IR-SIM: A LIGHTWEIGHT DECLARATIVE SIMULATOR FOR NAVIGATION LEARNING AND BENCHMARKING

Anonymous authors
Paper under double-blind review

ABSTRACT

Developing navigation policies requires simulation scenarios that support repeatable training and evaluation. Despite the availability of numerous simulators, constructing diverse navigation scenarios often involves writing simulator-specific code, using complex graphical interfaces, performing substantial manual configuration, or relying on resource-intensive computing platforms, making scenarios difficult to reproduce and reuse across experiments. To this end, we develop the Intelligent Robot Simulator (IR-SIM), a lightweight declarative simulator implemented as a Python library to support navigation learning and benchmarking through rapid construction of reusable and diverse scenarios and efficient execution on accessible platforms. IR-SIM represents scenarios as human-readable YAML configurations, in which necessary objects, behaviors, sensors, maps, and environment parameters can be flexibly specified and composed. This declarative representation makes IR-SIM friendly to large language model (LLM)-powered agents, enabling them to compose and modify executable navigation scenario files through the provided agent skills, instead of writing a large amount of code that is difficult to reuse. Despite its lightweight implementation, IR-SIM provides the essential components for navigation simulation, while retaining interfaces to high-fidelity simulators for downstream validation. Experiments demonstrate that IR-SIM runs up to several hundred times faster on the evaluated CPU platform, while agent skills reduce mean LLM-based scenario construction time by more than 50% across both evaluated models. The experiments further show ability to support reproducible benchmarking and RL-based navigation policy learning.

1 INTRODUCTION

Navigation scenario construction is a necessary step in developing and evaluating navigation algorithms in robotics research. It is increasingly important with the continued development of reinforcement learning (RL)-based navigation approaches, which typically require large numbers of diverse and reusable environments for policy training (Fan et al., 2020; Xu et al., 2025). Numerous general-purpose high-fidelity simulators, such as Gazebo (Koenig & Howard, 2004), CARLA (Dosovitskiy et al., 2017), and Isaac Sim (NVIDIA, 2025), provide realistic environments with 3D physics-based engines to support various navigation tasks (Li et al., 2025; Han et al., 2025; Xu et al., 2026). However, constructing scenarios can require learning simulator-specific APIs, configuring scene assets, and implementing agent behaviors. The computational demands of high-fidelity simulation can also limit repeated training and evaluation on accessible hardware. These challenges motivate simulators that combine rapid scenario construction with efficient execution.

Thus, to reduce scenario construction complexity and improve simulation efficiency, lightweight simulators have been developed for specific navigation tasks, such as multi-agent reinforcement learning (Bettini et al., 2024), social navigation (Chen et al., 2019), multi-vehicle simulation (Blanco-Claraco et al., 2023), differential-drive robot navigation (Rutherford et al., 2024), and multi-agent driving (Kazemkhani et al., 2025). These simulators provide efficient solutions for their target applications, but their designs typically focus on particular navigation subdomains or a limited set of robot models and geometries, or depend on specific software frameworks (e.g., ROS, PyTorch) or hardware prerequisites (e.g., GPUs). These constraints make it difficult to generate diverse scenarios and reuse them across different navigation learning and evaluation workflows.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

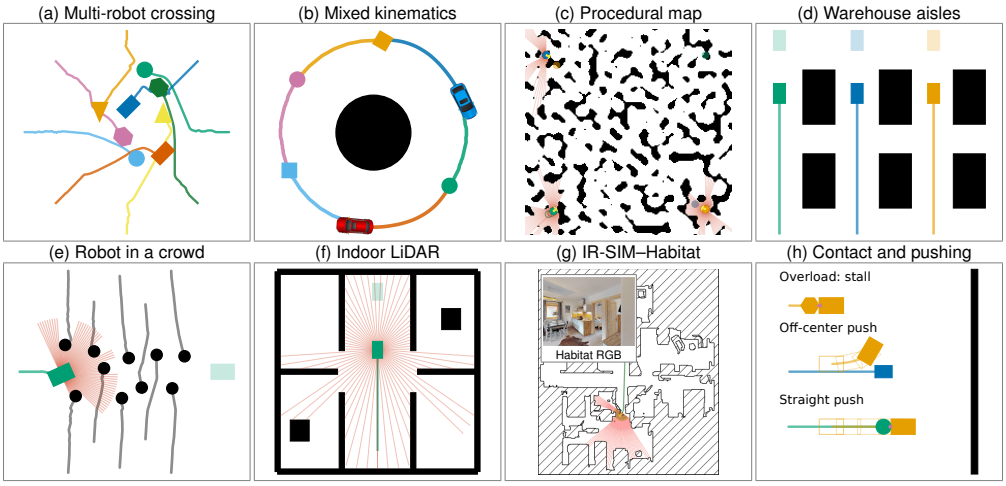


Figure 1: Navigation scenarios generated in IR-SIM. Declarative YAML configurations compose robot kinematics, geometry, agent behaviors, and sensing to support multi-agent encounters, social navigation, structured and procedural environments, and contact interactions.

Meanwhile, large language models (LLMs) have been used to generate robotic simulation tasks and training supervision (Wang et al., 2024a;b). LLM agents can construct scenes through simulator APIs, but successful authoring requires both correct API use and the intended runtime behavior. Generating simulator-specific scene and controller code can increase the amount of code that must be produced, inspected, and maintained. A compact scenario representation backed by reusable behavior implementations offers a way to simplify this authoring process for both users and LLM agents.

To address these challenges, we develop the **Intelligent Robot Simulator (IR-SIM)**, a lightweight declarative simulator implemented as a Python library for navigation learning and benchmarking. IR-SIM is fast, extensible, scalable, and LLM-friendly, with support for Windows, macOS, and Ubuntu. All the scenario components in IR-SIM, including geometry, kinematics, behaviors, sensing, contact, and maps, are declarative and represented through a human-readable YAML configuration, which can be easily modified and reused. This declarative representation reduces the need to implement scene logic and controllers repeatedly and also exploits LLM agents' ability to read and write structured text. The provided IR-SIM agent skills further enhance this ability to reduce token usage and improve the scenario-generation success rate. Figure 1 illustrates diverse scenarios in IR-SIM specified through YAML configuration files. Furthermore, to enable efficient execution on accessible platforms, IR-SIM performs planar geometric queries using Shapely (Gillies et al., 2025), numerical operations using NumPy (Harris et al., 2020), and vectorized and batched computation for world updates, particularly for LiDAR ray casting. Reusable configurations, seeded randomization, and shared evaluation protocols support navigation benchmarking, while RL interfaces connect the simulator to policy-learning libraries. Finally, we also provide bridges to external simulators and real robotic systems to further support downstream validation through dedicated adapters.

The contributions of this work can be summarized as follows:

- We develop **IR-SIM**, a lightweight declarative navigation simulator that combines rapid construction through reusable scenario components with efficient execution on accessible platforms. Generally, IR-SIM executes simulation steps faster than the compared simulators especially in large number of objects.
- We provide IR-SIM agent skills that guide LLM agents in constructing navigation scenarios, reducing mean authoring time by over 50% compared with IR-SIM without skills while achieving approximately 95% scenario validity across both evaluated models.
- We validate the effectiveness of IR-SIM through experiments on efficient execution, LLM-based scenario generation, and RL-based navigation policy learning and benchmarking. The code will be released as open source to facilitate future navigation research.

Table 1: Capability comparison of representative navigation simulators.

Simulator	Lightweightness	Declarative	Kine. diversity	Geo. diversity	Sensor sim.	Contact sim.	LLM authoring
PyRoboSim (pyrobosim Contributors, 2026)	High	✓	× (holo. only)	× (circles only)	✓	×	×
MVSim (Blanco-Claraco et al., 2023)	Medium	✓	✓	✓	✓	✓	×
VMAS (Bettini et al., 2024)	Medium	×	✓	✓	✓	✓	×
JaxNav (Rutherford et al., 2024)	Medium	×	× (diff. only)	✓	✓	×	×
CrowdNav (Chen et al., 2019)	Medium	×	✓	× (circles only)	×	×	×
GPUDrive (Kazemkhani et al., 2025)	Low	✓	✓	× (boxes only)	✓	×	×
Waymax (Gulino et al., 2023)	Low	×	✓	× (boxes only)	×	×	×
IR-SIM (ours)	High	✓	✓	✓	✓	✓	✓

Lightweightness qualitatively rates deployment burden from dependencies and installation profiles (High means lighter deployment).

2 Related Work

2.1 Lightweight Simulators for Navigation

Lightweight navigation simulators have a long history in robotics research. Early general-purpose examples include Stage (Vaughan, 2008) and ARGoS (Pinciroli et al., 2012), while a more recent example is PyRoboSim (pyrobosim Contributors, 2026). These simulators support multi-robot simulation in planar environments with simplified physics and rendering. More recent simulators increasingly target specific navigation tasks. For social navigation, representative examples include CrowdNav (Chen et al., 2019), SocialGym 2.0 (Chandra et al., 2024), and SocNavBench (Biswas et al., 2022), which focus on simulating multiple agents with social behaviors in shared spaces. RL requirements have also motivated simulators such as VMAS (Bettini et al., 2024) and JaxNav (Rutherford et al., 2024), which integrate with RL libraries to accelerate the training process. Finally, MVSim (Blanco-Claraco et al., 2023) supports mobile-robot and multi-vehicle simulation, while GPUDrive (Kazemkhani et al., 2025) and Waymax (Gulino et al., 2023) focus on autonomous driving. IR-SIM is a general-purpose lightweight simulator that supports a wide range of navigation tasks. Its advantages are the rapid construction of diverse, reusable scenarios and efficient execution on accessible platforms. Table 1 provides an overview of representative lightweight simulators in navigation research.

2.2 Large Language Models for Simulation

Recent work has explored the use of LLMs in generative robotic simulation, including generating simulation tasks (Wang et al., 2024a), training supervision (Wang et al., 2024b), and three-dimensional environments (Yang et al., 2024), as well as adapting environment configurations for RL (Zala et al., 2024). Another approach is for LLM agents to interact with existing software through structured tool interfaces or generated simulator-specific code. These workflows can use the Model Context Protocol (MCP) (Model Context Protocol Contributors, 2026) and agent skills (Anthropic, 2026a). In this work, IR-SIM provides agent skills to guide LLM agents in constructing and modifying declarative navigation scenarios, which reduces the amount of code that must be generated and maintained and improves the success rate of scenario generation.

2.3 Navigation Benchmarking

Navigation benchmarking requires diverse, reproducible scenarios with customizable behaviors, together with standardized evaluation metrics (Perille et al., 2020; Francis et al., 2025). Most benchmarks build on existing high-fidelity simulators for dynamic environments (Nair et al., 2022), social navigation (Biswas et al., 2022), and embodied navigation (Qiao et al., 2025). The difficulty of constructing scenarios and the need for high-fidelity simulators limit their broader adoption. In this work, IR-SIM provides reproducible scenarios that incorporate social behavior algorithms such as Optimal Reciprocal Collision Avoidance (ORCA) (van den Berg et al., 2011) and the Social Force Model (SFM) (Helbing & Molnár, 1995), together with evaluation metrics and protocols for benchmarking navigation algorithms.

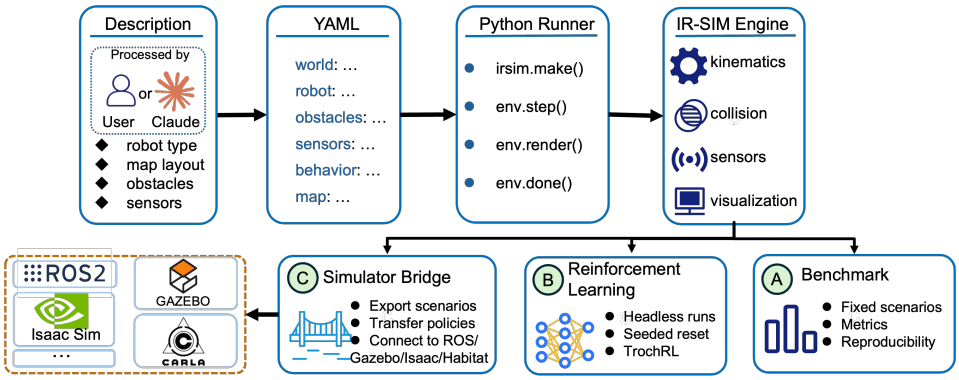


Figure 2: System overview of IR-SIM and skill-native workflows.

3 Simulator Design

3.1 System Overview

Fig. 2 summarizes the IR-SIM workflow from scenario description to downstream use. The left side shows scenario authoring. A user can either write the YAML file directly according to their needs or provide a natural-language scenario prompt, which is translated by an LLM agent, such as Codex (OpenAI, 2026) or Claude Code (Anthropic, 2026b), into a structured YAML scenario. The Python runner then parses the YAML file and instantiates the world, robots, behaviors, and sensors in the simulation loop. The physics engine next advances the simulation, and the Python library Matplotlib visualizes the scenario in real time. This design decouples scenario description, physics updates, and rendering, resulting in a stable simulator runner and flexible deployment options. The core simulator modules are implemented in Python, allowing IR-SIM to run directly across platforms including Windows, macOS, and Ubuntu. Finally, the Python API exposes functions for accepting control inputs, extracting state and sensor information, and generating demonstration files.

The bottom of Fig. 2 illustrates the main downstream applications, including navigation learning, benchmarking, and bridging to external simulators or real-world robots. To support RL-based navigation approaches, IR-SIM provides a Gym (Brockman et al., 2016) interface and wrappers for RL libraries such as TorchRL (Bou et al., 2024). Second, scenario artifacts with seeded randomization and built-in behavior configurations enable IR-SIM to run reproducible benchmarks and evaluate different navigation algorithms across diverse scenario families. Finally, users can use the flexible API to transfer scenarios to external simulators, e.g., Isaac Sim (NVIDIA, 2025), CARLA (Dosovitskiy et al., 2017), and Habitat-Sim (Savva et al., 2019), where the algorithm can be further validated in high-fidelity simulation and realistic environments. Details of these extensions can be found in Sec. F.

3.2 Simulation Engine

Physics engine: The physics engine of IR-SIM includes three main components: collision detection, kinematic state updates, and optional contact dynamics, which are executed at each time step. To achieve efficient execution that can run on accessible platforms, we have two main design choices. First, we only use two libraries to implement: Shapely (Gillies et al., 2025) for geometric queries and NumPy (Harris et al., 2020) for numerical operations. Specifically, Shapely is a widely used open-source Python package built on the GEOS geometry library for manipulating and analyzing planar geometric objects. Second, for the most computationally expensive LiDAR simulation, we implement a batched ray-casting by using NumPy and caches static geometry to reduce repeated geometric computation. These two features bring the strong scalability of IR-SIM, which can run hundreds of robots in real time on a single CPU core. Section 4.1 evaluates the execution cost of the resulting implementation.

The kinematic state update, implemented in NumPy, advances objects in discrete time according to their configured kinematic models, using either user-specified control inputs or actions generated by built-in behaviors. Commonly used kinematic models for mobile robots are supported, including

differential drive, omnidirectional, omnidirectional angular, and Ackermann. The optional contact dynamics module is also implemented in NumPy and resolves planar contacts using position-based constraint projection (Müller et al., 2007; 2020). It corrects overlaps using translations and rotations weighted by mass and inertia, updates object velocities, and estimates contact forces while accounting for friction. This allows for more realistic simulations of scenarios where robots or agents may collide or interact with each other or with obstacles in the environment.

Rendering engine: Following the lightweight design principle, IR-SIM implements default rendering with the Python library Matplotlib (Hunter, 2007) for simple 2D visualization. Additionally, keyboard and mouse controls are implemented through Matplotlib's event-handling system, allowing users to interact with the simulation in real time. For applications requiring high-fidelity rendering, such as visual sensor simulation, IR-SIM provides a bridge to external simulators, such as Isaac Sim (NVIDIA, 2025), CARLA (Dosovitskiy et al., 2017), and Habitat-Sim (Savva et al., 2019).

3.3 Scenario Representation

YAML configuration: In IR-SIM, to make scenarios easy to read and modify, all the components of a scenario are represented in a single human-readable YAML configuration file, which is parsed by the simulator to instantiate the world, robots, obstacles, behaviors, and sensors. The main YAML blocks are `world`, `robot`, `obstacle`, and `gui`; object configurations compose built-in geometry, kinematics, `behavior`, and `sensors`, while custom controllers and task logic can be implemented in Python. Each block contains relevant parameters and settings for the corresponding component. Additionally, to make the scenario more flexible and diverse, IR-SIM supports map specifications from images, where the map image is transformed into an occupancy grid map. Black pixels represent occupied cells and white pixels represent free cells. Such a grid map can be used for the development and evaluation of path planning algorithms, such as A* search (Hart et al., 1968) and Rapidly-exploring Random Trees (RRT) (LaValle, 1998).

Behavior modules: A scenario should not only contain robots and obstacles but also specify how they move and interact. IR-SIM provides behavior modules that define motion policies for each object or group, and the behavior parameters can also be configured directly through the YAML configuration file. The built-in social behaviors include Reciprocal Velocity Obstacles (RVO) (van den Berg et al., 2008), ORCA (van den Berg et al., 2011), SFM (Helbing & Molnár, 1995), and a simple goal-directed behavior called `dash`, which are commonly used in social navigation research to simulate pedestrian or dynamic agent behavior. To support diverse behaviors, IR-SIM also provides a registry that lets users define custom behaviors according to their needs and reference them in the YAML file without changing the simulator code interface.

3.4 LLM Authoring and Agent Skills

LLMs can generate code and follow natural-language instructions, which reduces the effort of scenario construction in IR-SIM and other similar tools. However, two challenges limit an unaided agent in IR-SIM: the API surface, which it may know only partly, and the runtime semantics of the built-in behaviors, which are not visible in the YAML. The agent otherwise learns these semantics only by running the scene. The IR-SIM agent skills address both challenges through a versioned package of task-focused skills for coding agents that supplies the verified API together with tested behavior parameters.

The IR-SIM package contains five skills. `irsim-scenario` authors, validates, and modifies YAML scenes through the blocks, it follows a YAML-first rule and carries reference sheets for the schema. `irsim-runner` handles the execution boundary through the simple Python loop. It also provides a verified table of accessors for reading the scene state and runner templates, including one that records every pose, the static geometry, and the sensor parameters in a trace file. `irsim-algorithm` exposes the path planners and the registration of custom behaviors and kinematics; `irsim-benchmark` structures fair comparisons through shared scenarios, seeds, episode counts, horizons, robot densities, and canonical metrics, and `irsim-dev` supports development of the library itself, including tests, linting, documentation, and releases.

The workflow from natural language to a scenario is illustrated in Fig. 3. The user provides the description shown in Fig. 3(a); the scenario skill extracts the scene layout, the number of robots, the

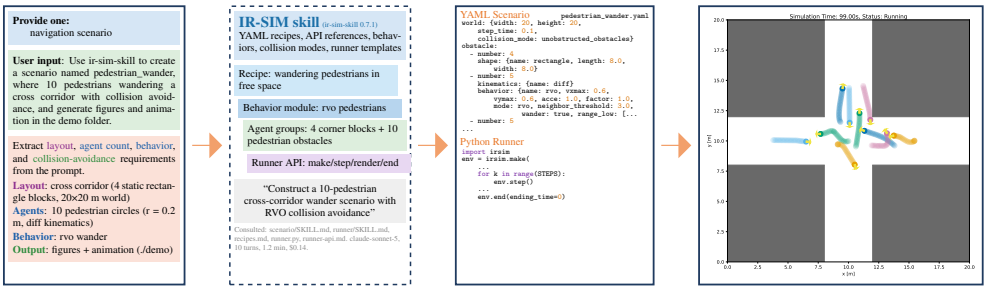


Figure 3: Construction of the `pedestrian wander` scenario. The pipeline contains four stages: (a) prompting and extracting scenario elements, (b) retrieving IR-SIM skill templates and API patterns, (c) assembling YAML and Python artifacts, and (d) rendering rollout snapshots and saving the demonstration animation.

behavior type, and the collision-avoidance requirement, retrieves the matching recipe and reference entries, and assembles the YAML scene and the Python runner; the runner skill renders rollout snapshots and saves the requested animation. The pipeline takes a few minutes from the prompt to the animated scenario, and the scene can be modified with a further prompt, after which the YAML file and the runner are updated. Section 4.2 quantifies the effect of the skills on agent-assisted scenario construction compared with other lightweight simulators in terms of scenario validity, execution success, and agent time.

4 Experiments

In this section, we conduct experiments to evaluate the performance of IR-SIM by addressing the following questions: (1) How efficiently can IR-SIM execute navigation scenarios? (2) Can LLM agents reliably construct navigation scenarios in IR-SIM, and can the provided agent skills further reduce the construction time and improve the generation success rate? (3) Can IR-SIM support navigation benchmarking and learning with randomized and reproducible evaluation? All the experiments are conducted on a desktop computer with an AMD Ryzen 9 7950X3D CPU and an NVIDIA GeForce RTX 4090 GPU with 24 GB of memory.

4.1 Execution Efficiency

We evaluate the execution efficiency of IR-SIM against four recent lightweight simulators: VMAS (Bettini et al., 2024), JaxNav (Rutherford et al., 2024), PyRoboSim (pyrobosim Contributors, 2026), and MVSim (Blanco-Claraco et al., 2023), on one shared scenario; MVSim, which cannot be stepped from Python, is timed through its C++ API. The scenarios include $N \in \{10, 100, 500, 1000\}$ differential-drive robots that replay the same recorded actions in a walled arena with unit-sized obstacles ($20\,\mathrm{m} \times 20\,\mathrm{m}$ up to $N = 100$, enlarged at constant density beyond). We run each scenario once with a 64-beam LiDAR on every robot, for which LiDAR simulation is computationally intensive, and once without sensors. Each simulation step comprises kinematic integration, collision handling, sensing, and observation construction. We evaluate the step latency of a single world on a CPU core and the batched throughput of many worlds on a GPU, following the agent steps per second (ASPS) protocol of GPUDrive (Kazemkhani et al., 2025). The results are shown in Figure 4.

For step latency in a single world, as shown in Figure 4 (a)(b), IR-SIM has the lowest LiDAR step latency for large teams, with $178\,\mathrm{ms}$ for 1000 robots ($1.6\times$ faster than MVSim, $27\times$ faster than JaxNav, $71\times$ faster than VMAS). It is on par with MVSim at 100 robots, and the compiled C++ engine of MVSim is faster for smaller teams. Key to this performance is the batched LiDAR scan, implemented in pure NumPy, the complete step latency with LiDAR grows only $11\times$ when the number of robots grows from 100 to 1000, a smaller increase than for MVSim ($17\times$), and others. Without LiDAR sensors, PyRoboSim, which also uses Shapely for collision detection, and the compiled MVSim perform comparably to IR-SIM with 10 robots, but IR-SIM achieves the lowest step latency of $59\,\mathrm{ms}$ for 1000 robots and scales better than others from 100 to 1000 robots, owing to its vectorized collision detection and kinematic integration in NumPy. With the optional Numba-compiled LiDAR kernel (Lam et al., 2015), IR-SIM further reduces the step latency with

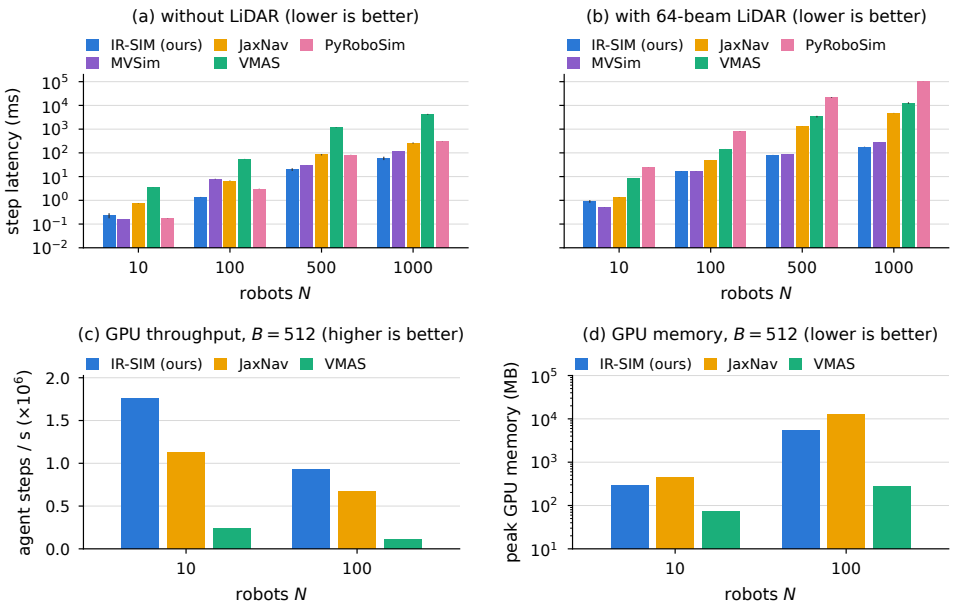


Figure 4: Execution efficiency on the shared scenario. (a), (b): single-core CPU step latency without and with a 64-beam LiDAR for N robots. (c): batched throughput on an RTX 4090 with $B = 512$ worlds per call; (d): peak GPU memory of the same runs.

LiDAR to 3.5 ms for 100 robots and 79 ms for 1000 robots, 4.8× and 2.2× faster than the NumPy version (Appendix B.1).

We also evaluate the batched throughput of 512 worlds on a GPU and test how many agent steps can be processed in 1 s, compared with JaxNav and VMAS, which are designed for GPU execution (PyRoboSim and MVSim have no GPU mode). Specifically, the GPU version of IR-SIM is implemented in PyTorch. As shown in Figure 4 (c)(d), IR-SIM achieves the highest ASPS: 1.76 and 0.93 million for 10 and 100 robots per world, respectively, against 1.13 and 0.67 million for JaxNav and 0.23 and 0.11 million for VMAS, with LiDAR at every step. IR-SIM has the second-lowest device memory usage, 5,364 MB at 512 worlds per call with 100 robots per world, while JaxNav and VMAS use 12,701 MB and 277 MB, respectively. In summary, IR-SIM is the most efficient simulator for large-scale multi-robot scenarios, with the lowest step latency and highest batched throughput, while maintaining a low memory footprint, owing to its NumPy- and Shapely-based implementation and batched LiDAR scanning. Further details about the scenario, timing boundaries, and step budgets are provided in Appendix B.1.

4.2 Scenario Construction

In this experiment, we measure how reliably and quickly an LLM agent builds navigation scenarios using IR-SIM, compare its performance with that on other lightweight simulators, and evaluate the effect of IR-SIM's agent skills described in Section 3. Thirteen simulator-independent construction briefs span five families: (A) static obstacles, (B) pedestrian encounters, (C) multi-robot navigation, (E) rich scenes with many robots and pedestrians moving according to standard behaviors, seeded crowds and layouts, and sensors with a limited field of view, and (H) further mixed scenes; all thirteen lie in the capability intersection of IR-SIM, VMAS, and PyRoboSim (JaxNav is evaluated on six compatible briefs), require no task-specific control code, and were solved by hand beforehand (Appendix C.1). Figure 5 shows twelve construction briefs with the scenes built from them.

We use Claude Code with `claude-opus-5-5` to receive the brief and write the deliverable once without executing anything. The agent attempts every supported brief-condition pair three times. Following the staged validation of GenSim (Wang et al., 2024a), an independent validator grades each deliverable as *executable* (it runs headlessly and writes a rollout trace) and *valid* (all acceptance checks of the brief pass, inspected through the simulator's API and the recorded rollout). Specification accuracy is the mean fraction of acceptance checks passed. Turns, Time, Cost, and Lines

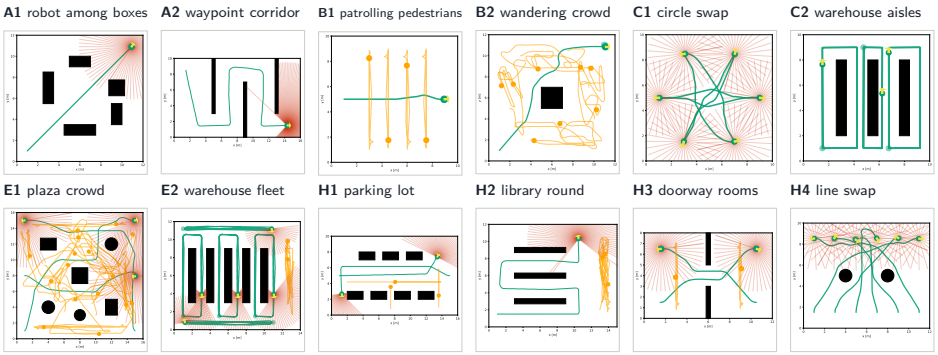


Figure 5: Twelve of the thirteen construction briefs and the scenes the IR-SIM + skills agent built from them in the single-shot protocol.

Table 2: Single-shot scenario construction with `claude-opus-5-5` and `claude-sonnet-5`. †JaxNav covers A1, A2, C1, C2, E3 and H4 only (18 attempts). ‡Human IR-SIM submissions on the same briefs, graded with the same validator.

Simulator	Executable (%)	Valid (%)	Spec. acc. (%)	Turns	Time (s)	Cost (USD)	Lines Python	Lines YAML
`claude-opus-5-5`: construction briefs A1, A2, B1, B2, C1, C2, E1, E2, E3, H1, H2, H3, H4								
IR-SIM + skills	**100.0**	94.9	**99.5**	**11.4**	**98**	**0.51**	**78**	**35**
IR-SIM	89.7	87.2	87.2	24.3	204	1.04	179	68
VMAS	97.4	**97.4**	97.4	23.2	238	1.02	316	0
PyRoboSim	**100.0**	89.7	99.1	24.4	212	1.00	293	9
JaxNav†	88.9	88.9	88.9	12.1	229	1.18	258	0
`claude-sonnet-5`: construction briefs A1, A2, B1, B2, C1, C2, E1, E2, E3, H1, H2, H3, H4								
IR-SIM + skills	**100.0**	**94.9**	**99.7**	**9.0**	**41**	**0.17**	**64**	**41**
IR-SIM	94.9	61.5	92.7	15.8	92	0.25	126	58
VMAS	71.8	64.1	71.3	19.1	137	0.34	222	0
PyRoboSim	**100.0**	87.2	98.8	22.2	127	0.38	210	8
JaxNav†	72.2	72.2	72.2	13.6	172	0.55	206	0
Humans: construction brief A1								
IR-SIM + Humans‡ ($n = 5$)	100.0	60.0	91.7	–	1194	–	69	34

report means over all attempts: documentation and file-tool turns, agent wall-clock time excluding validation, API list-price cost, and non-comment Python/YAML lines, respectively.

The results are listed in Table 2. IR-SIM with skills achieves the lowest generation time, API cost, and deliverable size on the construction briefs, with 95% validity. On these briefs, skills reduce mean agent time from 204 to 98 s and cost from \$1.04 to \$0.51 per attempt, while validity rises from 87% to 95%. Compared to human authors on A1, the agent is 19.2× faster, while achieving higher validity (3/3 versus 3/5). With the smaller `claude-sonnet-5` (second block of Table 2), the conditions also differ in validity: IR-SIM with skills is valid in 37 of 39 attempts at 41 s and \$0.17, against 62% for plain IR-SIM, 64% for VMAS, 87% for PyRoboSim, and 72% for JaxNav, at 92 to 172 s per attempt; attempts with code-generating interfaces fail because of runtime errors and controllers that miss goals, whereas the skills keep the declarative scene correct. IR-SIM with skills achieves the shortest generation time, lowest cost, and fewest lines of code among the compared agent conditions, with approximately 95% validity across both evaluated models and the highest validity with smaller model `claude-sonnet-5`. Detailed test results are in Appendix C.6.

4.3 Navigation Benchmarking and Learning

Multi-robot navigation benchmarking: We evaluate IR-SIM's support for reproducible navigation benchmarking by comparing eight reactive, optimization-based, and learning-based controllers under a shared scenario specification and evaluation protocol: ORCA (van den Berg et al., 2011), VO-polytope (Huang et al., 2023), AVOCADO (Martinez-Baselga et al., 2025), the RDA planner (Han et al., 2023), and the learned policies SARL (Chen et al., 2019), CADRL (Chen et al.,

Table 3: Example IR-SIM benchmark for multi-robot collision avoidance.

Method	5 robots				10 robots			
	Succ.↑	Coll.↓	Tout.↓	Time (s)↓	Succ.↑	Coll.↓	Tout.↓	Time (s)↓
ORCA (van den Berg et al., 2011)	**100**	**0**	**0**	5.05 ± 2.47	**90**	1	9	5.41 ± 3.04
VO-polytope (Huang et al., 2023)	83	17	**0**	$\mathbf{4.74} \pm 2.28$	45	55	**0**	$\mathbf{4.31} \pm 2.44$
AVOCADO (Martinez-Baselga et al., 2025)	**100**	**0**	**0**	5.23 ± 2.38	86	**0**	14	5.50 ± 2.75
RDA (Han et al., 2023)	20	**0**	80	5.03 ± 3.48	0	**0**	100	5.94 ± 6.04
SARL (Chen et al., 2019)	41	**0**	59	10.90 ± 7.80	26	11	63	10.96 ± 8.41
CADRL (Chen et al., 2017)	9	6	85	8.79 ± 6.87	0	40	60	8.93 ± 7.30
LSTM-RL (Everett et al., 2018)	39	13	48	9.97 ± 6.92	0	36	64	12.69 ± 11.02
RL-RVO (Han et al., 2022)	89	6	5	5.82 ± 3.46	73	20	7	6.87 ± 4.19

Table 4: Training CrowdNav's SARL policy: CrowdNav's original implementation and simulator versus IR-SIM with a batched TorchRL implementation of the same recipe.

Method	Simulator	Hardware	Worlds	5 humans				10 humans			
				Sim. (ms)	Net. (ms)	Time (h)	Speed-up	Sim. (ms)	Net. (ms)	Time (h)	Speed-up
CrowdNav	`CrowdSim`	CPU	1	6.86	27.77	6.99	1.0×	31.17	29.43	13.11	1.0×
CrowdNav	IR-SIM	CPU	1	2.50	27.00	6.13	1.1×	4.01	28.17	7.78	1.7×
Batched IR-SIM + TorchRL	IR-SIM	CPU	16	0.43	0.47	0.76	9.2×	0.72	0.76	1.27	10.3×
Batched IR-SIM + TorchRL	IR-SIM	CPU + GPU	16	**0.41**	**0.11**	**0.51**	**13.7**×	**0.70**	**0.12**	**0.70**	**18.8**×

2017), LSTM-RL (Everett et al., 2018), and RL-RVO (Han et al., 2022). The task comprises 5 or 10 differential-drive robots with random convex polygonal footprints in a $10\,\mathrm{m} \times 10\,\mathrm{m}$ world. Episode success requires every robot to reach its goal without collision within 500 steps. The results are listed in Table 3, which reports episode-level success, collision, and timeout rates (%) and navigation time of robots that reach their goals (mean $\pm$ standard deviation) for 100 randomized episodes per setting with a fixed seed. Specially, the CADRL, SARL, and LSTM-RL policies focus on social navigation and thus has lower success rates than the other multi-robot controllers, which align with our intuition. The experiment demonstrates that IR-SIM can combine shared scenario generation and metric collection to benchmark heterogeneous navigation controllers reproducibly. Controller settings, geometry representations, and policy provenance are detailed in Appendix D.

Reinforcement learning: We implement the classic RL-based social navigation approach CrowdNav (Chen et al., 2019) in our IR-SIM simulation with a TorchRL adapter to demonstrate the capability and efficiency of IR-SIM as a training backend for learning-based navigation policies in multi-agent scenarios. Specifically, we compare 1) the original CrowdNav algorithm (Chen et al., 2019) and 2) the original CrowdNav algorithm with IR-SIM backend and a TorchRL adapter (our framework). The results are listed in Table 4. We can see that the original CrowdNav algorithm with IR-SIM backend has 1.1 times the training speed, which time cost is mainly on the single world simulation step. While, by using our IR-SIM and TorchRL adapter, where the world simulation is batched, the training time is reduced significantly from $6.99\,\mathrm{h}$ to $0.76\,\mathrm{h}$ ($9.2\times$ speedup), which could further be reduced to $0.51\,\mathrm{h}$ ($13.7\times$ speedup) with GPU acceleration, while achieving the same policy performance.

5 Conclusion and Future Work

Conclusion: We presented IR-SIM, a lightweight declarative simulator for navigation learning and benchmarking, which enables users to rapidly construct diverse, reusable navigation scenarios and execute them efficiently on accessible platforms. Experiments demonstrate faster execution than the compared lightweight simulators in the evaluated LiDAR scenarios. With Claude agents, the provided agent skills reduce scenario-generation time and improve scenario validity compared with IR-SIM without skills. We further demonstrate that IR-SIM supports reproducible navigation benchmarking and that using IR-SIM with TorchRL reduces the training time for SARL, a classic RL-based social navigation approach, while maintaining the baseline success rate in the evaluated setting. We will open-source the simulator to enable further research.

Future work: IR-SIM is currently limited to 2D navigation and does not directly support 3D navigation or general manipulation tasks. It also lacks the complex robot dynamics required for legged locomotion and aerial vehicles, which limits its applicability to these settings. Future work will focus on extending IR-SIM to 3D navigation and more complex tasks, with richer robot dynamics and behaviors.

AI USE STATEMENT

An AI assistant was used to polish the writing of the abstract and introduction, and to check for grammar and spelling errors in the main text. AI assistants also provided feedback on experimental design and methodology, assisted with the implementation and debugging of experiment and analysis code, and supported the interpretation of experimental results.

REPRODUCIBILITY STATEMENT

Appendix A provides the kinematic and sensor models. Appendix B documents the CPU and GPU execution measurements, and Appendix C provides the scenario-construction protocol, validation checks, and complete results. Appendix D lists the scenario, evaluation protocol, and controller settings of the social navigation benchmark. We submit the core IR-SIM codebase as supplementary material and will open-source all the IR-SIM code, TorchRL wrappers, and bridges to external simulators upon acceptance of the paper.

REFERENCES

Anthropic. Agent skills overview. Online documentation, 2026a. URL https://platform.claude.com/docs/en/agents-and-tools/agent-skills/overview. Documentation snapshot, accessed 23 September 2026.

Anthropic. Claude Code: Overview. Online documentation, 2026b. URL https://code.claude.com/docs/en/overview. Documentation snapshot, accessed 23 September 2026.

Avidbots. Flatland: A 2D robot simulator. Software and documentation, 2026. URL https://github.com/avidbots/flatland. Repository snapshot 6601c65, accessed 22 September 2026.

Matteo Bettini, Ryan Kortvelesy, Jan Blumenkamp, and Amanda Prorok. VMAS: A vectorized multi-agent simulator for collective robot learning. In *Distributed Autonomous Robotic Systems*, volume 28 of *Springer Proceedings in Advanced Robotics*, pp. 42–56. Springer, 2024. doi: 10.1007/978-3-031-51497-5_4. URL https://link.springer.com/chapter/10.1007/978-3-031-51497-5_4. DARS 2022; proceedings published in 2024.

Abhijat Biswas, Allan Wang, Gustavo Silvera, Aaron Steinfeld, and Henny Admoni. SocNavBench: A grounded simulation testing framework for evaluating social navigation. *ACM Transactions on Human-Robot Interaction*, 11(3):1–24, 2022. doi: 10.1145/3476413. Article 26.

José-Luis Blanco-Claraco, Borys Tymchenko, Francisco José Mañas-Alvarez, Fernando Cañadas-Aránega, Ángel López-Gázquez, and José Carlos Moreno. MultiVehicle Simulator (MVSim): Lightweight dynamics simulator for multiagents and mobile robotics research. *SoftwareX*, 23: 101443, 2023. doi: 10.1016/j.softx.2023.101443. URL https://www.sciencedirect.com/science/article/pii/S2352711023001395.

Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. TorchRL: A data-driven decision-making library for PyTorch. In *International Conference on Learning Representations*, volume 2024, pp. 1778–1811, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/07bc8125400bf4b140c332010756bd9b-Abstract-Conference.html.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016. URL https://arxiv.org/abs/1606.01540.

Rohan Chandra, Zayne Sprague, and Joydeep Biswas. SOCIALGYM 2.0: Simulator for multi-robot learning and navigation in shared human spaces. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 23778–23780, 2024. doi: 10.1609/aaai.v38i21.30562. URL https://ojs.aaai.org/index.php/AAAI/article/view/30562. AAAI Demonstration Track, issue 21.

Changan Chen, Yuejiang Liu, Sven Kreiss, and Alexandre Alahi. Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. In *2019 International Conference on Robotics and Automation*, pp. 6015–6022. IEEE, 2019. doi: 10.1109/ICRA.2019.8794134.

Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P. How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation*, pp. 285–292. IEEE, 2017. doi: 10.1109/ICRA.2017.7989037.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg (eds.), *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pp. 1–16. PMLR, 13–15 Nov 2017. URL https://proceedings.mlr.press/v78/dosovitskiy17a.html.

Michael Everett, Yu Fan Chen, and Jonathan P. How. Motion planning among dynamic, decision-making agents with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3052–3059. IEEE, 2018. doi: 10.1109/IROS.2018.8593871.

Tingxiang Fan, Pinxin Long, Wenxi Liu, and Jia Pan. Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios. *The International Journal of Robotics Research*, 39(7):856–892, 2020. doi: 10.1177/0278364920916531.

Anthony Francis, Claudia Pérez-D'Arpino, Chengshu Li, Fei Xia, Alexandre Alahi, Rachid Alami, Aniket Bera, Abhijat Biswas, Joydeep Biswas, Rohan Chandra, Hao-Tien Lewis Chiang, Michael Everett, Sehoon Ha, Justin Hart, Jonathan P. How, Haresh Karnan, Tsang-Wei Edward Lee, Luis J. Manso, Reuth Mirsky, Sören Pirk, Phani Teja Singamaneni, Peter Stone, Ada V. Taylor, Peter Trautman, Nathan Tsoi, Marynel Vázquez, Xuesu Xiao, Peng Xu, Naoki Yokoyama, Alexander Toshev, and Roberto Martín-Martín. Principles and guidelines for evaluating social robot navigation algorithms. *ACM Transactions on Human-Robot Interaction*, 14(2):1–65, 2025. doi: 10.1145/3700599. Article 34.

Sean Gillies, Casper van der Wel, Joris Van den Bossche, Mike W. Taves, Joshua Arnott, Brendan C. Ward, and others. Shapely, may 2025. URL https://github.com/shapely/shapely. Version 2.1.1; BSD-3-Clause license.

Cole Gulino, Justin Fu, Wenjie Luo, George Tucker, Eli Bronstein, Yiren Lu, Jean Harb, Xinlei Pan, Yan Wang, Xiangyu Chen, John D. Co-Reyes, Rishabh Agarwal, Rebecca Roelofs, Yao Lu, Nico Montali, Paul Mougin, Zoey Yang, Brandyn White, Aleksandra Faust, Rowan McAllister, Dragomir Anguelov, and Benjamin Sapp. Waymax: An accelerated, data-driven simulator for large-scale autonomous driving research. In *Advances in Neural Information Processing Systems*, volume 36, pp. 7730–7742, 2023. doi: 10.52202/075280-0339. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1838feeb71c4b4ea524d0df2f7074245-Abstract.html.

Ruihua Han, Shengduo Chen, Shuaijun Wang, Zeqing Zhang, Rui Gao, Qi Hao, and Jia Pan. Reinforcement learned distributed multi-robot navigation with reciprocal velocity obstacle shaped rewards. *IEEE Robotics and Automation Letters*, 7(3):5896–5903, 2022. doi: 10.1109/LRA.2022.3161699.

Ruihua Han, Shuai Wang, Shuaijun Wang, Zeqing Zhang, Qianru Zhang, Yonina C. Eldar, Qi Hao, and Jia Pan. RDA: An accelerated collision free motion planner for autonomous navigation in cluttered environments. *IEEE Robotics and Automation Letters*, 8(3):1715–1722, 2023. doi: 10.1109/LRA.2023.3242138.

Ruihua Han, Shuai Wang, Shuaijun Wang, Zeqing Zhang, Jianjun Chen, Shijie Lin, Chengyang Li, Chengzhong Xu, Yonina C Eldar, Qi Hao, and Jia Pan. NeuPAN: Direct point robot navigation with end-to-end model-based learning. *IEEE Transactions on Robotics*, 41:2804–2824, 2025. doi: 10.1109/TRO.2025.3554252.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020. doi: 10.1038/s41586-020-2649-2. URL https://www.nature.com/articles/s41586-020-2649-2.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136.

Dirk Helbing and Peter Molnár. Social force model for pedestrian dynamics. *Physical Review E*, 51 (5):4282–4286, 1995. doi: 10.1103/PhysRevE.51.4282.

Jihao Huang, Jun Zeng, Xuemin Chi, Koushil Sreenath, Zhitao Liu, and Hongye Su. Velocity obstacle for polytopic collision avoidance for distributed multi-robot systems. *IEEE Robotics and Automation Letters*, 8(6):3502–3509, 2023. doi: 10.1109/LRA.2023.3269295.

J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3): 90–95, 2007. doi: 10.1109/MCSE.2007.55.

Saman Kazemkhani, Aarav Pandya, Daphne Cornelisse, Brennan Shacklett, and Eugene Vinitsky. GPUDrive: Data-driven, multi-agent driving simulation at 1 million FPS. In *International Conference on Learning Representations*, volume 2025, pp. 19320–19336, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/3107ddd4209e5f93c0371425763041a3-Abstract-Conference.html.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):1–14, 2023. doi: 10.1145/3592433.

Nathan P. Koenig and Andrew Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pp. 2149–2154. IEEE, 2004. doi: 10.1109/IROS.2004.1389727.

Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A LLVM-based Python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, pp. 1–6. ACM, 2015. doi: 10.1145/2833157.2833162.

Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Computer Science Department, Iowa State University, 1998. URL https://lavalle.pl/rrtpubs.html.

Guoliang Li, Ruihua Han, Shuai Wang, Fei Gao, Yonina C Eldar, and Chengzhong Xu. Edge accelerated robot navigation with collaborative motion planning. *IEEE/ASME Transactions on Mechatronics*, 30(2):1166–1178, 2025. doi: 10.1109/TMECH.2024.3419436.

Diego Martinez-Baselga, Eduardo Sebastián, Eduardo Montijano, Luis Riazuelo, Carlos Sagüés, and Luis Montano. AVOCADO: Adaptive optimal collision avoidance driven by opinion. *IEEE Transactions on Robotics*, 41:2495–2511, 2025. doi: 10.1109/TRO.2025.3552350.

Model Context Protocol Contributors. Model Context Protocol: Architecture overview. Online documentation, 2026. URL https://modelcontextprotocol.io/docs/2026-07-28/learn/architecture. Documentation version 2026-07-28; accessed 23 September 2026.

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007. doi: 10.1016/j.jvcir.2007.01.005.

Matthias Müller, Miles Macklin, Nuttapong Chentanez, Stefan Jeschke, and Tae-Yong Kim. Detailed rigid body simulation with extended position based dynamics. *Computer Graphics Forum*, 39(8):101–112, 2020. doi: 10.1111/cgf.14105.

Anirudh Nair, Fulin Jiang, Kang Hou, Zifan Xu, Shuozhe Li, Xuesu Xiao, and Peter Stone. DynaBARN: Benchmarking metric ground navigation in dynamic environments. In *2022 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pp. 347–352. IEEE, 2022. doi: 10.1109/SSRR56537.2022.10018758.

NVIDIA. Isaac Sim, 2025. URL https://github.com/isaac-sim/IsaacSim/releases/tag/v5.1.0. Version 5.1.0; Apache-2.0 license.

OpenAI. Codex CLI. Online documentation, 2026. URL https://learn.chatgpt.com/docs/codex/cli. Documentation snapshot, accessed 25 September 2026.

Daniel Perille, Abigail Truong, Xuesu Xiao, and Peter Stone. Benchmarking metric ground navigation. In *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pp. 116–121. IEEE, 2020. doi: 10.1109/SSRR50563.2020.9292572.

Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Mauro Birattari, Luca M. Gambardella, and Marco Dorigo. ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, 2012. doi: 10.1007/s11721-012-0072-5.

pyrobosim Contributors. pyrobosim: A ROS 2 enabled mobile robot simulator for behavior prototyping. Software and documentation, 2026. URL https://github.com/sea-bass/pyrobosim. Repository snapshot e904bcb, accessed 22 September 2026.

Yanyuan Qiao, Haodong Hong, Wenqi Lyu, Dong An, Siqi Zhang, Yutong Xie, Xinyu Wang, and Qi Wu. NavBench: Probing multimodal large language models for embodied navigation. In *Advances in Neural Information Processing Systems*, volume 38, pp. 95365–95385. Curran Associates, Inc., 2025. doi: 10.52202/085713-3188. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/8932bbe603280f51d425e2781cd6ea6e-Abstract-Conference.html.

Santhosh Kumar Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John M. Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X. Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-Matterport 3D Dataset (HM3D): 1000 large-scale 3D environments for embodied AI. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/34173cb38f07f89ddbebc2ac9128303f-Abstract-round2.html.

Alex Rutherford, Michael Beukman, Timon Willi, Bruno Lacerda, Nick Hawes, and Jakob Foerster. No regrets: Investigating and improving regret approximations for curriculum discovery. In *Advances in Neural Information Processing Systems*, volume 37, pp. 16071–16101. Curran Associates, Inc., 2024. doi: 10.52202/079017-0512. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/1d0ed12c3fda52f2c241a0cebcf739a6-Abstract-Conference.html.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied AI research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9339–9347, 2019. doi: 10.1109/ICCV.2019.00943.

Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *2008 IEEE International Conference on Robotics and Automation*, pp. 1928–1935. IEEE, 2008. doi: 10.1109/ROBOT.2008.4543489.

Jur van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics Research: The 14th International Symposium ISRR*, pp. 3–19. Springer, 2011. doi: 10.1007/978-3-642-19457-3_1.

Richard Vaughan. Massively multi-robot simulation in Stage. *Swarm Intelligence*, 2(2–4):189–208, 2008. doi: 10.1007/s11721-008-0014-4. URL https://link.springer.com/article/10.1007/s11721-008-0014-4.

Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. GenSim: Generating robotic simulation tasks via large language models. In *International Conference on Learning Representations*. OpenReview.net, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/143ea4a156ef64f32d4d905206cf32e1-Abstract-Conference.html.

Yufei Wang, Zhou Xian, Feng Chen, Tsun-Hsuan Wang, Yian Wang, Katerina Fragkiadaki, Zackory Erickson, David Held, and Chuang Gan. RoboGen: Towards unleashing infinite data for automated robot learning via generative simulation. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 51936–51983. PMLR, 21–27 Jul 2024b. URL https://proceedings.mlr.press/v235/wang24cc.html.

Zhefan Xu, Xinming Han, Haoyu Shen, Hanyu Jin, and Kenji Shimada. NavRL: Learning safe flight in dynamic environments. *IEEE Robotics and Automation Letters*, 10(4):3668–3675, 2025. doi: 10.1109/LRA.2025.3546069.

Zhefan Xu, Hanyu Jin, and Kenji Shimada. NavRL++: A system-level framework for improving sim-to-real transfer in reinforcement learning-based robot navigation. *arXiv preprint arXiv:2605.15559*, 2026. URL https://arxiv.org/abs/2605.15559.

Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, Chris Callison-Burch, Mark Yatskar, Aniruddha Kembhavi, and Christopher Clark. Holodeck: Language guided generation of 3D embodied AI environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16227–16237, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Yang_Holodeck_Language_Guided_Generation_of_3D_Embodied_AI_Environments_CVPR_2024_paper.html.

Abhay Zala, Jaemin Cho, Han Lin, Jaehong Yoon, and Mohit Bansal. EnvGen: Generating and adapting environments via LLMs for training embodied agents. In *Conference on Language Modeling*, 2024. URL https://envgen-llm.github.io/.

APPENDIX

CONTENTS

1 Introduction **1**

2 Related Work **3**

2.1 Lightweight Simulators for Navigation 3

2.2 Large Language Models for Simulation 3

2.3 Navigation Benchmarking 3

3 Simulator Design **4**

3.1 System Overview 4

3.2 Simulation Engine 4

3.3 Scenario Representation 5

3.4 LLM Authoring and Agent Skills 5

4 Experiments **6**

4.1 Execution Efficiency 6

4.2 Scenario Construction 7

4.3 Navigation Benchmarking and Learning 8

5 Conclusion and Future Work **9**

A Simulator Models and Implementation Details **16**

A.1 Differential-Drive Kinematics 16

A.2 Ackermann Kinematics 16

A.3 Omnidirectional Kinematics 17

A.4 Omnidirectional Kinematics with Yaw Control 17

A.5 Velocity Noise Model 17

A.6 LiDAR Simulation and Collision Checking 17

B Computational Efficiency **18**

B.1 Cross-Simulator Step-Time Measurements 18

B.2 GPU Execution and Batched Simulation 18

C Scenario Construction Study **21**

C.1 Briefs 21

C.2 Deliverable Contract and Validator 21

C.3 Agent Configuration and Budget 21

C.4 Reference Solutions 22

C.5 Coverage of Other Declarative Simulators 22

C.6 Complete Results 22
C.7 Human Reference 23

D Social Navigation Benchmark Protocol **23**

E Policy Training Protocol **24**

F Simulator Bridges and Robot Interfaces **25**
F.1 Habitat-Sim Map Conversion 25
F.2 Gaussian Splatting Visualization 25
F.3 CARLA Integration 25
F.4 Isaac Sim Integration 26
F.5 Physical Robot Interfaces 26

G Scenario Examples **26**

A Simulator Models and Implementation Details

This section summarizes representative kinematic and LiDAR models used by IR-SIM. Let Δt denote the simulation step time. The planar robot pose is written as $\mathbf{x} = [x, y, \theta]^\top$, where (x, y) is the robot position and θ is its heading. IR-SIM uses explicit discrete updates, so each object state is advanced from step k to $k + 1$ according to its configured kinematic model. Differential-drive, Ackermann, omnidirectional, and omnidirectional kinematics with yaw control are selected in the YAML scenario with `diff`, `acker`, `omni`, and `omni_angular`, respectively.

A.1 Differential-Drive Kinematics

For a differential-drive robot, the action is $\mathbf{u} = [v, \omega]^\top$, where v is the forward speed and ω is the angular speed. The continuous-time model is

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix}. \tag{1}$$

IR-SIM applies an Euler update,

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \begin{bmatrix} \cos\theta_k & 0 \\ \sin\theta_k & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_k \\ \omega_k \end{bmatrix}, \quad \theta_{k+1} \leftarrow \mathrm{wrap}_\pi(\theta_{k+1}), \tag{2}$$

where $\mathrm{wrap}_\pi(\cdot)$ wraps the heading angle to the range $(-\pi, \pi]$.

A.2 Ackermann Kinematics

For an Ackermann vehicle, IR-SIM stores the state as $\mathbf{x} = [x, y, \theta, \psi]^\top$, where ψ is the steering angle and L is the wheelbase. In the default steering mode, the action is $\mathbf{u} = [v, \delta]^\top$, where δ is the steering command stored for the next step. The pose update uses the current steering angle ψ_k:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \\ \psi_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \\ \psi_k \end{bmatrix} + \Delta t \begin{bmatrix} \cos\theta_k & 0 \\ \sin\theta_k & 0 \\ \tan\psi_k/L & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_k \\ \delta_k \end{bmatrix}, \quad \theta_{k+1} \leftarrow \mathrm{wrap}_\pi(\theta_{k+1}). \tag{3}$$

A.3 OMNIDIRECTIONAL KINEMATICS

IR-SIM also supports omnidirectional robots. For the `omni` model, the action is a body-frame velocity $\mathbf{u} = [v_f, v_l]^\top$, where v_f and v_l are forward and lateral speeds. The heading is preserved, and the world-frame velocity is obtained by rotating the body-frame velocity:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \Delta t \begin{bmatrix} \cos\theta_k & -\sin\theta_k \\ \sin\theta_k & \cos\theta_k \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_{f,k} \\ v_{l,k} \end{bmatrix}. \tag{4}$$

A.4 OMNIDIRECTIONAL KINEMATICS WITH YAW CONTROL

The `omni_angular` model extends the `omni` model with independent yaw control. The state is $\mathbf{x} = [x, y, \theta]^\top$, and the action is a body-frame velocity $\mathbf{u} = [v_f, v_l, \omega]^\top$, where v_f and v_l are the forward and lateral speeds and ω is the yaw rate. The continuous-time model is

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v_f \cos\theta - v_l \sin\theta \\ v_f \sin\theta + v_l \cos\theta \\ \omega \end{bmatrix}. \tag{5}$$

IR-SIM applies an Euler update,

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \begin{bmatrix} \cos\theta_k & -\sin\theta_k & 0 \\ \sin\theta_k & \cos\theta_k & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_{f,k} \\ v_{l,k} \\ \omega_k \end{bmatrix}, \quad \theta_{k+1} \leftarrow \mathrm{wrap}_\pi(\theta_{k+1}). \tag{6}$$

The translation is rotated by the current heading θ_k before the heading itself is updated. Unlike the `omni` model, whose heading stays fixed, the heading of an `omni_angular` robot is controlled independently of its travel direction, so the robot can translate in any direction while rotating.

A.5 VELOCITY NOISE MODEL

When velocity noise is enabled in the YAML scenario, IR-SIM perturbs the commanded velocity before applying the kinematic update. For differential-drive and Ackermann models, the noisy command is

$$\tilde{\mathbf{u}} = \mathbf{u} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \alpha_1 v^2 + \alpha_2 u_2^2 & 0 \\ 0 & \alpha_3 v^2 + \alpha_4 u_2^2 \end{bmatrix}\right), \tag{7}$$

where u_2 denotes the second action component, either angular speed or steering command depending on the kinematic model. For the `omni` and `omni_angular` models, IR-SIM applies independent zero-mean Gaussian noise to each body-frame velocity channel, with a constant variance that does not scale with the command: $\boldsymbol{\epsilon} \sim \mathcal{N}\big(\mathbf{0}, \mathrm{diag}(\alpha_f, \alpha_l)\big)$ for `omni` and $\boldsymbol{\epsilon} \sim \mathcal{N}\big(\mathbf{0}, \mathrm{diag}(\alpha_f, \alpha_l, \alpha_\omega)\big)$ for `omni_angular`.

A.6 LIDAR SIMULATION AND COLLISION CHECKING

For a 2D LiDAR with N beams and angular range ϕ, beam i has relative angle

$$\beta_i = -\frac{\phi}{2} + i\Delta\beta, \quad \Delta\beta = \frac{\phi}{N-1}, \quad i = 0, \ldots, N-1. \tag{8}$$

Given the sensor origin $\mathbf{o}$ and heading θ_s, beam i is the ray

$$\mathbf{p}_i(r) = \mathbf{o} + r \begin{bmatrix} \cos(\theta_s + \beta_i) \\ \sin(\theta_s + \beta_i) \end{bmatrix}, \quad 0 < r \le r_{\max}, \tag{9}$$

and the reported range is the distance to its nearest intersection with the obstacle boundaries,

$$r_i = \min\Big(r_{\max}, \min_{r \in \mathcal{I}_i} r\Big), \tag{10}$$

where $\mathcal{I}_i$ is the set of ranges at which beam i meets a boundary segment. IR-SIM implements the ray casting in NumPy: at each step, the beams of all LiDAR sensors are intersected with the obstacle boundary segments in one vectorized pass.

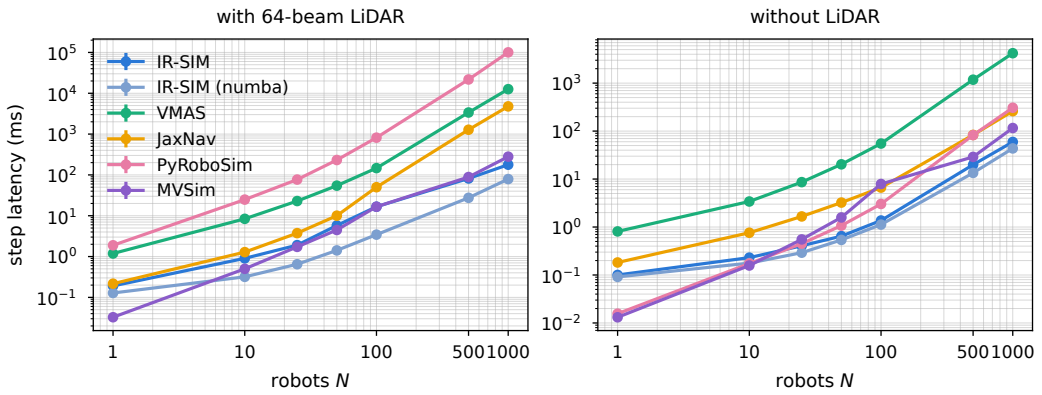


Figure 6: Single-core CPU scaling on the shared navigation scenario of Section 4.1: mean step latency against the number of robots with a 64-beam LiDAR on every robot (left) and without LiDAR (right); error bars give the standard deviation over five process-level runs, and both axes are logarithmic.

Collision checking is performed on planar geometric footprints. Let $\mathcal{G}_i(\mathbf{x}_i)$ denote the footprint of object i transformed by its current state. Two objects are considered in collision when

$$\mathcal{G}_i(\mathbf{x}_i) \cap \mathcal{G}_j(\mathbf{x}_j) \neq \emptyset. \tag{11}$$

IR-SIM implements this test with Shapely, using a spatial index to select candidate pairs before the exact intersection test.

B Computational Efficiency

B.1 Cross-Simulator Step-Time Measurements

Tables 5 and 5 and Figure 6 report the full single-core CPU measurements behind Figure 4, with and without LiDAR.

Scenario. All simulators receive the same obstacle geometry, initial poses, and recorded random actions. Differential-drive disc robots of radius $0.3\,\mathrm{m}$ move with a time step of $\Delta t = 0.1\,\mathrm{s}$ in a walled $20\,\mathrm{m} \times 20\,\mathrm{m}$ arena with 14 square obstacles; beyond $N = 100$, the arena grows as $20\sqrt{N/100}\,\mathrm{m}$ to keep the robot density constant. In the LiDAR condition, every robot carries a 64-beam, 360° LiDAR with a $6\,\mathrm{m}$ range.

Timing. One step covers kinematics, collision handling, LiDAR sensing when enabled, and observation construction. Each configuration runs on one pinned core of an AMD Ryzen 9 7950X3D in five independent processes, timing 1000 steps after 100 warm-up steps; at $N \geq 500$ the step budget is reduced, as marked in the tables. MVSim is timed through its C++ API with the same step content, since its Python interface cannot advance a single step; its robots are octagons of the same circumradius, and Box2D contact forces replace the stop rule. IR-SIM (numba) is the same IR-SIM build with its optional LiDAR kernel compiled by Numba (Lam et al., 2015); the main text reports the dependency-free NumPy build.

B.2 GPU Execution and Batched Simulation

This section evaluates the same scenario on an NVIDIA RTX 4090 GPU. IR-SIM provides an optional batched backend in PyTorch that steps B worlds of N robots as device tensors, with the same kinematics, collision rule, and LiDAR ray casting as the CPU path. The tables report its float32 mode, which uses the same precision as VMAS and JaxNav.

Single world. For one world, GPU execution is bound by kernel launches and data transfer rather than computation (Table 6a). With LiDAR, IR-SIM's backend steps one world in 1.9 to $3.0\,\mathrm{ms}$ for

Table 5: Single-core CPU results on the shared scenario (mean over five runs): step latency (ms) with its standard deviation and 95th percentile, agent steps per second, setup time (s), and peak resident memory (MB). Bold: best per N; ↑/↓: higher/lower is better. The arena side length is 20 m up to $N = 100$ and 45 m and 63 m at $N = 500$ and 1000. †: 100–200 timed steps instead of 1000. MVSim's setup excludes the Python start-up of the other simulators.

Simulator	N	with 64-beam LiDAR					without LiDAR				
		Mean ↓	p95 ↓	Steps/s ↑	Setup ↓	RSS ↓	Mean ↓	p95 ↓	Steps/s ↑	Setup ↓	RSS ↓
IR-SIM	1	0.19±0.00	0.21	5237	0.26	98	0.10±0.01	0.11	10067	0.31	97
IR-SIM	10	0.90±0.11	1.10	11260	0.28	100	0.23±0.06	0.34	45387	0.37	97
IR-SIM	25	1.91±0.03	2.20	13110	0.33	103	0.41±0.08	0.68	63718	0.44	98
IR-SIM	50	5.77±1.49	6.74	9693	0.49	107	0.65±0.01	0.81	77454	0.41	99
IR-SIM	100	16.6±0.13	18.1	6014	0.37	123	1.37±0.02	1.49	73122	0.47	101
IR-SIM	500†	81.8±0.38	86.7	6112	0.85	225	19.9±2.52	27.2	25478	0.90	113
IR-SIM	1000†	178±3.35	193	5622	1.34	353	59.4±9.65	73.3	17227	1.42	129
IR-SIM (numba)	1	0.13±0.00	0.16	7796	0.46	271	0.09±0.01	0.10	11096	0.28	97
IR-SIM (numba)	10	**0.32**±0.00	**0.43**	**31398**	0.48	272	0.18±0.01	**0.26**	56663	0.28	97
IR-SIM (numba)	25	**0.65**±0.02	**0.89**	**38529**	0.53	273	**0.29**±0.00	**0.48**	**85666**	0.32	98
IR-SIM (numba)	50	**1.42**±0.01	**1.59**	**35296**	0.65	275	**0.54**±0.01	**0.68**	**92893**	0.35	99
IR-SIM (numba)	100	**3.45**±0.02	**3.59**	**29025**	0.64	279	**1.12**±0.03	**1.29**	**88956**	0.44	101
IR-SIM (numba)	500†	**27.5**±0.20	**31.9**	**18186**	1.14	307	**13.4**±0.12	**17.6**	**37208**	0.64	113
IR-SIM (numba)	1000†	**79.1**±0.85	**86.9**	**12643**	1.83	342	**43.5**±0.33	**53.1**	**22968**	1.05	129
VMAS	1	1.19±0.00	1.21	841	0.06	536	0.81±0.00	0.83	1235	0.06	534
VMAS	10	8.40±0.02	8.50	1190	0.08	536	3.41±0.02	3.47	2935	0.07	535
VMAS	25	22.9±0.08	23.1	1091	0.09	537	8.63±0.05	8.71	2897	0.06	536
VMAS	50	54.6±0.15	55.0	916	0.14	538	20.3±0.17	20.4	2465	0.07	537
VMAS	100	147±0.19	148	681	0.28	540	55.0±0.55	55.5	1819	0.09	538
VMAS	500†	3377±258	3688	149	5.67	551	1180±24.4	1239	424	0.34	546
VMAS	1000†	12602±911	12945	80	15.96	560	4205±192	4288	238	0.22	558
JaxNav	1	0.22±0.00	0.23	4616	3.19	493	0.18±0.00	0.20	5483	3.29	493
JaxNav	10	1.28±0.02	1.31	7784	4.04	531	0.76±0.01	0.78	13160	3.98	523
JaxNav	25	3.75±0.03	3.78	6669	4.56	574	1.66±0.04	1.74	15047	4.37	563
JaxNav	50	10.0±0.02	10.1	4983	5.21	674	3.23±0.06	3.31	15465	4.78	646
JaxNav	100	50.0±0.09	51.1	1999	6.54	846	6.64±0.16	6.71	15072	5.70	796
JaxNav	500†	1279±5.63	1316	391	22.89	3102	83.2±4.83	91.6	6028	17.64	1900
JaxNav	1000†	4769±47.7	4817	210	48.78	7732	260±12.8	266	3849	33.80	2970
PyRoboSim	1	1.89±0.00	1.91	529	0.31	121	0.02±0.00	0.02	64003	0.32	121
PyRoboSim	10	24.8±0.12	25.2	403	0.32	122	0.17±0.00	0.27	57674	0.35	121
PyRoboSim	25	76.9±0.39	77.7	325	0.34	124	0.45±0.01	0.78	55551	0.40	121
PyRoboSim	50	232±0.97	233	216	0.36	126	1.07±0.02	1.43	46690	0.47	122
PyRoboSim	100	818±10.1	822	122	0.39	131	3.03±0.04	3.35	33060	0.64	123
PyRoboSim	500†	21767±766	22676	23	1.16	163	82.5±0.65	130	6063	2.08	127
PyRoboSim	1000†	100510±439	104449	10	2.70	202	307±3.08	473	3261	6.60	133
MVSim	1	**0.03**±0.00	**0.03**	**30606**	**0.00**	**76**	**0.01**±0.00	**0.01**	**76127**	**0.00**	**76**
MVSim	10	0.50±0.00	0.59	19970	**0.00**	**77**	**0.16**±0.00	0.27	**63459**	**0.00**	**76**
MVSim	25	1.73±0.01	2.02	14473	**0.00**	**77**	0.55±0.00	0.83	45259	**0.00**	**77**
MVSim	50	4.48±0.05	5.22	11169	**0.00**	**79**	1.58±0.05	2.40	31591	**0.00**	**78**
MVSim	100	16.6±0.13	21.2	6015	**0.01**	**81**	7.89±0.18	12.9	12673	**0.00**	**80**
MVSim	500†	88.7±0.17	115	5634	**0.02**	**95**	29.0±0.08	52.1	17252	**0.02**	**92**
MVSim	1000†	280±1.28	373	3575	**0.05**	**114**	116±0.15	187	8630	**0.04**	**108**

every N from 10 to 1000, the lowest GPU latency at every team size, whereas VMAS is 2 to 4× slower on the GPU than on a CPU core.

Batched throughput. Table 6b reports throughput when stepping B worlds per call and keeping the observations on the device, as in a GPU training loop. With 1024 worlds, IR-SIM reaches 2.05 and 0.91 million agent steps per second at $N = 10$ and $N = 100$, above JaxNav (1.84 million at $N = 10$; 0.67 million at $N = 100$ with 512 worlds, the largest batch that fits in memory) and VMAS (0.43 and 0.16 million).

Memory. On the CPU, IR-SIM uses 101 to 353 MB, compared with 80 to 114 MB for MVSim, 538 to 560 MB for VMAS and 796 to 7,732 MB for JaxNav (Table 7a). On the GPU with 1024 worlds of 100 robots, IR-SIM's backend uses 6,124 MB and VMAS 555 MB, while JaxNav runs out of memory.

Table 6: GPU execution on an RTX 4090 (IR-SIM's GPU backend in float32). (a) Step latency (ms) with one world per process; the CPU rows repeat Table 5. (b) Batched throughput with LiDAR (10^3 agent steps per second) for B worlds per call; the IR-SIM CPU row runs one world per core. Bold: best per column; ↑/↓: higher/lower is better; OOM: out of memory.

(a) Single-world step latency (ms, ↓)

Configuration	$N = 10$	$N = 100$	$N = 500$	$N = 1000$
with 64-beam LiDAR				
IR-SIM, CPU	**0.90**	16.6	81.8	178
VMAS, CPU	8.40	147	3377	12602
JaxNav, CPU	1.28	50.0	1279	4769
IR-SIM, GPU backend	1.85	**2.99**	**2.89**	**3.02**
IR-SIM, CPU step + CUDA tensors	0.94	16.1	67.9	147
VMAS, GPU	22.0	414	7212	27253
JaxNav, GPU	5.16	49.9	236	476
without LiDAR				
IR-SIM, CPU	**0.23**	**1.37**	19.9	59.4
VMAS, CPU	3.41	55.0	1180	4205
JaxNav, CPU	0.76	6.64	83.2	260
IR-SIM, GPU backend	0.40	1.65	**1.52**	**1.54**
IR-SIM, CPU step + CUDA tensors	0.45	2.23	16.7	49.5
VMAS, GPU	11.4	204	4113	15145
JaxNav, GPU	5.25	47.5	239	474

(b) Batched throughput (10^3 agent steps/s, ↑)

N	Configuration	$B = 1$	$B = 16$	$B = 128$	$B = 512$	$B = 1024$
10	IR-SIM, CPU (one world per core)	**11.3**	**127**	–	–	–
	IR-SIM, GPU backend	5.4	77.9	**551**	**1,762**	**2,051**
	VMAS, GPU	0.5	7.3	58.3	234	426
	JaxNav, GPU	3.2	54.3	418	1,130	1,837
100	IR-SIM, CPU (one world per core)	6.0	67.7	–	–	–
	IR-SIM, GPU backend	**33.7**	**507**	**777**	**928**	**912**
	VMAS, GPU	0.3	4.2	32.9	111	165
	JaxNav, GPU	3.6	55.8	364	673	OOM

Table 7: (a) Peak memory (MB): resident memory of one world on one CPU core, and device memory for $B = 1024$ worlds per call (bold: best per column; ↑/↓: higher/lower is better; OOM: out of memory). (b) Largest difference in robot state $\mathbf{x}$, LiDAR range r, and stop decisions between IR-SIM's GPU backend and its CPU path over 200 steps with a 64-beam LiDAR.

(a) Peak memory (MB, ↓)

Simulator	with LiDAR		without LiDAR	
	$N = 100$	$N = 1000$	$N = 100$	$N = 1000$
CPU, one world on one core				
IR-SIM	123	353	101	129
MVSim	**81**	**114**	**80**	**108**
VMAS	540	560	538	558
JaxNav	846	7,732	796	2,970
PyRoboSim	131	202	123	133
GPU, $B = 1024$ worlds, with LiDAR				
	$N = 10$	$N = 100$		
IR-SIM, float32	597	6,124		
IR-SIM, float64	1,153	11,918		
VMAS	**144**	**555**		
JaxNav	917	OOM		

(b) GPU backend vs. CPU path (↓)

Precision	N	max $\lvert\Delta\mathbf{x}\rvert$	max $\lvert\Delta r\rvert$ (m)	stop mismatches
float64	10	3.6×10^{-15}	$\mathbf{1.4 \times 10^{-13}}$	0 of 2,000
float64	100	$\mathbf{1.8 \times 10^{-15}}$	2.8×10^{-13}	0 of 20,000
float32	10	9.6×10^{-6}	2.7×10^{-4}	0 of 2,000
float32	100	1.6×10^{-5}	4.5	0 of 20,000

Exactness. In float64, the batched backend reproduces the CPU path: robot states agree to 4×10^{-15}, LiDAR ranges to 3×10^{-13} m, and all collision decisions are identical (Table 7b).

In float32, states agree to 2×10^{-5} and maximum range differences are 2.7×10^{-4} m at $N = 10$ and 4.54 m at $N = 100$, with hit/miss flips on grazing beams.

Table 8: Scenario-construction briefs reported in Table 2. All briefs require constructing a scene from scratch. All robots use differential-drive kinematics and circular footprints.

Brief	Family	Scenario and task	Steps
A1	static	12×12 m world, five rectangular obstacles; one robot ($r = 0.25$ m, 90-beam 360° LiDAR, 5 m) drives from $(1, 1)$ to $(11, 11)$	300
A2	static	16×10 m world, three wall segments forming a zigzag corridor; one robot ($r = 0.3$ m, 180-beam 270° LiDAR, 8 m) passes five ordered waypoints and reaches its goal	600
B1	pedestrians	10×10 m world; four pedestrians ($r = 0.2$ m, 0.6 m/s) walk back and forth across the robot's path; the robot ($r = 0.25$ m, no sensors) crosses from $(1, 5)$ to $(9, 5)$ without collision	400
B2	pedestrians	12×12 m world with a central 2×2 m pillar; eight pedestrians wander between random free-space targets at 0.5 m/s; the robot ($r = 0.3$ m) crosses the diagonal; the run must be reproducible under `--seed`	500
C1	multi-robot	six robots ($r = 0.2$ m, 36-beam LiDAR, 3 m) evenly placed on a circle of radius 4 m swap to diametrically opposite goals without colliding	500
C2	multi-robot	10×10 m warehouse with three shelves; three robots ($r = 0.2$ m, 0.8 m/s) cycle through their own four-waypoint loops, each waypoint at least twice, without colliding	800
E1	scene	16×16 m plaza with three boxes and three pillars; three robots ($r = 0.25$ m, 90-beam LiDAR, 6 m) drive to goals across the plaza among twelve pedestrians wandering through the free space and four patrolling ones; no collision; seeded reproducibility	600
E2	scene	14×12 m warehouse with five shelves; three robots ($r = 0.2$ m, 72-beam 270° LiDAR) cycle through loops around shelves at least twice, two robots (36-beam 180° LiDAR) patrol the open strips at least twice, two pedestrians wander inside the free strip on the right; seeded	800
E3	scene	14×14 m world with twelve circular pillars ($r = 0.5$ m) placed uniformly at random inside the central 10×10 m square with 1.8 m spacing using a seed; two robots ($r = 0.3$ m, 60-beam LiDAR) cross diagonally without collision; identical layout for equal seeds	500
H1	mixed	16×10 m parking lot with seven parked cars; two robots ($r = 0.25$ m, 48-beam 180° LiDAR) drive to parking bays past three patrolling pedestrians	500
H2	mixed	15×12 m library with three shelves; one robot ($r = 0.25$ m, 90-beam 270° LiDAR) passes five waypoints through the aisles; three pedestrians wander in a reading area; seeded	800
H3	mixed	12×8 m, two rooms joined by a doorway; two robots ($r = 0.25$ m, 60-beam LiDAR) cross via two waypoints each; two pedestrians patrol the rooms	600
H4	mixed	12×10 m, six robots ($r = 0.2$ m, 24-beam LiDAR) swap between two lines past two pillars	600

C Scenario Construction Study

This section summarizes the briefs, the validator, the agent setup, and the complete results of the scenario-construction study in Section 4.2.

C.1 Briefs

Each brief is a machine-readable specification from which the text shown to the agent is rendered, so the text and the validator cannot disagree. It fixes the world, the time step (0.1 s), the run length, the obstacles, the robots (differential drive, footprint, speed limit, sensors, and start pose), the task, and the pedestrians where present, and it ends with the list of acceptance checks. Table 8 summarizes the thirteen reported briefs; family E contains rich scenes, with crowds, seeded random layouts, and limited field-of-view sensors, that need no task-specific control code in any simulator.

C.2 Deliverable Contract and Validator

The deliverable is the scenario in the simulator's native form (a YAML file for IR-SIM, a `BaseScenario` subclass for VMAS, and a `create_world()` function for PyRoboSim) together with a script `run.py` that runs it headlessly and writes a rollout trace in a common JSON format. The validator executes `run.py`, loads the native artifact through the simulator's API, and evaluates the brief's acceptance checks: simulator usage, world size and step count, entity counts, obstacle geometry, robot kinematics, footprint, start pose and LiDAR parameters, task completion, speed limits and world bounds, pedestrian motion, collision freedom, and, where required, determinism under a seed. A deliverable is *executable* if `run.py` succeeds and the trace has the requested shape, and *valid* if every check passes; the *specification accuracy* is the fraction of checks passed.

C.3 Agent Configuration and Budget

The agent is Claude Code in non-interactive mode with `claude-opus-5-5` or `claude-sonnet-5`, using file and search tools but no shell, web access, or sub-agents. A read-only copy of the simulator's documentation and source is provided, and the skill condition

Table 9: Declarative simulators considered for the construction study and the requirements of the briefs they can express natively. Yes/no entries for the measured simulators were established by the reference solutions; the GPUDrive entries were read from its source (commit `aa48a43`); MVSim and Flatland entries come from their documentation and were not executed here.

Simulator	Scene description	Diff.-drive, per-robot radius	Boxes and pillars anywhere	LiDAR beams / FOV	Scripted pedestrians (own radius)	30–100 s runs at 0.1 s	Measured
IR-SIM	YAML objects, behaviors, sensors	yes	yes	yes	yes	yes	13 briefs
VMAS	Python scenario class	yes	yes	yes	yes	yes	13 briefs
PyRoboSim	Python / YAML world	yes	yes (locations)	yes	yes (extra robots)	yes	13 briefs
JaxNav	occupancy grid and agent states	one shared radius	yes (grid cells)	one shared model, ≥ 2 beams	no	yes	6 briefs
GPUDrive	Waymo-format JSON logs	vehicle models only	road polylines	fixed sample count	logged trajectories; non-vehicles removed by default	no (91 steps)	no
MVSim	XML world	yes	yes	yes	scripted vehicles	yes	not installable here
Flatland	YAML world and models	yes (plugins)	yes	yes (plugin)	scripted models	yes	ROS 1 only

additionally installs the five `ir-sim-skill` skills (version 0.9.1). Each attempt writes the deliverable once within 60 tool turns and is graded without repair. Work directories are isolated, and the session transcripts confirm that no attempt accessed files outside them.

C.4 Reference Solutions

All thirteen briefs were solved in IR-SIM, VMAS, and PyRoboSim, and six compatible briefs were solved in JaxNav before the study to confirm that each is achievable and to test the validator.

C.5 Coverage of Other Declarative Simulators

IR-SIM, VMAS, and PyRoboSim were chosen because all thirteen briefs lie within their common capabilities (Table 9).

JaxNav (Rutherford et al., 2024) is evaluated on the six briefs it can express (A1, A2, C1, C2, E3, H4), since all of its agents share one model and it has no moving obstacles. GPUDrive (Kazemkhani et al., 2025) supports only vehicle models and fixed 91-step episodes, and MVSim (Blanco-Claraco et al., 2023) and Flatland (Avidbots, 2026) require system or ROS installations, so they are not included.

C.6 Complete Results

Table 2 aggregates 174 attempts per model; none was stopped by an API rate limit. With `claude-opus-5-5`, most failures are specification errors rather than simulator limits: plain IR-SIM fails mainly because the agent adds a `name` key to the YAML `world` block (four of five failures), the skill condition fails twice on E1 (a robot–pedestrian collision), and PyRoboSim fails three of four times by simulating the scene without importing `pyrobosim`. With skills, attempts take an average of 98 s compared with 204 to 238 s for the other conditions (Mann–Whitney test, $p < 0.001$), while the validity gain over plain IR-SIM is not significant (Fisher's exact test, $p = 0.43$). The gap is largest on the rich scenes of family E (135 s and \$0.66 against 290 to 339 s and \$1.35 to \$1.54), where crowds, patrols, loops, and seeded layouts are YAML groups in IR-SIM but hand-written code elsewhere. Because `ir-sim-skill` 0.9.1 was refined on failures from these brief families, the skill result is an upper bound rather than an independent measurement.

With `claude-sonnet-5`, the skills raise validity over plain IR-SIM from 61.5% to 94.9% (Fisher's exact test, $p < 0.001$).

Table 10: Human single-shot submissions in IR-SIM, one row per participant, graded with the validator of the agent study: Docs: documentation only (docs) or documentation and the skill recipes (docs+skill); Fam.: self-rated IR-SIM familiarity (1–5) and years of Python experience; Time: active seconds from opening the brief to the submission (pauses excluded); Ran: the participant reported running the scene outside the page; Exec., Valid, Spec. acc. and the failed checks are defined as in the agent tables; Lines: non-comment Python / YAML lines. The row marked ‡ in the single-shot table aggregates these.

Participant	Docs	Fam. / Py	Task	Time (s)↓	Ran	Exec.	Valid	Spec. acc. (%)↑	Failed checks	Lines
P05	docs	3 / 7	A1	768	no	yes	yes	100	–	69 / 35
P100	docs	4 / 6	A1	1494	yes	yes	no	83	`obstacles_present`, `collision_free`	69 / 40
P11	docs	4 / 6	A1	1152	yes	yes	yes	100	–	69 / 40
p16	docs	3 / 8	A1	1380	no	yes	no	75	`robots_start`, `robots_lidar`, `collision_free`	69 / 25
P24	docs	4 / 8	A1	1176	no	yes	yes	100	–	69 / 30

C.7 Human Reference

Five participants authored brief A1 with the IR-SIM documentation through a single-page web study that follows the same single-shot protocol (Table 10). All five submissions are executable and three are valid. Authoring took 13 to 25 active minutes (mean 19.9); the failures involve specification errors, such as a misplaced obstacle or an incorrect initial heading. With five participants, these results indicate the scale of human authoring time rather than a statistical comparison.

D Social Navigation Benchmark Protocol

This section lists the settings behind Table 3; Table 11 gives every metric for every controller and team size.

Scenario. Robots have random convex polygonal footprints (average radius 0.1–$0.4\,\mathrm{m}$), random starts and goals, no static obstacles, velocity bounds $|v| \le 2\,\mathrm{m/s}$ and $|\omega| \le 3.14\,\mathrm{rad/s}$, a $0.1\,\mathrm{s}$ step, a 500-step horizon, and a $0.4\,\mathrm{m}$ arrival threshold; a collision stops the robots involved. Every episode reseeds the simulator from the base seed and the episode index, so all controllers see the same layouts. An episode with any collision counts as a collision, and one in which some robot has neither arrived nor collided at the horizon counts as a timeout.

Controllers. ORCA and AVOCADO use a $15\,\mathrm{m}$ neighbor distance, five neighbors, a $0.5\,\mathrm{s}$ time horizon, a $0.1\,\mathrm{m}$ safety buffer, and a $1.0\,\mathrm{m/s}$ preferred speed (AVOCADO with its default opinion-dynamics parameters); VO-polytope runs the released implementation in reciprocal mode with a $5\,\mathrm{m}$ neighbor region; RDA runs one model predictive controller per robot (horizon 5, at most five obstacles, two ADMM iterations, $0.2\,\mathrm{m}$ safety distance) without coordination. SARL, CADRL, and LSTM-RL are trained in the original CrowdSim environment with the repository's default recipe (3000 imitation learning and 10 000 reinforcement learning episodes) and evaluated with their original action selection; RL-RVO uses the released pretrained policy without retraining. ORCA, AVOCADO, and RL-RVO represent every robot by the largest enclosing circle in the scene, the CrowdNav policies observe each robot's own enclosing circle, and VO-polytope and RDA use the polygons directly. Planar velocity outputs $\mathbf{v}$ from controllers are mapped to a differential-drive command by $v = \|\mathbf{v}\| \max(0, \cos\Delta\theta)$ and $\omega = \mathrm{clip}(3\Delta\theta, -\omega_{\max}, \omega_{\max})$, where $\Delta\theta$ is the heading error between the robot and $\mathbf{v}$.

Runtime. All rows use the IR-SIM build of Section 4.1 on one core of an AMD Ryzen 9 7950X3D. A 100-episode cell, including policy inference, takes seconds to minutes for the reactive planners and tens of minutes for the per-robot MPC and the CrowdNav value-network policies (from $8\,\mathrm{s}$ for ORCA with five robots to about $110\,\mathrm{min}$ for RDA with ten robots).

Table 11: Complete social navigation benchmark results behind Table 3: episode-level outcome rates (%), and navigation time and path length of robots that reach their goals and speed over active robot frames (mean ± population standard deviation), for every controller and team size. Bold: best value per team size; ↑/↓: higher/lower is better.

Robots	Method	Success↑	Collision↓	Timeout↓	Time (s)↓	Speed (m/s)↑	Path (m)↓
5	ORCA (van den Berg et al., 2011)	**100.0**	**0.0**	**0.0**	5.05 ± 2.47	$\mathbf{0.897} \pm 0.125$	4.60 ± 2.26
5	VO-polytope (Huang et al., 2023)	83.0	17.0	**0.0**	$\mathbf{4.74} \pm 2.28$	0.888 ± 0.135	4.38 ± 2.21
5	AVOCADO (Martinez-Baselga et al., 2025)	**100.0**	**0.0**	**0.0**	5.23 ± 2.38	0.827 ± 0.139	4.49 ± 2.21
5	RDA (Han et al., 2023)	20.0	**0.0**	80.0	5.03 ± 3.48	0.601 ± 0.400	$\mathbf{4.14} \pm 2.14$
5	SARL (Chen et al., 2019)	41.0	**0.0**	59.0	10.90 ± 7.80	0.589 ± 0.222	6.62 ± 4.14
5	CADRL (Chen et al., 2017)	9.0	6.0	85.0	8.79 ± 6.87	0.578 ± 0.219	5.12 ± 3.07
5	LSTM-RL (Everett et al., 2018)	39.0	13.0	48.0	9.97 ± 6.92	0.632 ± 0.199	6.46 ± 3.94
5	RL-RVO (Han et al., 2022)	89.0	6.0	5.0	5.82 ± 3.46	0.827 ± 0.144	4.84 ± 2.47
10	ORCA (van den Berg et al., 2011)	**90.0**	1.0	9.0	5.41 ± 3.04	$\mathbf{0.850} \pm 0.151$	4.63 ± 2.40
10	VO-polytope (Huang et al., 2023)	45.0	55.0	**0.0**	$\mathbf{4.31} \pm 2.44$	0.835 ± 0.172	3.91 ± 2.33
10	AVOCADO (Martinez-Baselga et al., 2025)	86.0	**0.0**	14.0	5.50 ± 2.75	0.783 ± 0.160	4.49 ± 2.33
10	RDA (Han et al., 2023)	0.0	**0.0**	100.0	5.94 ± 6.04	0.356 ± 0.346	$\mathbf{3.64} \pm 2.44$
10	SARL (Chen et al., 2019)	26.0	11.0	63.0	10.96 ± 8.41	0.543 ± 0.215	5.89 ± 3.62
10	CADRL (Chen et al., 2017)	0.0	40.0	60.0	8.93 ± 7.30	0.558 ± 0.192	4.90 ± 3.20
10	LSTM-RL (Everett et al., 2018)	0.0	36.0	64.0	12.69 ± 11.02	0.535 ± 0.186	7.01 ± 5.38
10	RL-RVO (Han et al., 2022)	73.0	20.0	7.0	6.87 ± 4.19	0.750 ± 0.144	5.18 ± 2.94

E Policy Training Protocol

This section lists the settings behind Table 4.

Port. CrowdNav's learner (trainer, explorer, replay memory, value networks, and schedule) is used without modification; only its environment class is replaced by a subclass that keeps CrowdNav's episode sampler, seeding, observation, reward, collision test, and termination, and delegates the state update to an IR-SIM world generated from CrowdNav's `env.config` (a 0.25 s step, a holonomic robot of radius 0.3 m, and N ORCA-driven humans of radius 0.3 m with CrowdNav's ORCA settings). Because the humans do not observe the robot, the crowd commands are evaluated once per simulation step and reused for the 81 one-step lookaheads of CrowdNav's action selection, which is exact. Replaying 500 test episodes with recorded actions reproduces every human state, reward, and termination step of `CrowdSim` exactly with five humans; with ten humans the training runs of the two simulators differ slightly (Table 12).

Training. All runs use CrowdNav's default recipe: 3000 imitation-learning episodes from ORCA, then 10 000 reinforcement-learning episodes with 100 SGD batches after each episode, ϵ-greedy exploration decaying from 0.5 to 0.1, target-network updates every 50 episodes, validation every 1000 episodes, and a final test on 500 fixed cases. Each (policy, humans, seed) configuration is trained once in `CrowdSim` and once in IR-SIM with the same seed, and therefore the same episode layouts and exploration draws; the two runs of a pair train concurrently on two dedicated cores each, all on the same CPU chiplet.

Batched TorchRL implementation (C, D). The environment is a TorchRL `EnvBase` with a batch of 16 worlds, each using the IR-SIM port (D) or the unmodified `CrowdSim` (C) and starting the next training case as soon as its episode ends; the policy scores the 81 candidate actions of all worlds in one pass of the SARL value network, which reproduces the action of CrowdNav's `predict` in every checked decision. The recipe is CrowdNav's (same demonstrations, value targets, exploration schedule, SGD batches, target updates, validation, and test), so C and D differ from A and B only in the sampling order of the concurrent worlds, not in the algorithm; C evaluates the crowd twice per step where D evaluates it once, and D (GPU) runs the network passes and SGD updates on the RTX 4090.

Measurements. Simulator time per decision is the time spent in `step` and `onestep_lookahead` over the reinforcement-learning phase divided by the number of decisions (for C and D, the batched environment step, its observation, and the vectorized one-step reward); network time is the remaining rollout time per decision, i.e., the value-network passes and action selection (81 sequential calls per decision in A and B, one batched call per step shared by all

Table 12: Separating the simulator from the training implementation (SARL; five humans with seeds 0 and 1, ten humans with seed 0; speed-up relative to A in the same group). A and B run CrowdNav's original training code with `CrowdSim` and IR-SIM; C and D run the batched TorchRL implementation that steps 16 worlds together and scores the candidate actions of all worlds in one network pass. Each run uses two dedicated CPU cores on the same chiplet; D (GPU) additionally runs the network on an RTX 4090. Sim. and net. time are the simulator and value-network time of one robot decision as in Table 4. Bold: best value per group; ↑/↓: higher/lower is better.

Humans	Seed	Config	Method	Simulator	Hardware	Sim. time (ms)↓	Net. time (ms)↓	Time (h)↓	Speed-up↑	Success (%)↑	Collision (%)↓	Nav. time (s)↓
5	0	A	CrowdNav	`CrowdSim`	CPU	6.86	27.77	6.99	1.0×	100	0	10.54
5	0	B	CrowdNav	IR-SIM	CPU	2.50	27.00	6.13	1.1×	100	0	10.54
5	0	C	Batched `CrowdSim` + TorchRL	`CrowdSim`	CPU	**0.36**	0.44	0.74	9.4×	100	0	**10.38**
5	0	D	Batched IR-SIM + TorchRL	IR-SIM	CPU	0.43	0.47	0.76	9.2×	100	0	**10.38**
5	0	D (GPU)	Batched IR-SIM + TorchRL	IR-SIM	CPU + GPU	0.41	**0.11**	**0.51**	**13.7**×	100	0	10.57
5	1	A	CrowdNav	`CrowdSim`	CPU	6.98	28.10	7.07	1.0×	100	0	10.64
5	1	B	CrowdNav	IR-SIM	CPU	2.59	27.47	6.25	1.1×	100	0	10.64
5	1	C	Batched `CrowdSim` + TorchRL	`CrowdSim`	CPU	**0.38**	0.49	0.83	8.5×	100	0	**10.59**
5	1	D	Batched IR-SIM + TorchRL	IR-SIM	CPU	0.47	0.49	0.84	8.4×	100	0	**10.59**
5	1	D (GPU)	Batched IR-SIM + TorchRL	IR-SIM	CPU + GPU	0.45	**0.12**	**0.62**	**11.4**×	100	0	10.65
10	0	A	CrowdNav	`CrowdSim`	CPU	31.17	29.43	13.11	1.0×	100	0	11.91
10	0	B	CrowdNav	IR-SIM	CPU	4.01	28.17	7.78	1.7×	100	0	**11.64**
10	0	C	Batched `CrowdSim` + TorchRL	`CrowdSim`	CPU	1.41	0.75	1.44	9.1×	100	0	12.44
10	0	D	Batched IR-SIM + TorchRL	IR-SIM	CPU	0.72	0.76	1.27	10.3×	100	0	12.02
10	0	D (GPU)	Batched IR-SIM + TorchRL	IR-SIM	CPU + GPU	**0.70**	**0.12**	**0.70**	**18.8**×	100	0	12.25

worlds in C and D); rollout time covers the training and validation episodes of that phase, and the total adds imitation learning, the SGD updates, and the final test.

Additional runs. Table 12 repeats all configurations with a second seed and with ten humans; every trained policy achieves approximately 100% success on the 500 test episodes (rates rounded to whole percentages).

With the second seed, the batched implementation with IR-SIM (`CrowdSim`) is 8.4× (8.5×) faster than the original CrowdNav–CrowdSim pipeline, and 11.4× with the network on the GPU, which confirms the result of the main text. With ten humans, the simulator accounts for a larger share of the cost: replacing only the simulator shortens training from 13.11 to 7.78 h (1.7×), the batched implementation needs 0.72 ms of simulator time per decision with IR-SIM and 1.41 ms with `CrowdSim` (1.27 versus 1.44 h of training), and the GPU variant trains in 0.70 h (18.8×). With five humans, `CrowdSim`'s step is slightly cheaper than IR-SIM's (0.36 versus 0.43 ms) and the two batched runs take the same time.

F Simulator Bridges and Robot Interfaces

IR-SIM is designed for fast 2D scenario construction, but many navigation studies still require richer visual context, simulator-specific sensors, or physical robot validation. To support this workflow, IR-SIM provides bridge interfaces that preserve lightweight scenario logic while delegating perception, rendering, and physics simulation to external platforms.

F.1 Habitat-Sim Map Conversion

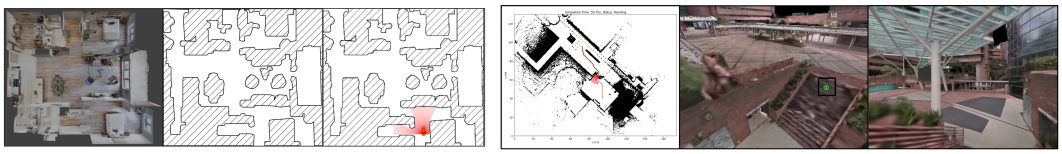
The left side of Fig. 7(a) illustrates the conversion from Habitat-Sim/HM3D (Savva et al., 2019; Ramakrishnan et al., 2021) to IR-SIM, where a 3D indoor scene is converted into a 2D IR-SIM occupancy map. This conversion allows rich indoor assets to be reused for 2D path planning, sensor simulation, and navigation benchmarking without requiring a full 3D simulation loop.

F.2 Gaussian Splatting Visualization

The right side of Fig. 7(a) shows the transfer from IR-SIM to Gaussian Splatting (Kerbl et al., 2023), where IR-SIM provides the navigation map and trajectory and the reconstructed 3D scene provides visual context for inspection.

F.3 CARLA Integration

The remaining bridge examples transfer IR-SIM scenarios or control policies into high-fidelity simulators and physical robots. As shown on the left side of Fig. 7(b), pedestrian trajectories generated



(a) Habitat-Sim/HM3D to IR-SIM occupancy map conversion (left) and IR-SIM trajectory and map in a Gaussian Splatting scene (right)



(b) IR-SIM pedestrian trajectories instantiated in CARLA (left) and NeuPAN control executed through IR-SIM with CARLA sensing and map information (right)



(c) IR-SIM pedestrian trajectories synchronized with Isaac Sim (left) and IR-SIM state and reference trajectory publishing for a real robot (right)

Figure 7: Bridge examples: (a) 3D scene assets, (b) CARLA, and (c) deployment to Isaac Sim and physical robots.

by IR-SIM are instantiated in a CARLA street scene to add crowd interaction. The right side of Fig. 7(b) shows NeuPAN control transferred from IR-SIM to CARLA, where CARLA provides LiDAR and map information while the NeuPAN policy (Han et al., 2025) is executed through IR-SIM.

F.4 Isaac Sim Integration

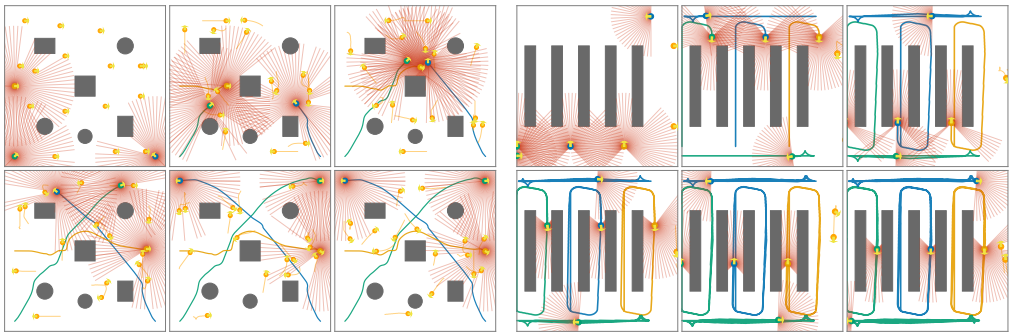
The left side of Fig. 7(c) shows an Isaac Sim warehouse scene, where pedestrian trajectories generated by IR-SIM using RVO behavior (van den Berg et al., 2008) are synchronized with the 3D simulator via ROS 2.

F.5 Physical Robot Interfaces

Finally, the right side of Fig. 7(c) applies the same idea to physical robots by publishing the robot state and reference trajectory through ROS. Together, these examples show how IR-SIM remains the lightweight scenario and control layer while external platforms provide richer rendering, sensing, or deployment environments, allowing a scenario authored once in IR-SIM to be reused across validation environments without reimplementing the scenario logic.

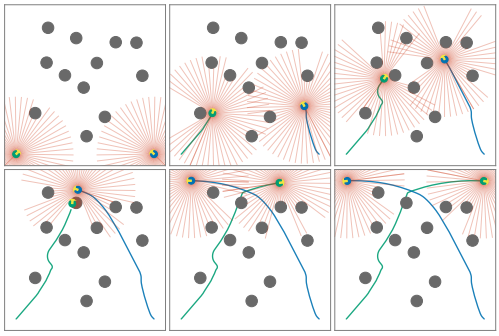
G Scenario Examples

Fig. 8 shows six rollout snapshots of each of eight IR-SIM scenes described in YAML, the reference scenes of the construction study (Appendix C) replayed from their recorded rollouts: crowded plazas, warehouse fleets with field-of-view LiDARs, seeded random layouts, parking lots, libraries, rooms joined by doorways, and intersections with patrolling and wandering pedestrians.

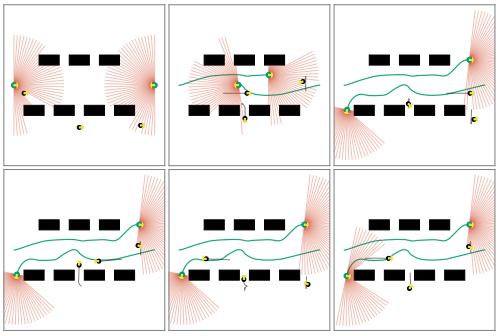


(a) Plaza crowd: three robots among sixteen pedestrians

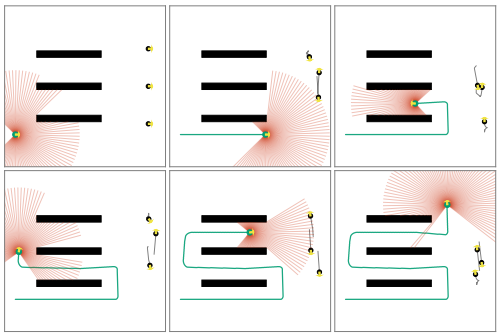
(b) Warehouse fleet with 270° and 180° LiDARs



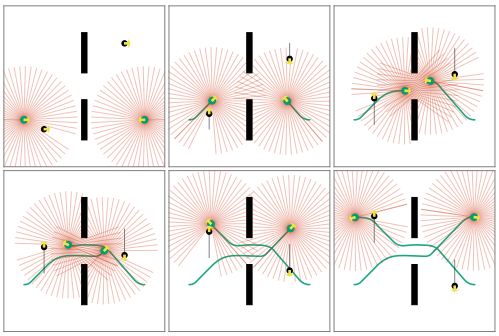
(c) Seeded random pillar field



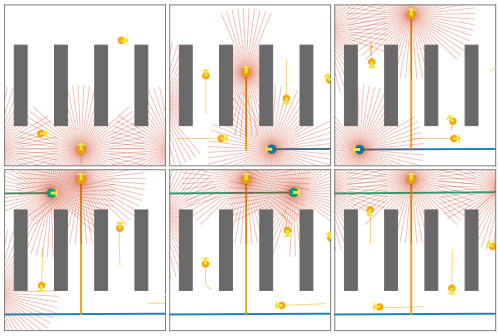
(d) Parking lot with patrolling pedestrians



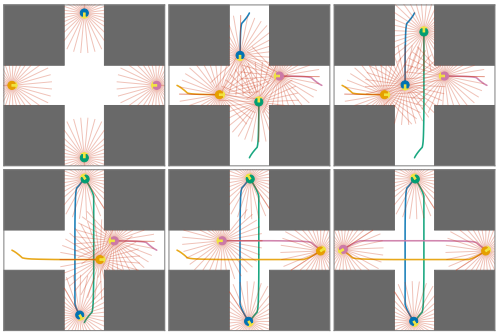
(e) Library aisles with wandering readers



(f) Two rooms joined by a doorway



(g) Warehouse crossing among patrols and wanderers



(h) Intersection square admitting one robot at a time

Figure 8: Rollout snapshots (six per scene) of complex IR-SIM scenes replayed from recorded rollouts: robots with their LiDAR beams and trajectories, pedestrians in orange with short trails, static obstacles in gray.