SIMULATION MANUAL

Data-Driven, Multimodal Freight Simulation Framework for Waterways and Ports

Simulation, Parking, Autonomous Vehicles, Routing, & Traffic Assignment Lab, The University of Texas at Austin

Version: 1.0
Date: September 3, 2025
Authors: Debojjal Bagchi, Kyle Bathgate, Stephen D. Boyles
Email:

Purpose

This manual provides instructions for preparing inputs, running scenarios, and interpreting outputs for the discrete-event simulation framework that models multimodal port operations across container, liquid bulk, and dry bulk terminals. It is intended for use of port modelling, bottleneck detection and scenario analysis.

Contents

1 Introduction and Scope

This framework is an open-source, modular, discrete-event simulation (DES) system, developed in Python using `simpy`, for analyzing port and waterway operations across three cargo domains: container, liquid bulk (tankers), and dry bulk/general cargo. The system supports planning, bottleneck detection, what-if scenario analysis, and capacity estimation based on real-world data and calibration.

The simulation framework is designed for ports organized with an **anchorage area** followed by terminals located along a **navigational channel**. Dummy data for a fictitious port are included for demonstration. The fictitious port is shown in Figure 1.1

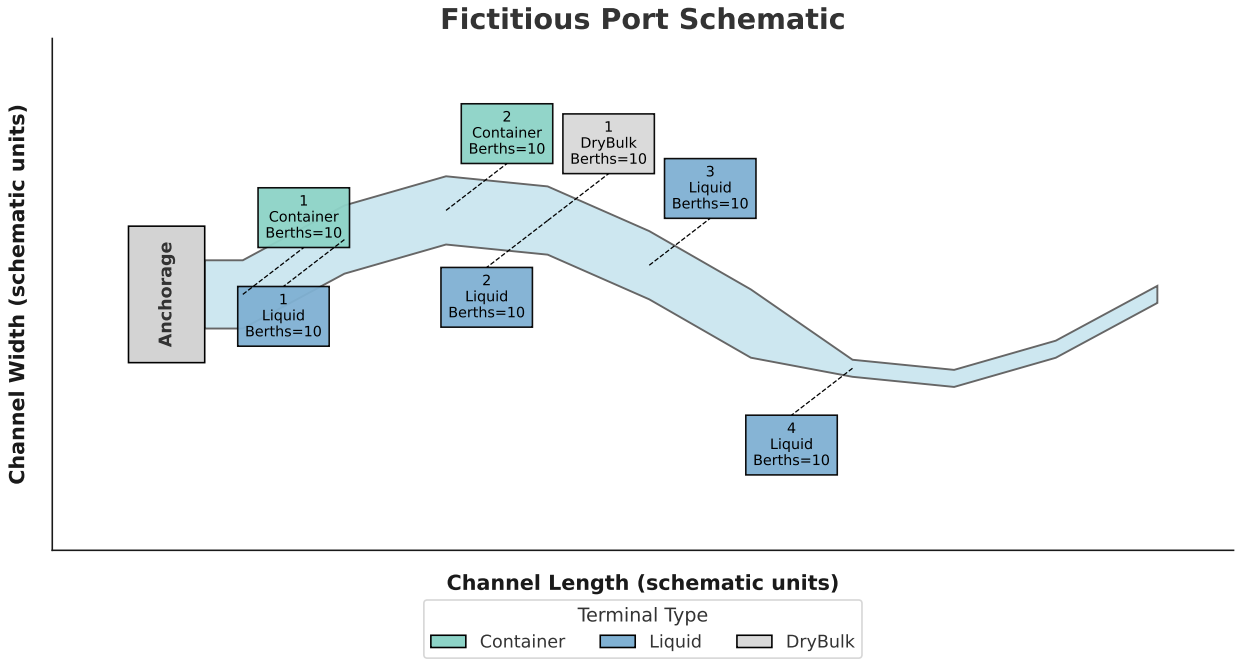

Figure 1.1: Fictitious port

Currently, the model supports:

- **Vessel types:** Container ships, tankers (liquid bulk), and general cargo/bulk vessels
- **Landside connections:** Truck, rail, and pipeline interfaces

1.1 Intended Use and Scope

The model aids public agencies and port authorities in:

- Identifying system bottlenecks across berths, cranes/pipelines/conveyors, pilots, tugboats, and navigational constraints.
- Analyzing vessel and cargo throughput and dwell/turnaround times under baseline and disrupted conditions.
- Evaluating performance under what-if scenarios (e.g., resource changes, disruptions, and arrival surges).
- Quantifying operating and ultimate capacity from simulation and analytical ODE approximations.

1.2 License

2 System Overview

2.1 Concept of Operations

Ports are modeled with an anchorage, a navigational channel segmented into sections, and terminals located along those sections. Vessels traverse the channel subject to speed, depth, beam, draft, daylight, and traffic-direction rules. Terminals process cargo using resources such as berths, cranes/pipelines/conveyors, gates, and storage. Inland modes (truck, rail, and pipelines) interact with terminal storage and transfer operations.

2.2 Capabilities

- **Vessel Types:** Container, Liquid (tankers), Dry Bulk/General Cargo.
- **Landside Connections:** Truck, rail, and pipeline interfaces.
- **Traffic Rules:** Pilot, tugboat, draft, beam, daylight, and minimal spacing restrictions.
- **Run modules:** Base simulation, bottleneck detection, capacity analysis, and disruption analysis (fog & hurricane)
- **Analytics:** Resource utilization; queue length and time distributions; dwell and turn times; channel restriction analysis; capacity estimation (operating and ultimate).

2.3 Code documentation

HTML API documentation is generated in `simulation_documentation/` folder (open `simulation_documentation/index.html` in a browser).

3 Required Inputs and Data Preparation

3.1 Overview

Accurate input preparation is essential before running the simulation. The following elements must be specified: channel geometry, fleet mix, terminal resources and throughput, landside connections, and operational activity flags. Dummy data are provided for demonstration, consisting of **10 channel sections**, **2 container terminals**, **4 liquid terminals**, and **1 dry bulk terminal**.

Before running a new port simulation, prepare:

- **Channel geometry:** Divide the navigational channel into sections where terminals are located. Each section requires lengths, widths, depths, and vessel speeds. The dummy dataset provides 10 channel sections. Multiple terminals can be located within a single section; the simulation will treat them as colocated.
- **Fleet mix:** Define vessel classes with size, draft, tonnage, and tug/pilot requirements. The dummy dataset assumes three vessel sizes (small, medium, large) across container, liquid bulk, and dry bulk categories.
- **Terminal resources:** Specify berths, cranes, pipelines, conveyors, and transfer rates.
- **Landside connections:** Configure truck and rail arrival rates, gates, storage capacities, payloads, and terminal connectivity.
- **Activity flags:** Identify terminals that are import-only, export-only, or allow both. Include pipeline connectivity where applicable.

Once essential data have been obtained, the following files should be updated replacing the dummy data.

Edit only the base files

Edit `config.py`, `inputs/channel_dimensions.csv`, `inputs/ship_sizes.csv`, and `inputs/terminal_data_base.csv`.
Do not edit `constants.py` or `inputs/terminal_data.csv`; these are internal copies generated by the system and may be overwritten.

3.2 File `config.py` (Global Run and Port Controls)

Purpose. The file `config.py` serves as the central control point for all simulation runs. It defines global parameters such as the simulation horizon, number of replications, random seed behavior, scenario toggles, and port-wide operational rules. It also specifies navigation restrictions, port resources, vessel arrival processes, and efficiency multipliers. This file must be carefully updated before every new simulation run, since all other modules depend on the values defined here.

Key Categories of Inputs.

- **Simulation Setup:** Number of months to simulate; warm-up period; number of random seeds/replications; CPU core allocation; run identifiers (`SCENARIO_NAME`); and logging/output cleanup toggles.
- **Scenario Toggles:** Binary flags for modeling extreme conditions (e.g., hurricane, fog), expanded channels, or what-if analyses (truck/pipeline overrides).
- **Navigation Restrictions:** Daylight-only navigation (start/stop hours); inbound/outbound closures (e.g., due to fog); maximum draft (ft) and beam (ft); two-way channel rules; entry spacing between vessels; and channel segment limitations for larger ships.
- **Port Resources:** Number of pilots (day/night ranges), tugboats, and service time ranges for channel entrance, tug steerage, and vessel turnaround.
- **Vessel Arrivals:** Poisson fit interarrival times (derived from AIS/port call data); arrival rate multipliers for scenario stress-tests; truncation limits for maximum interarrival times; and cargo-specific adjustments.
- **Efficiency and Anchorage Waits:** Calibrated anchorage waiting times (hrs) by cargo class; efficiency multipliers (0–1) for Container, Liquid, and Dry Bulk terminals.

- **Cargo Conversion Factors:** Non-cargo deadweight percentages; liquid density conversion factors (CBM/ton for crude oil, diesel, etc.); container payload conversions (20' vs. 40').

- **Landside Connections:** This field specifies how cargo is transferred between the terminal and external networks.

  - **Truck / Rail Availability Overrides:** For each terminal, the default assumptions about how many trucks or rail slots are available can be overridden here. If a terminal do not have rail or truck connection, it should be mentioned here. **The values override the terminal data input files.**

  - **Pipeline Transfer Rates (cbm/timestep):** Unlike berths that serve vessels directly, pipelines are modeled as continuous source/sink connections linking liquid bulk terminals with external tank farms or refineries. They do not represent vessel–terminal interactions, but instead specify the maximum flow capacity for pumping cargo in or out of storage per simulation step.

- **Daylight Windows:** Standard restriction (07:00–19:00) and expanded channel restriction (04:00–21:00) have been currently set and should be updated for port in consideration. Large vessels with beam greater than certain values are not allowed during day light restricted hours. These parameters must be updated for the port being modeled, based on local pilotage authority rules and operating manuals. The values are set in the function `simulation_classes/channel.daylight_restriction()`, which determines when a vessel can legally enter or transit the channel.

- **Narrow Channel Segments:** Some channel sections physically prevent wider ships from advancing beyond certain terminals. This is controlled by the parameters `NO_LARGE_SHIP_BEYOND_THIS_TERMINAL_CTR`, `NO_LARGE_SHIP_BEYOND_THIS_TERMINAL_LIQ`, and `NO_LARGE_SHIP_BEYOND_THIS_TERMINAL_DRYBULK`. These values specify the first terminal segment after which only smaller vessels are allowed to proceed. For example, setting `=2` for containers means that any container vessel wider than the threshold beam cannot transit past Terminal 2. The threshold is set as `MAX_BEAM_SMALL_SHIP` variable.

Guidance for modeling new ports. When adapting this framework to a new port system:

- **Simulation Setup:** Determine horizon and replications based on study design. Choose warm-up length to eliminate initialization bias.

- **Vessel Arrivals:** Estimate interarrival rates from AIS datasets and port call statistics. See Bathgate et al. (2025)[1] for recommended estimation procedures.

- **Efficiency and Anchorage Waits:** These parameters capture delays and productivity losses that are not explicitly modeled elsewhere in the simulation:

  - **Anchorage Waiting Times (hrs):** Represent unexplained or systemic delays before a vessel can enter the port or berth. For example, congestion, paperwork, or external factors (customs, inspections) may delay movement even if berths appear available in the model. Start by setting these to zero, then increase them gradually until simulated anchorage times align with observed averages.

  - **Terminal Efficiency Multipliers (0–1):** Scale the effective service rate at container, liquid, and dry bulk terminals. A value of 1.0 means the terminal operates at full design capacity with no slowdowns. Lower values reflect real-world inefficiencies such as crane downtime, workforce limitations, weather disruptions, or berth underutilization. Begin with 1.0 and reduce as needed during calibration until simulated dwell times match observed records.

  Together, these variables ensure that the model reproduces realistic vessel dwell and turnaround times by accounting for "hidden" inefficiencies not captured by physical resources alone.

- **Channel and Terminals:** Map the port into sequential channel sections where terminals are located; extract length, width, and depth from nautical charts. Enter actual berth counts and terminal IDs from port authority databases.

Important Notes.

- Default values in the distributed file are *dummy values* for the port shown in Figure 1.1; these must be replaced with actual port data prior to official runs.

- Clean-up toggles (`DELETE_EVERYTHING`, `DELETE_RESULTS_FOLDER`) should be used cautiously to prevent accidental loss of prior runs.

[1]Bathgate, Kyle, Debojjal Bagchi, and Stephen D. Boyles. *Use of AIS data to characterize vessel mix in Houston port operations for simulation.* Presented at the 104th Annual Meeting of the Transportation Research Board, Washington, D.C., 2025.

3.3 File `inputs/channel_dimensions.csv` (Channel Segments)

Purpose. Defines navigational segments for movement and restriction checks.

How the Channel is Modeled. The navigational channel is discretized into multiple sections arranged in order from the seaward entrance to the farthest inland terminal. Each **SectionID** acts as a waypoint:

- Vessels enter the first section from the anchorage.
- They proceed through subsequent sections, consuming time according to section **length** and permitted **speed**.
- Beam, draft, daylight and beam restriction are checked in each section.
- Terminals are mapped to particular sections; once a vessel reaches its assigned section, it attempts to dock at one of the berths defined for that terminal.

This structure allows realistic modeling of ports where terminals are spread along a long waterway, and where narrower or shallower stretches constrain certain vessel types.

Columns:

- **SectionID** — unique integer identifier.
- **length (miles)** [mile].
- **width (ft)** [ft].
- **depth (ft)** [ft].
- **speed (mph)** [mph].

Default values: Default values in the distributed file are *dummy values* for the port shown in Figure 1.1; these must be replaced with actual port data prior to official runs.

3.4 File `inputs/ship_sizes.csv` (Vessel Classes)

Purpose. Representative classes by cargo type with geometry, tonnage, and tug/pilot needs. Every vessel generated in the simulation is sampled from normal or uniform distribution with parameters set in this file.

Columns. Each row of this file represents a vessel class (defined by cargo type and size) and provides the distributional parameters needed to generate ships in the simulation.

- **ship_type**: Cargo category of the vessel. Allowed values are `Container`, `DryBulk`, and `Liquid`. Used to assign the vessel to a compatible terminal.
- **Size**: Class label (*Small*, *Medium*, or *Large*). This categorization allows multiple representative vessel sizes within each cargo type.
- **Avg_Length**, **Std_Length** [ft]: Average and standard deviation of vessel length. Sampled during ship generation to represent fleet variation.
- **Avg_Beam**, **Std_Beam** [ft]: Average and standard deviation of vessel beam (width). Used to check beam and daylight restrictions.
- **Avg_Draft**, **Std_Draft** [ft]: Average and standard deviation of vessel draft. Used to check draft restrictions.
- **Avg_Tonnage**, **Std_Tonnage** [ton]: Distribution of deadweight tonnage (DWT). Determines cargo capacity and influences turnaround times at terminals.
- **Count**, **Fraction**: Fleet mix calibration.
  - **Count:** number of vessels of this class observed in AIS/port call records.
  - **Fraction:** proportion of this class within its cargo category (sums to 1.0 for each `ship_type`).

  These ensure that generated vessels reflect the observed distribution of fleet sizes.
- **min_pilots/max_pilots** [count]: Minimum and maximum number of pilots required for this vessel class. Reflects real-world pilotage rules (larger vessels may need multiple pilots).
- **min_tugs/max_tugs** [count]: Minimum and maximum number of tugboats required for safe maneuvering. Larger or less maneuverable ships require more tug assistance.

Data sourcing. Cluster AIS and port call data to derive classes and shares; validate tug/pilot requirements with local providers.

3.5 File `inputs/terminal_data_base.csv` (Terminal Resources & Landside)

Purpose. Declares per-terminal resources, storage, and landside throughput. The system copies this to `inputs/terminal_data.csv` internally.

Random ranges. Any numeric field may be specified as `a,b` to indicate uniform sampling in $[a, b]$ per seed.

Columns:

- **Cargo**: `Container`, `Liquid`, `DryBulk`.
- **Terminal**: identifier (string).
- **Segment**: channel **SectionID** where the terminal is located.
- **Berths**: count.
- **import/export**: binary flags (0/1).
- **storage volume**: TEU, FEU (container); tons (dry bulk); CBM (liquid).
- **transfer units per berth**: cranes/pipelines/conveyors (count).
- **transfer rate per unit**: moves/hour (container), tons/hour (dry), CBM/hour (liquid).
- **truck arrival rate**: trucks/hour; **truck gates**; **truck bays/chassis** (count).
- **truck loading/unloading rate**: per hour; **truck payload size**: FEU/TEU or tons/truck.
- **train arrival rate**: trains/hour; **train car storage**; **train car loading bays** (count).
- **train cargo transfer rate**: per hour; **train car payload size**: TEU, FEU/tons/CBM per car; **train car amount**: cars/train.
- **pipeline source/pipeline sink**: import/export via pipeline (0/1).

Data sourcing. Terminal operator fact sheets, planning studies, and interviews.

4 Run Procedures and Scenario Execution

4.1 Environment Setup

Before running the simulation, set up a Python virtual environment and install the required packages:

```
python -m venv .venv
source .venv/bin/activate      # Windows: .venv\Scripts\activate
pip install -r requirements.txt
```

4.2 Base Simulation

To run a standard (base) simulation:

1. Edit `config.py` to set global parameters (simulation horizon, seeds, resources, etc.).
2. Execute the simulation:

   ```
   python main.py
   ```

Outputs (plots, logs, and statistics) are written into per-seed directories named `.Results_*`. Parallel execution automatically uses the CPU core count specified in `config.py`.

4.3 Bottleneck Detection Scenario

The bottleneck analysis procedure systematically tests how sensitive port performance is to reductions in specific resources (e.g., berths, storage, transfer rates, pilots, tugboats, truck/rail arrivals). By applying controlled reductions (typically ±10%; cen be set in

`bottleneck_analysis.py`) to one resource at a time, the analysis identifies which elements constrain overall capacity and where delays or queues grow fastest. The outputs allow the user to rank-order bottlenecks and quantify their impact on system throughput.

1. Set `SCENARIO_NAME = "BottleneckDetection"` in `config.py`.
2. Run the bottleneck analysis script:

```
python bottleneck_analysis.py
```

Results are written under `bottleneckAnalysis/`, with sensitivity logs and a consolidated capacity report.

Running Multiple Scenarios. The main bottleneck script allows to test multiple scenarios in a single batch run. In the file `bottleneck_analysis.py`, a scenario list is defined at the end of the script:

```
# Example: to run multiple cases
scenario_name_list = [
   'Base',
   'LowCtrBerths', 'LowCtrStorage', 'LowCtrTransfer',
   'LowLiqBerth', 'LowLiqStorage', LowLiqTransfer',
   LowDryBlkBerth', LowDryBlkStorage', LowDryBlkeTransfer',
   LowPilots', HighPilots', LowTugs', HighTugs',
   SlowTruckArrival', FastTruckArrival',
   SlowTrainArrival', FastTrainArrival'
]
```

By uncommenting and editing this list, the modeler may pick and choose any combination of scenarios. Each entry triggers an automated run where the script adjusts `terminal_data.csv` or `constants.py`, executes `main.py` under multiple arrival factors, and records results.

This design allows the user to explore both single-resource bottlenecks and combinations in a structured, repeatable way.

Standalone operation. The bottleneck analysis can be run independently of other scripts. It uses the simulation framework defined in `main.py` but performs its own parameter updates, execution loops, and plotting routines.

4.4 Ultimate Capacity Breakpoint Analysis

The capacity breakpoint analysis is used to determine the point at which the port system becomes unstable, meaning that vessel queues begin to grow rapidly rather than remaining bounded. This provides an estimate of the *ultimate capacity* of the system under given resource constraints. The method works by gradually increasing vessel arrival rates, running the simulation at each level, and then fitting an ODE-based capacity model to the results.

Operating capacity estimates are done in every base case simulation run; however to get ultimate capacity several simulation runs are needed. For more details see Bagchi et al. (2025)[1].

1. Set `SCENARIO_NAME = "BreakpointAnalysis"` in `config.py`.
2. Run the breakpoint analysis script:

```
python breakpoint_analysis.py
```

What the script does. The breakpoint script automates a series of simulation runs:

- Updates `constants.py` with new arrival-rate multipliers.
- Runs `main.py` repeatedly at increasing arrival rates.
- Extracts results from log files (queue lengths, channel utilization, ships entered/exited).
- Plots throughput curves (entered vs. exited ships) and anchorage queue growth for each cargo type.
- Fits an ODE model to the observed exit rates, yielding two key parameters:
  - C_u: ultimate capacity of the port system.
  - θ: shape parameter governing the steepness of the transition from stable to unstable.

Outputs. Results are written under `breakpointAnalysis/` as both figures (`.pdf`) and CSV data files, including:

[1]Bagchi, Debojjal, Kyle Bathgate, and Stephen Boyles. *A queuing-theory-based operating capacity model for multimodal port operations.* Presented at the 104th Annual Meeting of the Transportation Research Board, Washington, D.C., 2025.

- Throughput plots for all ships and by cargo type.
- Anchorage queue length plots (per cargo and combined).
- Channel utilization plots with error bars.
- CSV files recording arrival rate multipliers, mean entered/exited ships, and utilization statistics.
- ODE fit plots showing C_u and θ.

Running multiple stress levels. By default, the script defines a list of arrival-rate multipliers (e.g., 0.5, 1.0, 1.5, ..., 4.0). These are iterated automatically to test system stability across a range of traffic intensities. Users can modify this list inside `breakpoint_analysis.py` to expand or narrow the tested range.

Standalone operation. The breakpoint analysis can be run independently of other scripts. It uses the simulation framework defined in `main.py` but performs its own parameter updates, execution loops, and plotting routines.

4.5 User-Editable Input Files

Only the following files should be directly modified by the user:

- `config.py` — defines global run controls (user-editable). *Note: `constants.py` is auto-generated and must not be edited.*
- `inputs/terminal_data_base.csv` — baseline terminal data (user-editable). *Note: `inputs/terminal_data.csv` is system-generated from the base file.*
- `inputs/channel_dimensions.csv` — channel section definitions (user-editable).
- `inputs/ship_sizes.csv` — fleet composition and vessel class definitions (user-editable).

5 Outputs and Interpretation

5.1 Per-Seed Output Directory

Each simulation seed generates a dedicated results folder named with the following convention: `.Results_<SEED>_<DURATION>_<VERSION>/` . This directory contains the following subfolders and files:

- `bottlenecks/` — Summary statistics of resource utilization, including pilots, tugboats, and terminal berths.
- `logs/` — Detailed event logs recording vessel arrivals, departures, allocations, and time stamps.
- `plots/` — Diagnostic figures organized into subfolders:
  - `DwellTimes/` — Raw dwell time traces for individual vessels.
  - `DwellTimesDist/` — Histograms of dwell time distributions.
  - `TurnTimes/` — Raw turnaround times (anchorage + dwell).
  - `TurnTimesDist/` — Histograms of turnaround time distributions.
  - `shipDistribution/` — Vessel type and size class distribution plots.
  - `truckDistribution/` — Truck arrival and service distribution plots.
  - `TruckArrival/` — Time series of truck arrivals.
  - `TruckDwell/` — Histograms and plots of truck dwell times.
  - `TerminalProcessCharts/` — Process-flow charts for individual terminals.
  - `TerminalProcessDist/` — Histograms of terminal-level process times.
  - `scenarios/` — Scenario-specific charts and diagnostic outputs.

- **Selected Key Figures and Files:**
  - `dwell_time_distribution.jpg` — Histogram of vessel dwell times.
  - `turn_time_distribution.jpg` — Histogram of vessel turnaround times.
  - `anchorage_queue_all.pdf` — Anchorage queue length plot for all vessel types.
  - `anchorage_queue_container.pdf` — Anchorage queue plot for container ships.
  - `anchorage_queue_liquid.pdf` — Anchorage queue plot for liquid bulk ships.
  - `anchorage_queue_drybulk.pdf` — Anchorage queue plot for dry bulk ships.
  - `channel_utilization.pdf` — Average and variability of vessels using the channel.
  - `ChannelSections.pdf` — Diagram of modeled channel sections.
  - `pilots_availability.pdf` — Pilot utilization over time.
  - `tugs_availability.pdf` — Tugboat utilization over time.
  - `moves.pdf` — Summary of crane movements.

5.2 Collated Results

After all seeds complete, results are aggregated under `collatedResults/`. Each run is stored in a separate subdirectory, for example: `collatedResults6_months_2_runs_run_100/`

This directory typically includes:

- `data/` — Combined CSV outputs from all seeds.
- `distPlots/` — Aggregate dwell and turnaround distributions.
- `queuePlots/` — Combined anchorage and system queue charts.
- `restrictionPlots/` — Channel and daylight restriction visualizations.
- `timePlots/` — Time-series plots across cargo types.
- Key reports: `capacity_simulation_analysis.pdf`, `channel.pdf`, and run-specific details such as `run_details_run_100.txt`.

6 Software Architecture and Modules

6.1 Execution Modules

The following python files can be run for different kind of analysis:

Base Simulation (`main.py`): Executes the primary simulation, producing data analysis plots and tracking resource utilization over time. Also supports disruption scenarios such as fog and hurricanes (can be set on the config file).

Bottleneck Analysis (`bottleneck_analysis`): Identifies and evaluates system bottlenecks by varying arrival rates under reduced capacity of particular resource.

Capacity Analysis (`breakpoint_analysis.py`): Performs ultimate capacity analysis by varying arrival rates over a large number of points and fitting an ODE model.

6.2 Project Structure

```
.
 main.py                          # Master simulation controller
 breakpoint_analysis.py           # Analyze ultimate capacity
 bottleneck_analysis.py           # Finds resource bottlenecks

 config.py                        # Set input parameters for the simulation
 constants.py                     # Internal copy (DO NOT CHANGE)

 inputs/
    channel_dimensions.csv       # Channel dimensions
    ship_sizes.csv               # Vessel dimensions/classes
    terminal_data_base.csv       # Editable terminal attributes
    terminal_data.csv            # Internal copy (DO NOT CHANGE)
```

```
simulation_analysis/
   capacity.py                  # Operating & ultimate capacity functions
   collate_results.py           # Aggregation across runs
   resource_utilization.py      # Berth/crane/pipeline utilizations
   results.py                   # Plots and logs per seed
   whatif_scenarios.py          # What-if and sensitivity scripts

simulation_classes/
   channel.py                   # Navigational channel logic
   pipeline.py                  # Liquid pipeline model
   port.py                      # Port resources (terminals, pilots, tugs)
   terminal_container.py        # Container terminal operations
   terminal_drybulk.py          # Dry bulk terminal operations
   terminal_liquid.py           # Liquid terminal operations
   train.py                     # Rail interface
   truck.py                     # Gate-truck processes

simulation_handler/
   generators.py                # Arrival & demand generators
   helpers.py                   # Utilities
   preprocess.py                # Input validation/derivations
   run_simulation.py            # Entrypoint for programmatic runs

simulation_documentation/
   generate_documentation       # pdoc build script(s)
   home_logo.png                # Project logo
   *.html                       # HTML docs (open index.html)

.Results_<SEED>_<DURATION>_<VERSION>/  # Per-seed outputs
collatedResults/                        # Aggregated results
bottleneckAnalysis/                     # Bottleneck scenario outputs
breakpointAnalysis/                     # Breakpoint scenario outputs
```

7 Frequently Asked Questions

Q: What measurement units are used throughout the simulation?
A: The framework uses consistent engineering units: [hour] (time), [ft] (length, depth, beam, draft), [mph] (channel speed), [CBM] (cubic meters for liquid volume), [tons] (cargo weight and vessel deadweight), and [TEU, FEU] (container capacity). Always check data sources and convert to these units before entering values.

Q: How are terminals placed along the channel if channel segments and terminals are defined in different files?
A: Channel geometry is defined in `inputs/channel_dimensions.csv` (by **SectionID**), while terminals are defined in `inputs/terminal_data_base.csv`. Each terminal row includes a **SectionID** column. This value must match an existing **SectionID** in the channel file, so that the terminal is correctly assigned to a channel segment. If the IDs do not match, the simulation will not be able to locate the terminal along the waterway.

Q: Which files should I edit to change port inputs?
A: Only the following files are user-editable:

- `config.py` — global simulation controls and scenarios.
- `inputs/channel_dimensions.csv` — channel geometry (sections, widths, depths, speeds).
- `inputs/ship_sizes.csv` — fleet definitions by cargo type and vessel class.
- `inputs/terminal_data_base.csv` — baseline terminal data (berths, storage, transfer rates).

Do not edit `constants.py` or `inputs/terminal_data.csv`, as these are automatically generated during each run and will be overwritten. .

A Abbreviations and Units

AIS	Automatic Identification System
CBM	Cubic Meter (liquid volume unit)
DES	Discrete-Event Simulation
FEU	Forty-foot Equivalent Unit (container)
TEU	Twenty-foot Equivalent Unit (container)

B Run Checklists

Pre-Run Checklist

Files to Verify Before Running

- `inputs/channel_dimensions.csv` — Channel sections, lengths, widths, depths, and speeds updated for the study port.
- `inputs/ship_sizes.csv` — Vessel classes (Container, Liquid, Dry Bulk) with correct dimensions, tonnage, and tug/pilot requirements.
- `inputs/terminal_data_base.csv` — Terminal berths, storage, and transfer rates set for each terminal.
- `config.py` — Scenario name (`SCENARIO_NAME`), number of random seeds, CPU cores, and clean-up toggles adjusted.

Post-Run

- Verify `.Results_*/` created for each seed.
- Review queue and dwell distributions for anomalies.
- Check final results in `collatedResults/`.