applied sciences

MDPI

Review

Experimenting with Agent-Based Model Simulation Tools

Alessia Antelmi [1], Gennaro Cordasco [2], Giuseppe D'Ambrosio [1,*], Daniele De Vinco [1] and Carmine Spagnuolo [1]

[1] Dipartimento di Informatica, Università degli Studi di Salerno, 84084 Fisciano, Italy
[2] Dipartimento di Psicologia, Università degli Studi della Campania "Luigi Vanvitelli", 81100 Caserta, Italy
* Correspondence: gdambrosio@unisa.it

Abstract: Agent-based models (ABMs) are one of the most effective and successful methods for analyzing real-world complex systems by investigating how modeling interactions on the individual level (i.e., micro-level) leads to the understanding of emergent phenomena on the system level (i.e., macro-level). ABMs represent an interdisciplinary approach to examining complex systems, and the heterogeneous background of ABM users demands comprehensive, easy-to-use, and efficient environments to develop ABM simulations. Currently, many tools, frameworks, and libraries exist, each with its characteristics and objectives. This article aims to guide newcomers in the jungle of ABM tools toward choosing the right tool for their skills and needs. This work proposes a thorough overview of open-source general-purpose ABM tools and offers a comparison from a two-fold perspective. We first describe an off-the-shelf evaluation by considering each ABM tool's features, ease of use, and efficiency according to its authors. Then, we provide a hands-on evaluation of some ABM tools by judging the effort required in developing and running four ABM models and the obtained performance.

Keywords: agent-based model; agent-based simulations; agent-based tools; open-source software

check for updates

Citation: Antelmi, A.; Cordasco, G.; D'Ambrosio, G.; De Vinco, D.; Spagnuolo, C. Experimenting with Agent-Based Model Simulation Tools. *Appl. Sci.* **2022**, *13*, 13. https://doi.org/10.3390/app13010013

Academic Editor: Vincent A. Cicirello

Received: 28 October 2022
Revised: 13 December 2022
Accepted: 16 December 2022
Published: 20 December 2022

1. Introduction

Simulation models are one of the most effective and successful methods for studying real-world phenomena [1]. The term model refers to an abstract and simplified representation of the object of study that only considers the aspects relevant to the investigation; the concept of simulation indicates a model's manifestation realized through software for reproducing its dynamics and providing analyzable results. Among simulation models, agent-based models (ABMs) are one of the most adopted techniques for representing reality using a bottom-up approach [2]. Starting from a simple independent entity called an agent, the modeler can shape the global behavior of a complex system. An agent is an autonomous, independent entity that acts within an environment and interacts with it as well as with other agents, according to a set of rules the modeler defines. The combination of these interactions creates a reproduction of the reality under investigation that the modeler can evaluate to extract valuable data and meaningful information from its emergent behaviors. The resulting model will exhibit patterns, structures, and behaviors that were not explicitly programmed but arise through agents' interactions [3]. For these reasons, ABMs turn out to be a valuable tool for studying, explaining, and predicting complex phenomena, supporting researchers in investigating how the macroscopic behavior of a system depends on the micro-level properties, constraints, and rules [3–5].

ABMs are extremely powerful for studying problems centered on an individual's interactions with other individuals or the surrounding environment [4]. This intrinsic characteristic makes ABMs particularly suited to being applied in diverse research fields. Nowadays, ABMs are widely used by researchers in various disciplines, allowing them to test and assess new theories and observe and notice mechanisms never considered before. Applications domains for ABMs include social science [6], economics [7,8], climate change [9,10], epidemiology [11,12], transportation and logistics [13,14], and many others [2,5,15]. The widespread use of ABMs in these various application fields created the need for tools to easily and quickly develop simulations without requiring significant coding skills. Several ABM tools and platforms have been introduced to overcome this requirement by abstracting the complexity of the simulation implementation, allowing modelers to focus on the reality under investigation rather than the coding component.

Existing ABMs reviews. The plethora of existing tools, frameworks, and libraries for developing ABM simulations makes it difficult for modelers to choose the right tool. Over the years, several works analyzed and compared the available ABM software to identify their pros and cons from different perspectives and support modelers towards the right choice for their needs.

The first review about general-purpose ABM software dates back to 2006 when Railsback et al. [16] compared the execution speed and the functionalities offered by some pillar ABM platforms, including MASON, NetLogo, Repast, and Swarm, and identified development priorities for future ABM platforms. In 2010, Allan et al. [15] provided a comprehensive exploration of all the ABM software packages available then, describing the usage of ABMs in different domains and exploring some of the most meaningful works. Five years later, Kravari et al. [17] supplied a comparative review of ABM platforms based on given evaluation criteria to classify ABM software and characterize their ideal usage situation. In 2016, Rousset et al. [18] proposed a detailed description of ABM platforms running simulations on high-performance architectures; specifically, the authors focused on parallel/distributed multi-agent simulation tools and a performance assessment of the HPC-compliant platforms. A year later, Abar et al. [5] presented a concise characterization of all the ABM frameworks and tools existing back then, providing a valuable reference for researchers and developers, and identified future priorities around adopting ABMS platforms. In 2020, Pal et al. [19] recently presented a review of the existing ABM platforms, including active platforms and those no longer under development or unclear status, providing a historical perspective of this domain area. Table 1 compares the existing reviews about ABM tools with our article.

Table 1. Comparison of the existing ABMs reviews. The symbol "-" means that the authors did not explicitly state any constraints on the ABM tools considered in their review.

Reference	Year	Selection Criteria	Number of Tools	Comparison Criteria	Benchmark	Hands-on Experience
Railsback et al. [16]	2006	Publicly released ABM tools	4	Agent-related functionalities (e.g., movement, scheduling), scalability, reproducibility	✓	✓
Allan et al. [15]	2010	Authors' curated list	31			
Kravari et al. [17]	2015	Actively supported ABM tools	24	Security, performance, user support, scalability, license, learning curve		
Rousset et al. [18]	2016	Parallel and distributed ABM computing tools	10	Agent-related functionalities (e.g., movement, scheduling), scalability, reproducibility	✓	✓
Abar et al. [5]	2017	-	85	Programming language, GUI, OS Support, ease of use, scalability, license		
Pal et al. [19]	2020	-	134			
Ours	2022	Open-source actively supported ABM tools, associated with peer-reviewed articles	23	Available features (e.g., visual programming, HPC-related), ease of use, efficiency	✓	✓

Contribution. This paper proposes an overview of the open-source general-purpose ABM tools from the two-fold perspective of their available features and the perceived programming experience. Specifically, we first compare the existing ABM tools based on their characteristics (e.g., programming model, analytical features), ease of use, and efficiency according to their authors and based on the referring peer-reviewed articles, documentation, and websites. Then, we assess the experience of developing and running four ABM models with some available ABM tools. We finally report the obtained performance. Although several works examine existing ABM tools, this article reviews the state of the art from the novel perspective of standing as a guide for newcomers in the world of ABM tools. With this work, we aim to provide the reader with a clear perspective of what each tool can offer, how fast it can be, and the skills required to build and run simulation models. The contributions of our paper can be summarized as follows:

- A discussion about what can be considered an ABM developing tool and which desiderata each tool should meet;
- A description of the existing open-source general-purpose ABM tools and their comparison under the perspectives of their intrinsic characteristics (e.g., programming language, simulation space), functionalities offered to the user (e.g., GUI, analysis), and scaling capabilities (e.g., parallel/distributed execution);
- A comparison of ABM tools based on their efficiency and ease of use according to their authors' statements;
- A hands-on comparison of ABM tools by empirically evaluating their ease of installation and setup, the extensiveness of their documentation, and the effort to implement and run some example models;
- A suite of experiments to analytically evaluate each tool's efficiency and scalability.

Outline. The remainder of this article is organized as follows. Section 2 introduces the main concepts related to agent-based models and discusses what ABM tools are and which desiderata each tool should offer. Section 3 overviews the existing general-purpose, open-source ABM simulation tools. Section 4 compares the collected tools from the two-fold perspective of the available features and the trade-off between ease of use and efficiency. Section 5 describes the developing experience in using each tool and its actual performance. Finally, Section 6 concludes this work.

2. Agent-Based Models and Simulations

In this section, we introduce the reader to the main concepts underpinning this work. First, we characterize ABMs through their key components; then, we focus on ABM tools, overviewing their main features and the desiderata each tool should have.

2.1. What Is an ABM?

Although no single formal definition of ABM exists in literature, we can easily identify some key components that ABMs share: agents, environment, and rules [2]. Agents model the living population, the environment determines the setting where the agents act, and the rules define the potential agent-to-agent and agent-to-environment interactions [20].

The main aspect of an agent is its ability to act autonomously in response to the surrounding environment while making decisions to achieve its internal goals. The modeler must define this decision-making process and compose the agent's behavior, which determines how the agent relates to other agents and environmental factors [21]. The agent's behavior includes simple actions such as moving and communicating, but also more complex operations allowing the population to evolve. Each agent maintains its attributes and the information acquired during the simulation within its state, which may vary during its life cycle. Agents live and act within an environment defining how agents can move and are connected to other agents. An environment has a well-defined topology defined by fields such as spatial grids, continuous spaces, and networks. When the simulation environment must reproduce real-world places, ABMs can exploit Geographic Information System (GIS) data to replicate existing locations like buildings or towns [3,21,22]. All other information related to fields and other non-active objects is included in the environment state. The collection of all agents' and environment's states represents the simulation state, which holds all the information about the model delineating its status at a specific time of the simulation.

The modeler must be able to identify, model, and code agents, environment, and behavioral rules to realize an ABM.

2.2. What Is an ABM Tool?

The effectiveness of ABMs collides with the difficulty of developing a model for researchers with low expertise in computer science. Therefore, the ABM community made a considerable effort to provide standardized software platforms to design, build, and execute ABMs. ABM tools take away many of the complexities of the model implementation, allowing the user to focus on the simulation outcomes rather than the development process [20,23,24]. ABM tools usually come in the form of frameworks and libraries, providing developers with (i) a framework consisting of a set of standard concepts for designing and describing a model and (ii) a library for implementing the framework, containing tools for the execution and the analysis of the simulation [15,16].

Over the years, numerous ABM platforms have been developed with different objectives and targets. The first discriminant factor is identifiable in the platform's purpose. A tool can be either general or special-purpose [17,19]. In the former case, this characteristic denotes the user can use the ABM platform to model any system of interest. In the latter case, the term special purpose implies that the system is oriented to a specific domain, thus including functionalities to address peculiar situations of a given research field. A further differentiation concerns the main development objective of the tool, usually identifiable in better ease of use or improved efficiency. Some ABM platforms emphasize easy-to-use interfaces with a reasonable learning curve that allows non-experienced programmers to produce models quickly [5,18,25]. The drawback of this approach is low scalability since graphical tools and domain-specific programming languages are not focused on performance. Several ABM toolkits for mainstream programming languages also exist. In this case, they offer high-performance capabilities but require technical skills to use them appropriately [20,23].

2.3. What Are the Desiderata of an ABM Tool?

Although there is a lack of standardized criteria to analyze ABM tools, we can enumerate a series of desiderata deriving from existing work. In the following, we discuss the desirable requirements either explicitly mentioned in the literature by previous ABM-related articles or implicitly addressed and implemented by current ABM tools.

Researchers use ABMs to investigate and analyze complex phenomena to understand how each component and the interaction with other elements affect their emerging behavior [5]. Conducting this kind of experiment often demands building elaborated models in terms of the number of agents and the parameters regulating their interactions. The need for implementing and running such models implies the first two fundamentals desiderata of an ABM tool: efficiency and ease of use [5,17,20,26,27]. These two aspects are heavily influenced by the design of the ABM platform and its programming model and are often two conflicting objectives. Some ABM tools expose their functionalities via a GUI (Graphical User Interface) to enhance user experience and grant high ease of use while limiting the achievable model complexity by preventing the user from personalizing and/or adding more complex dynamics. Conversely, frameworks and libraries based on standard programming languages give room for developing complex ABM by offering generic facilities. As a downside, these tools demand adequate technical knowledge, which may result in a higher perceived difficulty [16], usually mitigated via proper documentation, examples, and tutorials [20,25,28].

Developing an ABM means defining different common patterns involving agents' behavior, environment, and interactions. ABM tools should provide ready-to-use methods and interface covering those patterns, allowing the modeler to easily and quickly implement standard actions such as movement and communication, agents' lifecycle and internal state management, environments creation using grids, continuous spaces, and networks, and interaction with the environment [20,29]. Defining a realistic simulation field is a critical feature to accommodate since the simulation environment may provide a rich set of information influencing the agents' behavior. As 80% of the data have a spatial/geographical nature or a geographical component, the ability to work with Geographic Information Systems (GIS) data becomes a fundamental requirement for any ABM tool. GIS-based systems use multiple spatial data models for representing and storing information about phenomena with spatial location and extent [21,22]. In the ABM context, GIS data are particularly crucial, especially for the extensive use of simulation in transportation, urban mobility [30,31], epidemic, and pandemic models [32,33] that incorporate spatial and network topologies to model, for instance, people's realistic activity [34]. Additionally, ABM should transparently handle multiple types of agents and fields within the same simulation without the developer's involvement.

Other desiderata relating to the analysis of simulations include facilities for creating a graphical model visualization, tools for statistical and non-statistical analysis, real-time monitoring, and data visualization [3,5,16,20,23,29]. Among tools for statistical analysis, random number generation assumes tremendous importance since ABMs usually include stochastic processes [3,16,35,36], i.e., processes influenced by a specific random component causing the simulation results to be volatile. Researchers need to handle this stochasticity through random number generators that enable them to reliably reproduce a model's behavior and investigate how random distributions affect it [16,20,37]. The intrinsic stochastic nature of most ABMs also affects the reproducibility of such models. In this context, the automated validation process becomes a critical step in ABM development as simulations must be run several times and their results aggregated [36,38,39].

As the last desideratum, we include model exploration and optimization capabilities since modelers need to explore the model's parameter space experimenting with how the simulation behaves when given parameters vary [20,23,36,38].

3. ABM Tools Overview

The increasing adoption of ABMs in various research fields led to the emergence of numerous platforms to guide and facilitate the design and development of ABMs. In this section, we describe the process we adopted to select the final set of ABM tools to include in our review. A concise description of each tool follows.

3.1. Methodology

In this work, we focus our analysis on open-source general-purpose platforms supported by peer-reviewed academic works that are still actively supported. To compile an initial list of candidates, we gathered data from the bibliographic database Scopus since it represents a comprehensive yet accurate database of peer-reviewed research papers on ABM-related topics. From this source, we collected all articles describing ABM frameworks, platforms, or tools. We then filtered out from this list all articles related to commercial or proprietary platforms or discussing domain-specific ABM tools. We finally included additional ABM frameworks collected from the code hosting service GitHub to consider other free, open-source works described in peer-reviewed articles not indexed in Scopus. In Section 4, we compare the main characteristics of each framework by exploiting the information gathered from each platform's related article(s), website, and documentation.

3.2. ABM Tools Description

ActressMAS [40] is an agent-based framework written in .NET with the primary objective of being simple to learn and easy to use. ActressMAS is designed to allow the user to focus on the model logic rather than learning the framework, enhancing its accessibility at the expense of performance. According to its developers, ActressMAS should be used for applications that do not require fast execution speed or do not include a considerable number of agents.

AgentPy [41] is an open-source Python library for developing and analyzing ABMs integrated with IPython and Jupyter Notebooks, a web-based interactive development environment. AgentPy is designed for scientific applications and provides features for model exploration, numeric experiments, and advanced data analysis. The library offers functionalities to easily create models and their visualization that can be embedded within Jupyter notebooks. Moreover, AgentPy allows the modeler to run simulations in a parallel environment without writing parallel code.

Agents.jl [42] is a recent framework for agent-based simulations for implementing, running, and visualizing models exploiting the Julia programming language. This framework is mainly centered on granting efficiency and ease of use by exposing methods that allow the user to develop models with few lines of code. Agents.jl is available as a Julia library and is easily usable with the plethora of analytical tools of the Julia ecosystem. It offers the most common ABM-related features, including different environments, support for GIS data, and model exploration capabilities. Agents.jl also supports parallel and distributed computing to empower simulation execution.

Care HPS [43] is a C++ tool for modeling and executing ABMs on high-performance architectures while hiding the complexity of parallel and distributed programming. The tool abstracts the modeler from crucial and tricky tasks such as agent distribution, load balancing, and synchronization. Still, Care HPS remains easily extensible by expert developers.

Cormas (Common-Pool Resources and Multi-Agent Systems) [44] is a simulation platform based on the VisualWorks programming environment and the Smalltalk language. This platform is mainly dedicated to non-computer scientists and offers facilities to build, design, and analyze ABMs; however, it exchanges this ease of use with limited efficiency and scalability. Cormas editor allows the user to define agent behaviors through activity diagrams without including sophisticated features to keep its interface as simple as possible.

CppyABM [45] is a library for ABM development that combines the efficiency of C++ with the availability of Python libraries and exploits CMake to be platform-free. CppyABM offers all functionalities in both languages, enabling users to choose their preferred programming language. CppyABM relies on third-party packages to provide additional functionality while remaining a lightweight library. Other Python or C++ libraries can be installed separately and integrated into CppyABM.

EcoLab [46] is a framework for developing ABMs in C++ and executing them using TCL (Tool Command Language). This tool provides a GUI through the Tk toolkit and supports parallel and distributed processing by exposing utilities to manage communication. The user has to handle synchronization and partitioning manually.

Evoplex [47] is a platform for developing ABM based on C++, using CMake scripts to facilitate compilation and setup, thus making it cross-platform. Evoplex adopts a fully modular approach that separates the core library from the GUI and visualization tools. The APIs exposed by the core library allow the user to develop the model. At the same time, additional components are available to improve ease of use with an interactive GUI and a web visualization tool.

FLAME (FLexible Agent Modeling Environment) [48] is an agent-based modeling system for creating models runnable on most computing systems, ranging from laptops to HPC supercomputers. FLAME provides a formal framework for creating models based on the XXML language, a dialect of XML, that it uses to generate the source code for the simulation in C automatically. The FLAME engine automatically generates parallel code without any effort by the modeler by adopting a new programming language easy to understand.

FLAME GPU [49] is an extended version of the FLAME framework to write ABMs for Graphics Processing Units (GPUs) using the FLAME standard formal XXML language.

Thanks to FLAME GPU, the user does not need to explicitly understand GPU programming languages or optimization strategies since the API available uses the FLAME template to generate the simulation program in CUDA for target GPU devices. Visualization of the simulation is available even for a massive agent population without suffering performance loss.

GAMA (Gis & Agent-based Modelling Architecture) [22] is an agent-oriented generic modeling and simulation platform. GAMA grants high ease of use by providing a simple agent-based programming language called GAML that offers a simple formalism to describe all the characteristics of the entities of an ABM. Moreover, the platform can manage simulations with hundreds of thousands of agents with good performance. The modular architecture, separating each aspect of the model into a specific component, and the facilities provided makes GAMA an accessible tool for non-expert developers with minimum learning requirements. This characteristic is enhanced by the full integration of GAMA with the Eclipse IDE, which provides convenient features such as auto-compilation, auto-completion, and the use of templates. Finally, the platform supports the integration of external modules to introduce additional functionalities, such as GAMAR [50], that enable the analysis of simulation results with R.

Insight Maker [51] is a graphical modeling and simulation tool focused on accessibility and availability of features rather than performance. This tool is a web application accessible through a standard web browser and includes features specific to a web environment, like user management and model searching and sharing. The main advantage of Insight Maker resides in its VPL (Visual Programming Language), which offers access to all its functionalities, including tools for data analysis and model exploration and validation. The simulator also includes API to build models and analyze their results programmatically.

JADE (Java Agent Development Framework) [52] is an industry-driven Java FIPA-compliant framework aiming to simplify the implementation of multi-agent systems. Thanks to its features, this tool has established itself as one of the most popular platforms in academic and industrial communities [17]. JADE provides a powerful and useful GUI enabling the user to control and configure the simulation during its execution and also supports debugging and development tasks. Moreover, this tool is designed to work on distributed systems abstracting most of the inherent complexities to the modeler. The core characteristics of JADE make the tool highly scalable, robust, easy to learn, and compatible with most Java-based platforms. Moreover, its popularity grants high user support, with complete documentation and many tutorials and examples available.

JAS-mine (Java Agent-based Simulation library—Modelling In a Networked Environment) [53] is a Java-based toolkit for discrete-event simulations designed to aid ABM development. Specifically, this platform aims to speed up model development, facilitate model documentation, and foster model testing and sharing. The core capabilities of JAS-mine reside in integrating I/O communication functionalities in the form of embedded relational database management systems tools and automatic CSV table creation. The database explorer included in the platform enables the user to inspect the database through Structured Query Language (SQL) style commands.

krABMaga [54] is a fast, reliable, discrete-event multi-agent simulation toolkit based on the Rust language for developing ABMs. Designed to be a ready-to-use tool for the ABM community, krABMaga embraces the architectural concepts of the well-adopted MASON simulation library to provide modelers with a familiar programming environment and decrease the learning curve of the framework. However, krABMaga re-engineered some aspects of the MASON architecture to exploit Rust's peculiarities and programming model. This framework comprises all functionalities required for developing and executing a model, including a visualization component and a convenient UI. Additional functionalities relate to running model exploration jobs on parallel, distributed, and cloud architectures.

MaDKit (Multi-agent Development Kit) [55] is a lightweight Java library for designing and simulating agent systems. The tool follows an organization-centered rather than an agent-centered approach based on the AGR (Agent/Group/Role) model. MaDKit provides several functionalities via APIs, including agents' lifecycle management and distribution, being mainly designed to be used by users with some programming knowledge.

MASON [56] is a discrete-event simulation toolkit written in Java for designing, executing, and visualizing ABMs. MASON provides functionalities and API supporting the most common needs of a modeler, including common agents' behavior, environment creation, and scheduling management. One of the main advantages of MASON is its snapshot system enabling the user to stop and save a simulation and resume it in another machine thanks to the compatibility provided by the Java Virtual Machine. Moreover, thanks to the existing extensions, additional features are available, including GIS data with GeoMASON [57], model exploration with ECJ [58], or the possibility of executing a simulation on distributed systems and Cloud Computing with DistributedMASON [59]. Further, MASON is well-suited to computationally intensive models or long-running simulations.

MASS (Multi-Agent Spatial Simulation) [60] is a multi-agent and spatial simulation library designed to address the need for parallel ABMs. The architecture is based on the coordinator–worker approach, where the coordinator process spawns workers at different computing nodes to run parallel simulations. MASS automatically manages agent execution and migration as well as the simulation space through several APIs, which facilitate the model development (if the user has some basic knowledge of Java).

Mesa [61] is a Python-based ABM framework providing built-in core components to easily create, visualize, and analyze simulations. Mesa is one of the most used and actively supported ABM libraries, which exploits Python's popularity to provide ease of use and accessibility. One of the main advantages of Mesa is its extensibility allowing users to develop and share their components through an open-source ecosystem. This approach created a rich community providing extensions for any need, including the possibility to exploit a multi-processor system, support for GIS data, and advanced analysis.

NetLogo [62] is an agent-based modeling environment implemented in Java and Scala, and it is considered the standard platform for developing ABMs. The importance and popularity of NetLogo rose to prominence thanks to its community, which is continuously providing extensions such as GIS data usage, 3D visualization, and integration with other languages, such as Python with PyNetLogo [63] or Pylogo [64], or R with RNetLogo [65]. Other relevant extensions worth to be mentioned are HubNet [66] for creating participatory simulations and BehaviorSpace [67] for providing parameter-sweeping capabilities using distributed and parallel techniques. NetLogo allows modelers to develop their models through a simple-to-use dedicated modeling language while offering a VPL to create and edit components to realize any simulation. However, its accessibility leads to significant limitations regarding model complexity.

Pandora [68] is an ABM framework for large-scale distributed simulation providing two identical programming interfaces exposing the same functionalities in two different programming languages. pyPandora allows non-expert developers to develop models using Python quickly. C++ Pandora offers a more efficient interface in C++ to implement complex models, including the automatic generation of parallel and distributed code. Pandora includes Cassandra, a GUI tool with functionalities to design and analyze a single model execution or to set up a model exploration process. This tool can run large-scale ABMs, and deal with thousands of agents with complex behavior.

Repast (REcursive Porous Agent Simulation Toolkit) [69] is a family of agent-based modeling and simulation platforms available in several programming languages. Repast Simphony [70] is a Java-based modeling system that provides automated methods to perform all the common tasks required in a simulation and supports several crucial additional functionalities. The Simphony platform is based on a modular architecture adopting a plugin system that enables adding a wide range of external tools. Any other Repast version implements the core features of Repast Simphony. Repast4Py [71] is a Python-based framework that includes functionalities to develop distributed ABMs.

RepastHPC (Repast for High-Performance Computing) [72] is another member of the Repast suite; specifically, it is a C++-based modeling system designed for running on large computing clusters and supercomputers. This toolkit enables the execution of massive simulations containing hundreds of thousands of agents of very complex behavior whose execution requires high computational power. Although some built-in functions are available for developing a model, RepastHPC still requires users to have good programming experience since they have to manage different aspects of the parallel execution.

4. ABM Tools Comparison

The plethora of ABMS tools available in the literature can easily overwhelm any scientists interested in adopting a platform to develop ABMs with less effort. In this section, we compare the leading open-source general-purpose platforms (see Section 3) from the two-fold perspective of the available features and the trade-off between their declared ease of use and efficiency. It is worth noting that (as previously mentioned) we impartially gathered all information from each platform's related article(s), website, and documentation.

4.1. Available Features

Every ABM tool should offer some fundamental features to assist the user in creating their model [16]. Table 2 lists these properties, which are based on the desiderata discussed in Section 2.3 and includes facilities to write, run, analyze, and optimize ABMs. Tables 3 and 4 compare the tools according to whether they support each desirable feature.

Table 2. Description of the desirable features for ABM platforms.

Feature	Description
Programming Language	Programming language adopted for model development.
GUI	Availability of a Graphical User Interface (GUI) to control the simulation execution.
Visual programming	Use of a Visual Programming Language (VSL) to support the model development phase.
Simulation environment	Fields and topologies available to create the simulation environment, including GIS data.
Visualization	Possibility to graphically visualize the model.
Snapshot and checkpoint	Presence of functionalities to pause and save the simulation state and resume it later.
Modularity and reusability *	Adoption of a modular design supporting the reuse of portions of simulation code that can be combined to realize other simulations.
Inspector	Facilities for retrieving the model's information during its execution.
Analysis tools *	Presence of tools for statistical and non-statistical analysis, creation of charts and plots.
Random number generator *	Support for the custom generation of random numbers.
Batch runner *	Facilities to automatically perform multiple runs of the same simulation to assess whether stochastic processes influence the model's output.
Continuous Integration (CI)/ Continuous Development (CD)	Facilities to let developers automate code continuous integration and delivery while keeping track of the simulation versions.
Model exploration and optimization * (MEO)	Facilities for exploring the model's behavior when its input parameters vary and optimizing its outcomes. This feature also includes automated testing.
HPC	Exploitation of parallel and distributed computing as well as integration with cloud platforms for in-model execution or optimization.

* We only report whether the given feature is available without considering its comprehensiveness.

We do not explicitly report the feature "Modularity and reusability" as a column since all frameworks support this property to some extent, primarily thanks to their model development language. In particular, most of the tools exploit an object-oriented programming (OOP) approach, natively benefiting from the concepts of inheritance and polymorphism. Further, OO programming languages, such as Java and Scala, represent the building blocks for domain-specific languages (like NetLogo and GAML), which directly inherit their OO characteristics. In this context, it is worth mentioning how GAMA is the first (and unique) tool introducing the concept of modularity by design thanks to the co-modeling mechanism [22], which enables users to incorporate previously defined agents or models inside new models to create more complicated simulations.

Table 3. Comparison of ABM tools based on their features.

↓ Tool/Feature →	Programming Language	GUI	VSL	Simulation Environment	Visualization	Snapshot and Checkpoint	Code
ActressMAS	C#	✓		Generic			[73]
AgentPy	Python	✓		Grid, continuous, network	2D		[74]
Agents.jl	Julia	✓		Grid, continuous, network, GIS	2D, 3D		[75]
Care HPS	C++						[43]
Cormas	VisualWorks	✓	✓	Generic	2D	✓	[76]
CppyABM	C++ and Python			Generic	2D, 3D		[77]
EcoLab	C++ and TCL	✓		Grid, continuous, network	2D	✓	[78]
EvoPlex	C++	✓		Grid, continuous, network	2D		[79]
FLAME	C++ and XXML			Generic			[80]
FLAME GPU	C++ and Python			Generic	3D		[81]
GAMA	GAML	✓	✓	Grid, continuous, network, GIS	2D, 3D	✓	[82]
Insight Maker	JavaScript	✓	✓	Grid, network			[83]
Jade	Java	✓		Generic		✓	[84]
JAS-mine	Java	✓		Grid			[85]
krABMaga	Rust	✓		Grid, continuous, network	2D		[86]
MADKIT	Java	✓				✓	[87]
MASON	Java	✓		Grid, continuous, network, GIS with GeoMASON [57]	2D, 3D	✓	[88]
MASS	Three versions available: Java, C++, CUDA	✓	✓	Grid, network	2D		[89]
Mesa	Python	✓		Grid, continuous, network, GIS with Mesa-Geo [90]	2D, 3D with Mesa 3D [91]		[92]
NetLogo	NetLogo, Python [93,94] and R [95] with extension	✓	✓	Grid, continuous, network, GIS with extension [96]	2D, 3D with extension [97]	✓	[98]
Pandora	C++ and Python	✓		Generic, GIS		✓	[99]
Repast	Three version available: C++, Java, Python	✓		Grid, continuous, network, GIS	2D, 3D	✓	[100]
RepastHPC	XXML			Generic	3D	✓	[101]

Table 4. Comparison of ABM tools based on their features.

↓ Tool/Feature →	Inspector	Analysis Tools	Random Number Generator	Batch Runner	CI/CD	MEO	HPC	
							In-Model	MEO
ActressMAS	✓	✓	✓				Parallel, distributed	
AgentPy		✓	✓	✓	With extension [102]	✓		Parallel
Agents.jl	✓	✓	✓	✓	With extension [103]	✓		Distributed via the `Distributed` module [104]
Care HPS			✓				Parallel, distributed	

Table 4. *Cont.*

↓ Tool/Feature →	Inspector	Analysis Tools	Random Number Generator	Batch Runner	CI/CD	MEO	HPC	
							In-Model	MEO
Cormas	✓			✓	With extension [105]	✓		
CppyABM		✓	✓	✓			Parallel	
EcoLab	✓		✓	✓	With extension [106]	✓	Parallel, distributed	
EvoPlex	✓	✓	✓	✓	With extension [107]		Parallel	
FLAME		✓	✓				Parallel, distributed	
FLAME GPU			✓	✓	With extension [108]		Parallel, distributed	
GAMA	✓	With gamar [50]	✓	✓	✓	✓	Parallel	Parallel
Insight Maker	✓	✓	✓	✓		✓		
Jade	✓	✓			✓[109]		Parallel, distributed	
JAS-mine	✓	✓	✓	✓		✓		
krABMaga		✓	✓	✓	With extension [110]	✓		Parallel, distributed, cloud
MADKIT	✓			✓	With extension [111]		Distributed	
MASON	✓	✓	✓	✓		With ECJ [112]	Distributed With Distributed MASON [113]	Parallel, distributed
MASS	✓	✓	✓	✓	With extension [114]		Parallel, distributed, cloud	
Mesa		✓	✓	✓	With extension [115]	✓		Parallel
NetLogo	✓	With extension [116]	With BehaviorSpace [117]	With extension [118]	With extension [119]	With BehaviorSpace [117]		Parallel
Pandora	✓	✓	✓	✓		✓	Parallel, distributed	
Repast	✓	✓	✓	✓		With EMEWS [120]	Parallel, distributed	Distributed
RepastHPC		✓	✓	✓			Parallel	

4.2. Declared Ease of Use vs. Efficiency

Ease of use and efficiency are two essential and often conflicting criteria commonly employed to assess software platforms, and ABM tools are no exception [51]. In this work, the term "ease of use" refers to the effort required for installation and setup procedures, the presence of examples, and the clarity of the documentation provided. The term "efficiency" refers to the capability of the ABM tool to handle large and complex models granting low execution time.

We evaluated the ease of use of each ABM platform by adopting a five-level Likert scale. The scale is based on the criteria included in the work of Macal et al. [2], and it considers the programming model adopted by each ABM tool and the (number of) available functionalities as reported in the corresponding documentation, websites, and article (declared ease of use). A description of the scale follows.

- *Very low*: poor modeling/execution APIs are provided by the tool, and the developer is also responsible for many execution/technology-related tasks.
- *Low*: good modeling, but poor execution APIs are provided by the tool.
- *Medium*: comprehensive modeling/execution APIs.
- *High*: improves the previous level by adopting a well-known programming language, such as Python or Java, which widens the number of developers ready to use the tool.
- *Very high*: tools that offer the previous level by using a Domain-specific Language or a Visual Programming Language specifically developed for ABM.

Broadly stated, the ease of use of a tool strictly depends on the programming language used (higher-level languages are generally easier to approach), the number of available functionalities (the higher, the better), and the availability of a dedicated GUI or a VSL (which can further reduce the need for coding).

The classification of ABM tools according to their efficiency is based on how the authors position themselves within the state-of-the-art regarding the potential to handle large-scale models and the efficiency in executing them. In this case, we used a five-level Likert scale, ranging from *Very low* to *Very high*, to evaluate the efficiency of each tool as a function of its underlying technology and HPC capabilities. Table 5 classifies all ABM tools based on their ease of use and efficiency.

Table 5. Trade-off between the declared ease of use and efficiency of the ABM platforms.

		DECLARED EASE OF USE				
		Very low	**Low**	**Medium**	**High**	**Very high**
DECLARED EFFICIENCY	**Very low**				Cormas Mesa	Insight Maker NetLogo GAMA
	Low				AgentPy	
	Medium			FLAME MADKIT Repast	ActressMAS Evoplex	
	High		EcoLab	Agents.jl JADE krABMaga MASON MASS Pandora	CppyABM JAS-mine	
	Very high	Care HPS FLAMEGPU	RepastHPC			

5. ABM Tools Evaluation

Part of the remarkable popularity of ABM tools comes from the improved developer experience that allows non-expert programmers and lay users to design and (possibly efficiently) run a simulation model. In this section, we analyze the collected ABM tools from the developer's perspective, assessing each tool's experienced support and performance.

5.1. Hands-on Developer Experience

A critical quality of each ABM tool is the perceived hands-on experience. To evaluate this characteristic, we considered the following elements.

Installation and setup. Perceived difficulty in installing and configuring the tool.

- *N/A*: installation guide not available.
- *Easy*: an installation script or installer is provided, or the tool can be used as a standard library.
- *Hard*: information about the installation procedure is vague or it requires technical expertise.

Documentation and examples. Extensiveness of the documentation and presence of examples and tutorials.

- *N/A*: no documentation and/or tutorials are available.
- *Basic*: most functionalities are not documented or only essential information is provided; few or no examples are given.
- *Good*: all functionalities are documented; some examples are given.
- *Extensive*: all functionalities are extensively documented; several comprehensive examples exist.

Effort. Effort required to implement a model, measured in hours.

Problems. Errors and issues that prevented the tool's installation or usage.

Hands-on evaluation process. The hands-on programming experience was evaluated by two developers with a strong background in computer science, relevant expertise in the ABM field, and some experience with most of the ABM tools considered. Both developers independently installed each ABM tool reviewed in this work and tried to solve any arisen problems before moving on to the model development phase. At the end of this process, both developers rated each tool according to (i) its installation and setup procedures, (ii) the comprehensiveness of its documentation and examples, (iii) the effort required to develop a single model, and (iv) documented whether they encountered any issues in the process (as detailed above). Finally, both developers filled in Table 6, discussing any discrepancy until agreement. The upper part of the Table includes all tools correctly installed and used; the lower part lists the remaining tools, clarifying the problems encountered in the installation phase or when running examples. The number of hours describing the effort required refers to the average time needed to develop a single model since each has different peculiarities. In the case of ready-to-use ABM models (e.g., Flockers in NetLogo), the developers estimated the effort required to adapt them to the experiment's requirements (see Section 5.2).

Table 6. Evaluation of ABM tools based on the perceived easiness in installing and using each tool. N/V stands for Not Verifiable due to the inability to install or use the corresponding tool.

Tool	Installation and Setup	Documentation and Examples	Effort	Problems
ActressMAS	N/A	Basic	∼4 h	No installation information provided; only works with VisualStudio.
AgentPy	Easy	Good	∼2 h	
Agents.jl	Easy	Extensive	∼2 h	
CppyABM	Easy	Basic	∼4 h	
GAMA	Easy	Extensive	∼2 h	
krABMaga	Easy	Good	∼3 h	
MASON	Easy	Extensive	∼3 h	
Mesa	Easy	Extensive	∼2 h	
NetLogo	Easy	Extensive	∼3 h without VSL	
Repast	Easy	Good	∼4 h	

Table 6. *Cont.*

Tool	Installation and Setup	Documentation and Examples	Effort	Problems
Care HPS	N/A	N/A	N/V	Only documented within the article.
Cormas	Easy	N/A	N/V	The application crashes after few operations.
EcoLab	Easy	Basic	N/V	The installation script does not work.
EvoPlex	Easy	Basic	N/V	All the examples do not run; no usage information provided.
FLAME	Easy	Basic	N/V	The installation script requires fixes to work properly; no more supported.
FLAME GPU	Easy	Good	N/V	Not included because of the GPG involvement.
Insight Maker	Easy	Extensive	N/V	The web application cannot handle huge workloads.
Jade	Easy	Extensive	N/V	Not included due to the different programming model
JAS-mine	Easy	Basic	N/V	Installation fails due to configuration errors.
MADKIT	Easy	Basic	N/V	Installation fails due to the JAR file.
MASS	Easy	Basic	N/V	The building process fails.
Pandora	Easy	Basic	N/V	No more supported.
RepastHPC	Hard	Good	N/V	The installation script provided does not work.

5.2. Performance Evaluation

To evaluate the efficiency and scalability of each ABM tool, we exploited four ABM models with different characteristics regarding data structures, agents' behaviors, and environment type. A brief description of each model follows:

- *Flockers*. Developed by Craig Reynolds, this is one of the most famous ABM simulating a flock's flying behavior. In this model, the agents move within a continuous toroidal space according to a simple set of rules.
- *Schelling*. This is a simple segregation model based on a 2D grid in which agents decide whether to move into a new cell based on the status of their neighbors.
- *Wolf, Sheep, and Grass*. This multi-agent model simulates the population dynamics of predators and prey coexisting in a shared environment.
- *ForestFire*. This stochastic spreading model is realized as a cellular automaton to reproduce the fire diffusion in a forest.

To perform a fair comparison, we conceived a meta-model for each of the above models based on their NetLogo implementation, describing the agents' behavior, the simulation environment, and the possible interactions happening among agents and between the agents and the environment. We used these meta-models to ensure that a specific model simulation would expose the same behavior in all tested platforms. Specifically, in the case we had to implement a given model for a specific platform from scratch, we followed the tool's suggested modeling choices and the guidelines imposed by the meta-model to guarantee the expected behavior. Conversely, if the model was already available from the platform's repository, we verified its behavior and applied small changes in its implementation to match the meta-model if needed. Still, despite our effort to reproduce the same model for each ABM tool, some differences could exist due to the variance in the frameworks, mainly because of the different architectures and programming languages involved. It is important to notice that in all the evaluated platforms, it is possible to reuse code portions, such as agent definitions and behavior, in other models (see Section 4.1). For instance, by incrementally adapting its behavior, we included the Flocker agent model in the Wolf, Sheep, and Grass simulation. Specifically, we expanded a Flocker agent's basic movement behavior within a two-dimensional environment to a Wolf agent's more complex behavior, which also includes eating and reproducing.

Table 7 reports whether a given model was already available as an example on each functioning platform (see Section 5). The implementations of all models and the benchmark are freely available on the following GitHub repository: https://github.com/isislab-unisa/ABM_Comparison (accessed on 15 December 2022).

Table 7. Summary of whether each model was available for a given platform (●) or has been developed from scratch (○). We only considered the ABM tools that could be installed and used.

↓ Tool/Model →	Flockers	Schelling	Wolf, Sheep, and Grass	ForestFire
ActressMAS	○	●	○	○
AgentPy	●	○	●	●
Agents.jl	●	●	●	●
CppyABM	○	●	○	○
GAMA	●	○	●	○
krABMaga	●	●	●	●
MASON	●	●	○	○
Mesa	●	●	●	●
NetLogo	●	●	●	●
Repast	●	●	●	○

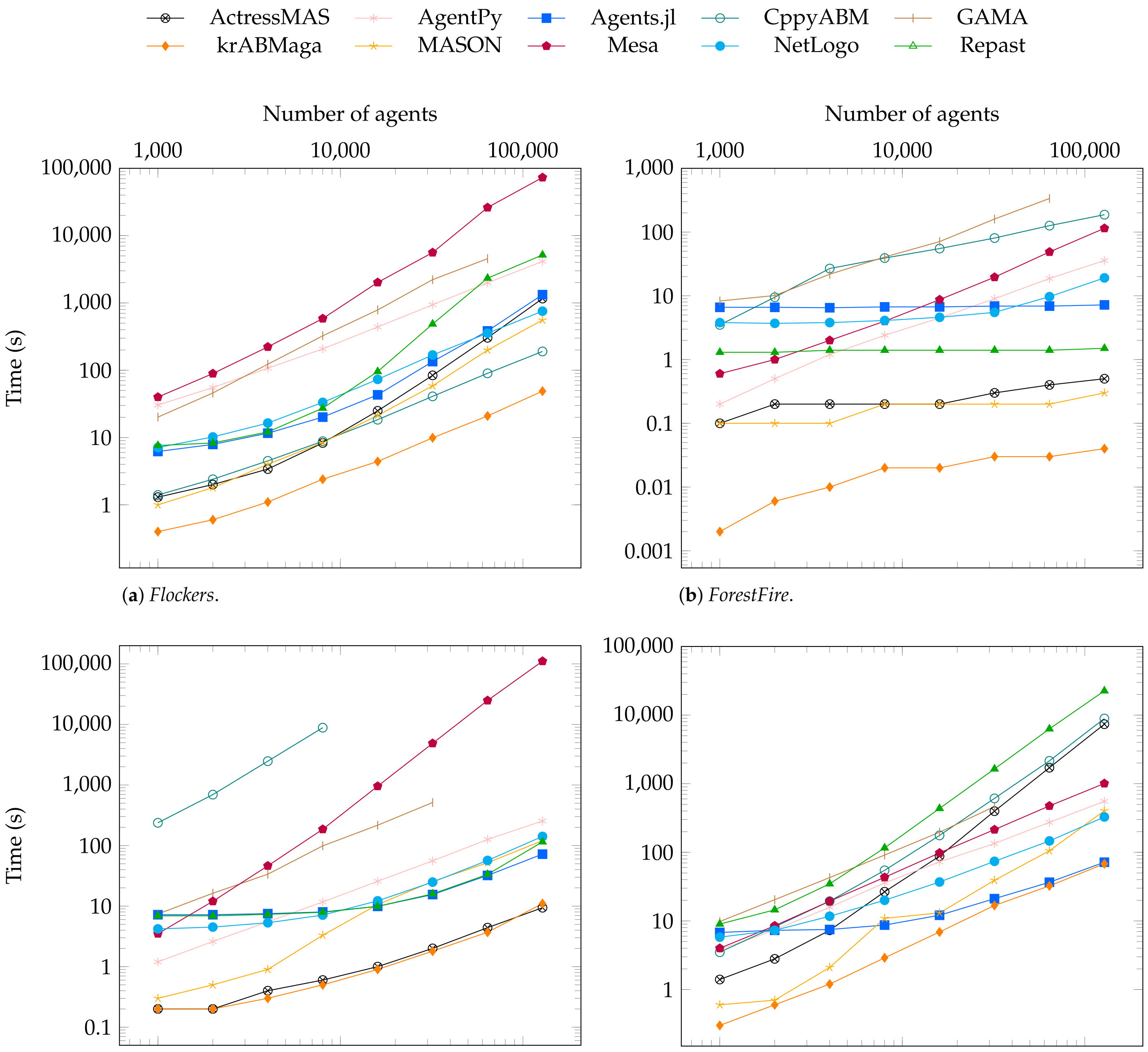
Benchmark configurations. All experiments were performed on the same Ubuntu 22.04 LTS x86_64 machine with kernel version 5.15.0-48-generic and equipped with an Intel i7-8700T (12) @ 4.000 GHz CPU, an NVIDIA GeForce GTX 1050 Mobile GPU, and 16GB RAM. The performance of each framework was tested with different models configurations, starting with a field of size 100 × 100, 1000 agents, and 200 steps, while maintaining an agent density of $\cong$ 10%, calculated as $\frac{\text{width}\times\text{height}}{\text{number of agents}}$. We obtained the other configurations by doubling the number of agents and changing the field dimension to preserve the agent density. Table 8 lists all experiment configurations.

Table 8. Experiment configurations for evaluating ABM tools' performance.

#	Agents	Field Size	#	Agents	Field Size
1	1000	100 × 100	5	16,000	400 × 400
2	2000	141 × 141	6	32,000	565 × 565
3	4000	200 × 200	7	64,000	800 × 800
4	8000	282 × 282	8	128,000	1131 × 1131

Results. Figure 1 depicts the running times of all the benchmarked ABM tools varying the computational load over the four models. The missing data in the graphs means that the specific tool failed in performing the simulation with the given configuration.

Generally, most tools achieve the same performance stated in the referring articles. Nevertheless, we can note a few exceptions. Better-than-expected results come from ActressMAS, reaching execution times comparable with tools specifically designed for obtaining high performance. This outcome probably derives from the efficiency of C# in managing the simple data structures required to manage the simulation. The platform only struggles with the WSG model, which includes more complex data structures and requires multi-agent management. Likewise, NetLogo performs better than expected in all models, providing good efficiency and scalability. Despite not being designed for simulating complex models, NetLogo grants medium to low execution times and handles intensive workloads. Conversely, CppyABM struggled with huge workloads, incurring execution failures or delivering high execution times. This behavior led to higher computational times than awaited in all models but Flockers. The reasons for these results must be further investigated and may also be influenced by our inexperience with the platform and the need for more documentation and examples.



(**a**) *Flockers*. (**b**) *ForestFire*. (**c**) *Schelling*. (**d**) *Wolf, Sheep and Grass*.

Figure 1. Running times of each framework on the four ABMs, varying the computational load by increasing the number of agents while preserving the agent density.

6. Conclusions

ABMs are an effective technique for studying complex systems via a bottom-up approach as, through ABM simulations, researchers from different fields can investigate phenomena that are too difficult to understand using traditional methods. ABMs represent an interdisciplinary approach to examining complex systems, and the heterogeneous background of ABM users demands comprehensive, easy-to-use, and efficient environments to develop ABM simulations. Over the years, many tools, frameworks, and libraries have been developed, each with its characteristics and objectives.

This article aims to guide scientists in choosing the proper ABM tool according to their needs and skills. Specifically, we propose a thorough overview of open-source general-purpose ABM tools and offer a comparison from a two-fold perspective. We first describe an off-the-shelf evaluation by considering each ABM tool's features, ease of use, and efficiency according to its authors. Then, we provide a hands-on evaluation of some ABM tools by judging the experience in developing and running four ABM models and the obtained performance. In particular, this work empirically demonstrates that most platforms meet the goals and priorities set by their authors. Extreme ease of use characterizes tools such as GAMA and NetLogo, which represent the optimal choice for lay users as they do not require any technical skill to be used effectively and provide a VSL along with extensive documentation. MASON and krABMaga offer a variety of convenient features and solid documentation, enabling modelers with some coding experience to easily develop and even run complex simulations. Moreover, they also provide a good trade-off between ease of use and efficiency. When performance is critical and HPC systems are available, FLAME GPU, RepastHPC, and MASON (using its distributed extension) embody the best choices, even though these platforms require extended training and expertise. In this context, we must emphasize that ABM applications and domains span profoundly diverse disciplines; therefore, the choice of the right ABM tool is often strongly influenced by the features needed for the specific use case. For instance, modelers that need deeper insights into the simulations' behavior will find Agents.jl and Mesa as their best options thanks to their functionalities and a programming language ideal for data analysis.

This review points out that there is no perfect ABM platform, but modelers must choose the right tool primarily based on their technical skills and application requirements (e.g., elevated computational loads, GIS data management, visualization). Specifically, modelers must evaluate their confidence in coding in a specific programming language, the amount of advanced and non-advanced functionalities required, and the scale and complexity of the models. Based on these parameters, it is possible to identify the tool that best suits the user's needs, finding the right compromise between ease of use and efficiency. Given their inherent distributed nature, ABMs lend themselves well to analyzing phenomena from the ever more attractive distributed computing domains such as federated learning [121–124] and blockchain systems [125–128]. An interesting perspective worth exploring in future work could be assessing the current ABM tool landscape's availability and suitability in such domains to inform possible feature development in this context.

Author Contributions: Conceptualization, G.D. and C.S.; methodology, G.D. and D.D.V. and C.S.; software, D.D.V.; validation, A.A. and C.S.; formal analysis, C.S. and G.C.; investigation, G.D. and D.D.V.; data curation, A.A. and G.D. and D.D.V.; writing—original draft preparation, G.D.; writing—review and editing, A.A. and G.D. and C.S.; visualization, A.A. and G.D.; supervision, A.A. and C.S. and G.C.; project administration, C.S. and G.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Donkin, E.; Dennis, P.; Ustalakov, A.; Warren, J.; Clare, A. Replicating complex agent based models, a formidable task. *Environ. Model. Softw.* **2017**, *92*, 142–151. [CrossRef]
2. Macal, C.M. Everything you need to know about agent-based modelling and simulation. *J. Simul.* **2016**, *10*, 144–156. [CrossRef]
3. Macal, C.M.; North, M.J. Tutorial on agent-based modelling and simulation. *J. Simul.* **2010**, *4*, 151–162. [CrossRef]
4. Siebers, P.O.; Macal, C.M.; Garnett, J.; Buxton, D.; Pidd, M. Discrete-event simulation is dead, long live agent-based simulation! *J. Simul.* **2010**, *4*, 204–210. [CrossRef]
5. Abar, S.; Theodoropoulos, G.K.; Lemarinier, P.; O'Hare, G.M. Agent Based Modelling and Simulation tools: A review of the state-of-art software. *Comput. Sci. Rev.* **2017**, *24*, 13–33. [CrossRef]
6. García-Magariño, I.; Lombas, A.S.; Plaza, I.; Medrano, C. ABS-SOCI: An Agent-Based Simulator of Student Sociograms. *Appl. Sci.* **2017**, *7*, 1126. [CrossRef]
7. Farmer, J.D.; Foley, D. The economy needs agent-based modelling. *Nature* **2009**, *460*, 685–686. [CrossRef] [PubMed]
8. Bert, F.; North, M.; Rovere, S.; Tatara, E.; Macal, C.; Podestá, G. Simulating agricultural land rental markets by combining agent-based models with traditional economics concepts: The case of the Argentine Pampas. *Environ. Model. Softw.* **2015**, *71*, 97–110. [CrossRef]
9. Farmer, J.D.; Hepburn, C.; Mealy, P.; Teytelboym, A. A Third Wave in the Economics of Climate Change. *Environ. Resour. Econ.* **2015**, *62*, 329–357. [CrossRef]
10. Hailegiorgis, A.; Crooks, A.; Cioffi-Revilla, C. An Agent-Based Model of Rural Households' Adaptation to Climate Change. *J. Artif. Soc. Soc. Simul.* **2018**, *21*, 4. [CrossRef]
11. Waleed, M.; Um, T.W.; Kamal, T.; Khan, A.; Zahid, Z.U. SIM-D: An Agent-Based Simulator for Modeling Contagion in Population. *Appl. Sci.* **2020**, *10*, 7745. [CrossRef]
12. Antelmi, A.; Cordasco, G.; Spagnuolo, C.; Scarano, V. A Design-Methodology for Epidemic Dynamics via Time-Varying Hypergraphs. In Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, Auckland, New Zealand, 9–13 May 2020; International Foundation for Autonomous Agents and Multiagent Systems: Richland, SC, USA, 2020; pp. 61–69. [CrossRef]
13. Kato, T.; Kamoshida, R. Multi-Agent Simulation Environment for Logistics Warehouse Design Based on Self-Contained Agents. *Appl. Sci.* **2020**, *10*, 7552. [CrossRef]
14. Serrano-Hernandez, A.; Faulin, J.; Hirsch, P.; Fikar, C. Agent-based simulation for horizontal cooperation in logistics and transportation: From the individual to the grand coalition. *Simul. Model. Pract. Theory* **2018**, *85*, 47–59. [CrossRef]
15. Allan, R.J. *Survey of Agent Based Modelling and Simulation Tools*; Science & Technology Facilities Council: New York, NY, USA, 2010.
16. Railsback, S.F.; Lytinen, S.L.; Jackson, S.K. Agent-based Simulation Platforms: Review and Development Recommendations. *Simulation* **2006**, *82*, 609–623. [CrossRef]
17. Kravari, K.; Bassiliades, N. A Survey of Agent Platforms. *J. Artif. Soc. Soc. Simul.* **2015**, *18*, 11. [CrossRef]
18. Rousset, A.; Herrmann, B.; Lang, C.; Philippe, L. A survey on parallel and distributed multi-agent systems for high performance computing simulations. *Comput. Sci. Rev.* **2016**, *22*, 27–46. [CrossRef]
19. Pal, C.; Leon, F.; Paprzycki, M.; Ganzha, M. A Review of Platforms for the Development of Agent Systems. *arXiv* **2020**, arXiv:2007.08961.
20. Castle, C.; Crooks, A. *Principles and Concepts of Agent-Based Modelling for Developing Geospatial Simulations*; Working Paper. CASA Working Papers (110); Centre for Advanced Spatial Analysis (UCL): London, UK, 2006; Volume 110.
21. Brown, D.G.; Riolo, R.; Robinson, D.T.; North, M.; Rand, W. Spatial process and data models: Toward integration of agent-based models and GIS. *J. Geogr. Syst.* **2005**, *7*, 25–47. [CrossRef]
22. Taillandier, P.; Gaudou, B.; Grignard, A.; Huynh, Q.N.; Marilleau, N.; Caillou, P.; Philippon, D.; Drogoul, A. Building, composing and experimenting complex spatial models with the GAMA platform. *GeoInformatica* **2019**, *23*, 299–322. [CrossRef]
23. Gilbert, N.; Bankes, S. Platforms and methods for agent-based modeling. *Proc. Natl. Acad. Sci. USA* **2002**, *99*, 7197–7198. [CrossRef]
24. Bordini, R.H.; Braubach, L.; Dastani, M.; Fallah-Seghrouchni, A.E.; Gómez-Sanz, J.J.; Leite, J.; O'Hare, G.M.P.; Pokahr, A.; Ricci, A. A Survey of Programming Languages and Platforms for Multi-Agent Systems. *Informatica* **2006**, *30*, 33–44.
25. Nikolai, C.; Madey, G. Tools of the Trade: A Survey of Various Agent Based Modeling Platforms. *J. Artif. Soc. Soc. Simul.* **2009**, *12*, 1–2.
26. Theodoropoulos, G.; Minson, R.; Ewald, R.; Lees, M.; Uhrmacher, A.; Weyns, D., Simulation Engines for Multi-Agent Systems. In *Multi-Agent Systems: Simulation and Applications*; Taylor & Francis: Abingdon, UK, 2009; Chapter 3.
27. Suryanarayanan, V.; Theodoropoulos, G.; Lees, M. PDES-MAS: Distributed Simulation of Multi-agent Systems. *Procedia Comput. Sci.* **2013**, *18*, 671–681. [CrossRef]
28. Tobias, R.; Hofmann, C. Evaluation of free Java-libraries for social-scientific agent based simulation. *J. Artif. Soc. Soc. Simul.* **2004**, *7*, 1–6.
29. Gupta, R.; Kansal, G. A Survey on Comparative Study of Mobile Agent Platforms. *Int. J. Eng. Sci. Technol.* **2011**, *3*, 1943–1948.
30. Bartley, B. Mobility impacts, reactions and opinions: Traffic demand management options in Europe: The MIRO Project. *Traffic Eng. Control* **1995**, *36*, 596–602.

31. Zia, K.; Farrahi, K.; Riener, A.; Ferscha, A. An agent-based parallel geo-simulation of urban mobility during city-scale evacuation. *Simulation* **2013**, *89*, 1184–1214. [CrossRef]
32. Carley, K.; Fridsma, D.; Casman, E.; Yahja, A.; Altman, N.; Chen, L.C.; Kaminsky, B.; Nave, D. BioWar: Scalable agent-based model of bioattacks. *IEEE Trans. Syst. Man Cybern. Part Syst. Hum.* **2006**, *36*, 252–265. [CrossRef]
33. Epstein, J.M.; Goedecke, D.M.; Yu, F.; Morris, R.J.; Wagener, D.K.; Bobashev, G.V. Controlling Pandemic Flu: The Value of International Air Travel Restrictions. *PLoS ONE* **2007**, *2*, e401. [CrossRef]
34. Zhuge, C.; Shao, C.; Wang, S.; Hu, Y. An agent- and GIS-based virtual city creator: A case study of Beijing, China. *J. Transp. Land Use* **2018**, *11*, 1231–1256. [CrossRef]
35. Axelrod, R. Advancing the Art of Simulation in the Social Sciences. In *Proceedings of the Simulating Social Phenomena*; Conte, R., Hegselmann, R., Terna, P., Eds.; Springer: Berlin/Heidelberg, Germany, 1997; pp. 21–40.
36. Axtell, R.; Axelrod, R.; Epstein, J.M.; Cohen, M.D. Aligning simulation models: A case study and results. *Comput. Math. Organ. Theory* **1996**, *1*, 123–141. [CrossRef]
37. Heath, B.; Hill, R.; Ciarallo, F. A Survey of Agent-Based Modeling Practices (January 1998 to July 2008). *J. Artif. Soc. Soc. Simul.* **2009**, *12*, 9.
38. Bankes, S.C. Agent-based modeling: A revolution? *Proc. Natl. Acad. Sci. USA* **2002**, *99*, 7199–7200. [CrossRef]
39. Brown, D.; Page, S.; Riolo, R.; Zellner, M.; Rand, W. Path dependence and the validation of agent-based spatial models of land use. *Int. J. Geogr. Inf. Sci.* **2005**, *19*, 153–174. [CrossRef]
40. Leon, F. ActressMAS, a .NET Multi-Agent Framework Inspired by the Actor Model. *Mathematics* **2022**, *10*, 382. [CrossRef]
41. Foramitti, J. AgentPy: A package for agent-based modeling in Python. *J. Open Source Softw.* **2021**, *6*, 3065. [CrossRef]
42. Datseris, G.; Vahdati, A.R.; DuBois, T.C. Agents.jl: A performant and feature-full agent-based modeling software of minimal code complexity. *Simulation* **2022**, 003754972110688. [CrossRef]
43. Borges, F.; Gutierrez-Milla, A.; Luque, E.; Suppi, R. Care HPS: A high performance simulation tool for parallel and distributed agent-based modeling. *Future Gener. Comput. Syst.* **2017**, *68*, 59–73. [CrossRef]
44. Bommel, P.; Becu, N.; Le Page, C.; Bousquet, F. Cormas: An Agent-Based Simulation Platform for Coupling Human Decisions with Computerized Dynamics. In *Proceedings of the Simulation and Gaming in the Network Society*; Springer: Singapore, 2016; pp. 387–410. [CrossRef]
45. Nourisa, J.; Zeller-Plumhoff, B.; Willumeit-Römer, R. CppyABM: An open-source agent-based modeling library to integrate C++ and Python. *Softw. Pract. Exp.* **2022**, *52*, 1337–1351. [CrossRef]
46. Standish, R.K.; Leow, R. EcoLab: Agent Based Modeling for C++ programmers. *arXiv* **2004**, ARXIV.CS/0401026. [CrossRef]
47. Cardinot, M.; O'Riordan, C.; Griffith, J.; Perc, M. Evoplex: A platform for agent-based modeling on networks. *SoftwareX* **2019**, *9*, 199–204. [CrossRef]
48. Holcombe, M.; Coakley, S.; Smallwood, R. A general framework for agent-based modelling of complex systems. In Proceedings of the 2006 European Conference Complex Systems, Paris, France, 25–29 September 2006.
49. Coakley, S.; Gheorghe, M.; Holcombe, M.; Chin, S.; Worth, D.; Greenough, C. Exploitation of High Performance Computing in the FLAME Agent-Based Simulation Framework. In Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems, Liverpool, UK, 25–27 June 2012; pp. 538–545. [CrossRef]
50. Hai, H.B.; Contamin, L.; Choisy, M.; Brugière, A. Gamar: An R Interface to the GAMA Platform. Available online: https://github.com/r-and-gama/gamar (accessed on 31 October 2022).
51. Fortmann-Roe, S. Insight Maker: A general-purpose tool for web-based modeling & simulation. *Simul. Model. Pract. Theory* **2014**, *47*, 28–45. [CrossRef]
52. Bellifemine, F.; Poggi, A.; Rimassa, G. Developing multi-agent systems with a FIPA-compliant agent framework. *Softw. Pract. Exp.* **2001**, *31*, 103–128. [CrossRef]
53. Richiardi, M.G.; Richardson, R.E. JAS-mine: A new platform for microsimulation and agent-based modelling. *Int. J. Microsimulation* **2017**, *10*, 106–134. [CrossRef]
54. Antelmi, A.; Cordasco, G.; D'Auria, M.; De Vinco, D.; Negro, A.; Spagnuolo, C. On Evaluating Rust as a Programming Language for the Future of Massive Agent-Based Simulations. In *Proceedings of the Methods and Applications for Modeling and Simulation of Complex Systems*; Springer: Singapore, 2019; pp. 15–28.
55. Gutknecht, O.; Ferber, J. The MadKit Agent Platform Architecture. In *Proceedings of the Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*; Wagner, T., Rana, O.F., Eds. Springer: Berlin/Heidelberg, Germany, 2001; pp. 48–55.
56. Luke, S.; Cioffi-Revilla, C.; Panait, L.; Sullivan, K.; Balan, G. MASON: A Multiagent Simulation Environment. *Simulation* **2005**, *81*, 517–527. [CrossRef]
57. Sullivan, K.; Coletti, M.; Luke, S.; Crooks, A. GeoMason: Geospatial Support for MASON. 2010. Available online: https://cs.gmu.edu/~eclab/projects/mason/extensions/geomason/ (accessed on 31 October 2022).
58. White, D.R. Software review: The ECJ toolkit. *Genet. Program. Evolvable Mach.* **2012**, *13*, 65–67. [CrossRef]
59. Cordasco, G.; Scarano, V.; Spagnuolo, C. Distributed MASON: A scalable distributed multi-agent simulation environment. *Simul. Model. Pract. Theory* **2018**, *89*, 15–34. [CrossRef]

60. Chuang, T.; Fukuda, M. A Parallel Multi-agent Spatial Simulation Environment for Cluster Systems. In Proceedings of the 2013 IEEE 16th International Conference on Computational Science and Engineering, Sydney, Australia, 3–5 December 2013; pp. 143–150. [CrossRef]
61. Kazil, J.; Masad, D.; Crooks, A. Utilizing Python for Agent-Based Modeling: The Mesa Framework. In *Proceedings of the Social, Cultural, and Behavioral Modeling*; Springer International Publishing: Berlin/Heidelberg, Germany, 2020; pp. 308–317.
62. Uri, W. Modeling nature's emergent patterns with multi-agent languages. In Proceedings of the EuroLogo 2001, Linz, Austria, 21–25 August 2001.
63. Jaxa-Rozen, M.; Kwakkel, J.H. PyNetLogo: Linking NetLogo with Python. *J. Artif. Soc. Soc. Simul.* **2018**, *21*, 4. [CrossRef]
64. Abbott, R.; Lim, J. PyLogo: A Python Reimplementation of (Much of) NetLogo. In Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications—SIMULTECH, INSTICC, SciTePress, Online, 7–9 July 2021; pp. 199–206. [CrossRef]
65. Thiele, J.C. R Marries NetLogo: Introduction to the RNetLogo Package. *J. Stat. Softw.* **2014**, *58*, 1–41. [CrossRef]
66. Jiang, L.; Zhao, C. The Netlogo-Based Dynamic Model for the Teaching. In Proceedings of the 2009 Ninth International Conference on Hybrid Intelligent Systems, Shenyang, China, 12–14 August 2009; Volume 2, pp. 49–53. [CrossRef]
67. Railsback, S.F.; Ayllón, D.; Berger, U.; Grimm, V.; Lytinen, S.; Sheppard, C.; Thiele, J. Improving Execution Speed of Models Implemented in NetLogo. *J. Artif. Soc. Soc. Simul.* **2017**, *20*, 3. [CrossRef]
68. Rubio-Campillo, X. Pandora: A Versatile Agent-Based Modelling Platform for Social Simulation. In Proceedings of the SIMUL 2014, The Sixth International Conference on Advances in System Simulation, Nice, France, 12–16 October 2014; pp. 29–34. [CrossRef]
69. North, M.J.; Collier, N.T.; Vos, J.R. Experiences creating three implementations of the repast agent modeling toolkit. *ACM Trans. Model. Comput. Simul.* **2006**, *16*, 1–25. [CrossRef]
70. North, M.J.; Collier, N.T.; Ozik, J.; Tatara, E.R.; Macal, C.M.; Bragen, M.; Sydelko, P. Complex adaptive systems modeling with Repast Simphony. *Complex Adapt. Syst. Model.* **2013**, *1*, 1–26. [CrossRef]
71. Collier, N.T.; Ozik, J.; Tatara, E.R. Experiences in Developing a Distributed Agent-based Modeling Toolkit with Python. In Proceedings of the 2020 IEEE/ACM 9th Workshop on Python for High-Performance and Scientific Computing (PyHPC), Atlanta, GA, USA, 13 November 2020; pp. 1–12. [CrossRef]
72. Collier, N.; North, M. Parallel agent-based simulation with Repast for High Performance Computing. *Simulation* **2013**, *89*, 1215–1235. [CrossRef]
73. Florin, L. ActressMas. Available online: https://github.com/florinleon/ActressMas (accessed on 31 October 2022).
74. Foramitti, J. AgentPy—Agent-Based Modeling in Python. Available online: https://agentpy.readthedocs.io/en/latest/ (accessed on 31 October 2022).
75. Datseris, G.; Vahdati, A.R.; DuBois, T.C. Agents.jl. Available online: https://juliadynamics.github.io/Agents.jl/ (accessed on 31 October 2022).
76. Bommel, P.; Becu, N.; Le Page, C.; Bousquet, F. CORMAS: COmmon-Pool Resources and Multi-Agent Simulations. Available online: http://cormas.cirad.fr/indexeng.html (accessed on 31 October 2022).
77. Nourisa, J. CppyABM. Available online: https://pypi.org/project/cppyabm/ (accessed on 31 October 2022).
78. Standish, R.K.; Leow, R. EcoLab. Available online: https://ecolab.sourceforge.net/ (accessed on 31 October 2022).
79. Cardinot, M.; O'Riordan, C.; Griffith, J.; Perc, M. Evoplex–Agent-Based Modeling on Networks. Available online: https://evoplex.org/ (accessed on 31 October 2022).
80. Laboratory, S.R.A. FLAME. Available online: http://flame.ac.uk/ (accessed on 31 October 2022).
81. Richmond, P. FLAME GPU. Available online: https://flamegpu.com/ (accessed on 31 October 2022).
82. Taillandier, P.; Gaudou, B.; Grignard, A.; Huynh, Q.N.; Marilleau, N.; Caillou, P.; Philippon, D.; Drogoul, A. GAMA Platform. Available online: https://gama-platform.org/ (accessed on 31 October 2022).
83. Fortmann-Roe, S. Insight Maker. Available online: https://insightmaker.com/ (accessed on 31 October 2022).
84. Bellifemine, F.; Caire, G.; Rimassa, G.; Poggi, A.; Bergenti, F.; Trucco, T.; Gotta, D.; Cortese, E.; Quarta, F.; Vitaglione, G. JADE Site | Java Agent Development Framework. Available online: https://jade.tilab.com/ (accessed on 31 October 2022).
85. Richiardi, M.G.; Richardson, R.E. JAS-Mine. Available online: http://jas-mine.net/ (accessed on 31 October 2022).
86. Spagnuolo, C.; D'Ambrosio, G.; De Vinco, D.; Postiglione, L.; Foglia, F.; Caramante, P. krABMaga. Available online: https://krabmaga.github.io/ (accessed on 31 October 2022).
87. Gutknecht, O.; Ferber, J. MaDKit. Available online: https://www.madkit.net/madkit/ (accessed on 31 October 2022).
88. Sean, L.; Catalin Balan, G.; Sullivan, K.; Panait, L. MASON Multiagent Simulation Toolkit. Available online: https://cs.gmu.edu/~eclab/projects/mason/ (accessed on 31 October 2022).
89. Munehiro, F.; Freksa, C.; Salathe, E.; Wooyoung, K.; Yasushi, K. MASS: A Parallelizing Library for Multi-Agent Spatial Simulation. Available online: https://depts.washington.edu/dslab/MASS/ (accessed on 31 October 2022).
90. Boyu, W.; Kazil, J. Mesa-Geo: GIS Extension for Mesa Agent-Based Modeling. Available online: https://github.com/projectmesa/mesa-geo (accessed on 31 October 2022).
91. Mesa Community. Mesa 3D Graphics Library. Available online: https://github.com/Mesa3D/mesa (accessed on 31 October 2022).

92. Kazil, J.; Masad, D.; Crooks, A. Mesa: Agent-Based Modeling in Python 3+. Available online: https://mesa.readthedocs.io/en/latest/ (accessed on 31 October 2022).
93. Abbott, R.; Lim, J. PyLogo. Available online: https://pylogo.sourceforge.net/ (accessed on 31 October 2022).
94. Jaxa-Rozen, M.; Kwakkel, J.H. pyNetLogo. Available online: https://pynetlogo.readthedocs.io/en/latest/ (accessed on 31 October 2022).
95. Thiele, J.C. RNetLogo. Available online: http://rnetlogo.r-forge.r-project.org/ (accessed on 31 October 2022).
96. Russell, E.; Hovet, J. NetLogo Gis Extension. Available online: https://ccl.northwestern.edu/netlogo/docs/gis.html (accessed on 31 October 2022).
97. Brady, C.; Jeremy; Grider, R.; Brandes, A. NetLogo View2.5D Extension. Available online: https://github.com/NetLogo/View2.5D (accessed on 31 October 2022).
98. Uri, W.; Hjorth, A.; Bain, C.; Payette, N.; Head, B.; Bertsche, J. NetLogo. Available online: https://ccl.northwestern.edu/netlogo (accessed on 31 October 2022).
99. Rubio-Campillo, X. Pandora. Available online: http://xrubio.github.io/pandora/ (accessed on 31 October 2022).
100. Collier, N.; Murphy, J.T.; Ozik, J.; Rimer, S.; Sheeler, D.; Tatara, E. Repast Suite. Available online: https://repast.github.io/ (accessed on 31 October 2022).
101. Collier, N.; Murphy, J.T.; Ozik, J.; Rimer, S.; Sheeler, D.; Tatara, E. Repast HPC. Available online: https://repast.github.io/repast_hpc.html (accessed on 31 October 2022).
102. Foramitti, J. AgentPy CI/CD. Available online: https://github.com/JoelForamitti/agentpy/blob/master/.github/workflows/test.yml (accessed on 9 December 2022).
103. Datseris, G.; Vahdati, A.R.; DuBois, T.C. Agents.jl CI/CD. Available online: https://github.com/JuliaDynamics/Agents.jl/blob/main/.github/workflows/ci.yml (accessed on 9 December 2022).
104. Datseris, G.; Vahdati, A.R.; DuBois, T.C. Distributed Computing—The Julia Language. Available online: https://docs.julialang.org/en/v1/stdlib/Distributed/ (accessed on 31 October 2022).
105. Bommel, P.; Becu, N.; Le Page, C.; Bousquet, F. CORMAS CI/CD. Available online: https://github.com/cormas/cormas/blob/master/.github/workflows/test.yml (accessed on 9 December 2022).
106. Standish, R.K.; Leow, R. EcoLab CI/CD. Available online: https://github.com/highperformancecoder/ecolab/blob/master/.github/workflows/main.yml (accessed on 9 December 2022).
107. Cardinot, M.; O'Riordan, C.; Griffith, J.; Perc, M. Evoplex—CI/CD. Available online: https://github.com/evoplex/evoplex/blob/master/.travis.yml (accessed on 9 December 2022).
108. Richmond, P. FLAME GPU CI/CD. Available online: https://github.com/FLAMEGPU/FLAMEGPU2/blob/master/.github/workflows/Draft-Release.yml (accessed on 9 December 2022).
109. Bellifemine, F.; Caire, G.; Rimassa, G.; Poggi, A.; Bergenti, F.; Trucco, T.; Gotta, D.; Cortese, E.; Quarta, F.; Vitaglione, G. JADE CI/CD. Available online: https://jade-project.gitlab.io/docs/add-on/JADE_TestSuite.pdf (accessed on 9 December 2022).
110. Spagnuolo, C.; D'Ambrosio, G.; De Vinco, D.; Postiglione, L.; Foglia, F.; Caramante, P. krABMaga CI/CD. Available online: https://github.com/krABMaga/krABMaga/blob/main/.github/workflows/rust-ci.yml (accessed on 9 December 2022).
111. Gutknecht, O.; Ferber, J. MaDKit CI/CD. Available online: https://github.com/fmichel/MaDKit/blob/master/.travis.yml (accessed on 9 December 2022).
112. George Mason University's ECLab Evolutionary Computation Laboratory. ECJ A Java-Based Evolutionary Computation Research System. Available online: https://cs.gmu.edu/~eclab/projects/ecj/ (accessed on 31 October 2022).
113. Wang, C.H.; Wei, E.; Lather, R.S.; Patel, R.; Dinh, L.; D'Auria, M.; D'Ambrosio, G.; Vinco, D.D.; Moffatt, R.; Osman, Z.; et al. Distributed MASON. Available online: https://cs.gmu.edu/~eclab/projects/mason/extensions/distributed/ (accessed on 31 October 2022).
114. Munehiro, F.; Freksa, C.; Salathe, E.; Wooyoung, K.; Yasushi, K. MASS CI/CD. Available online: https://bitbucket.org/mass_library_developers/mass_java_core/src/master/bitbucket-pipelines.yml (accessed on 9 December 2022).
115. Kazil, J.; Masad, D.; Crooks, A. Mesa CI/CD. Available online: https://github.com/projectmesa/mesa/blob/main/.github/workflows/build_lint.yml (accessed on 9 December 2022).
116. Staelin, C. NetLogo Stats Extension. Available online: https://github.com/cstaelin/Stats-Extension (accessed on 31 October 2022).
117. Tisue, S.; Wilensky, U. BehaviorSpace. Available online: https://ccl.northwestern.edu/netlogo/docs/behaviorspace.html (accessed on 31 October 2022).
118. Rngs—Random Number Generator extension for NetLogo. Available online: https://github.com/AFMac/rngs (accessed on 31 October 2022).
119. Uri, W.; Hjorth, A.; Bain, C.; Payette, N.; Head, B.; Bertsche, J. NetLogo CI/CD. Available online: https://github.com/NetLogo/NetLogo/blob/hexy/.github/workflows/main.yml (accessed on 9 December 2022).
120. Ozik, J.; Collier, N.; Wozniak, J.; Spagnuolo, C.; An, G. EMEWS: Extreme-Scale Model Exploration with Swift. Available online: https://emews.github.io/ (accessed on 31 October 2022).
121. McMahan, H.B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Proceedings of the International Conference on Artificial Intelligence and Statistics, Cadiz, Spain, 9–11 May 2016.

122. Polato, M. Federated Variational Autoencoder for Collaborative Filtering. In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 18–22 July 2021; pp. 1–8. [CrossRef]
123. Ghimire, B.; Rawat, D.B. Recent Advances on Federated Learning for Cybersecurity and Cybersecurity for Federated Learning for Internet of Things. *IEEE Internet Things J.* **2022**, *9*, 8229–8249. [CrossRef]
124. Imteaj, A.; Amini, M.H. Leveraging asynchronous federated learning to predict customers financial distress. *Intell. Syst. Appl.* **2022**, *14*, 200064. [CrossRef]
125. Roesch, M.; Linder, C.; Zimmermann, R.; Rudolf, A.; Hohmann, A.; Reinhart, G. Smart Grid for Industry Using Multi-Agent Reinforcement Learning. *Appl. Sci.* **2020**, *10*, 6900. [CrossRef]
126. Zhang, Z.; Yang, T.; Liu, Y. SABlockFL: A blockchain-based smart agent system architecture and its application in federated learning. *Int. J. Crowd Sci.* **2020**, *4*, 133–147. [CrossRef]
127. Połap, D.; Srivastava, G.; Yu, K. Agent architecture of an intelligent medical system based on federated learning and blockchain technology. *J. Inf. Secur. Appl.* **2021**, *58*, 102748. [CrossRef]
128. Rincon, J.; Julian, V.; Carrascosa, C. FLaMAS: Federated Learning Based on a SPADE MAS. *Appl. Sci.* **2022**, *12*, 3701. [CrossRef]