Example Use Case

- Boltzmann Wealth Model (simple statistic)
  In the Boltzmann Wealth Model, a simulation runs with **N** agents over **T** timesteps. If we collect data for all agents at every timestep, the space complexity is:

  $$O(N \, x \, T \, x \, C)$$

  where **C** is the number of columns (agent attributes). This can lead to high memory consumption.

  Instead of storing all agent attributes, suppose the researcher is only interested in:

  1. **Maximum wealth at each timestep**
  2. **Number of agents reaching the maximum wealth**

  With the **Stats Collection**, we only store **T × 2** values (max wealth and agent count per timestep), significantly reducing memory usage while preserving key insights.

Approach	Space Complexity
Storing all agent attributes	$O(N \times T \times C)$
Using DataCollector	$O(T \times 2)$

```
model.datacollector.model_data
```

shape: (100, 4)

timestep	max_wealth	num_agents_max_wealth
i64	f64	i64
0	1.0	100
1	4.0	2
2	4.0	2
3	4.0	2
4	4.0	3
...	...	...
95	5.0	3
96	6.0	2
97	7.0	1
98	6.0	1
99	6.0	1

- Wolf-Sheep (count of a statistic)
  In the Wolf-Sheep model, assume we start with S sheep and W wolves and run the model for T timesteps. If we track every agent's state, the space complexity would be T × (S + W) (where S and W vary over time).

  However, if the researcher only needs:

  1. The total number of surviving sheep per timestep
  2. The total number of surviving wolves per timestep

  Then, instead of storing every agent's full state, the Stats Collection will store only T × 2 values (count of alive sheep and wolves). This drastically reduces storage without losing key behavioral trends.

Approach	Space Complexity
Storing all agent attributes	$O(TxW + TxS)$
Using DataCollector	$O(T \times 2)$

2. Event Driven Collection

- **Motivation:** Not all timesteps produce meaningful data. Collecting information at every timestep can lead to unnecessary overhead, especially when relevant data only appears during specific events. Focusing data collection on key moments improves efficiency and ensures that only valuable information is captured.

- **Solution**: Implement an Event-Driven DataCollector, which stores data only when predefined conditions are met, rather than collecting at every timestep.

  Example Use Case

- Wolf-Sheep model

  Suppose a researcher wants to collect data *only when the sheep population experiences a drastic drop*. Instead of collecting data at every timestep, we define an event:

  - If the sheep count at **time t+1** is **less than 50% of its value at time t**, the DataCollector stores the **before-and-after data** for that timestep.

  For instance, if at timestep **20**, the sheep count drops from **200 to 98**, and at timestep **42**, it drops from **150 to 50**, only these two points are stored: pre and post is included for easier analysis when collecting more than 2 consecutive points

timestep	state	sheep_count	wolves_count
19	pre	200	300
20	post	98	400
41	pre	150	260
42	post	50	300
61	pre	400	420
62	post	100	600

- Boltzmann wealth

  A researcher may be interested in tracking when the number of agents reaching max wealth exceeds above a critical threshold.

  - Define an event where data is only collected when the number of agents reaching max wealth drops exceeds **0.05% of the initial count**.
  - Instead of tracking wealth distribution at every timestep, the DataCollector **only logs data when this event occurs**.

This approach ensures that **only significant economic shifts** are recorded, making analysis **more focused and memory-efficient**. So if initial number of Agents were 100 then the below data would be collected

```
model.datacollector.model_data
```

shape: (4, 4)

timestep	max_wealth	num_agents_max_wealth
i64	f64	i64
3	4.0	6
8	4.0	6
11	3.0	12
12	3.0	8

- **Flexible Event Conditions**: Allow users to define custom event conditions, enabling them to capture data based on the dynamics of their specific agent-based models.
- The function will evaluate the predefined condition at each timestep and return **True** if data should be captured or **False** otherwise.

3. External Data Storage Integration

- **Motivation**

  Large-scale agent-based simulations can generate substantial amounts of data at every timestep. Storing all of this data in memory may lead to performance bottlenecks, excessive RAM usage, and even system crashes. For long-running or high-frequency simulations, an efficient mechanism is essential to persist data externally without overloading system resources.

- **Solution**

  To prevent excessive memory usage while ensuring data is available for post-simulation analysis, we implement **external storage integration** that:

  - **Streams collected data directly to a database or file-based storage**, eliminating in-memory storage overhead.
  - **Supports multiple storage backends**, currently plan on using S3 and PostgreSQL, will be confirmed with Mentor

- **Allows configurable storage settings**, enabling users to toggle between different external storage methods based on their needs.

- **Approach**

  External data storage integration is implemented as an optional feature within the existing DataCollector. The key design decisions include:

  1. *Storage Backends & Flexibility*
     - Users can configure different storage backends (e.g., SQL, NoSQL, file-based) based on the simulation requirements.
     - Storage backends include:
       - Relational Databases (PostgreSQL, MySQL): Ideal for structured querying & analytics.
       - File-based Storage (CSV, Parquet, JSON): Suitable for archiving or large-scale batch processing.

  2. *Data Streaming & Asynchronous Writes*
     - Instead of waiting for the entire simulation to complete before saving, data is streamed in real-time to the chosen external storage.
     - By using **asynchronous** I/O operations, simulation performance is not slowed down by storage operations.
     - Batching mechanisms ensure efficient write operations while keeping latency low.

  3. *Scalability & Performance Considerations*
     - **Configurable Buffering**: Data is temporarily stored in a buffer before writing, reducing write amplification.
     - **Parallel Processing**: Writes to external storage happen in the background, so they don't interfere with simulation speed.
     - **Fault Tolerance**: If external storage becomes unavailable, the system can temporarily store data in memory and retry writes later.

  4. *User Configuration & Control*
     - The integration is fully configurable—users can enable or disable external storage based on the size and duration of the simulation.
     - Settings like buffer size, write interval, and storage type can be adjusted for optimal performance.
     - Users can specify retention policies, choosing to store only **aggregated data** if raw data is not needed.

Beyond These Deliverables

While these are the primary goals, software development is inherently dynamic. New challenges—such as bug fixes, feature requests, or architectural improvements—may arise. I am prepared to address these as they emerge, prioritizing them effectively to ensure Mesa-Frames remain stable, efficient, and well-maintained.

Additionally, if time permits during the GSoC period—or even after its completion—I would like to focus on:

- Transition Polars operations from eager to lazy execution to unlock significant performance improvements
- Optimizing vectorization, converting non-vectorized operations where possible.

Beyond performance optimizations, I aim to improve **Mesa-Frames' usability** by adopting **Mesa's modular** documentation **[***Link***]**. This will make it **more intuitive** for new users and **easier to extend** for future contributors.

Current Code DRAFT

Future drafts will be submitted as pull requests on GitHub.

1. DataCollector

```
class DataCollector:
    def __init__(self, model, reporters=None, trigger=None, stat_config=None):
        self.model = model
        self.reporters = reporters or {}
        self.trigger = trigger or (lambda model: True)
        self.stat_config = stat_config or {}
        self.data = pl.DataFrame()
        self._stat_methods = {
            "max": lambda df, var: df[var].max(),
            "min": lambda df, var: df[var].min(),
            "mean": lambda df, var: df[var].mean(),
            "sum": lambda df, var: df[var].sum(),
        }
    def collect(self):
        if not self.trigger(self.model):
            return

        row = {"timestep": self.model._steps}

        for name, func in self.reporters.items():
            row[name] = func(self.model)
        if self.stat_config:
            agent_df = self.model._agents._agentsets[0]
            row.update(self.compute_stats(agent_df))
        self.data = self.data.vstack(pl.DataFrame([row]))
    def get_data(self):
```

```
        return self.data
    def compute_stats(self, df: pl.DataFrame) -> dict:
        results = {}
        cache = {}
        for key, stats in self.stat_config.items():
            for stat in stats:
                if stat.startswith("count"):
                    parts = stat.split(":")
                    mode = parts[1] if len(parts) > 1 else None
                    val = self.get_count(df, key, mode, cache)
                    label = f"{key}_count" if not mode else f"{key}_{mode}_count"
                    results[label] = val
                elif stat in self._stat_methods:
                    val = self._stat_methods[stat](df, key)
                    results[f"{key}_{stat}"] = val
                    cache[stat] = val

        return results
    def get_count(self, df, var, mode=None, cache=None):
        if mode in self._stat_methods:
            val = cache.get(mode) or self._stat_methods[mode](df, var)
            cache[mode] = val
            return (df[var] == val).sum()
        return len(df)
    def external_data_storage():
        # This function is a placeholder for the external data storage logic.
        pass
```

Schedule of Deliverables

Community Bonding Period (May 8 - June 1)

- Finalize the design for a `DataCollector` with statistical, event-driven and external data storage.
- Consult with mentors to ensure compatibility with Mesa-Frames goals.
- Develop a structured plan for implementation and integration.

Coding Phase 1 (June 2 - July 14)

🎯 **Goal: Build the core unified `DataCollector` with stats and event-based logging**

Week 1–2 (June 2 – June 15)

- Implement the initial `DataCollector` structure using Polars
- Add support for core statistics: `mean`, `max`, `min`, `sum`, `count`
- Ensure integration with Mesa-Frames agent and model design

Week 3–4 (June 16 – June 30)

- Add support for event-driven data collection via user-defined triggers
- Enable custom reporter functions
- Begin writing unit tests and benchmarks

Week 5–6 (July 1 – July 14)

- Implement external data collector
- Add support for external storage formats (CSV, DB)
- Implement optional memory-efficient streaming mode

📌 **Midterm Evaluation (July 14 – 18)**

- Submit progress report
- Receive feedback and adjust plan as needed

Coding Phase 2 (July 14 - August 25)

🎯 **Goal: Extend features and optimize performance**

Week 7–8 (July 15 – July 28)

- Finalize advanced statistics (e.g., `count:max`, `count:condition` and other statistics)
- Add support for custom **group by** fields and nested configurations
- Integrate Polars **Lazy API** when possible for optimized performance

Week 9–10 (July 29 – August 11)

- Extend the trigger system for compound and multiple conditions
- Ensure seamless integration between stats and event triggers

Week 11–12 (August 12 – August 25)

- Final debugging and performance tuning
- Write documentation and example notebooks
- Prepare final submission and polish all components

📌 **Final Submission (August 25 – September 1)**

- Submit final code and documentation
- Complete final evaluations