Appendix to the contribution: Towards Reproducible Benchmarking in Warehouse Logistics: An Open-Source Simulation

Für ein reproduzierbares Benchmarking in der Lagerlogistik: Eine Open-Source-Simulation

David Heik, Fouad Bahrpeyma, Dirk Reichelt
Hochschule für Technik und Wirtschaft Dresden, Dresden (Germany),
david@heik.science, bahrpeyma@ieee.org, dirk.reichelt@htw-dresden.de.

Abstract: This appendix provides a detailed technical overview of the modular benchmark framework introduced in the main paper. The evaluation of warehouse logistics strategies often suffers from a lack of reproducibility, opaque assumptions, and reliance on proprietary software tools. To address these issues, we present an open-source simulation environment designed to support transparent and standardized benchmarking. The framework consists of three interoperable components: a browser-based layout editor, a Python-based simulation engine, and an interactive visualization module. It supports the integration of custom layouts and solving strategies, including heuristics, metaheuristics, and (multi-agent) reinforcement learning, under controlled experimental conditions. In addition, the system enables the creation and reuse of documented test scenarios, facilitating method comparison and community-driven validation.

1 Proposed Approach and Contribution

The improvement of warehouse systems through data-driven methods is currently hindered by fragmented toolchains, proprietary simulation environments, and inconsistent evaluation procedures. Researchers often rely on custom-made setups that limit the reproducibility of experiments and obstruct meaningful comparison across methods. To address these challenges, we introduce a modular and open-source simulation framework that supports the structured development, execution, and evaluation of warehouse layout designs and picker routing strategies. The system is composed of three tightly integrated components, each responsible for a key aspect of the experimental workflow:

- A web-based layout editor (Module 1) for interactive modelling of warehouse topologies,
- A Python-based simulation engine (Module 2) that executes picker movements and applies routing logic under controlled conditions,
- A visualization frontend (Module 3) for the graphical inspection and side-by-side comparison of simulation results.

As shown in Figure 1, the architecture of the framework is a strict separation between:

- the framework logic (open-source codebase) and
- the user-defined assets (custom layouts, scenarios, and algorithms).

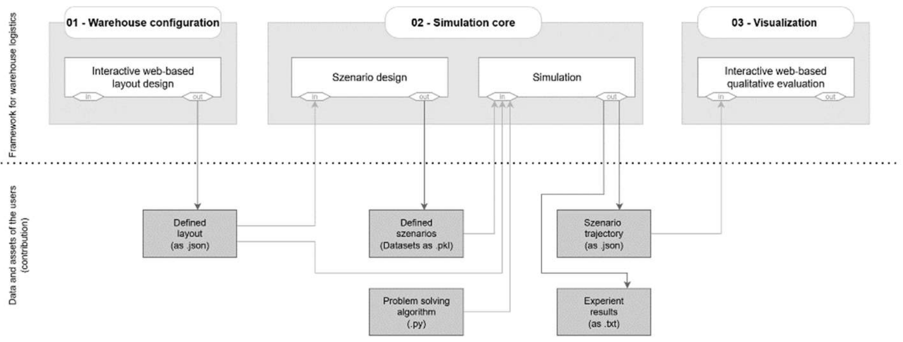
Framework for warehouse logistics
01 - Warehouse configuration
Interactive web-based layout design
in
out
02 - Simulation core
Szenario design
in
out
Simulation
in
out
03 - Visualization
Interactive web-based qualitative evaluation
in
out
Data and assets of the users (contribution)
Defined layout (as .json)
Defined szenarios (Datasets as .pkl)
Szenario trajectory (as .json)
Problem solving algorithm (.py)
Experient results (as .txt)

Figure 1 *Modular System Architecture for Benchmarking in Warehouse Logistics*

Users are welcome to further enhance the framework itself, but the primary focus is on utilizing the already established functionality and contributing the following elements:

- a defined layout (as .json), created via Module 1 (examples are shown in Figure 2),
- or individual (novel) problem-solving algorithms (e.g. heuristics or learning-based strategies, as .py files),
- and defined scenarios (e.g., picking datasets as .pkl-files) with fixed order configurations (within Model 2)

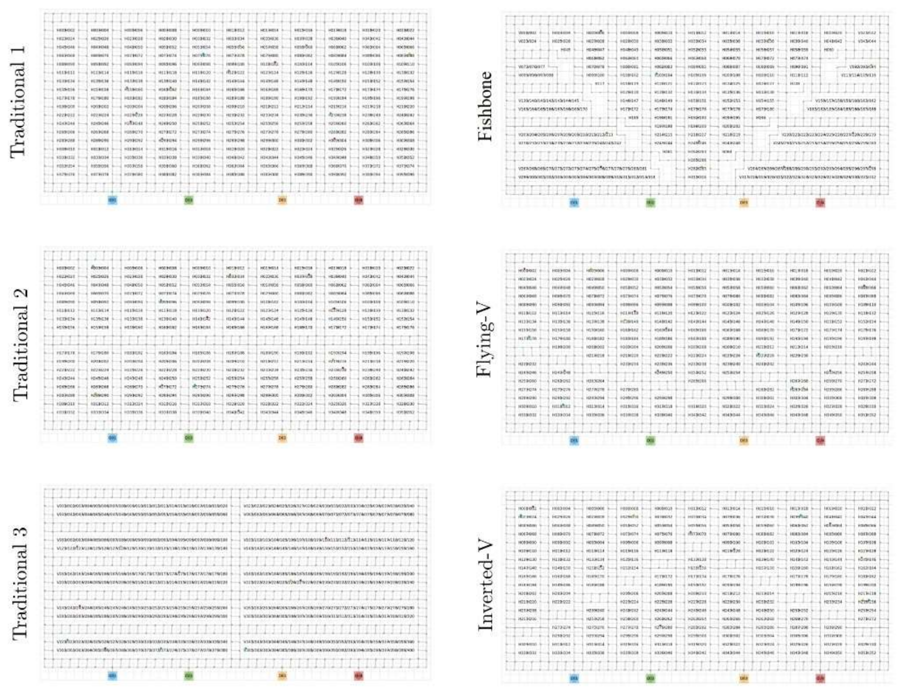

***Figure 2**: Common warehouse layout types within visualization module*

These inputs feed into the simulation process (Module 2), which generates two distinct outputs:

- **Scenario trajectories** (.json) – detailed stepwise movement logs for each agent.
- **Experiment results** (.txt) – aggregated performance metrics such as total steps, waiting times, for comparison with other solutions or warehouse arrangements.

These scenario trajectories artefacts are then loaded into the visualization module (Module 3), which enables the qualitative comparison of strategies and layout decisions. All modules communicate via standardized, human-readable .json formats that enable transparent experiments and extensible integration.

In addition to the core functionality, a fourth component is planned: a benchmarking and comparison platform that enables structured, cross-method evaluation on a shared set of scenarios. While this component is not yet technically implemented or hosted online, the current strategy follows a decentralized approach. Researchers are encouraged to share documented layouts, scenarios, and solver results either via publication appendices or as forks of the official GitHub repository (https://github.com/david-dd/Attempt-to-benchmark-warehouse-logistics).

This design deliberately avoids centralized administration in the initial phase. Instead, authors and reviewers of individual publications or conference contributions remain free to decide which methods or results are included in their comparative analyses. This flexible model supports gradual community uptake and lowers the barrier for contribution and reuse, while paving the way for future consolidation into a more formal benchmarking platform.

1.1 Module 1: Layout Design Tool

The layout design tool provides an intuitive interface for defining and configuring warehouse environments. It allows users to create realistic and flexible floor plans that serve as the basis for all subsequent simulations.

Key functionalities include:

- Specification of overall warehouse dimensions
- Definition of navigable aisle structures, including uni- and bidirectional paths
- Placement of storage locations (e.g., shelves, racks) within the layout
- Configuration of picker start positions and working areas
- Designation of free zones or inaccessible regions, allowing for realistic modeling of pillars, machinery, or restricted areas

This component generates a machine-readable layout file (JSON format), which serves as a standardized input for the simulation module.

1.2 Module 2: Simulation Core

The simulation core constitutes the central execution environment for evaluating warehouse operations. It integrates various algorithmic approaches and executes scenario-specific simulations. Core functionalities include:

- Import of layout configurations generated by the design module
- Modular integration of different problem-solving strategies, such as heuristics, metaheuristics, or reinforcement learning agents
- Scenario generation and configuration, enabling the definition of individual picking tasks for each agent (picker), including the randomized or controlled placement of target items
- Execution of simulation runs under consistent and repeatable conditions

The simulation kernel is implemented in Python and designed for extensibility. Results are exported in a structured JSON format, enabling downstream analysis.

1.3 Module 3: Visualization Module

The visualization module supports the qualitative validation and graphical analysis of simulation outcomes. While not mandatory for running experiments, it serves as a valuable tool for interpreting results. Key features include:

- Graphical rendering of picker trajectories, highlighting route differences between solvers
- Side-by-side comparison of different solution strategies, such as classical heuristics and learning-based methods
- Identification of bottlenecks and critical interaction zones, enabling a more detailed understanding of inefficiencies within the layout

This component is particularly useful during development and benchmarking phases, as it enables researchers to better understand agent behavior and route allocation patterns.

1.4 Module 4: Benchmarking and Comparison Platform (Decentralized Extension)

Although not yet fully integrated as a technical component, we envision a fourth module dedicated to the transparent sharing and structured comparison of results across the research community. Given the open-source nature of the framework, contributors are encouraged to publish their solution strategies alongside documented scenarios – either within conference appendices or as forks of the official GitHub repository (https://github.com/david-dd/Attempt-to-benchmark-warehouse-logistics). This decentralized approach lowers the barrier to entry and fosters community-driven growth without the need for immediate centralized infrastructure.

2 Start-up and first steps

2.1 Module 1: Layout Design Tool

2.1.1 Installation instructions

The layout design tool is provided as a browser-based web application and requires a local Apache server environment for deployment. We recommend using solutions such as XAMPP or LAMP for a straightforward setup.

- Install a local Apache server environment (e.g., XAMPP for Windows/macOS (https://www.apachefriends.org) or LAMP for Linux).
- Copy the contents of **Module 1** into a subdirectory, e.g., C:\xampp\htdocs\warehouse.
- Start the Apache server and open the tool in your browser at: http://localhost/warehouse

2.1.2 Initial usage steps:

Once launched, the layout editor provides a configurable grid interface for defining the warehouse layout. We recommend selecting a slightly larger grid than the anticipated layout size to allow for future extensions.

Element definitions and interactions:

- Storage Locations: Each grid cell can host a single abstract storage unit. These may represent ground-level zones, racks, or multi-level shelves. For each unit, the user must define whether it is accessible via horizontal or vertical pathways, reflecting operational constraints.
- Aisle Structures: Aisles define which movement directions are permitted from each cell (up, down, left, right). Users can configure these as uni- or bidirectional, supporting both one-way and two-way flow. This structure defines the navigable topology for agent movement.
- Picker Positions: One or more picker start locations can be defined. These positions serve as the origin and return point for pickers during simulation runs.
- Free or Blocked Zones: Non-usable areas (e.g., structural columns, technical installations, or reserved spaces) are empty slots and therefore the default and do not need to be explicitly marked.

Advanced features:

- Copy & Paste: Users can duplicate layout structures using a relative copy tool. The first selected slot acts as a reference point; all copied elements are inserted with preserved offsets.

- Undo/Redo: Any action can be reverted or restored to support iterative design workflows.
- Deletion Tool: Individual slots can be cleared using a dedicated deletion function, ensuring flexible editing.

The resulting layout is automatically saved in a standardized JSON format and serves as the basis for subsequent simulation steps.

2.2 Module 2: Simulation Core

2.2.1 Installation Steps

The simulation core is implemented in Python and requires a local Python environment (≥ 3.8). We recommend setting up a virtual environment to manage dependencies in an isolated and reproducible manner.

- Install Python 3.8+ if not already available.
- (Optional) Create and activate a virtual environment:

```
python -m venv venv
source venv/bin/activate    # On Linux/macOS
venv\Scripts\activate.bat  # On Windows
```

- Install required packages via pip:

```
pip install -r requirements.txt
```

2.2.2 Initial usage steps:

To launch a batch simulation using predefined layouts and routing strategies, run the following command: **python start_multi.py**

This will:

- Load a selected dataset (e.g., DS01),
- Iterate through all registered warehouse layouts (e.g., Heik_Traditional_1, Heik_Flying-V),
- Apply the selected solver (e.g., OptimalRoutingCoordinator),
- Save results to disk, including episode statistics and/or exportable simulation traces in JSON format.

By default, results are stored in the output_images/ and Result_*.json files. These can later be analyzed or visualized using the optional visualization module (see Section 2.3).

2.2.3 Functionality

The simulation core constitutes the algorithmic heart of the benchmark framework. It executes the operational dynamics of the warehouse system under reproducible conditions and allows for a systematic evaluation of different layout and routing strategies.

- Import of layout files in standardized JSON format, generated via the design tool.
- Initialization of warehouse environments, including storage slots, aisle connectivity, and picker positions.
- Assignment of order sets, defined as randomized or predefined item lists to be picked and returned by each agent.
- Execution of multi-agent simulation episodes under a configurable maximum step count.
- Export of results in structured JSON and log formats for post-hoc analysis and visualization.

Routing Coordination and Planning:

At the beginning of each episode, the system triggers a route planning phase for each picker. In the case of the OptimalRoutingCoordinator, all permutations of the item list are evaluated to identify the shortest possible tour (i.e., minimal total path length) from the picker's current position, through all required items, and back to the drop-off zone. The planner checks the accessibility of all targets and only includes feasible paths using the integrated pathfinding logic. This ensures that only reachable item sequences are considered.

Execution Workflow (based on Heik_Traditional_1 and optimal strategy):

- Environment Reset: The warehouse simulation is initialized using the loaded layout (Heik_Traditional_1.json), which reflects a common block layout with parallel aisles.
- Order Restoration: Predefined orders (e.g., 4–6 items per picker for Dataset 1) are loaded.
  Please note that only a partial of the first data set is stored here, please download the complete data set:
  https://zenodo.org/records/15828553/files/ReferenceData.zip?download=1

- Path planning: For each picker, the OptimalRoutingCoordinator computes the most efficient tour via exhaustive permutation and path evaluation.
- Simulation Execution: The pickers follow their computed paths. At each time step, the environment advances by one action for each picker until all routes are completed or the step limit is reached (e.g., 400 steps).
- Export: Upon completion of an episode, the results are written to disk in JSON format if desired, including full motion traces and wait times, ready for analysis and visualization.

As it takes a while, the simulation ends by default after an export and does not run through the complete dataset.

Technical Notes:

- The simulation is implemented in Python and can be extended with additional coordination strategies (e.g., S-Shape, Return, Mid-Point).
- Evaluation settings, such as the number of pickers or the order size range, can be adjusted in the start_multi.py configuration.
- Performance metrics, such as the number of steps required and planning time, are logged per episode and aggregated across runs.

This component ensures consistent benchmarking conditions and forms the analytical backbone for quantitative comparison between layout configurations and routing algorithms.

2.3 Module 3: Visualization Module

The visualization module provides an interactive web-based interface for the qualitative evaluation of simulation results. It enables researchers to inspect picker trajectories, analyze system dynamics, and compare routing strategies side-by-side. Although not essential for running simulations, this module significantly enhances transparency and interpretability.

2.3.1 Installation Instructions

The visualization component is implemented as a standalone browser-based application. Like the layout module (see 2.1.1), it requires a local Apache server environment.

Installation Steps:

- Install XAMPP or a similar Apache-based environment.
- Copy the contents of Module 3 into a subdirectory, e.g., C:\xampp\htdocs\visualization.
- Start the Apache server and open the tool in your browser at: http://localhost/visualization

2.3.2 Usage Instructions

Once started, the tool provides a graphical interface to inspect simulation outputs. Users can select a previously exported JSON result and explore the following visualizations.
Supported Data Format: The visualization module expects result files in the JSON format exported by **Module 2**.

Key Features:

- Trajectory Visualization: Graphical rendering of picker routes over time, based on the stored movement traces. Each picker is assigned a unique color for clear distinction.
- Step-by-Step Playback: An interactive timeline allows users to step through each simulation frame, observing picker positions, movements, and interactions.
- Comparison View: Side-by-side comparison of different routing strategies (e.g., Optimal vs. S-Shape) within the same warehouse layout, highlighting behavioral differences and bottlenecks.