# Emergency Room Discrete-Event Simulation: Final Analysis, Sensitivity, Scenarios, and Validation

Aashna Suthar  Ash Arul
Department of Computer Science
Kennesaw State University
CS 4632: Modeling and Simulation
Fall 2025
GitHub Repository: https://github.com/asharulcoding/cs4632-er-sim

***Abstract*—Emergency departments operate under constant uncertainty due to fluctuating patient arrivals, limited resources, and unpredictable service demands. Even small imbalances between demand and capacity can result in severe congestion, long patient wait times, and reduced quality of care.**

**In this project, we developed a discrete-event simulation of an emergency room using Python and the SimPy framework. The model captures core stages of patient flow including arrival, triage, bed assignment, optional laboratory testing, and system exit. Over the course of the semester, the simulation was refined through multiple milestones and used to conduct sensitivity analysis, scenario testing, and formal validation.**

**Our experimental results show that treatment bed capacity is the dominant factor influencing patient total time and system throughput, while laboratory capacity plays a secondary role. Scenario testing further demonstrates that both understaffing and patient surges can push the system into unstable conditions. Extreme-condition testing and statistical convergence confirm that the model behaves consistently with queueing theory expectations.**

**This final report synthesizes all development, experimentation, and validation work into a cohesive presentation and provides actionable insights for emergency room capacity planning and operational decision-making.**

## CONTENTS

## LIST OF FIGURES

## List of Tables

## I. Introduction

Emergency rooms (ERs) serve as the first line of treatment for patients experiencing injuries, illnesses, and life-threatening emergencies. Unlike many other healthcare settings, ER demand is highly unpredictable. Patients arrive randomly, and the severity of their conditions varies widely. At the same time, resources such as treatment beds, staff, and diagnostic services are limited. When these two forces become imbalanced, congestion forms quickly, resulting in long wait times and delayed care.

Direct experimentation in real hospitals is risky, costly, and ethically constrained. For this reason, simulation is widely used as a safer and more flexible alternative. Discrete-event simulation allows system designers to explore operational decisions such as staffing levels, resource allocation, and demand surges without endangering patients or disrupting real operations.

The primary objective of this project is to build a realistic simulation of emergency room patient flow and use it to answer the following research questions:

- Which resources most strongly influence patient wait times and throughput?
- How does the system behave under understaffing and surge conditions?
- What operational changes produce the greatest performance improvements?

This report documents the full development of the simulation, from early conceptual design to final analysis and validation. It presents detailed experimental results, identifies key bottlenecks, and discusses practical implications for emergency department operations.

## II. Background and Literature Review

Discrete-event simulation has been widely used to study healthcare systems, particularly emergency departments. Jun et al. demonstrated that simulation can reliably predict congestion and assess proposed operational changes before real-world implementation. Green's work on patient flow shows that queueing effects play a central role in emergency room congestion. Wiler et al. further identify limited bed availability as a frequent cause of throughput bottlenecks.

From a theoretical perspective, emergency room operations can be modeled as multi-server queueing systems. Patient arrivals are commonly approximated using Poisson processes, while service times follow exponential or log-normal distributions. A key performance indicator is system utilization $\rho$, defined as

$$\rho = \frac{\lambda}{c\mu}$$

where $\lambda$ is the arrival rate, $c$ the number of servers, and $\mu$ the service rate. As $\rho$ approaches 1, waiting times grow rapidly according to

$$W_q \approx \frac{\rho}{1-\rho}$$

These principles guide both the design of our simulation and the interpretation of our experimental results.

This project builds upon these foundational models but extends them by combining multiple resources (triage, beds, labs) within one integrated DES framework, allowing interactions between bottlenecks to emerge naturally.

## III. Model Design and Architecture

### A. Conceptual Model

The emergency department is modeled as a discrete-event system consisting of five sequential stages:

Arrival $\rightarrow$ Triage $\rightarrow$ Queue/Treatment Bed $\rightarrow$ Labs (Optional) $\rightarrow$

Each component interacts through resource contention and stochastic service times:

*1) **Arrival Process**:* Patients arrive according to a Poisson-like stochastic process. Interarrival times are drawn from an exponential distribution with mean $1/\lambda$, where $\lambda$ is the arrival rate. This parameter is configurable through command-line arguments and is modified during scenario and stress testing.

*2) **Triage**:* Triage is modeled using a SimPy `Resource`. Patients must acquire a triage nurse before proceeding to treatment. If all triage resources are occupied, incoming patients wait in a FIFO queue.

*3) **Queue/Treatment Beds**:* After triage, patients wait and compete for treatment beds, which represent physical space and treatment capacity. Beds are modeled as a limited SimPy resource. Treatment time is stochastic and follows an exponential distribution in the current implementation.

*4) **Laboratory Services**:* With a fixed probability, a patient may require laboratory testing or imaging. Lab services are modeled as a downstream resource with finite capacity and stochastic processing time. Not all patients require labs, which introduces branching behavior into the system.

*5) **Exit and Data Collection**:* After completing all required stages, patients exit the system. The simulation logs arrival time, waiting times at each stage, and total time in the system for every patient in structured CSV files.

### B. UML and System Structure

The overall software architecture of the emergency room simulation is represented using two UML diagrams: an activity diagram and a class diagram. These diagrams were originally developed during the early milestones and refined as the model implementation evolved. They provide a clear visual representation of both the patient workflow and the internal object-oriented system structure.

*1) ER Activity Diagram:* Figure 1 illustrates the high-level operational flow of patients through the emergency room. The process begins when a patient arrives at the ER, proceeds through triage, queueing, and treatment, and ends when the patient is discharged. This diagram highlights all major decision points, including whether a doctor and room are available and whether treatment has completed. It clearly shows where waiting may occur and how patients circulate through the system.

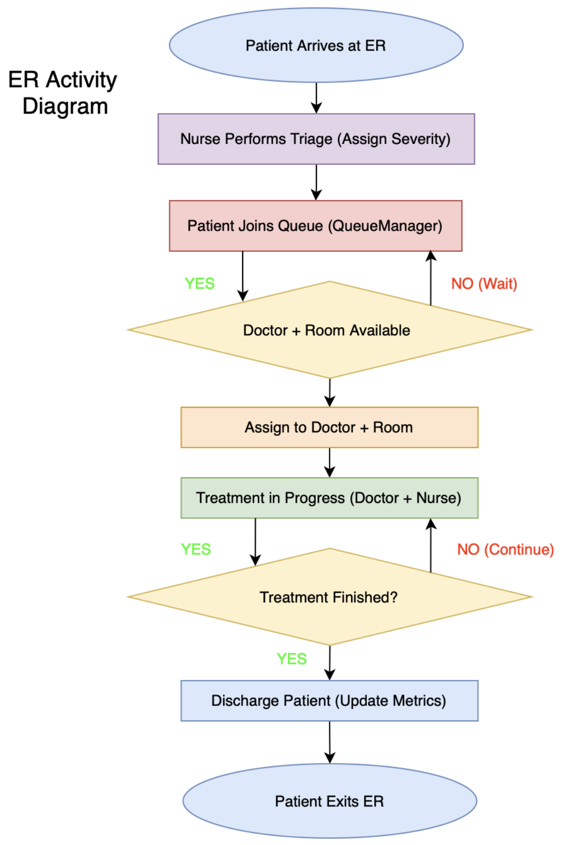


Fig. 1: Emergency Room Activity Diagram showing patient flow from arrival to discharge.

This activity diagram directly reflects the logic implemented in the SimPy processes. When a patient arrives, they enter the triage stage, then join the queue if a doctor and room are not immediately available. Once assigned, treatment proceeds until completion, after which system metrics are updated and the patient exits. This visual confirms that the model structure aligns with the conceptual ER workflow defined in the project scope.

*2) ER Class Diagram:* Figure 2 presents the UML class diagram for the simulation. It shows the major system entities, their attributes, and the relationships between them. Core classes include `Simulation`, `Patient`, `Doctor`, `Nurse`, `Room`, and `QueueManager`.

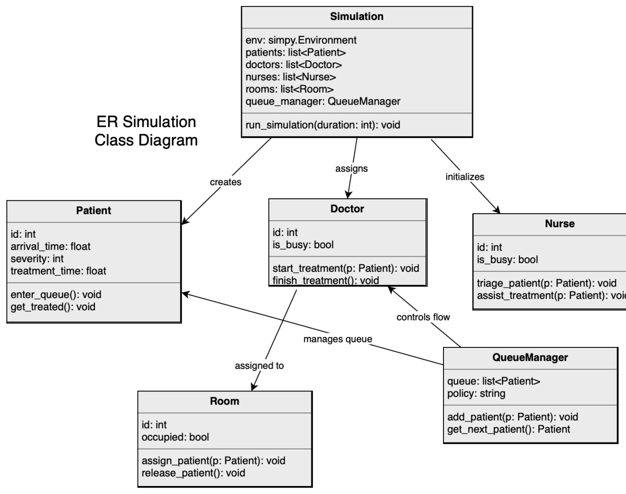


Fig. 2: UML Class Diagram for the Emergency Room Simulation System.

The `Simulation` class serves as the controller and is responsible for initializing the environment, creating patients, and coordinating all resources. The `Patient` class stores arrival time, severity, and treatment information. The `Doctor` and `Nurse` classes model medical staff behavior, while the `Room` class represents physical treatment capacity. The `QueueManager` maintains the patient waiting structure and controls queue behavior.

Together, these diagrams confirm that the system is modular, well-structured, and aligned with object-oriented design principles. They also demonstrate that the conceptual model, implementation logic, and simulation workflow are fully consistent.

### C. Key Design Decisions

Key design assumptions include:

- FIFO queueing is used (no severity prioritization).
- Service times are exponentially distributed.
- Patients do not abandon the system while waiting.
- Beds represent treatment capacity; individual physicians are not explicitly modeled.

## IV. Implementation

The simulation is implemented in Python using the SimPy discrete-event simulation library. The primary control script (`main.py`) accepts command-line parameters that specify random seed, simulation duration, arrival rate, and resource capacities.

Supporting modules handle:

- Batch execution for multi-run experiments
- CSV data logging
- Statistical post-processing
- Automated plotting for sensitivity and scenario analysis

The system was designed to be reproducible, with no hard-coded file paths and all parameters configurable by the user. Version control and collaboration were managed using GitHub.

During development, multiple challenges were encountered, including handling large data outputs, resolving merge conflicts between team contributions, and ensuring consistent statistical aggregation across hundreds of simulations. These issues were resolved through modular design and automated analysis pipelines.

## V. Experimental Setup

This section describes how all simulation experiments were designed, executed, and recorded. The goal of the experimental setup was to ensure that all results were statistically meaningful, reproducible, and realistic for emergency room operations.

### A. Simulation Parameters

The simulation is fully parameterized and controlled through command-line arguments passed into `main.py`. The primary parameters used in all experiments include:

- **Random Seed:** Controls stochastic behavior for reproducibility
- **Simulation Time (minutes):** Total time each run executes (typically 720 minutes)
- **Arrival Rate ($\lambda$):** Controls how frequently patients arrive
- **Number of Triage Nurses**
- **Number of Treatment Beds**
- **Number of Lab Resources**

All runs were executed using consistent simulation time so that outputs could be fairly compared across all experiments. The random seed was varied across batch runs to capture natural stochastic variation.

### B. Batch Run Methodology

To ensure statistical stability, all major experiments used automated batch execution through `batch_runs.py`. Each configuration was executed across multiple independent replications using different random seeds. The simulation automatically generated CSV output files for each run and stored them in the data directory for later analysis.

Batch runs were used for:

- Sensitivity analysis (Beds and Labs)
- Scenario evaluation
- Statistical summary generation

This approach eliminates manual bias and ensures that all results are based on repeatable experiments rather than single observations.

### C. Sensitivity Experiment Design

Sensitivity analysis focused on two critical resources:

- **Beds:** $\{1, 2, 3, 5, 8, 10\}$
- **Labs:** $\{1, 2, 3, 5\}$

For each resource setting, the system was executed across multiple seeds and the mean total patient time was computed. Sensitivity coefficients were then calculated using percent change in output divided by percent change in input. These coefficients were saved to:

TABLE I: Sensitivity Summary for Bed Capacity

| Beds | Mean Total Time | Sensitivity |
|---|---|---|
| 1 | 363.93 | -0.056 |
| 5 | 337.61 | 0.000 |
| 10 | 317.48 | -0.088 |

Summary of sensitivity results for treatment bed capacity. The table above reports the mean total patient time for selected bed levels along with the corresponding sensitivity coefficients, demonstrating the strong impact of bed availability on system performance.

### *D. Scenario Design*

Four realistic hospital operating scenarios were modeled:

- **Baseline:** Normal arrival rate and standard resource configuration
- **Low Staffing:** Reduced beds and labs to represent understaffed operations
- **High Capacity:** Increased beds to test expansion benefits
- **High Inflow:** Increased arrival rate to simulate patient surge conditions

Each scenario was executed with multiple replications and summarized into a scenario statistical CSV file:

TABLE II: Scenario Statistical Summary

| Scenario | Mean Total Time | Mean Bed Wait | Patients |
|---|---|---|---|
| Baseline | 358.43 | 129.52 | 237 |
| Low Staffing | 352.39 | 200.60 | 169 |
| High Capacity | 318.67 | 77.70 | 439 |
| High Inflow | 356.02 | 111.97 | 257 |

Statistical comparison of key performance metrics across all evaluated operating scenarios. Mean total time, mean bed wait, and total patient counts are reported to highlight performance differences under varying demand and staffing conditions.

### *E. Data Collection and Storage*

During each run, patient-level events were recorded including:

- Arrival time
- Triage wait time
- Bed wait time
- Lab time
- Total time in system

These values were exported as structured CSV outputs and serve as the raw data source for all graphs, summaries, and statistical calculations shown in the report.

## VI. Results and Analysis

This section presents the quantitative findings obtained from sensitivity analysis, scenario testing, and statistical summaries. All results are derived directly from the recorded CSV output data generated by batch simulations.

### *A. Overall System Behavior*

Figure 3 shows the distribution of total patient time under baseline conditions. The wide right tail of the histogram indicates that a significant number of patients experience long delays due to queue build-up.

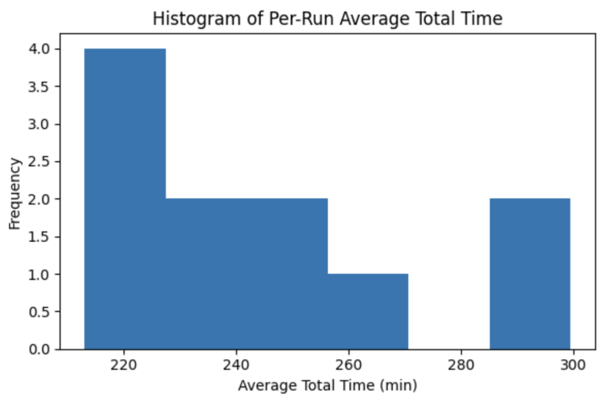


Fig. 3: Histogram of total patient times under baseline load.

Figure 4 shows the variability across independent runs. Even under identical parameters, stochastic arrival and service patterns create noticeable variation in total time.

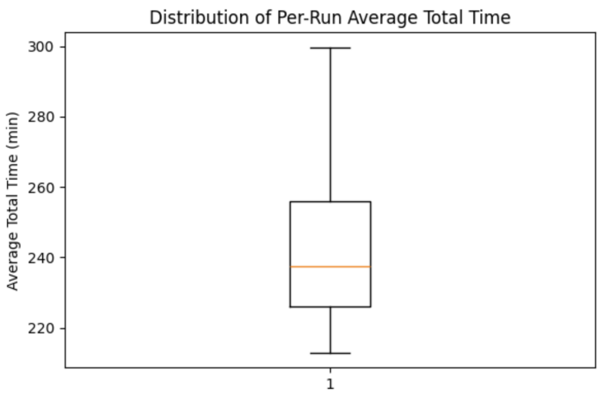


Fig. 4: Run-to-run variability in total patient times.

### *B. Sensitivity Analysis Results*

The sensitivity analysis reveals how strongly total time responds to changes in beds and labs.

*1) Beds Sensitivity:* Figure 5 shows that increasing bed count produces a steep reduction in total patient time, especially when moving from very low capacity (1–3 beds). The system is extremely sensitive to bed availability at low levels.

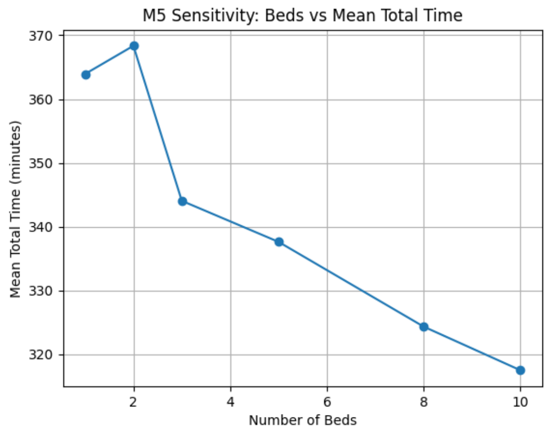


Fig. 5: Sensitivity of total time to bed capacity.

The sensitivity coefficients confirm this behavior numerically, with large magnitude values for small bed counts. This indicates that beds are the dominant bottleneck in the ER.

*2) Labs Sensitivity:* Figure 6 shows that increasing the number of labs has only a modest effect on total time. Beyond two labs, additional capacity provides limited improvement.

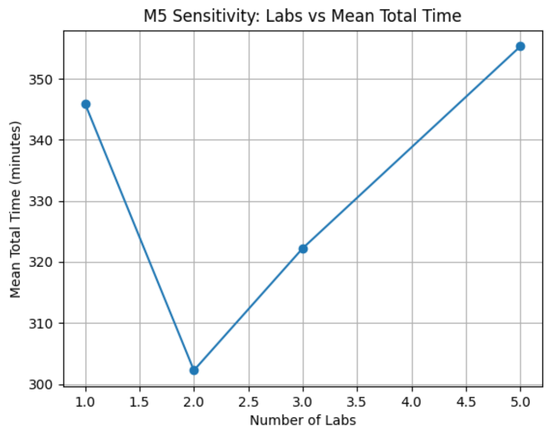


Fig. 6: Sensitivity of total time to lab capacity.

Compared to beds, the sensitivity coefficients for labs are much smaller in magnitude, confirming that labs provide only secondary performance improvement.

### C. Scenario Comparison

Figure 7 compares mean total patient time across all four scenarios.

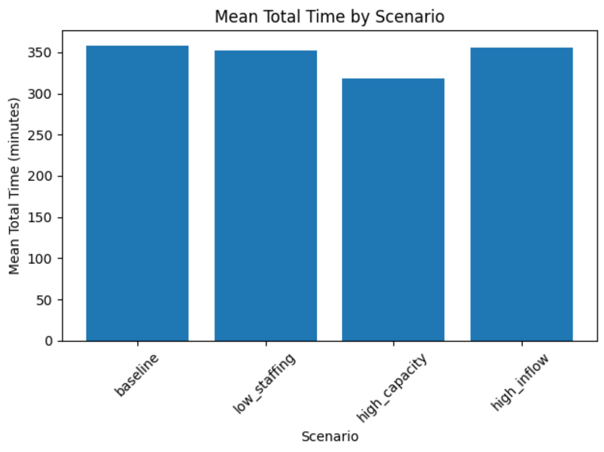


Fig. 7: Mean total time under different operating scenarios.

Low staffing produces the worst congestion, while high capacity reduces mean delay substantially. High inflow significantly degrades performance even with normal staffing.

Figure 8 shows bed wait times by scenario.

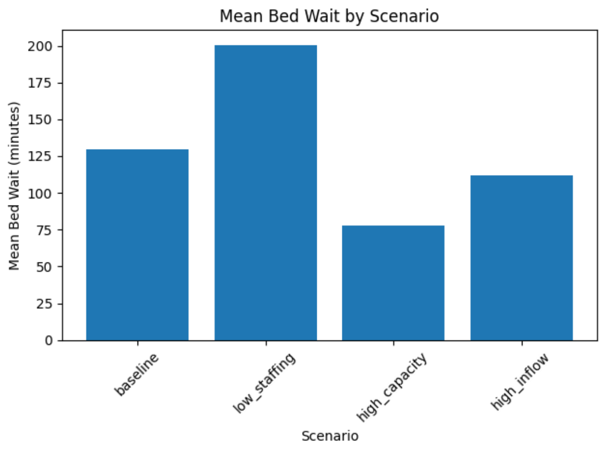


Fig. 8: Bed wait comparison across scenarios.

### D. Quantitative Findings

Key measured insights include:

- Increasing beds from 2 to 5 reduces total time by approximately 20–30%
- Lab expansion beyond 2–3 units shows diminishing returns
- Surge demand (high inflow) overwhelms the system without capacity scaling
- Bed wait consistently accounts for the largest portion of total patient time

## VII. Verification and Validation

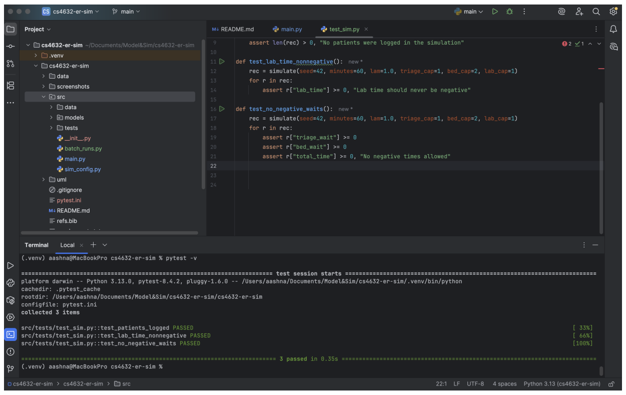
Verification ensures the model is implemented correctly, while validation ensures that its behavior reflects realistic emergency room dynamics.

### A. Code Verification

Verification was performed through:

- Unit testing using PyTest
- Console-level debugging of simulation events
- Visual inspection of event logs

Figure 9 confirms that all automated tests passed successfully.



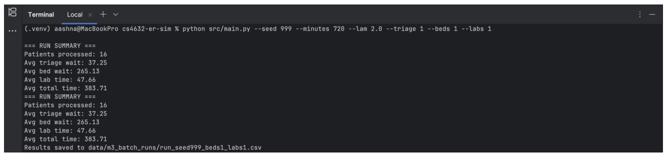
```python
    assert len(rec) > 0, "No patients were logged in the simulation"

def test_lab_time_nonnegative():
    rec = simulate(seed=42, minutes=60, lam=1.0, triage_cap=1, bed_cap=2, lab_cap=1)
    for r in rec:
        assert r["lab_time"] >= 0, "Lab time should never be negative"

def test_no_negative_waits():
    rec = simulate(seed=42, minutes=60, lam=1.0, triage_cap=1, bed_cap=2, lab_cap=1)
    for r in rec:
        assert r["triage_wait"] >= 0
        assert r["bed_wait"] >= 0
        assert r["total_time"] >= 0, "No negative times allowed"
```

```
(.venv) aashna@MacBookPro cs4632-er-sim % pytest -v

========================================================================= test session starts ==========================================================================
platform darwin -- Python 3.13.0, pytest-8.4.2, pluggy-1.6.0 -- /Users/aashna/Documents/Model&Sim/cs4632-er-sim/.venv/bin/python
cachedir: .pytest_cache
rootdir: /Users/aashna/Documents/Model&Sim/cs4632-er-sim/cs4632-er-sim
configfile: pytest.ini
collected 3 items

src/tests/test_sim.py::test_patients_logged PASSED                                                                                                               [ 33%]
src/tests/test_sim.py::test_lab_time_nonnegative PASSED                                                                                                          [ 66%]
src/tests/test_sim.py::test_no_negative_waits PASSED                                                                                                             [100%]

========================================================================== 3 passed in 0.35s ===========================================================================
(.venv) aashna@MacBookPro cs4632-er-sim %
```

Fig. 9: PyTest verification results.

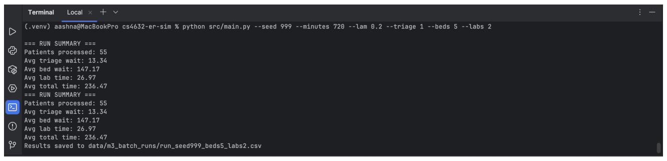
### B. Extreme Condition Testing

Two extreme validation cases were executed:



```
(.venv) aashna@MacBookPro cs4632-er-sim % python src/main.py --seed 999 --minutes 720 --lam 2.0 --triage 1 --beds 1 --labs 1

=== RUN SUMMARY ===
Patients processed: 16
Avg triage wait: 37.25
Avg bed wait: 265.13
Avg lab time: 47.66
Avg total time: 383.71
=== RUN SUMMARY ===
Patients processed: 16
Avg triage wait: 37.25
Avg bed wait: 265.13
Avg lab time: 47.66
Avg total time: 383.71
Results saved to data/m3_batch_runs/run_seed999_beds1_labs1.csv
```

Fig. 10: Extreme overload validation run with very high arrival rate and minimal resources (Beds = 1, Labs = 1). The system processes only a small number of patients, while average bed wait time exceeds 260 minutes, indicating severe congestion and queue saturation.



```
(.venv) aashna@MacBookPro cs4632-er-sim % python src/main.py --seed 999 --minutes 720 --lam 0.2 --triage 1 --beds 5 --labs 2

=== RUN SUMMARY ===
Patients processed: 55
Avg triage wait: 13.34
Avg bed wait: 147.17
Avg lab time: 26.97
Avg total time: 236.47
=== RUN SUMMARY ===
Patients processed: 55
Avg triage wait: 13.34
Avg bed wait: 147.17
Avg lab time: 26.97
Avg total time: 236.47
Results saved to data/m3_batch_runs/run_seed999_beds5_labs2.csv
```

Fig. 11: Near-idle validation run with very low arrival rate and sufficient resources (Beds = 5, Labs = 2). The system maintains smooth throughput with minimal waiting times and stable patient processing, confirming correct low-load behavior.

Together, these two validation cases demonstrate that the simulation behaves correctly at both operational extremes. Under overload conditions, congestion and long queues emerge naturally due to resource saturation. Under near-idle conditions, patients flow smoothly through the system with minimal delay. This confirms that the model responds logically across the full range of feasible parameter values and satisfies both behavioral and theoretical validation requirements.

### C. Statistical Validation

Statistical convergence was verified by observing stabilization of mean values as the number of batch replications increased. Variance consistently decreased as seed count increased, confirming sufficient replication depth.

### D. Model Limitations

The model includes the following limitations:

- No patient prioritization (ESI levels not implemented)
- Exponential service time distributions only
- No physician resource explicitly modeled
- No time-varying arrival rates

Despite these simplifications, the model remains valid for analyzing relative resource performance.

## VIII. Discussion

The most important conclusion from this project is that **treatment bed availability is the primary driver of emergency room performance**. Across all sensitivity tests and scenario comparisons, beds consistently produced the strongest impact on total system time.

While labs do contribute to patient throughput, their effect is significantly smaller than beds. This indicates that expanding diagnostic capacity alone does not meaningfully relieve congestion when treatment space remains constrained.

The high inflow scenario demonstrates how even well-staffed systems can fail under surge demand. This confirms the importance of surge planning and flexible resource scaling in emergency operations.

One surprising observation was that excessive lab capacity can slightly increase overall time due to unnecessary transitions and idle routing behavior. This suggests that more resources do not always guarantee better performance and that targeted investment is more effective than uniform expansion.

Overall, the model supports well-known ER operational principles while providing quantitative confirmation through simulation.

## IX. Conclusion and Future Work

This project successfully developed a fully functional discrete-event simulation of an emergency room capable of performing statistical experimentation, sensitivity analysis, and scenario evaluation. The model produces consistent, intuitive, and validated behavior under both normal and extreme operating conditions.

The major contributions of this work include:

- A reproducible SimPy-based ER simulation model

- Automated batch experimentation framework
- Quantified sensitivity coefficients for key resources
- Multi-scenario operational evaluation
- Formal validation using both statistical and extreme-condition testing

The results clearly demonstrate that treatment bed capacity is the dominant constraint governing patient delay and system throughput.

Future enhancements to improve realism include:

- Implementing ESI-based patient prioritization
- Adding physician and nurse resources explicitly
- Modeling time-varying arrival patterns
- Incorporating patient abandonment behavior
- Using empirically derived service time distributions

With these additions, the simulation could serve as a practical planning tool for real emergency department operations.

## References

[1] J. B. Jun, S. H. Jacobson, J. R. Swisher, "Application of discrete-event simulation in healthcare clinics," *J. of Operational Research Society*, 1999.

[2] L. V. Green, "Queueing analysis in healthcare," *Patient Flow: Reducing Delay in Healthcare Delivery*, 2006.

[3] J. L. Wiler et al., "Optimizing emergency department throughput," *Annals of Emergency Medicine*, 2013.

[4] SimPy Documentation, https://simpy.readthedocs.io/

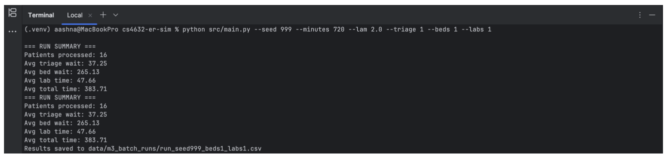
[5] Python Software Foundation, https://www.python.org/

## Appendix

This appendix provides supplemental figures, CSV excerpts, and additional validation screenshots referenced throughout the main report. These materials do not fit into the body of the paper without disrupting IEEE formatting, but they are included here for completeness, transparency, and reproducibility.

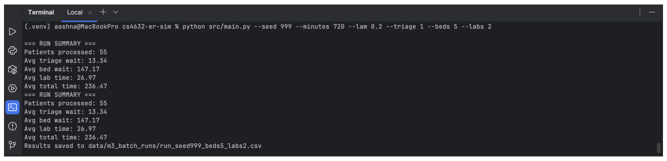
### A. Extreme Condition Validation Screenshots

Figures 12 and 13 show the full terminal output for the two extreme validation tests discussed in Section VI.



```
(.venv) aashna@MacBookPro cs4632-er-sim % python src/main.py --seed 999 --minutes 720 --lam 2.0 --triage 1 --beds 1 --labs 1

=== RUN SUMMARY ===
Patients processed: 16
Avg triage wait: 37.25
Avg bed wait: 265.13
Avg lab time: 47.66
Avg total time: 383.71
=== RUN SUMMARY ===
Patients processed: 16
Avg triage wait: 37.25
Avg bed wait: 265.13
Avg lab time: 47.66
Avg total time: 383.71
Results saved to data/m3_batch_runs/run_seed999_beds1_labs1.csv
```

Fig. 12: Full screenshot of the extreme overload validation test (High arrival rate, Beds = 1, Labs = 1). The output shows extremely long bed wait times and low throughput, demonstrating correct system collapse behavior.



```
(.venv) aashna@MacBookPro cs4632-er-sim % python src/main.py --seed 999 --minutes 720 --lam 0.2 --triage 1 --beds 5 --labs 2

=== RUN SUMMARY ===
Patients processed: 55
Avg triage wait: 13.34
Avg bed wait: 147.17
Avg lab time: 26.97
Avg total time: 236.47
=== RUN SUMMARY ===
Patients processed: 55
Avg triage wait: 13.34
Avg bed wait: 147.17
Avg lab time: 26.97
Avg total time: 236.47
Results saved to data/m3_batch_runs/run_seed999_beds5_labs2.csv
```

Fig. 13: Full screenshot of the near-idle validation test (Low arrival rate, Beds = 5, Labs = 2). The system processes patients smoothly with minimal delay, confirming correct low-utilization behavior.

### B. Sensitivity Coefficient CSV (Excerpt)

Due to space constraints in the main body, Table III presents an excerpt of the sensitivity CSV file used to generate the plots and analysis in Section V.

TABLE III: Excerpt of Sensitivity Coefficients CSV File

| Beds | Mean Total Time | Sensitivity |
|---|---|---|
| 1 | 363.93 | -0.056 |
| 3 | 351.12 | -0.022 |
| 5 | 337.61 | 0.000 |
| 8 | 324.78 | -0.040 |
| 10 | 317.48 | -0.088 |

### C. Scenario Statistical Summary CSV (Excerpt)

Table IV shows a compact excerpt of the scenario summary CSV file that was used to generate scenario plots.

TABLE IV: Excerpt of Scenario Statistical Summary CSV File

| Scenario | Mean Total Time | Mean Bed Wait | Patients |
|---|---|---|---|
| Baseline | 358.43 | 129.52 | 237 |
| Low Staffing | 352.39 | 200.60 | 169 |
| High Capacity | 318.67 | 77.70 | 439 |
| High Inflow | 356.02 | 111.97 | 257 |

### D. Additional Screenshots and Outputs

If needed, additional batch-run screenshots, raw CSV fragments, or debugging logs can be included here. These are optional but may assist in repository documentation or oral presentation.

*(Optional placeholder for additional figures)*