SUSTAINHUB: OPEN SOURCE COMMUNITY SUSTAINABILITY

Vidhi Rohira
Veermata Jijabai Technological Institute (VJTI)
Mumbai University
Mumbai, 400 019
vjti.ac.in
Other insightful authors: Bradly Alicea, Jesse Parent, Morgan Hough, Sarrah Bastawala, Mehul Arora

August 28, 2025

ABSTRACT

Open-source communities thrive on collaborative contributions, but maintaining long-term sustainability requires understanding how contributors engage, resolve issues, and manage knowledge. SustainHub addresses this challenge by introducing an adaptive agent-based model that simulates contributor behavior, task management, and knowledge sharing to optimize workload distribution and enhance community stability. The model introduces diverse agent roles: Innovators, Maintainers, Knowledge Curators, and Contributors, each contributing to project sustainability. SustainHub uses Multi-Armed Bandit (MAB) algorithms for task allocation and Q-Learning/SARSA to refine decision-making, ensuring adaptive task assignments. It incorporates Harmony Index and Resilience Quotient to assess community health, task diversity, and contributor engagement. This project aligns with the mission of the Orthogonal Research and Education Lab (OREL), an open-science lab focused on computational science and the societal impact of AI. By contributing to SustainHub, the project supports OREL's commitment to advancing open science and sustaining thriving open-source ecosystems.

SustainHub builds on prior research in open-source community sustainability using agent-based models [10].

1 Introduction

Open-source communities rely on collaboration and knowledge sharing, but sustaining long-term engagement is often difficult due to uneven task distribution, contributor dropouts, and knowledge silos. SustainHub addresses these challenges with an adaptive agent-based model that simulates contributor behavior, task allocation, and community dynamics.

The framework defines four agent roles Maintainers, Innovators, Knowledge Curators, and Maintainers to reflect real-world contributions. Task allocation is optimized using a Multi-Armed Bandit (MAB) algorithm, while Q-Learning and SARSA refine decision-making to balance workload and encourage consistent participation.

Community health is measured through two key metrics: the Harmony Index, which tracks efficiency and collaboration, and the Resilience Quotient, which evaluates adaptability during contributor dropouts. By combining adaptive task management with lightweight, modular design, SustainHub offers a scalable way to strengthen open-source ecosystems in line with the mission of the Orthogonal Research and Education Lab (OREL).

2 Reinforcement Learning in the Project

SustainHub employs Reinforcement Learning (RL) to model realistic decision-making and adaptability within open-source communities. RL provides a mathematical framework in which agents learn to make better decisions through trial-and-error interactions with the environment. In SustainHub, RL algorithms govern two key layers: (1) global task allocation through **Multi-Armed Bandits (MAB)**, and (2) local agent learning through **SARSA**.

This dual-layer design ensures that both community-wide task distribution and individual agent behavior evolve adaptively based on past outcomes.

2.1 Multi-Armed Bandits (MAB)

Building on the rule-based baseline: we introduced the **Multi-Armed Bandit (MAB)** model—specifically, *Thompson Sampling*—to make task allocation both adaptive and grounded in agent performance data.

2.1.1 Rethinking Task Assignment: The Exploration–Exploitation Tradeoff

Open-source communities thrive when contributors of all backgrounds are given opportunities, yet the best use of talent requires learning who excels at what. This creates a persistent challenge:

- **Exploration:** Trying new or lesser-used agents to discover potential and prevent stagnation. - **Exploitation:** Relying on experienced agents with a proven history of success.

The Multi-Armed Bandit framework naturally models this balance. In the SustainHub simulation:

- Agents act as the *"levers"*—each can be assigned different task types. - Tasks act as the *"pulls"*—each assignment is an experiment yielding either a success or a failure. - The system's objective is to maximize successful task completions over time, while continuously discovering and adapting to contributors' evolving strengths.

Formally, at each step t, the Maintainer selects agent a_t for a task and observes reward $r_t \in \{-1, 0, +1, +3\}$. Thompson Sampling samples from each agent-task success distribution and selects the one with the highest probability of reward. As discussed in [5], the Multi-Armed Bandit problem provides a natural framework for balancing exploration and exploitation.

2.1.2 Theoretical Value: More Than Just Math

- **Probabilistic Reasoning:** Thompson Sampling updates decisions based on observed outcomes, infusing fairness and adaptability. - **Equity by Design:** No agent is permanently sidelined; exploration ensures potential is recognized alongside past performance. - **Resilience:** The system adapts when new agents join or when agent performance drifts, reflecting real-world OSS dynamics.

2.1.3 Immediate Outcomes and Learnings

- The task assignment process became realistically unpredictable—occasionally surprising, inherently fair, and consistently adaptive. - The simulation began surfacing hidden talent, as agents with fewer prior opportunities succeeded, boosting overall project resilience. - The codebase became more modular, laying the foundation for future adaptive approaches (e.g., UCB, ϵ-greedy).

In summary, the MAB approach brought SustainHub closer to real-world dynamics of open-source collaboration by balancing fairness, adaptability, and efficiency in task allocation.

The Multi-Armed Bandit framework was a natural fit for modeling and balancing this tension. In the OSS context, agents serve as the "levers" while tasks are the "pulls." The system's goal is to maximize successful task completions while adapting to evolving strengths of contributors, aligning with broader advancements in sequential decision-making research [9].

2.2 SARSA — State-Action-Reward-State-Action

While the Maintainer (via MAB) manages global task allocation, the **individual agents** (Contributors, Innovators, Knowledge Curators) must decide how to act when tasks are assigned. For this, SustainHub employs the **SARSA** algorithm, a classic *on-policy* Temporal Difference (TD) learning method.

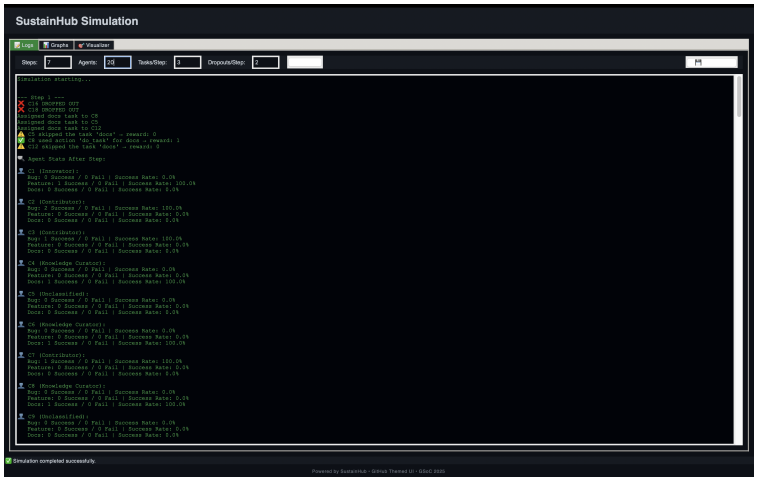

Figure 1: Illustration of Logs.

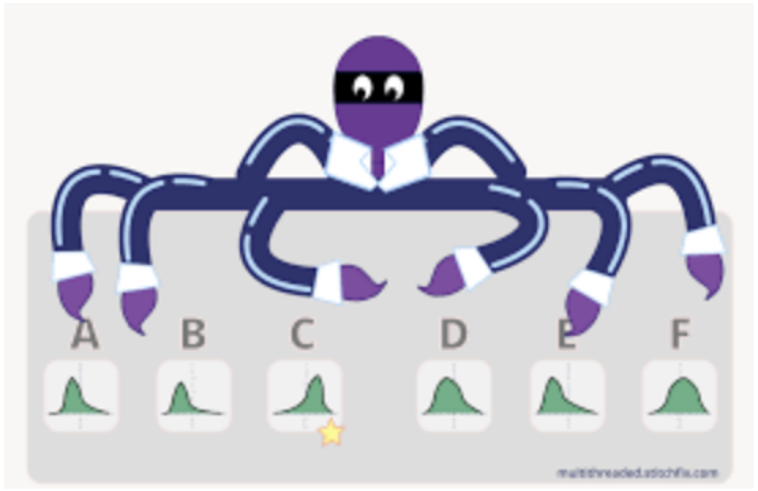

Figure 2: Multi-Armed-Bandits

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big[r + \gamma Q(s',a') - Q(s,a)\Big]$$

Where: - $Q(s,a)$ = action-value estimate for performing action a in state s. - α = learning rate (how quickly the agent adapts). - γ = discount factor (how much the agent values future rewards). - r = immediate reward (success, failure, or skip). - (s', a') = the next state and action chosen by the same policy.

2.2.1 Agent-Level Learning Process

Each agent maintains its own Q-table, where rows represent **states** and columns represent **actions**.

- **State** (s): Current situation, defined by
  - Task type (bug, feature, documentation, other).
  - Current workload (light, moderate, heavy).
  - Historical success rate.
- **Action** (a): Possible responses to a task.
  - Accept and attempt the task.
  - Skip the task (do nothing).

  - Attempt the task outside specialization.
- **Reward (r)**:
  - $+3$ for success in specialized tasks (e.g., Innovator completing a feature).
  - $+1$ for success in non-specialized tasks.
  - 0 for skipping tasks.
  - -1 for task failure.

At each step: 1. The agent observes state s. 2. Selects an action a using an ϵ-greedy policy (mostly exploit, sometimes explore). 3. Executes the action and receives reward r. 4. Transitions to new state s', chooses next action a', and updates $Q(s, a)$.

2.2.2 Why SARSA in SustainHub?

- **On-Policy Learning:** SARSA learns based on the agent's actual behavior, balancing exploration and exploitation rather than assuming optimality.
- **Encourages Specialization:** Rewards are higher for domain-specific work (bug fixes, features, docs), leading to natural role development.
- **Supports Flexibility:** Agents are not locked to one role—they can attempt other tasks, though with lower payoff.
- **Discourages Idleness:** Skipping gives zero reward, preventing free-riding behaviors.
- **Adaptive Contributors:** Models how OSS contributors realistically learn from experience, adapting strategies over time.
- SustainHub uses the **SARSA** algorithm, a Temporal Difference (TD) learning method widely applied in sequential decision-making [9].

2.2.3 Illustrative Example

Consider an Innovator agent:

1. **State s:** Assigned a *feature* task, with moderate workload.
2. **Action a:** Accepts the task.
3. **Reward r:** Completes successfully, receives $+3$.
4. **Next state s':** Task queue now contains a bug.
5. **Next action a':** Chooses to attempt the bug (non-specialized).
6. **Update:** The Q-value $Q(\text{feature}, \text{accept})$ is updated using the SARSA rule.

Through repeated episodes, the Innovator learns: - Always accept feature tasks (high expected reward). - Occasionally attempt bugs (moderate reward, useful for exploration). - Avoid overloading when failure probability is high (to minimize -1 penalties).

2.2.4 Impact on Simulation

By combining MAB (for *which agent gets the task*) with SARSA (for *how the agent responds*), SustainHub achieves a layered learning structure:

- **System-Level (MAB):** Maintainers learn allocation policies. - **Agent-Level (SARSA):** Contributors adapt behavior based on experience.

This dual mechanism creates a more realistic and adaptive ecosystem, where task assignment and task execution both improve over time.

This approach resonates with recent advancements such as State-Separated SARSA, which refines traditional SARSA to improve recovery from poor actions and stabilize long-term learning [9].

2.3 Synergy of MAB and SARSA

- **MAB (Global)**: Optimizes *who* should be assigned a task, learning which agent-task combinations yield higher rewards. - **SARSA (Local)**: Optimizes *how* an agent behaves when given a task, improving decision-making based on experience.

This layered RL approach ensures that both system-level efficiency (fair allocation, resilience) and agent-level intelligence (learning, adaptation) evolve together. By combining MAB and SARSA, SustainHub captures the complexity of real OSS communities where both coordination and individual learning are essential for long-term sustainability.

This layered design reflects modern perspectives in cooperative multi-agent learning, where hierarchical or hybrid approaches outperform single-level decision systems [5, 9].

3 Detailed Description of the Project

3.1 Agents

The system models an open-source community where four types of agents interact to contribute, review, and maintain the project.

3.1.1 Maintainer

Maintainers play a central role in coordinating the community by managing task allocation. They use the Multi-Armed Bandit (MAB) algorithm to assign tasks dynamically to agents based on availability, expertise, and prior success rates. Task completion is incentivized through a reward mechanism: agents receive higher positive rewards when they successfully complete tasks within their area of expertise, and lower (but still positive) rewards when completing tasks outside their expertise. This encourages flexibility while still valuing specialization. Skipping tasks yields zero reward, ensuring that inactivity is not reinforced, while poorly executed or incomplete tasks result in a penalty of -1. This system ensures that maintainers continuously balance efficiency and fairness in task distribution, optimizing community productivity. Great maintainers exhibit both technical and social skills [1].

3.1.2 Contributor: Bugs or Bug Fixes

Contributors focusing on bug-related tasks form one category of agents within SustainHub. Their primary responsibility is identifying, analyzing, and fixing software bugs. These agents are rewarded with higher positive values for successfully completing bug-related tasks, reflecting their specialized role in maintaining software stability. They may receive smaller positive rewards for contributing to other types of tasks (such as documentation or feature development), ensuring that their participation in non-primary areas is still recognized but less emphasized. This reward structure mirrors real-world open-source environments, where bug fixing is critical to project health, but cross-task contributions also strengthen the overall community. Understanding contributor behavior is essential for OSS sustainability [2].

3.1.3 Innovator: Features

Innovators are agents responsible for proposing and implementing new features, thereby driving the growth and evolution of the project. Innovators are responsible for proposing and implementing new features, thereby driving the growth and evolution of the project. Prior research highlights the importance of the core team in sustaining innovation within open-source communities [3]. They earn higher rewards for successfully completing feature-related tasks, such as designing new functionalities or improving existing systems. Similar to other roles, they can also contribute to tasks outside their expertise but will receive comparatively lower rewards. This encourages innovation while still allowing for flexibility and cross-collaboration. By prioritizing feature-related tasks, Innovators ensure that the project continues to evolve, adapt to user needs, and remain competitive over time.

3.1.4 Knowledge Curator: Documentation

Knowledge Curators specialize in maintaining and enhancing project documentation, tutorials, and knowledge repositories. They ensure that information is accessible, well-structured, and continuously updated, which is essential for onboarding new contributors and sustaining long-term engagement. These agents receive higher rewards for completing documentation-related tasks, reflecting the importance of knowledge management in community sustainability. They may also participate in other roles, receiving lower rewards to acknowledge their secondary contributions. By promoting transparency and knowledge sharing, Knowledge Curators reduce information silos and enhance collaboration across the community.

3.1.5 Agent Summary

The table below summarizes the four agent types in SustainHub, their primary responsibilities, and the associated reward structure:

Agent Type	Specialization	Reward (Success)	Reward (Failure / Skip)
Maintainer	Task allocation using Multi-Armed Bandit (MAB)	Indirect rewards via community efficiency	
Contributor	Bug fixing and stability improvement	+3 (bug tasks), +1 (others)	-1 (failure), 0 (skip)
Innovator	New feature design and implementation	+3 (feature tasks), +1 (others)	-1 (failure), 0 (skip)
Knowledge Curator	Documentation, tutorials, knowledge sharing	+3 (documentation tasks), +1 (others)	-1 (failure), 0 (skip)

Table 1: Summary of Agent Roles and Reward Mechanisms in SustainHub

3.2 Why we need the above agents?

These agents ensure the smooth functioning and sustainability of an open-source community by distributing responsibilities effectively—maintainers uphold code quality, contributors drive development, innovators bring fresh ideas, and knowledge curators preserve essential project information for long-term accessibility. Their collaboration fosters continuous improvement, stability, and inclusivity within the project ecosystem.

3.3 Metrics

To assess the sustainability and efficiency of the community, SustainHub introduces three core evaluation metrics: the Harmony Index (HI), the Resilience Quotient (RQ), and the Reassignment Overhead (RO). These metrics provide a multi-dimensional perspective on balance, adaptability, and efficiency in the system, ensuring that both short-term performance and long-term sustainability are captured.

3.3.1 Harmony Index (HI)

The Harmony Index captures how evenly tasks are distributed among agents and how successfully they are completed. An optimal system should maintain both a high success rate and a low workload imbalance. The Harmony Index has been widely applied to evaluate coordination in multi-agent systems [6].

$$HI = 0.6 \times \text{Avg Success Rate} + 0.4 \times \frac{1}{1 + \text{Load Variance}}$$

Here, the average success rate reflects the ability of agents to complete tasks, while load variance measures fairness in workload distribution. If one agent is consistently overloaded while others remain underutilized, the variance increases, lowering the Harmony Index.

The index ranges between 0 (poor balance) and 1 (perfect balance). A high HI indicates that the community is functioning efficiently, with contributors equally engaged and completing tasks successfully. A low HI highlights inefficiencies such as uneven workload or reduced collaboration.

Importance: The Harmony Index is critical because sustainable communities require both efficiency and fairness. Even if tasks are completed, overburdening a subset of contributors leads to burnout and dropout risks. By rewarding balanced workload and success, HI ensures long-term stability and active participation.

3.3.2 Resilience Quotient (RQ)

The Resilience Quotient measures the community's ability to adapt and recover from disruptions such as agent dropouts, inactivity, or sudden workload spikes. It captures how quickly the system restores stability after shocks.

$$RQ = 0.4 \times TRE + 0.3 \times SRR + 0.3 \times HS$$

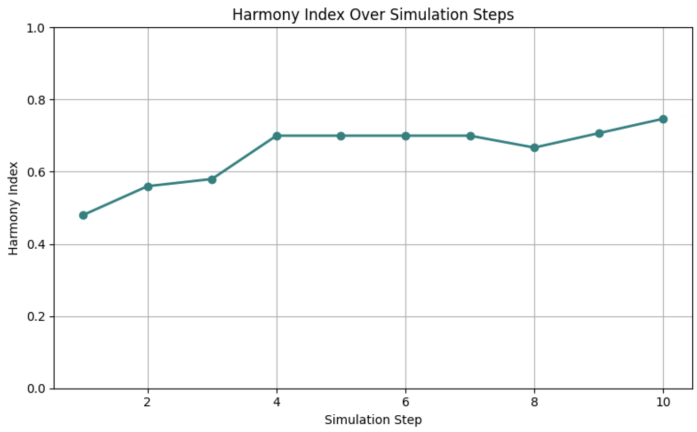

Figure 3: Illustration of Harmony Index for the first run.

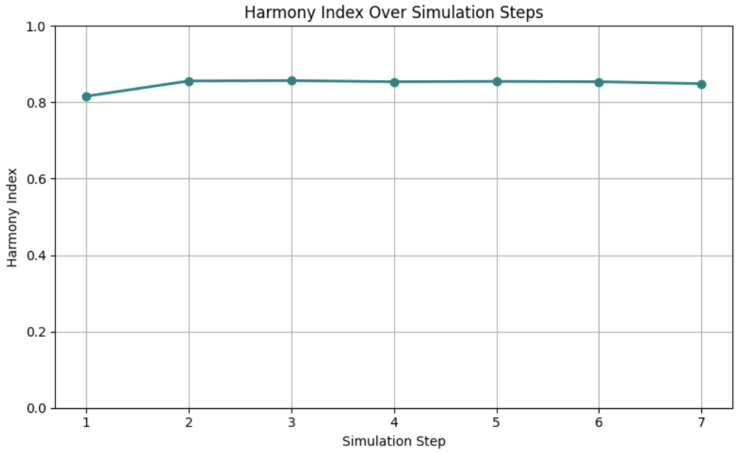

Figure 4: Illustration of Harmony Index after a few runs.

Where TRE (Task Reallocation Efficiency) evaluates how effectively tasks are reassigned, SRR (Success Rate Recovery) tracks the ability to regain pre-disruption task success levels, and HS (Harmony Stability) measures whether balance is maintained during recovery.

The quotient ranges between 0 (low resilience) and 8 (high resilience). A system with high RQ can recover smoothly from disruptions, while a low RQ indicates vulnerability to stagnation or collapse.

Importance: Open-source communities often experience fluctuating participation, with contributors leaving or becoming inactive. The Resilience Quotient ensures that the system is not only efficient during normal operations but also robust when faced with disruptions. High resilience reduces the risks of backlog accumulation and project stagnation, making it a vital metric for long-term sustainability. The concept of Resilience Quotient has been proposed to evaluate adaptive capacity in systems [7].

3.3.3 Reassignment Overhead (RO)

The Reassignment Overhead quantifies the frequency of task reallocations within the system. Ideally, the initial allocation should minimize reassignments, as frequent changes introduce inefficiency and instability.

$$RO = \frac{\text{Reassigned Tasks}}{\text{Total Tasks Assigned}}$$

Figure 5: Illustration of Resilience Quotient with high dropouts.

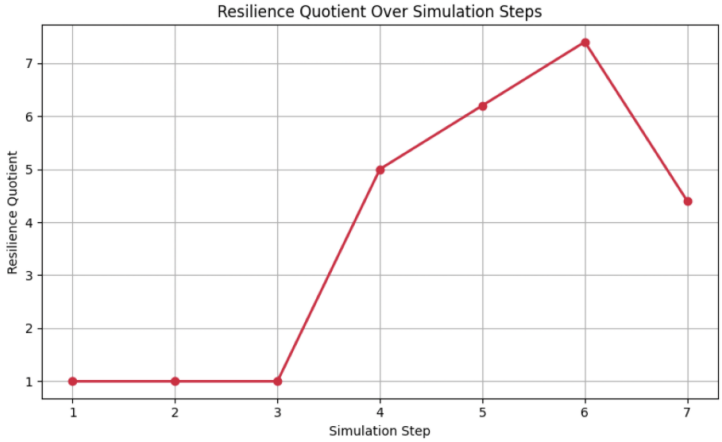

Figure 6: Illustration of Resilience Quotient with low dropouts.

This ratio ranges between 0 and 1. A lower RO indicates that most tasks are correctly allocated the first time, demonstrating efficiency and effective use of resources. A higher RO suggests instability, poor initial matching, or frequent agent dropouts requiring constant task redistribution.

Importance: Tracking RO is essential because excessive reassignments waste contributor effort and delay project progress. While some reassignment is natural in dynamic systems, minimizing overhead ensures that contributors remain focused and that resources are utilized efficiently. A low RO supports steady task completion and reduces friction in collaboration.

3.3.4 Metric Summary

Together, these three metrics provide a comprehensive evaluation framework. The Harmony Index ensures balance and fairness in workload distribution, the Resilience Quotient evaluates adaptability to disruptions, and the Reassignment Overhead highlights allocation efficiency and system stability. By combining these perspectives, SustainHub can measure not just performance but also the sustainability and robustness of open-source communities.

4 Graphical User Interface

The Graphical User Interface (GUI) of SustainHub was implemented using Python's `Tkinter` framework. The design was inspired by GitHub's dark theme to provide a clean, modern, and developer-friendly environment for interacting with the simulation. The GUI is divided into three primary sections, organized into tabs: **Logs**, **Graphs**, and the

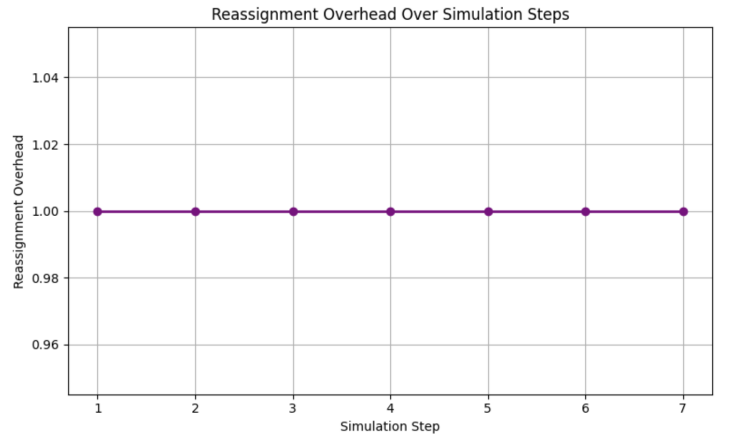

Figure 7: Illustration of Reassignment Overhead (RO).

Metric	Purpose and Importance	Range
Harmony Index (HI)	Ensures balanced workload and high success rates, preventing contributor burnout and maintaining fairness.	0 (imbalanced/low) $\rightarrow$ 1 (balanced/high)
Resilience Quotient (RQ)	Evaluates adaptability to disruptions, ensuring the community recovers quickly from dropouts or workload shifts.	0 (low resilience) $\rightarrow$ 8 (high resilience)
Reassignment Overhead (RO)	Measures efficiency of task allocation by tracking reassignments. Lower values indicate stable, efficient operations.	0 (not reassigned) $\rightarrow$ 1 (reassigned)

Table 2: Summary of Metrics in SustainHub

Visualizer. This organization allows the user to switch seamlessly between monitoring textual output, analyzing quantitative metrics, and observing a dynamic visual representation of the simulation.

4.1 Logs

The **Logs tab** serves as the primary console output and configuration panel for running simulations. At the top of the tab, users can configure simulation parameters: the number of *steps*, the number of *agents*, the number of *tasks per step*, and the number of *dropouts per step*. These inputs are provided as text entry fields with intuitive labels, ensuring that users can customize the simulation without modifying the underlying code. The "Run" button triggers the simulation with the provided parameters, while a dedicated button is provided to export the textual logs as a `.txt` file for future reference.

Below the configuration panel, a scrollable text window is embedded using the `ScrolledText` widget. This output area is styled with a monospaced font (`Courier New`), green text on a black background, replicating a classic terminal look. Standard output from the simulation is redirected to this area, ensuring all events, agent behaviors, and performance messages are visible to the user in real time. The text area is locked to prevent accidental edits, but it automatically scrolls down to display the latest updates as the simulation progresses. Together, these design choices make the Logs tab both a control hub and a transparent debugging console.

4.2 Graphs

The **Graphs tab** provides a visual representation of three key performance metrics:

- **Harmony Index** – quantifying coordination among agents.
- **Resilience Quotient (RQ)** – measuring adaptability to dropouts and failures.
- **Reassignment Overhead (RO)** – indicating the additional cost of reallocating tasks.

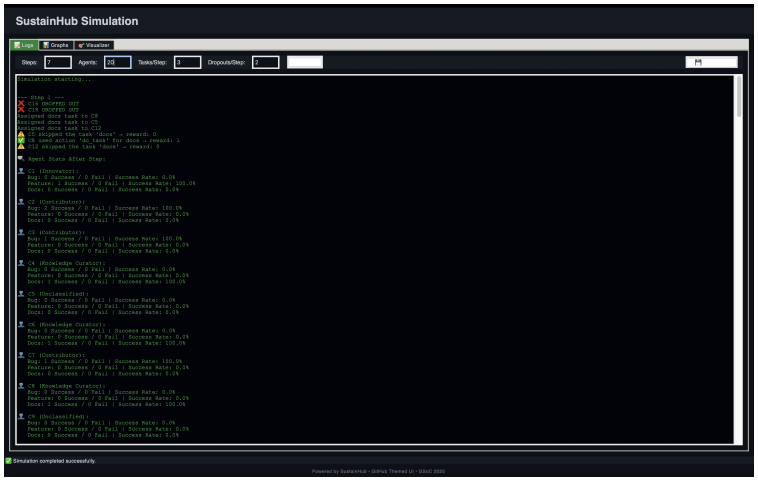

Figure 8: Illustration of Logs.

These metrics are displayed in a vertically stacked layout of three subplots, implemented using the `Matplotlib` library. Each graph is rendered on a dark background (#161b22) with white axis labels and colored line plots (cyan, orange, and magenta respectively) for visual clarity. This consistent color-coded representation allows users to distinguish metrics at a glance and identify performance patterns across simulation steps.

A "Save Graph" button is provided to export the plotted figures as a `.png` file, enabling users to document results or include visual evidence in reports. The Graphs tab thus acts as a performance dashboard, transforming raw simulation data into an intuitive and analyzable form.

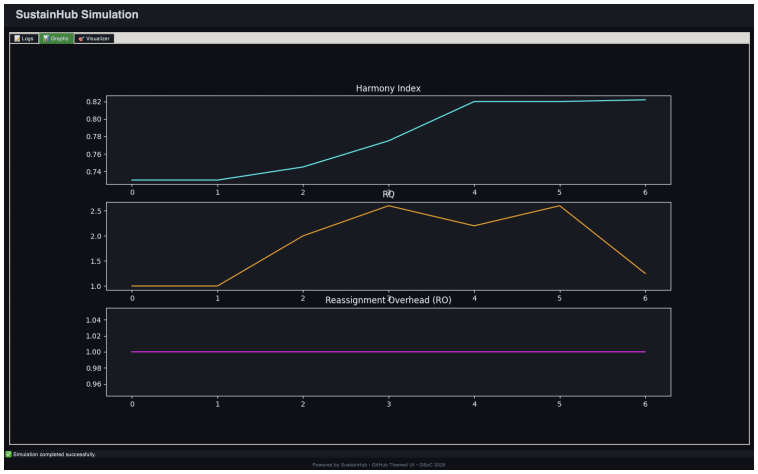

Figure 9: Illustration of Graphs.

4.3 Visualizer

The **Visualizer tab** is designed to present a more interactive and illustrative perspective of the simulation. It uses a `Canvas` widget to represent agents as blue circles and tasks as yellow squares, positioned randomly within the simulation space. Once the simulation runs, agents animate step-by-step, moving incrementally toward their assigned tasks. This motion conveys how agents coordinate in real time to complete work, providing an immediate visual understanding of allocation dynamics.

The animation is controlled using Tkinter's `after` scheduling method, which updates positions at timed intervals, thereby simulating continuous movement. This design not only improves engagement but also assists in intuitively grasping abstract concepts such as reassignment and task distribution.

Additionally, a button labeled “Launch NetLogo” is available in this tab. If the NetLogo integration module is available, it allows the simulation to be visualized in the NetLogo environment, where more complex and agent-based visualization can be explored. This optional feature demonstrates the extensibility of the interface, bridging lightweight Tkinter visualization with advanced modeling tools.

Overall, the Visualizer tab provides an accessible way for users to see the abstract computations of SustainHub come alive, making the system more transparent and comprehensible.

Figure 10: Illustration of Visualisation.

5 Outputs Explanation

5.1 Harmony Index

The Harmony Index (HI) reflects how evenly distributed the workload is across agents while considering the overall success rate of task completion. - A **higher HI value (closer to 1)** indicates that contributors are not overloaded, work is fairly shared, and most tasks are completed successfully. - A **lower HI value (closer to 0)** signals imbalance, meaning some agents are overburdened while others remain idle, or many tasks are failing.

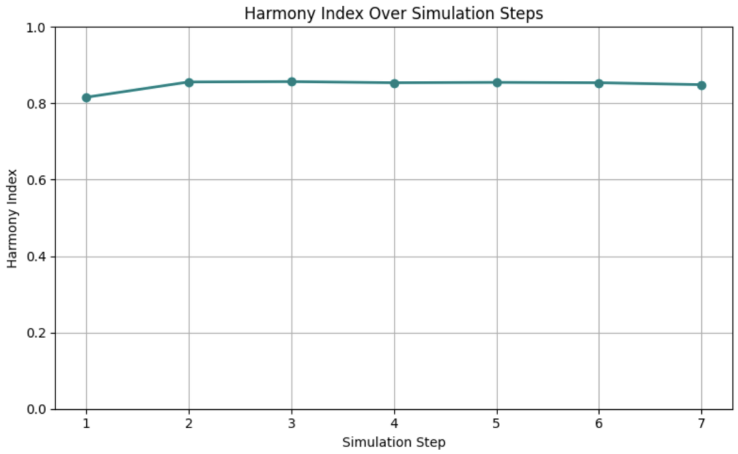

Figure 11: Harmony Index over simulation steps. A higher value indicates balanced collaboration.

—

5.2 Resilience Quotient (RQ)

The Resilience Quotient (RQ) captures how well the community adapts to disruptions, such as sudden agent dropouts or spikes in workload. - A **high RQ (above 0.7)** means the system quickly reallocates tasks and recovers from failures. - A **low RQ** indicates that dropouts or disruptions cause significant delays and instability.

Figure 12: Resilience Quotient showing adaptability under disruptions.

—

5.3 Reassignment Overhead (RO)

The Reassignment Overhead (RO) measures how many tasks needed to be reassigned because the original agent skipped or failed them.

$$RO = \frac{\text{Reassigned Tasks}}{\text{Total Tasks Assigned}}$$

- A **good RO is high (closer to 1)**, meaning tasks that were initially skipped or failed are successfully reassigned and completed. - A **low RO** suggests inefficiency, where reassigned tasks are still not being completed effectively.

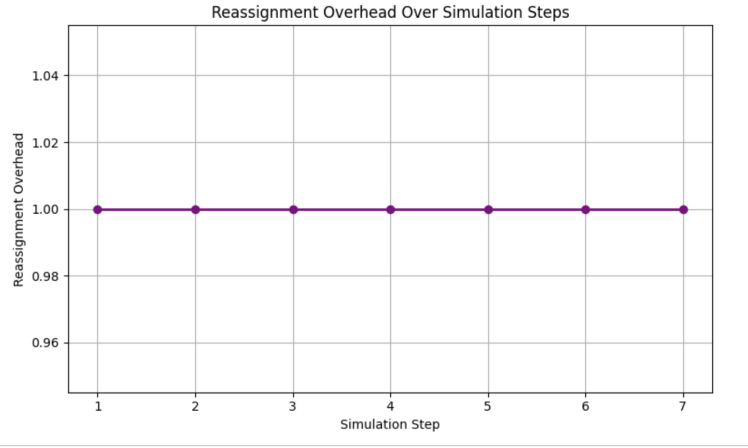

Figure 13: Reassignment Overhead across steps. A higher value indicates effective reassignment.

—

5.4 Success Heatmap

The success heatmap visualizes how agents perform across different task types. - **Darker regions** indicate higher success rates. - **Lighter regions** show areas where agents frequently fail or skip tasks.

This helps identify role specializations and weaknesses — for example, contributors performing strongly on bug fixes while innovators excel on feature tasks.

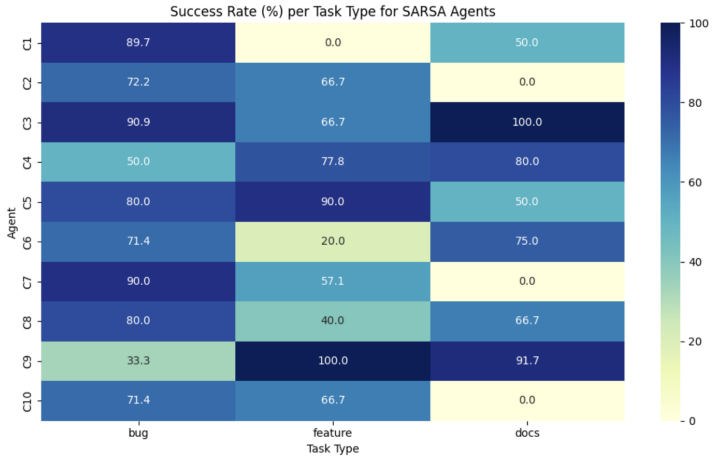

Figure 14: Heatmap of agent success rates across task types. Darker is better.

—

5.5 Logs

Logs provide a **step-by-step textual record** of the simulation as it runs. They include details such as: - Which agent received which task. - Whether the task was completed, skipped, or failed. - Success or failure counts per step. - Updates to learning values (SARSA Q-values).

The log system makes the simulation **transparent and debuggable**, allowing researchers to trace how decisions were made and how metrics evolved.

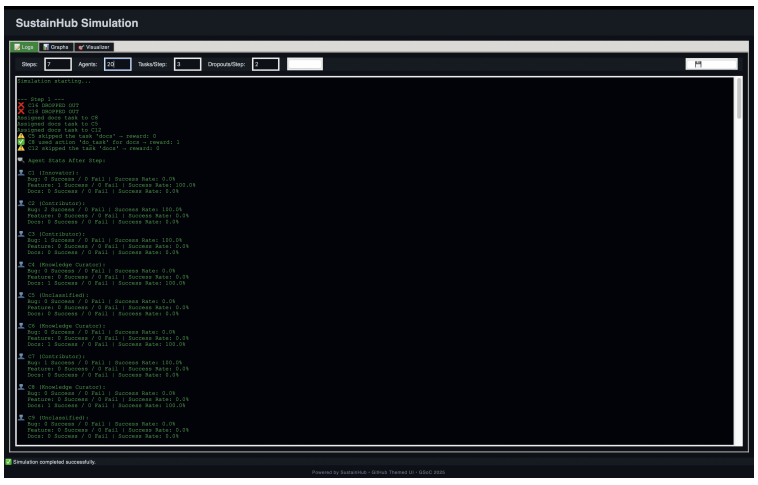

Figure 15: Example of real-time logs generated during the simulation.

—

5.6 Visualisation

The visualization tab provides a **graphical view of the simulation dynamics**: - **Blue circles** represent agents (contributors, innovators, maintainers, knowledge curators). - **Yellow squares** represent tasks waiting to be completed. - The number of **blue circles** reflects how many active agents remain (dropouts reduce this number over time). - The movement of agents toward tasks shows allocation and completion in real-time.

This provides an **intuitive understanding** of how tasks are distributed, reassigned, and completed, complementing the numerical metrics.

Figure 16: Visualization of agents (blue) and tasks (yellow) during simulation.

References

[1] Edson Dias, Paulo Roberto Miranda Meirelles, Fernando Castor, and Igor Steinmacher. What Makes a Great Maintainer of Open Source Projects. May 2021.

[2] Jinghui Cheng and Jin L.C. Guo. Analysis of Open Source Software Contributors. March 13, 2019.

[3] Rocio Martinez-Torres, S.L. Toral, M. Perales, and Federico Barrero. Analysis of the Core Team Role in Open Source Communities. June 2011.

[4] Javier Luis Cánovas Izquierdo and Jordi Cabot. On the Analysis of Non-Coding Roles in Open Source Development. September 28, 2021.

[5] Aleksandrs Slivkins. Introduction to Multi-Armed Bandits. April 2020.

[6] Darryl Roman, Noah Ari, and Johnathan Mell. The Harmony Index: Evaluating, Predicting, and Visualizing Effectiveness in Multi-Agent Team Dynamics. 2010.

[7] Melissa De Iuliis, Omar Kammouh, and Gian Paolo Cimellaro. Resilience Quotient. June 20, 2022.

[8] Afshin Oroojlooy and Davood Hajinezhad. A Review of Cooperative Multi-Agent Deep Reinforcement Learning. April 30, 2021.

[9] Yuto Tanimoto and Kenji Fukumizu. State-Separated SARSA: A Practical Sequential Decision-Making Algorithm with Recovering Rewards. March 18, 2024.

[10] Bradly Alicea, Hussain Ather, Himanshu Chougule, Jesse Parent, and others. Open-source Community Sustainability using Agent-based Models. March 2023. Orthogonal Research and Education Laboratory. License: CC0.