Journal of Computer Sciences and Applications, 2025, Vol. 13, No. 1, 16-28
Available online at http://pubs.sciepub.com/jcsa/13/1/2
Published by Science and Education Publishing
DOI:10.12691/jcsa-13-1-2

Temporal Analysis of an IoT Distributed Ledger Simulation using NetLogo and Agents.jl

Peter Kimemiah Mwangi*, Stephen T. Njenga, Gabriel Ndung'u Kamau

School of Computing Information Technology, Murang'a University of Technology, Murang'a County, Kenya
*Corresponding author: pkimemiah@student.mut.ac.ke

Received May 16, 2025; Revised June 18, 2025; Accepted June 26, 2025

Abstract Agent-Based Modelling (ABM) tools provide a cost-effective way to simulate complex systems like an Internet of Things Distributed Ledger Technology (IoT-DLT) networks, where nodes operate as autonomous agents. While physical testbeds are expensive, ABMs offer scalable and efficient alternatives. However, few studies compare ABM performance on standard consumer hardware. In this research, we evaluate NetLogo 6.3 and Agents.jl (Julia 1.9) by simulating an IoT-DLT model across two laptop configurations. Results show that Agents.jl runs up to 9× faster on newer hardware and 4× faster on older hardware compared to NetLogo, though it requires more setup. NetLogo remains user-friendly but underutilises system resources like GPU and multicore processing. The research uses inferential analysis tools, such as regression analysis, to rigorously evaluate the performance differences between the ABM tools and hardware configurations. This research helps researchers choose efficient ABM tools for large-scale simulations on personal computers, demonstrating that emerging tools like Agents.jl are promising candidates for future simulations.

Keywords: *Agent Based Modelling, Distributed Ledger Technology, Internet of Things, Julia, NetLogo, Performance, Simulation*

Cite This Article: Peter Kimemiah Mwangi, Stephen T. Njenga, and Gabriel Ndung'u Kamau, "Temporal Analysis of an IoT Distributed Ledger Simulation using NetLogo and Agents.jl." *Journal of Computer Sciences and Applications*, vol. 13, no. 1 (2025): 16-28. doi: 10.12691/jcsa-13-1-2.

1. Introduction

Complex systems consist of interacting individual components that exhibit emergent global behaviour. Multi-Agent Systems (MAS) are widely used to model such systems, simulating real-world interactions where agents operate autonomously and engage in social behaviour like cooperation and negotiation [1]. Agent-Based Modelling (ABM) has emerged as a key tool for empirical research, enabling "bottom-up" analysis of complex systems [2]. In ABM, agents follow predefined rules, and their collective interactions reveal emergent properties, making ABM particularly valuable for studying Internet of Things with Distributed Ledger Technology (IoT-DLT) networks, where scalability and cost pose challenges for physical testbeds [3] and [4].

While ABM tools like NetLogo offer abstraction layers for model development, their performance on standard hardware remains understudied. Efficient resource utilisation, such as multicore/GPU support, can significantly reduce simulation time and costs. This research addresses this gap by comparing two ABM tools: NetLogo 6.3 (a user-friendly, established platform) and Agents.jl (Julia 1.9) (a high-performance alternative touted for scalability). This study evaluates these tools using an IoT-DLT simulation, ported from NetLogo to Agents.jl, with the following goals:

- Evaluate temporal Performance measurement: Comparison of mean simulation times across different network sizes.
- Perform a temporal runtime difference between each ABM tool on different generations of laptops.
- Identify the trade-off between performance and non-performance strengths and limitations of each ABM tool.
- Measure speed-up and show how modern tools like Agents.jl can outperform traditional ABMs in efficiency.

This research provides a practical framework for selecting ABM tools when modelling transaction-intensive, multi-agent systems like IoT-DLT networks, using the IoT-Direct Acyclic Graph-Based Distributed Ledger Technology Simulation (IoT-DAG-DLTSim) model as a benchmark case. By establishing performance baselines that demonstrate execution speed differences between NetLogo and Agents.jl across hardware generations, the study offers empirically validated criteria for tool selection.

The research also presents a refined experimental approach, which included rerunning the simulations to ensure robustness. Furthermore, this work incorporates inferential statistical analysis and specifically regression modelling to provide a rigorous and statistically validated understanding of the factors influencing simulation

performance, such as the choice of ABM tool, hardware configuration, and simulation scale.

The research develops a decision matrix comparing computational efficiency, development complexity, and hardware utilisation capabilities specifically for simulations requiring high transaction throughput and concurrent agent interactions. Furthermore, it validates how emerging ABM architectures (Agents.jl) overcome scalability limitations of traditional platforms (NetLogo) in large-scale IoT-DLT scenarios, providing researchers with both quantitative metrics and qualitative evaluation guidelines for tool adoption.

The rest of the paper is structured as follows: Section 2 reviews related work; Section 3 details the materials and methods; Section 4 presents the results obtained; Section 5 discusses implications of the results; and Section 6 concludes with a summary of the work and future directions for IoT-DAG-DLT scalability performance research.

1.1. Related Studies

Recent research has emphasised the growing role of Agent-Based Modelling (ABM) tools in simulating complex systems, particularly for domains like IoT and distributed ledgers. Surveys by [5,6] and [7] provide overviews of widely used ABM frameworks and their applicability in various research domains. These reviews collectively highlight tools such as NetLogo [8], MASON [9], Mesa[10], FLAME GPU [11], and the more recent Agents.jl [12], each offering trade-offs between usability, flexibility, and performance.

Several studies have attempted to benchmark the computational performance of ABM platforms, especially when simulating models with high agent counts or requiring intensive concurrent interactions. For instance, in [11], the authors introduce FLAME GPU 2, a high-performance ABM framework written in C++ with GPU acceleration capabilities. Their work demonstrates how parallel execution and minimised data movement can achieve simulation speedups of up to 10× compared to traditional GPU baselines. However, such tools often require specialised hardware (e.g., NVIDIA V100 GPUs) and expertise in parallel programming, limiting accessibility for researchers using standard laptops or desktops.

The ABM tool Agents.jl, developed in Julia, has emerged as a performant and flexible alternative for ABM development. In [12], the authors compare Agents.jl to NetLogo, Mesa, and MASON across a suite of well-known ABM benchmarks, such as Flocking and Schelling. The study evaluates both runtime performance and development effort, showing that Agents.jl delivers competitive performance with significantly fewer lines of code, making it suitable for large-scale simulations on modern CPUs. However, their analysis does not explore performance variability across different hardware generations.

The ABM tool NetLogo, by contrast, remains a widely adopted ABM platform due to its low barrier to entry and integrated modelling environment. Its strengths lie in accessibility and ease of use, particularly for education and early-stage prototyping [9]. Yet, several studies (e.g., [11,12]) have noted its limitations in utilising multicore processors or GPUs, which can restrict its scalability for computation-heavy simulations like those seen in IoT-DLT networks.

MASON, as discussed in [9], offers extensibility and modular design through its Java-based architecture. It separates model logic from visualisation, facilitating headless simulations and batch runs. Though it performs better than NetLogo in some benchmarks, MASON still falls short of tools like FLAME GPU or Agents.jl in terms of raw speed, especially for large-scale agent interactions.

While these works offer valuable comparisons across frameworks, few have examined how ABM tools perform on everyday consumer-grade hardware, an increasingly relevant consideration for researchers without access to high-performance computing resources. This study addresses this gap by benchmarking NetLogo and Agents.jl using the same IoT-DLT simulation model across two common laptop configurations, quantifying runtime differences and identifying practical trade-offs in tool adoption for real-world deployment contexts.

In this research, an experiment was performed by running simulations of the IoT-DAG-DLTSim model on two different multicore/GPU laptops, hosted on a multithreaded, multicore operating system. The model was designed to study the effect of horizontal scalability on IoT-DAG-DLTSim, and the effect of using different algorithms to attach light IoTs, which are not capable of storing their own distributed ledger database to full nodes. The results of the simulation were collected using CSV files and analysed to provide the required results based on the objectives.

2. Methods and Materials

In this research, an experiment was performed by running simulations of the IoT-DAG-DLTSim model on two different multicore/GPU laptops, hosted on a multithreaded, multicore operating system. The model was designed to study the effect of horizontal scalability on IoT-DAG-DLTSim, and the effect of using different algorithms to attach light IoTs, which are not capable of storing their own distributed ledger database to full nodes. The results of the simulation were collected using CSV files and analysed to provide the required results based on the objectives.

2.1. Model Design

The IoT-DAG-DLTSim model was designed using the Agent Unified Modelling Language (A-UML) approach and implemented using two tools: NetLogo 6.3, an ABM based on Java Virtual Machine (JVM), and Agents.jl Julia 1.9, an ABM based on Low-Level Virtual Machine (LLVM). Model performance was measured and focused on IoT device scalability, transaction throughput, latency, and model completion times.

The IoT-DAG-DLTSim ecosystem design is shown in Figure 1(a) includes a set of Agents, $A \in [A_1, A_2 \ldots A_j]$, some having active ledgers, others with no ledger, and the Agents connect to each other via the network. This is depicted as a three-layer model, Figure 1(a), as shown, and includes random network internet connections

between agents. Individual agents were designed to use a gossip-like protocol to receive, process, and send messages. Transaction processing involved handling new data and replication requests, local processing, and gossiping information to neighbours.

During each epoch, 0.1% of the Agents randomly produced transactions. Agents receive or generate transactions and update their local database. A Gossip SIR [13] protocol is employed to propagate these transactions to neighbouring agents. New transactions are added to the Direct Acyclic Graph (DAG) Ledger database as new tips by attaching them to two existing tips in the local database. The transactions are then gossiped using the Gossip SIR algorithm to neighbouring agents for replication. When transactions for replication are received, the DAG Ledger attaches them to the corresponding DAG tips, if they have been received. If not received, the DAG Ledger waits for the required tips. If waiting takes too long, agents request the missing DAG transactions from neighbours using a Gossip SIR-like algorithm.

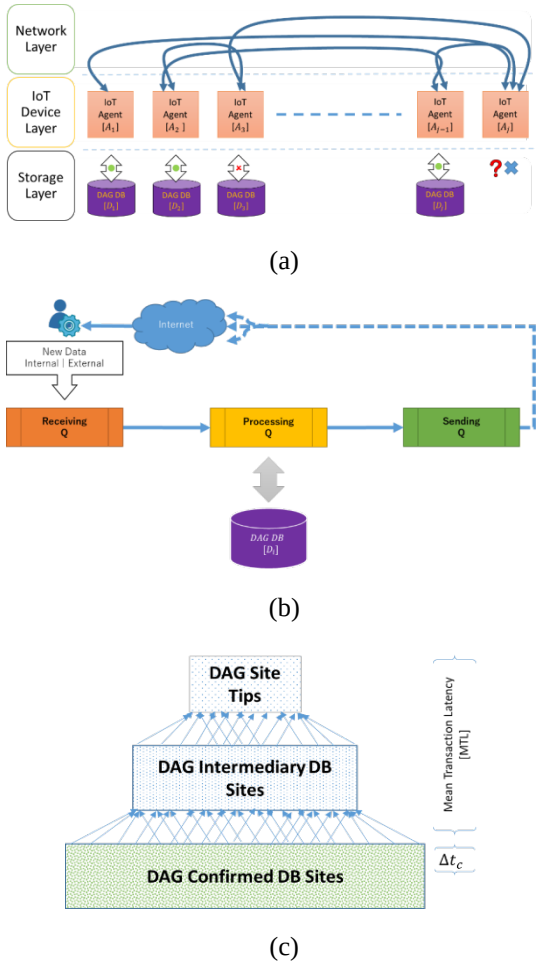


Figure 1. (a) Three-layer model Architecture of the IoT-DAG-DLT ecosystem with random peer-to-peer connections and Local DAG-DLT databases storage. (b) Individual process cycle of local agents. The feedback loop communicates to neighbouring agents via a Gossip SIR-like protocol. (c) A global view of the DAG-DLT with local tips, which are then replicated to neighbouring agents until more than ¾ of DAG-DLTs' DBs have received the same transaction.

The key components of the agent model are:

- Agents: Classified as Full-agents (with a local DAG-Ledger database) or light-agents (with no DAG-Ledger database).
- Agent network: Agents are interconnected using a scale-free random network. This uses the Barabasi and Albert Preferential Attachment (PA) algorithm [14], simulating internet-like connections with a k-degree of 2.3, based on empirical data.
- DAG Ledger: Each agent maintains a local database to store new transactions and replicate neighbours' transactions. The DAG Ledger is built using an IOTA Tangle [15] like random algorithm.
- Gossip Protocol: Agents send and receive messages using the random Gossip SIR model. This protocol performs a push for new messages and a pull for any missing DAG tips based on the age of the updates.

The Figure 1(c) shows the impact of the percolation of the DAG Ledger transactions over time, starting as database (DB) tips, then intermediary tips as the databases became synchronised with each other. The final stage is reached when ¾ of the DAG-Ledger DB sites have received the same transaction tips, this achieves the required Byzantine Fault Tolerance to allow a transaction to be deemed confirmed [16].

2.2. Model Parameters

The Table 1, shows the parameters used to set-up and run the experiment. The input parameters determine the size of the PA network, which changes during successive experiments in order to simulate horizontal scalability. The remaining inputs remain the same. There are various output parameters which can be configured. For this experiment, only the time taken to set up and run the experiment is used.

Table 1. Model Input and Output Parameters

Parameter	Values
Input	
No of agents in the DLT-DAG network (N_{total})	[100, 200, 400, 800, 1600, 3200, 6400]
Agent Types (N_{full}, N_{light})	[1 ,10]
Agent Ratio	$N_{light} = 0.1 * N_{total}$, $N_{full} = 0.9 * N_{total}$
Mean degree of agent connectivity ($k - degree, \kappa$)	2.3
Agent Heterogeneity (H)	0
Epochs per experiment (i)	$i \epsilon [1 \ldots 1000]$
Transaction generated per epoch (λ_i)	$0.01 * N_{full}$
Output	
Time taken to setup experiment(seconds)	T_{setup}
Time taken to run experiment(seconds)	$T_{experiment}$
Transactions per epoch	λ_i
Transactions generated per experiment (Λ)	$\sum_{i=1}^{1000} \lambda_i$
Global Transaction Confirmation (β)	$\lambda(t_{confirm})_i, t_{confirm} \neq -1, \sum_{j=0}^{N_{full}} t_{confirm} > 0.67$
Transaction Latency (δ)	$\delta = \lambda(t_{confirm} - t_{creation})_i$
Confirmed Transactions (i)	$\sum_{i=1}^{n} \lambda_i, t_{i+n} \neq -1$

2.3. Experimental Setup

The mean simulation time was measured over 30 runs of the model, each running for 1000 time epochs. These experiments were conducted on two machines, with hardware specifications detailed in Table 2. The Figure 2 presents a high-level pseudocode illustrating the model design. It includes two main routines: "**Pseudo Code 1** Setup", initialising and setting up the simulation, and "**Pseudo Code 2** Run Experiment" shows the behaviour of each full node, which involves receiving data, processing and storing it in the DAG-based DLT, and broadcasting transactions to neighbouring agents via a gossip protocol. The IoT agent network topology was generated using the Preferential Attachment (PA) algorithm from [17], with a degree parameter $K = 2.3$, which approximates real-world internet router connectivity patterns.

Table 2. Experimental Tools Setup and Configuration, Two Machines Used in the Experiment

Configuration A - Toshiba i3 (TOi3)
• Toshiba Satellite C850 • Intel, Core i3-2348M, 2-Core Processor, 4 logical processors 2.3GHZ, 12GB DDR3 RAM, L1 128KB, 512KB, 3MB • Intel HD Graphics 3000, • 400GB SATA HDD • Windows 10 pro, 10.0.19045,64bit • Java SE JDK 17.0.4 (64bit) • NetLogo 6.3, extensions (array bitarray csv ls lt nw) profiler py table time) • Julia 1.9, Agents.jl 5.17
Configuration B – HP Victus I5 (HPi5)
• HP Victus 15 FAO • Intel i5-, 2.0GHz, 32 GB DDR4 RAM L1 704KB, L2 7MB, L3 12MB • Intel HD Graphics 4000, NVIDIA GTX 1650, • 400GB SSD Disk • Windows 10 Pro, 10.0.19045 • Java SE JDK 17.0.4 (64bit) • NetLogo 6.3, extensions (array, bitarray[1], csv, ls, lt, nw, profiler, py, table, time) • Julia 1.9, Agents.jl 5.17

Pseudo Code 1: Setup

```
1: do setup
2:     create-agents [ 1:10] N_total
3:     create-PA-network (2.3) #k − degree
4:     create-local DAG DB Ledger for full node
5:     create Global DAG DB for global view
5: end
```

Pseudo Code 2: Run Experiment

```
1: do go
2:   for ℜ in 1:30 # experimental runs
3:     for epoch in 1:1000 #perform epoch iterations
4:         for each A_i # agent
5:             generate-transactions ρ = 0.01
6:             receive-message (A_0 or A_j)
7:             process-data
8:             add-to-local DAGLedger (λ_i)
               # new tips or replicas
9:             update Global DAG DB
10:            send-message (A_i ....)
               #send message to neighbouring agents
11:          end
12:        extract data to csv # collect data
13:    end
14:  end
15: end
```

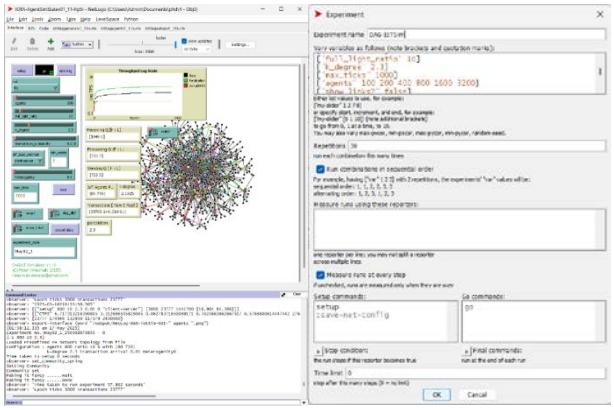
Figure 2. IoT-DLT model - high-level pseudo code showing a snippet of setup and running of the experiment

[1] Custom developed extension

In Figure 2, $\Re$ represent the experiment run being executed, N_{total} represents the total number of nodes simulated, $k - degree$ is how the full agents interconnection topology, $epoch$ is the time sequence, A_i is the agent, ρ is the probability of an agent generating a new transaction, λ_i is the received transaction that need to be attached to the local DAG Ledger, either new or replication.

2.3.1. NetLogo Setup

NetLogo 6.3 comes with a Graphical User Interface (GUI) and a behaviour module that allows you to set up experiments, their input parameters, and the output parameters. Figure 3(a) shows the standard GUI of NetLogo tool and the designed interface, while Figure 3(b), shows the tool used to configure and run the experiments and perform parameter iterations.



(a) (b)

Figure 3. (a) IoT-DAG-DLTSim NetLogo 6.3. GUI Interface, (b) NetLogo 6.3 Behaviour Experiment Setup tool used to run experiments based on parameters under investigation

Note that only the number of agents where changed i.e. 100 to 6400 for each experiment, and the experiment was repeated 30 times. The data was exported from the experiment using the NetLogo CSV extension [18], and the data loaded into Julia 1.9 using CSV,jl and analysed using the Julia 1.9 DataFrames.jl [19], StatsBase.jl [20], HypothesisTesting.jl [21], GLM.jl [22], Plots.jl [23], CurveFit.jl [24] package libraries to generate the descriptive, inferential and visualisations.

During development, the IoT-DAG-DLTSim application was tuned using the data profiling tools available as an extension in Julia. This allowed for the selection of improved techniques to improve the performance [25]. The JupyterLab was used for the data analysis to allow documentation of the data collection and analysis process.

2.3.2. Agents.jl Julia 1.9

Agents.jl Julia ABM does include a default GUI and users manage the tool via VS Code IDE or the native Julia command prompt (see Figure 4). A JupyterLab [26] interface is also available. Julia offers a REPL (read, evaluate, print, loop), allowing users to input commands and see the results displayed immediately. The model interaction can be invoked using either the Makie.jl, WGLMakie.jl or GLMakie.jl [27] libraries. These libraries

allow for image, video capture or custom GUI interactions. In this experiment, the Agent.jl GUI was not used during experimental runs, opting for non-interactive mode to improve performance. However, the GUI graphics were utilised extensively in creating visualisations, providing user-selectable high-quality outputs in PNG(Portable Network Graphics), SVG(Scalable Vector Graphics), and PDF (Portable Document Format) file formats.

The data was loaded into Julia 1.9 DataFrames.jl [19], StatsBase.jl [20], HypothesisTesting.jl [21], GLM.jl [22], Plots.jl [23], CurveFit.jl [24] package libraries to generate the descriptive, inferential and visualisations. The JupyterLab Notebook Interface was used for the data analysis, to allow documentation of the data collection and analysis process. During development, the Profiling.jl [28] Julia Package was used to improve performance and reduce bottlenecks. The JupyterLab [26] tool was used for the data analysis to allow documentation of the data collection and analysis process.

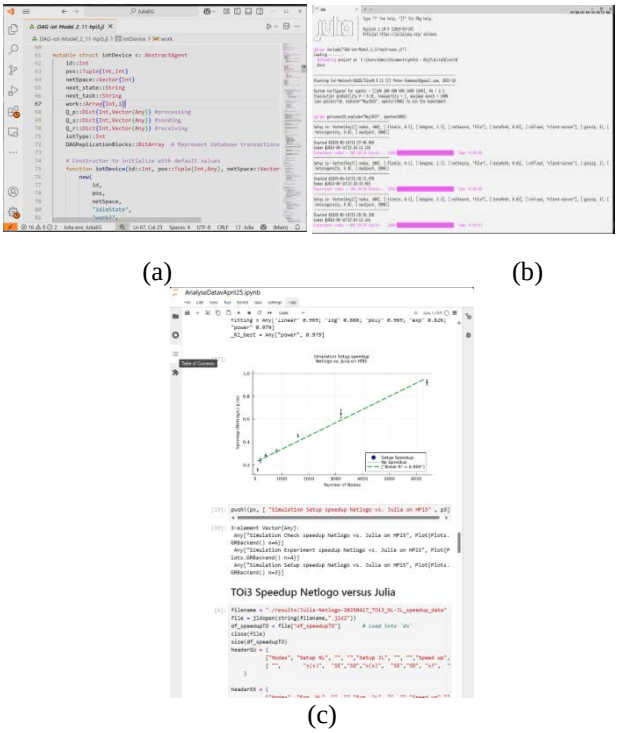


Figure 4. Using Julia in (a) VS Code IDE interface, (b) REPL Command Interface, (c) JupiterLab IDE Interface – Analysis and Documentation

2.4. Data Gathering and Analysis

For each experiment, a CSV file was generated, and the results of each experiment run was appended to the file. The data was extracted from the Global-DAG-DLT database stored in a Julia Dictionary in the Agents.jl Julia 1.9 ABM and from table structure in NetLogo. The data was then analysed using the tools described in section 0 and 0. The results were stored in a results folder and used in the results section of this document.

Descriptive statistics, including mean simulation times and speed-up ratios, were initially calculated to summarise the performance results across varying agent counts and hardware configurations.

To provide a deeper, statistically robust evaluation of the factors affecting simulation performance, inferential statistical analysis was performed using regression modelling. The data collected from the 1680 experimental runs were analysed using the Julia 1.9 ecosystem, leveraging libraries such as DataFrames.jl, StatsBase.jl, HypothesisTesting.jl, and GLM.jl (Generalized Linear Models). Two regression models were constructed: one to predict the "setup_time" and another to predict the "exp_time" (experiment execution time). The predictor variables in these models included the ABM tool (NetLogo or Julia Agents.jl), the machine (TOi3 or HPi5), the nodes (number of agents), and the total number of transactions simulated. The regression analysis allowed for the identification of the statistical significance and magnitude of the effects of factors on the measured time metrics.

3. Results

The data collected was analysed using descriptive, inferential statistics and visualisation. For each of the models implemented using the two ABM tools and on the two hardware configurations, the results were compared for performance speed-up. In computer science, speed-up [29, p. 6] is measured by taking the base speed and dividing it by the speed of the item being measured, as shown in Equation (1). A positive value greater than one (>1) shows an improvement in performance, while a value of less than one shows a negative speed-up (slower). In our case, Speed-up is defined in terms of the work, while the time is kept constant.

$$Speed-up = \frac{Base\ model\ time\ taken}{Improved\ model\ time\ taken} \quad (1)$$

Each model was run 30 times for varying network sizes of N_{total} 100, 200, 400, 800, 1600, 3200 and 6400 IoT agents but with a fixed PA topology of K-degree = 2.3. In this experiment, all other parameters were kept constant i.e. $\left[N_{full}, N_{light}\right] = [1:10], \kappa = 2.3, \lambda = 0.01, \iota = [1,1000], Homogentity = 1$. A total of x30 runs, for x7 Agent configurations, for x2 computing equipment, for x2 languages were formed creating a total of 1680 data samples.

The two models implemented on Julia ABM abbreviated as (JL) and NetLogo ABM abbreviated as (NL) in the table and are run on two different computer platforms TOi3 and HPi5. The Figure 5(a) shows the number of agents for each experiment set and the various combinations. The Figure 5 (b) shows the number of agents, both full nodes and light nodes. The Figure 5(b) shows the transaction arrival rate produced by the agents, the simulated transaction arrival rate per epoch, the replications per epoch that occurs per full agent, the Confirmed Transaction per second (CTPS) and the mean transaction latency (MTL) before 2/3 nodes have received the transactions.

The Figure 6 shows a visual representation of the seven (7) PA topologies used in the experiment to interconnect the agents. To allow for a fair comparison, the same PA configuration was used when performing each experiment with the same number of Agents.

In the following sections, we start by presenting the

results of different ABMs on the same hardware. Finally, we calculate the overall speedup for performing the entire experiment by comparing the speed-up across different hardware and tools.

3.1. Agents.jl ABM Versus NetLogo ABM on the Same Hardware Configuration

This section shows the performance comparison of the two ABMs and the relative speedup using different equipment.

3.1.1. TOi3 Hardware Configuration

The Table 3 shows the setup performance between NetLogo and Julia and the resultant speed-up when running the IoT-DAG-DLTSim on TOi3 configuration. The Table 4 shows the results of executing the experiment simulation and the resulting speed-up when running the IoT-DAG-DLTSim on TOi3 configuration. The Figure 7 plots the visualisation of the speed-up by performing a linear and non-linear regression fit against the results in Table 3 and Table 4.

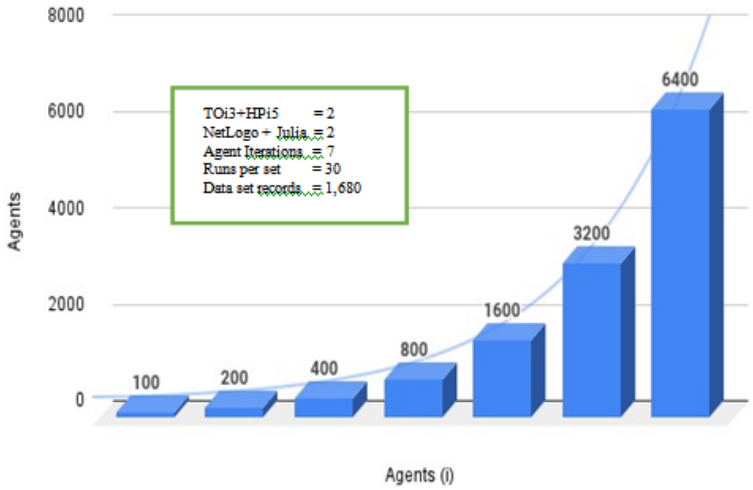
The Table 4 shows the results of executing the experiment simulation and the resulting speed-up when running the IoT-DAG-DLTSim on TOi3 configuration. The Figure 7 plots the visualisation of the speed-up by performing a linear and non-linear regression fit against the results in Table 3 and Table 4.

The Figure 8 shows the calculated estimated linear and non-linear regression fit for the results in Table 3 and Table 4.

The results of Table 3, Table 4and Table 5 are plotted in Figure 7 to visualise the results and the curve fitted to evaluate their trend and predict their behaviour.

3.1.2. HPi5 Hardware Configuration

The Table 6 shows the setup performance between NetLogo and Julia and the resultant speed-up when running the IoT-DAG-DLTSim on HPi5 configuration. The Table 7 shows the results of executing the experiment simulation and the resulting speed-up when running the IoT-DAG-DLTSim on HPi5 configuration. The Table 8 shows the calculated estimated linear and non-linear regression fit for the results in Table 6 and Table 7. The Figure 8 plots the visualisation of the speed-up by performing a linear and non-linear regression fit for the results in Table 6, Table 7 and Table 8.



(a)

			Transactions per Epoch				
Agents	**Full Agents**	**Light Agents**	**Theoretical λ**	**Actual λ**	**Replication rate**	**CTPS (μ)**	**MTL (μ)**
100	10	90	1	0.98	13.04	2.46	12.22
200	20	180	2	1.80	52.89	2.52	15.13
400	40	360	4	3.58	181.87	4.47	16.66
800	80	720	8	7.21	754.84	8.09	19.02
1,600	160	1,440	16	14.64	2,925.20	15.90	20.55
3,200	320	2,880	32	29.11	10,724.59	31.75	21.37
6,400	640	5,760	64	62.62	26,488.68	63.55	22.64

(b)

Figure 5. (a) Agent Scaling used in IoT-DLT performance experiment (b) Sampled Transaction Throughput by IoT-DAG-DLTSim model, with transaction arrival (λ) per epoch, replication and mean(μ) transaction latency(MTL), mean confirmed transaction processing rate (CTPS)

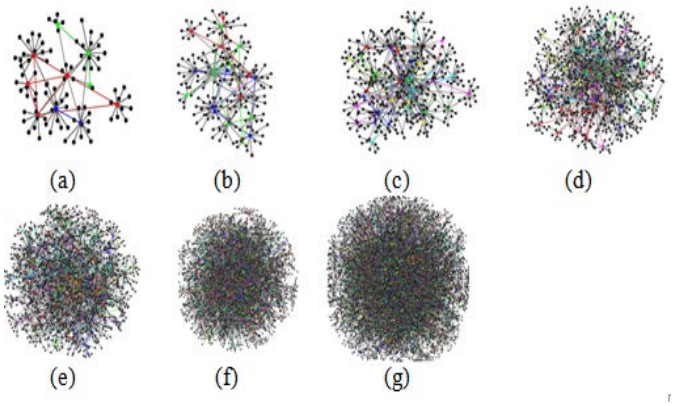


Figure 6. PA topologies k-degree =2.3 showing full and light agents connections, for seven (7) topologies with (a) = 100, (b) = 200, (c) = 400, (d) = 800, (e) = 1600, (f) = 3200, (g) = 6400 agents respectively

Table 3. Setup, Average Execution Time, Standard Error, Standard Deviation, and Speedup for NetLogo and Julia Simulations for Different Node Sizes on TOi3 Hardware

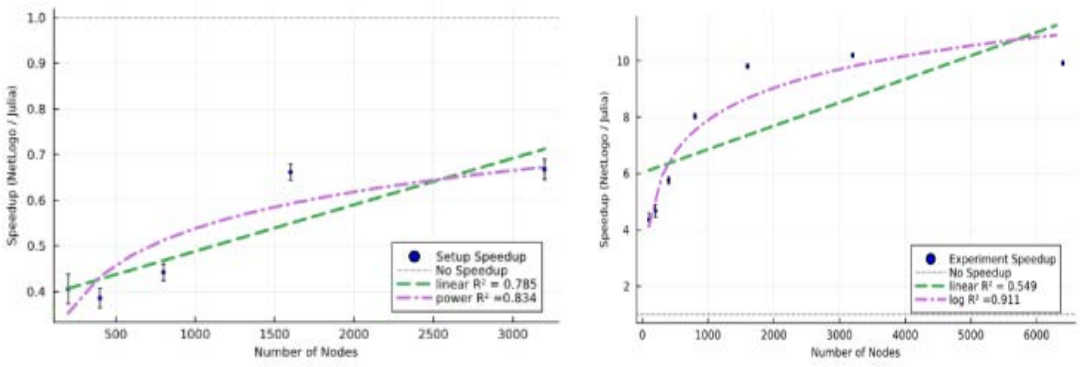
Nodes	Setup NL (seconds)			Setup JL (seconds)			Speed-up		
	μ	±sε	±σ	μ	±sε	±σ	μ	±sε	±σ
100	0.243	0.063	0.333	0.151	0.010	0.055	1.608	0.431	2.283
200	0.083	0.004	0.022	0.204	0.012	0.063	0.406	0.032	0.166
400	0.106	0.004	0.019	0.275	0.012	0.062	0.386	0.022	0.110
800	0.240	0.008	0.043	0.542	0.013	0.072	0.442	0.018	0.098
1600	0.696	0.011	0.055	1.053	0.023	0.119	0.662	0.018	0.091
3200	2.046	0.039	0.205	3.063	0.082	0.451	0.668	0.022	0.119
6400	9.201	0.363	1.986	6.548	0.155	0.835	1.405	0.065	0.352
	12.615			11.836			0.797		

Table 4. Experiment, Average Execution Time, Standard Error, Standard Deviation, and Speedup for NetLogo and Julia Simulations for Different Node Sizes on TOi3 Hardware"

Nodes	Exp. NL(seconds)			Exp. JL(seconds)			Speed-up		
	μ	±sε	±σ	μ	±sε	±σ	ratio	±sε	±σ
100	1.591	0.073	0.386	0.365	0.011	0.056	4.355	0.238	1.253
200	4.164	0.122	0.671	0.890	0.033	0.183	4.677	0.223	1.222
400	11.379	0.086	0.441	1.974	0.039	0.216	5.765	0.123	0.668
800	44.600	0.329	1.742	5.551	0.045	0.245	8.035	0.088	0.474
1600	181.634	1.155	6.112	18.511	0.066	0.335	9.812	0.071	0.375
3200	732.629	3.328	17.924	71.828	0.284	1.532	10.200	0.061	0.331
6400	2946.440	14.138	73.465	296.948	1.348	7.261	9.922	0.066	0.347
	3922.437			396.067			7.538		

Table 5. Linear and Non-Linear Curve Fi for Setup and Run Experiment Speed-up NetLogo versus Julia

Parameter	Linear Fit R^2	Non-Linear R^2
Setup speed-up	R^2 = 0.785	power R^2 = 0.834
Run Experiment Speedup	R^2 = 0.549	log R^2 = 0.911



(a) (b)

Figure 7. Linear and Non-linear fit of Speed-up on NetLogo versus Julia on TOi3 configuration for (a) Setting up the experiment, Running the experiment

Table 6. Setup, Average Execution Time, Standard Error, Standard Deviation, and speed-up for NetLogo and Julia simulations at different Node Sizes on HPi5 hardware

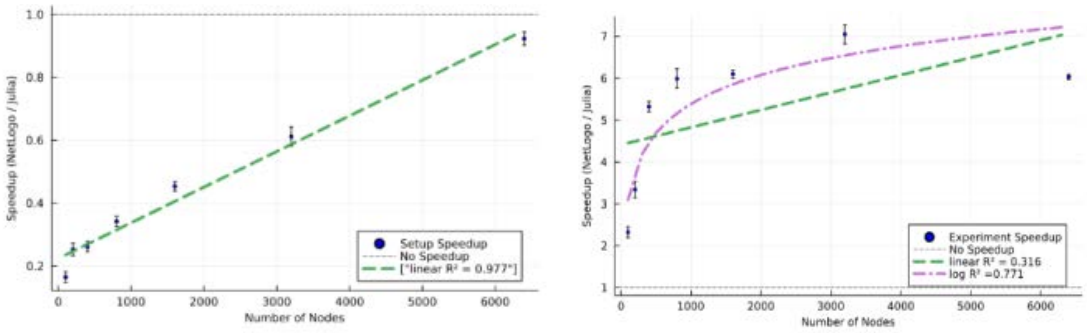
Nodes	**Setup NL (seconds)**			**Setup JL (seconds)**			**Speed-up (×?)**		
	μ	±sε	±σ	μ	±sε	±σ	μ	±sε	±σ
100	0.025	0.002	0.010	0.152	0.011	0.058	0.165	0.017	0.092
200	0.041	0.002	0.011	0.163	0.010	0.053	0.253	0.020	0.107
400	0.068	0.002	0.009	0.260	0.015	0.077	0.262	0.017	0.085
800	0.129	0.005	0.026	0.376	0.012	0.062	0.342	0.017	0.089
1600	0.339	0.010	0.049	0.747	0.010	0.053	0.453	0.015	0.073
3200	1.395	0.061	0.315	2.282	0.059	0.306	0.612	0.031	0.160
6400	4.732	0.103	0.552	5.127	0.034	0.177	0.923	0.021	0.112
	6.729			9.107			0.430		

Table 7. Experiment, Average Execution Time, Standard Error, Standard Deviation, and Speedup for NetLogo and Julia Simulations for Different Node Sizes on HPi5 hardware

Nodes	Exp. NL (s)			Exp. JL (s)			Speed-up (×?)		
	μ	±sε	±σ	μ	±sε	±σ	μ	±sε	±σ
100	0.712	0.024	0.128	0.307	0.014	0.073	2.321	0.130	0.693
200	2.224	0.072	0.395	0.666	0.032	0.176	3.339	0.194	1.064
400	8.066	0.036	0.175	1.516	0.034	0.184	5.322	0.120	0.656
800	24.862	0.900	4.928	4.150	0.060	0.321	5.991	0.233	1.275
1600	83.837	1.338	7.081	13.735	0.065	0.343	6.104	0.102	0.538
3200	370.170	11.421	62.554	52.478	0.457	2.463	7.054	0.226	1.237
6400	1325.714	11.796	63.521	219.790	0.877	4.642	6.032	0.059	0.316
	1814.873			292.642			5.166		

Table 8. Linear and Non-Linear Curve Fi for Setup and Run Experiment Speed-up NetLogo versus Julia

Parameter	**Linear Fit R^2**	**Non-Linear R^2**
Setup speed-up	$R^2 = 0.977$	-
Run Experiment Speedup	$R^2 = 0.316$	$\log R^2 = 0.771$



(a) (b)

Figure 8. Linear and Non-linear fit of Speed-up on NetLogo versus Julia on HPi5 configuration for (a) Setting up the experiment, (b) Running the experiment.

3.2. Temporal Analysis for Full Experiment

The results of running the full experiment on different hardware configurations were collected and the results presented in table and visual formats. The Table 9 shows the results of running the experiment using NetLogo and Julia on Toi3 configuration.

Table 9. Time Taken to Execute 30 runs, Including Setup Time and Experiment Time on TOi3 and Comparing JL and NL

Model	NL TOi3 (seconds)			JL TOi3 (seconds)		
	μ	±sε	±σ	μ	±sε	±σ
Setup	12.614	0.370	2.025	11.836	0.179	0.965
Experiment	3922.433	14.575	75.892	396.066	1.381	7.438
Total	3935.047	13.878	75.919	407.902	1.369	7.500

Table 10. Time Taken to Execute 30 runs, Including Setup Time and Experiment Time on HPi3 and Comparing JL and NL

Model	NL HPi5 (seconds)			JL HPi5 (seconds)		
	μ	±sε	±σ	μ	±sε	±σ
Setup	6.729	0.120	0.638	9.107	0.073	0.379
Experiment	1815.585	16.498	89.569	292.641	0.994	5.283
Total	1822.314	16.353	89.571	301.748	0.967	5.297

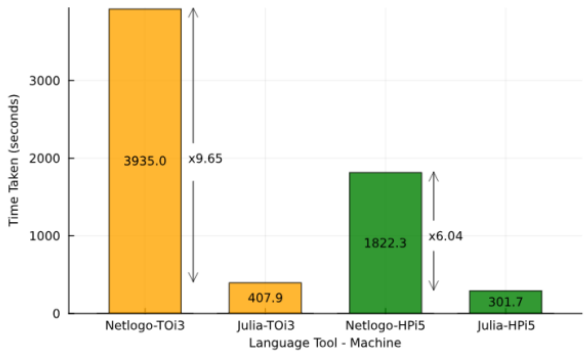


Figure 9. Experiment Run times for Julia versus NetLogo ABM tool on TOi3 and HPi5 equipment mean times taken to run 30 iterations.

The Table 10 shows the results of running the experiment using NetLogo and Julia on HPi5 configuration.

The results of Table 9 and Table 10 are visualised in Figure 9 along with speed-up calculation between the ABM tools.

3.3. Inferential Analysis

In this section, a regression analysis was performed on the setup and the experiment run. To complement the descriptive performance metrics and speed-up analysis, a rigorous inferential analysis was conducted using linear regression models to statistically evaluate the impact of the ABM tool, hardware, node count, and transaction count on simulation setup and execution times. Table 11 presents the regression results for the "setup time" model, while Table 12 presents the regression results for the "exp_time" model.

The interpretation of the regression results of Table 11 is as follows:

- Intercept: The intercept (-0.344 for setup time and -359.092 for execution time) represents the baseline when the tool is Julia and the machine is HPi5, with 0 nodes and 0 transactions. The negative intercepts indicate slightly negative baseline values for both setup time and execution time.
- Tool: NetLogo: For setup time, using NetLogo results in a 0.230 unit increase compared to the reference category, Julia, but this effect is not statistically significant (p-value = 0.527). For execution time, using NetLogo leads to a significant increase (342.122 units) compared to Julia, with a very small p-value (< 1e-32), indicating a robust effect.
- Machine: TOi3: In both models, using TOi3 increases the respective times compared to HPi5. The effect is modest for setup time (0.301 units) and large for execution time (161.161 units), with both coefficients being statistically significant.
- Nodes: For both setup time and execution time, the number of nodes has a negative effect, meaning that as the number of nodes increases, the times tend to decrease. This effect is statistically significant for execution time (p-value < 1e-04), but not for setup time (p-value = 0.103).
- Transactions: Each additional transaction increases setup time slightly (0.002 units), though only

marginally significant for setup time (p-value = 0.0875), and increases execution time significantly (0.434 units, p-value 1e-05).

Table 11. Regression Results for the "Setup_Time" Model, Including the Coefficients for the Intercept, Tool (NetLogo), Machine (TOi3), Number of Nodes, and Number of Transactions. Provided are the Estimated Coefficients, Standard Errors, t-values, p-values, and 95% Confidence Intervals for each Variable in the Model

setup_time ~ 1 + tool + machine + nodes + trans
Coefficients:

Coef.	Std. Error	t	Pr(>|t|)	Lower 95%	Upper 95%
(Intercept)	-0.343776	0.347716	-0.99	0.3231	-1.02628 0.338723
tool: NetLogo	0.23004	0.36343	0.63	0.5269	-0.483304 0.943384
machine: TOi3	0.300765	0.359871	0.84	0.4035	-0.405593 1.00712
nodes	-0.0207369	0.0126959	-1.63	0.1028	-0.0456565 0.00418274
trans	0.00217322	0.00127041	1.71	0.0875	-0.00032035 0.00466678

The interpretation of the regression results of Table 11 is as follows:

- Intercept: The intercept (-0.344 for setup time and -359.092 for execution time) represents the baseline when the tool is Julia and the machine is HPi5, with 0 nodes and 0 transactions. The negative intercepts indicate slightly negative baseline values for both setup time and execution time.
- Tool: NetLogo: For setup time, using NetLogo results in a 0.230 unit increase compared to the reference category, Julia, but this effect is not statistically significant (p-value = 0.527). For execution time, using NetLogo leads to a significant increase (342.122 units) compared to Julia, with a very small p-value (< 1e-32), indicating a robust effect.
- Machine: TOi3: In both models, using TOi3 increases the respective times compared to HPi5. The effect is modest for setup time (0.301 units) and large for execution time (161.161 units), with both coefficients being statistically significant.
- Nodes: For both setup time and execution time, the number of nodes has a negative effect, meaning that as the number of nodes increases, the times tend to decrease. This effect is statistically significant for execution time (p-value < 1e-04), but not for setup time (p-value = 0.103).
- Transactions: Each additional transaction increases setup time slightly (0.002 units), though only marginally significant for setup time (p-value = 0.0875), and increases execution time significantly (0.434 units, p-value 1e-05).

The Table 12 shows the results for the "exp_time" model. Each result includes the estimated coefficients, standard errors, t-values, and p-values for each predictor variable, indicating their statistical significance.

The interpretation of the regression results in Table 12 is as follows:

- Intercept (-359.092): The intercept represents the baseline execution time when the tool is Julia and the machine is HPi5, which are the reference categories. The negative value of the intercept indicates that, for the baseline categories, the execution time is negative (perhaps suggesting a computational anomaly or non-standard baseline, which could warrant further investigation).
- Tool: NetLogo (342.122): This coefficient reflects the change in execution time when using NetLogo instead of Julia. The large positive value (342.122) indicates that using NetLogo results in a significant increase in execution time compared to Julia.
- Machine: TOi3 (161.161): This coefficient shows the change in execution time when using TOi3 instead of HPi5. A positive value of 161.161 indicates that using TOi3 increases the execution time compared to HPi5.
- Nodes (-4.151): This negative coefficient indicates that as the number of nodes increases, the execution time decreases, which is statistically significant (p-value < 0.001).
- Transactions (0.434): For each additional transaction, the execution time increases by 0.434 units. This effect is statistically significant (p-value < 1e-05).

Table 12. Regression results for the "exp_time" Model, Including the Coefficients for the Intercept, Tool (NetLogo), Machine (TOi3), Number of Nodes, and Number of Transactions, Provided are the Estimated Coefficients, Standard Errors, t-values, p-values, and 95% Confidence Intervals for each Variable in the Model

exp_time ~ 1 + tool + machine + nodes + trans
Coefficients:

Coef.	Std. Error	t	Pr(>|t|)	Lower 95%	Upper 95%
(Intercept)	-359.092	26.0882	-13.76	<1e-38	-410.299 -307.886
tool: NetLogo	342.122	27.2672	12.55	<1e-32	288.601 395.642
machine: TOi3	161.161	27.0002	5.97	<1e-08	108.165 214.157
nodes	-4.15117	0.952539	-4.36	<1e-04	-6.02083 -2.28152
trans	0.433977	0.0953152	4.55	<1e-05	0.246891 0.621063

4. Discussion

In this section, the results are discussed, and inferences and implications are drawn from the results produced.

4.1. Speed-up Time for Julia versus NetLogo

The speed-up analysis of Table 5 and Figure 7:for the TOi3 configuration, a linear and non-linear curve fitting was conducted to model the speed-up from NetLogo to Julia. For the setup phase, a power model achieved a better fit ($R^2 = 0.834$) than the linear model ($R^2 = 0.785$), indicating a non-linear relationship. For the experiment execution phase, the non-linear (logarithmic) model had a very strong fit ($R^2 = 0.911$) compared to the linear model ($R^2 = 0.549$). This suggests that while setup performance improvements were moderate, experiment runtime improvements scaled strongly in favour of Julia in a non-linear fashion.

The total execution time for the different ABMs in Table 9, Table 10 and Figure 9, shows a comparison across both machines, TOi3 and HPi5 and confirms Julia's superiority. On TOi3, executing 30 runs required 3935.0 seconds for NetLogo but only 407.9 seconds for Julia — an approximately 9.65× speed-up. On HPi5, NetLogo took 1822.3 seconds compared to Julia's 301.7 seconds, a 6.04× improvement. These results are illustrated in Figure 9, which shows consistent and substantial reductions in total simulation time when using Julia.

The inferential statistical analysis derived from the data shows the two models' setup-time and experiment-time. The setup time regression shows no statistically significant predictors for setup time, although the number of transactions (trans) approached significance ($p \approx 0.0875$), suggesting minimal influence of model complexity on setup duration. The experiment time regression shows highly significant predictors. Notably that:

- The use of NetLogo significantly increased experiment times ($\beta = 342.12$, $p < 0.001$).
- TOi3 machines had longer experiment times compared to HPi5 ($\beta = 161.16$, $p < 0.001$).
- More nodes significantly decreased runtime ($\beta = -4.15$, $p < 0.001$).
- More transactions significantly increased runtime ($\beta = 0.43$, $p < 0.001$).

The regression models statistically validated the observed experimental speed-ups, showing that tool choice (Julia vs NetLogo) and hardware (TOi3 vs HPi5) substantially affect performance outcomes.

4.2. Model Development Effort

The effort to develop the NetLogo 6.3 ABM is significantly easier initially, but requires significantly more effort to optimise performance. NetLogo has an extensive community of users and hundreds of existing model examples. However, NetLogo 6.3 requires third-party tools to perform analysis of the data. For the IoT-DAG-DLTSim model, an extension to handle bitwise array operations was developed, and required significant resources of time and effort. In Julia, the customised "bitarray" structure or any new data structure were easy to develop, and took a minimum amount of time and effort.

The Agents.jl Julia 1.9 ABM tool has a steep learning curve, but with fewer lines of code. The language is more intuitive as you progress, particularly in the use of arrays and data dictionaries. Agents.jl Julia 1.9 ABM did not require third-party tools to analyse the data, as this comes built-in with the Julia 1.9 ecosystem. Performance improvement can be attributed to the LLVM framework.

4.3. Comparative Analysis of ABM

As shown in Table 12, the regression analysis for "exp_time" revealed that both the ABM tool used (NetLogo vs. Julia Agents.jl) and the hardware configuration (TOi3 vs. HPi5) are highly statistically significant predictors of execution time, with p-values less than 1e-08. Specifically, using NetLogo significantly increased execution time compared to Agents.jl (coefficient = 342.122, $p < 1e\text{-}32$). Similarly, using the older TOi3 machine significantly increased execution time compared to the HPi5 (coefficient = 161.161, $p < 1e\text{-}08$). The number of transactions also had a significant positive impact on execution time (coefficient = 0.434, $p < 1e\text{-}05$), as expected. Interestingly, the analysis showed a statistically significant negative relationship between the number of nodes and execution time (coefficient = -4.15117, $p < 1e\text{-}04$), which may warrant further investigation. For "setup time" (Table 11), the effects of tool and transactions were not statistically significant at the conventional 0.05 level, while the machine type and nodes had significant positive effects.

NetLogo 6.3 ABM is a well-proven tool for model development and is suitable for developing proof of concepts and small models. It is suitable for researchers new to the field of model development and simulations, and provides a rich source of support, including books. However, it is limited when it comes to developing your custom extensions and native analysis and reporting, though it has a rich ecosystem of third-party integration to tools such as R, Python and Mathematica. Performance optimisation requires a good understanding of the Java Language and tools, and can prove to be time-consuming and hence expensive. In this research, it has been shown that NetLogo is not suitable for large models with many agents and calculations.

Agents.jl Julia 1.9 ABM is a recent tool with a much smaller user base and a less-known track record. However, it has been specifically designed for performance nearing that of C++, is as easy to use as Python and eliminates the need for a third-party ecosystem for data analysis. Agents.jl Julia 1.9 ABM is well suited for running large models and makes good use of computer hardware resources, resulting in shorter model run times.

5. Conclusion

This study successfully addressed its four research objectives through a systematic comparison of NetLogo 6.3 and Agents.jl in IoT-DLT simulations through temporal performance measurement, Hardware Equipment Performance Analysis and finally, tool Trade-off evaluation.

The research quantified significant execution time differences, with Agents.jl achieving 4× 10× faster simulation runs than NetLogo across varying network sizes. Benchmark data revealed consistent performance advantages for Agents.jl, particularly in large-scale scenarios exceeding 1,000 agents. Runtime comparisons across laptop configurations demonstrated that hardware improvements benefited Agents.jl disproportionately - newer hardware yielded up to 10× speedup versus 4× on older systems. This identifies Agents.jl's superior utilisation of modern multicore architectures. The study systematically contrasted the platforms' strengths: NetLogo's simplified development environment versus Agents.jl's computational efficiency. A decision matrix was developed weighing setup complexity against runtime performance for different research scenarios.

The advantages of modern ABM were clearly demonstrated in the execution phase benchmarks, which conclusively proved Agents.jl's superiority for compute-intensive models, while also identifying NetLogo's continued relevance for prototyping and educational

applications. The study's findings on the performance comparison of NetLogo and Agents.jl were empirically validated and statistically robust due to the use of inferential analysis based on the regression techniques. Therefore, these findings provide researchers with empirically validated guidelines for ABM tool selection, particularly for transaction-intensive IoT-DLT simulations. The results confirm that modern frameworks like Agents.jl can overcome traditional scalability limitations, though future work should explore parallel agent execution and memory optimisation to further enhance performance. This research establishes a foundation for evaluating ABM tools in emerging distributed ledger applications while highlighting pathways for both methodological and technical improvements in agent-based simulation.

Future work should investigate how to leverage Julia's multithreading or other parallel features to allow multiple agents to process transactions concurrently. This could potentially further improve execution times significantly, taking full advantage of multi-core processors, thereby enabling researchers to simulate even larger networks or explore a wider range of model parameters.

ACKNOWLEDGEMENTS

This research has not received any external funding and is fully self-funded.

Statement of Competing Interests

To the best of my knowledge, the authors have no competing interests related to this research.

List of Abbreviations

A-UML	Agent Unified Modelling Language
ABM	Agent-Based Modelling
CTPS	Confirmed Transactions per second
DAG	Direct Acyclic Graph
DB	Database
GPU	Graphical Processing Unit
GUI	Graphical User Interface
HPi5	HP Victusi5 (Hardware configuration B)
IDE	Integrated Development Environment
IoT-DAG-DLTSim	IoT-Direct Acyclic Graph-Based Distributed Ledger Technology Simulation
IoT-DLT	Internet of Things Distributed Ledger Technology
JL	Julia ABM
JVM	Java Virtual Machine
LLVM	Low-Level Virtual Machine
MAS	Multi-Agent Systems
MTL	Mean Transaction Latency
NL	NetLogo ABM
PA	Preferential Attachment
PDF	Portable Document Format
PNG	Portable Network Graphics
REPL	Read, evaluate, print, loop
SIR	Susceptible, Infected, Recovered
SVG	Scalable Vector Graphics
TOi3	Toshiba i3 (Hardware configuration A)

References

[1] Michael Wooldridge., *An Introduction to MultiAgent Systems, 1st Edition | Wiley*, 1st ed. 2002. Accessed: Jan. 17, 2022. [Online]. Available: https://www.wiley.com/en-sg/ An+ Introduction+ to+MultiAgent+Systems%2C+2nd+Edition-p-9780470519462.

[2] J. M. Epstein, 'Agent-based computational models and generative social science', *Complexity*, vol. 4, no. 5, pp. 41–60, May 1999.

[3] K. Batool and M. A. Niazi, 'Modeling the internet of things: a hybrid modeling approach using complex networks and agent-based models', *Complex Adapt. Syst. Model.*, vol. 5, no. 1, p. 4, Mar. 2017.

[4] M. Brumbulli and E. Gaudin, 'Towards Model-Driven Simulation of the Internet of Things', in *Complex Systems Design & Management Asia*, M.-A. Cardin, S. H. Fong, D. Krob, P. C. Lui, and Y. H. Tan, Eds., Cham: Springer International Publishing, 2016, pp. 17–29.

[5] S. Abar, G. K. Theodoropoulos, P. Lemarinier, and G. M. P. O'Hare, 'Agent Based Modelling and Simulation tools: A review of the state-of-art software', *Comput. Sci. Rev.*, vol. 24, pp. 13–33, May 2017.

[6] Cynthia Nikolai and Gregory Madey, 'Tools of the Trade: A Survey of Various Agent Based Modeling Platforms', *J. Artif. Soc. Soc. Simul. Vol 12 No 2 2*, vol. 12, Mar. 2009, [Online]. Available: https://www.jasss.org/12/2/2.html.

[7] Z. Wrona *et al.*, 'Overview of Software Agent Platforms Available in 2023', *Information*, vol. 14, no. 6, p. 348, Jun. 2023.

[8] *Modeling Emergent Phenomena with NetLogo'*. 2001. [Online]. Available: http://ccl.northwestern.edu/papers/MEE/.

[9] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, 'MASON: A Multiagent Simulation Environment', *SIMULATION*, vol. 81, no. 7, pp. 517–527, Jul. 2005.

[10] J. Kazil, D. Masad, and A. Crooks, 'Utilizing Python for Agent-Based Modeling: The Mesa Framework', in *Social, Cultural, and Behavioral Modeling*, R. Thomson, H. Bisgin, C. Dancy, A. Hyder, and M. Hussain, Eds., Cham: Springer International Publishing, 2020, pp. 308–317.

[11] P. Richmond, R. Chisholm, P. Heywood, M. K. Chimeh, and M. Leach, 'FLAME GPU 2: A framework for flexible and performant agent based simulation on GPUs', *Softw. Pract. Exp.*, vol. 53, no. 8, pp. 1659–1680, Aug. 2023.

[12] G. Datseris, A. R. Vahdati, and T. C. DuBois, 'Agents.jl: A performant and feature-full agent based modelling software of minimal code complexity', *SIMULATION*, p. 003754972110688, Jan. 2022.

[13] M. Jelasity, 'Gossip-based Protocols for Large-scale Distributed Systems', Doctorial, University of Szeged, Szeged, 2013.

[14] R. Albert and A.-L. Barabási, 'Statistical mechanics of complex networks'', *Rev Mod Phys*, vol. 74, no. 1, pp. 47-97, Jan. 2002.

[15] T. Alsboui, Y. Qin, R. Hill, and H. Al-Aqrabi, 'Towards a Scalable IOTA Tangle-Based Distributed Intelligence Approach for the Internet of Things', in *Intelligent Computing*, K. Arai, S. Kapoor, and R. Bhatia, Eds., in Advances in Intelligent Systems and Computing, vol. 2. Cham: Springer International Publishing, 2020, pp. 487–501.

[16] M. S. Alaslani, 'Toward Improving the Internet of Things: Quality of Service and Fault Tolerance Perspectives', Doctorial, King Abdullah University of Science and Technology, Thuwal, Kingdom of Saudi Arabia, 2021. [Online]. Available: https://repository.kaust.edu.sa/handle/10754/668723.

[17] R. Albert and A.-L. Barabási, 'Statistical mechanics of complex networks'', *Rev Mod Phys*, vol. 74, no. 1, pp. 47-97, Jan. 2002.

[18] E. Beeker and M. Koehler, 'NetLogo Meets Discrete Event Simulation', in *Proceedings of the 2018 Conference of the Computational Social Science Society of the Americas*, T. Carmichael and Z. Yang, Eds., in Springer Proceedings in Complexity. Cham: Springer International Publishing, 2020, pp. 145–164.

[19] M. Bouchet-Valat and B. Kamiński, 'DataFrames.jl : Flexible and Fast Tabular Data in *Julia*', *J. Stat. Softw.*, vol. 107, no. 4, 2023.

[20] 'Getting Started StatsBase.jl'. Accessed: Jan. 09, 2025. [Online]. Available: https://juliastats.org/StatsBase.jl/stable/.

[21] Simon Kornblith, 'HypothesisTests.jl a Julia package that implements a wide range of hypothesis tests.' Accessed: Apr. 27, 2025. [Online]. Available: https:// juliastats.org/ HypothesisTests.jl/dev/.

[22] Douglas Bates, 'GLM.jl: Linear and generalized linear models in Julia'. Accessed: Apr. 27, 2025. [Online]. Available: https://juliastats.org/GLM.jl/stable/.

[23] Thomas Breloff, 'Plots.jl: powerful convenience for visualization in Julia'. Accessed: Apr. 27, 2025. [Online]. Available: https://docs.juliaplots.org/stable/.

[24] Paulo Jabardo, 'CurveFit.jl:A package that implements a few curve fitting functions.', GitHub. Accessed: Apr. 27, 2025. [Online]. Available: https:// github.com/ pjabardo/ CurveFit.jl/blob/master/README.md.

[25] Jason Bertsche, Seth Tisue, Robert G, and Jeremy B, 'NetLogo Profiler Extension: Improve Code Tune Performance'. Accessed: Apr. 27, 2025. [Online]. Available: https:// ccl.northwestern.edu/ netlogo/4.0/docs/profiler.html.

[26] Cameron Davidson-Pilon; *et al.*, 'Jupyter Notebook Viewer'. Accessed: Jan. 18, 2021. [Online]. Available: https:// nbviewer.jupyter.org/github/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/ blob/ master/ Prologue/ Prologue.ipynb.

[27] S. Danisch and J. Krumbiegel, 'Makie.jl: Flexible high-performance data visualization for Julia', *J. Open Source Softw.*, vol. 6, no. 65, p. 3349, Sep. 2021.

[28] Jeff Bezanson, Stefan Karpinski, and Viral B. Shah, 'Profiling.jl: Performance Profiling to identify performance bottlenecks'. Accessed: Apr. 27, 2025. [Online]. Available: https:// docs.julialang.org/ en/v1/manual/profile/.

[29] M. Madiajagan and S. S. Raj, 'Parallel Computing, Graphics Processing Unit (GPU) and New Hardware for Deep Learning in Computational Intelligence Research', in *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, A. K. Sangaiah, Ed., Academic Press, 2019, pp. 1–15.