Computer-aided simulations - Second Delivery

Alessandro Maini
s346219@studenti.polito.it

***Abstract*—This document provides an overview of the laboratory activities from 5 to 9 of the *Computer-aided simulations lab* course. In particular, these laboratories concern 3 main objectives: the generation and testing of random variables, the development of an airport lounge simulator and the development of a Markov co-evolution model.**

I. Generating and testing R.V.s

The first task undertaken is the implementation of functions to generate different types of random variables (R.V.s). The following R.V.s are considered:

- **Hyper-exponential**
- **Erlang-k**
- **Pareto**

A. Generation Approaches

For each type of R.V., a different generation approach is implemented.[1]

a) Hyper-exponential: Since the probability density function of a hyper-exponential can be expressed as the weighted sum of the PDFs of different exponential R.V.s, as can be seen in Equation 1, the most fitting approach to generate this kind of variables is the **composition method**.

$$f(x) = \sum_{i=1}^{n} p_i \lambda_i e^{-\lambda_i x}, \quad x \geq 0 \tag{1}$$

The composition method returns instances of the desired hyper-exponential random variable by applying the following steps iteratively:

1) Generate an instance of the random variable I, such that
$$\mathbb{P}(I = i) = p_i$$
2) Generate an instance of the exponential random variable with PDF
$$f_i(x) = \lambda_i e^{-\lambda_i x}, \quad x \geq 0$$

b) Erlang-k: For the Erlang-k distribution, the **convolution method** is convenient, since this type of r.v. is just the sum of k exponentially distributed random variables with the same lambda, as can be seen in Equation 2.

$$f(x; k, \lambda) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!}, \quad x \geq 0 \tag{2}$$

The convolution method returns instances of the Erlang-k r.v. by applying these simple steps:

1) Generate k instances of an exponentially distributed r.v. with parameter λ: $\{y_i\}_k$
2) Sum all the k values to generate an instance of the Erlang-k r.v.
$$x = \sum_{i=1}^{k} y_i, \quad x \sim E_k(\lambda)$$

c) Pareto: The Pareto distribution, differently from the previous ones, is not directly related to any other distribution, but since its cumulative distribution function (Equation 3) is invertible, the **inverse-transform technique** can be applied.

$$F(x) = \begin{cases} 1 - \left(\frac{x_m}{x}\right)^{\alpha} & x \geq x_m \\ 0 & x < x_m \end{cases} \tag{3}$$

Using the inverse-transform method, instances of the Pareto r.v. can be generated as follows:

1) Generate an instance u of a uniformly distributed r.v. $U(0, 1)$
2) Calculate $x = F^{-1}(u)$, where the inverted CDF is
$$F^{-1}(u) = \frac{x_m}{(1-u)^{\frac{1}{\alpha}}}$$
3) Return $x \sim \text{Par}(\alpha, x_m)$

B. Distribution Test

After generating the requested R.V.s, some tests are designed to assess the correctness of the implemented routines.

The first test aims to verify that the distributions empirically obtained from the previous functions correspond to the theoretical distributions of the random variables. To prove this hypothesis, a **chi-square goodness-of-fit** test is implemented. The structure of the test is as follows:

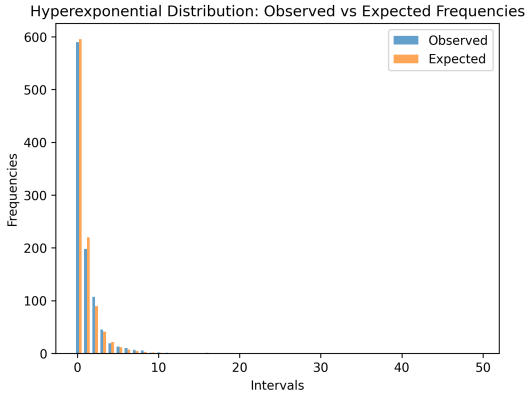
1) Define n intervals in the r.v. support, with the interval i defined by (a_i, a_{i+1})
2) Generate N instances of the r.v. using the previously defined routines and compute the number O_i of instances falling in the i-th interval and compare it with the expected value, E_i
$$E_i = N[F(a_{i+1}) - F(a_i)]$$
3) Compute
$$X = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$
4) If the hypothesis is true and N is large, X is distributed according to a chi-square distribution with $n-1$ degrees of freedom.

So, by comparing X and the chi-square distribution with $n-1$ degrees of freedom, the null hypothesis H_0 (empirical

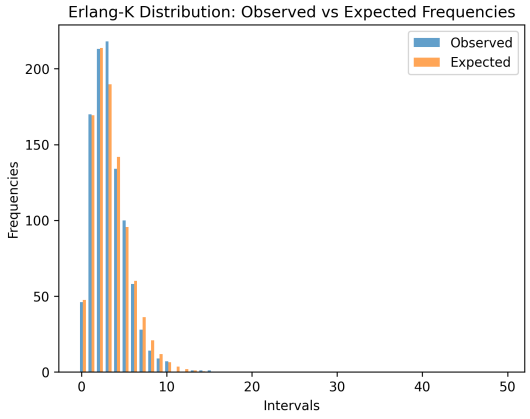
[1] Code implementation can be found in `rv_generation.py`

and theoretical distributions are the same) will be rejected if $X > \chi^2_{n-1,\alpha}$ (α is the level of significance). The values of $\chi^2_{n-1,\alpha}$ are known and tabulated.

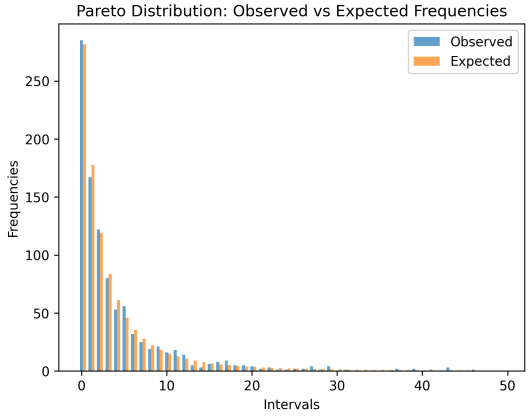
All the routines developed in I-A successfully passed the goodness-of-fit test, proving that the proposed generation approaches work well for the various types of random variables. Figure 1 shows the difference between the theoretical distributions and the empirical distributions obtained through the routines.



(a) Hyper-exponential r.v.



(b) Erlang-k r.v.



(c) Pareto r.v.

Fig. 1: Distribution chi-square goodness-of-fit tests.

C. Independence Test

The second test to be implemented aims to verify the independence of the generated samples. To check this property, an estimate of the **autocorrelation function** is computed according to Equation 4.

$$\hat{R}(k) = \frac{1}{(n-k)S^2} \sum_{t=1}^{n-k} (X_t - \bar{X})(X_{t+k} - \bar{X}) \quad \text{for } 0 \leq k < N$$
$$\text{where: } S^2 = \frac{\sum_{i=1}^{n} X_i - \bar{X}}{n-1}, \quad \bar{X} = \frac{\sum_{i=1}^{n} X_i}{n} \tag{4}$$

If the samples are completely independent, then the autocorrelation function should have value 1 in $\hat{R}(0)$ (lag zero) and value 0 for all other lags.

Figure 2 plots the estimated autocorrelation functions for 1000 samples generated by the different routines, respectively hyper-exponential, Erlang-k and Pareto.

It is easy to see that all three estimated autocorrelation functions behave very similarly to the ideal scenario of complete independence: in zero, they assume the value 1 and for all other lags, they are close to zero. For these reasons, the independence test is also considered passed.

D. Performance Evaluation - Queuing System

Now, these random variables are used to generate arrival times and service times in a basic queuing system. The queue has a single server and a capacity of 1000; the inter-arrival times have a constant mean of 1 second and the service times have a constant mean of 0.8 seconds.

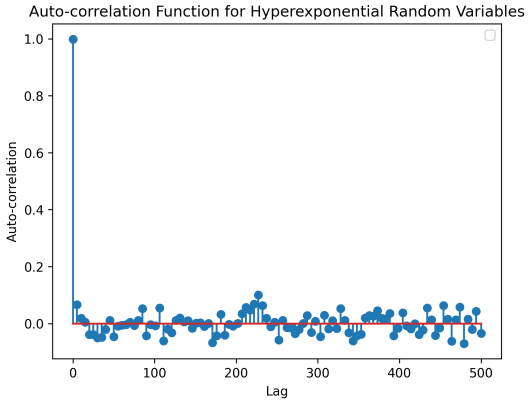
The tested configurations are summarized in Table I. Each configuration is run 10 times with different seeds, then the average metrics across the runs are computed. The metrics taken under consideration are:

- Average waiting time
- Average queue length
- Standard deviation of inter-arrival times
- Standard deviation of service times
- Server utilization

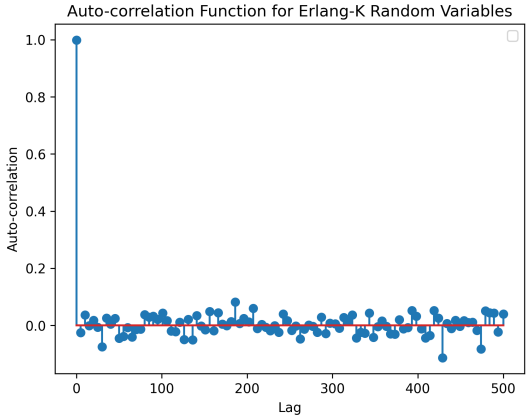
The results obtained are summarized in Table II and Figure 3.

These results show, as expected, that the various random variables generated have very different variances, while maintaining the same mean. Indeed, the Erlang-k distribution has the lowest standard deviation, while both the hyper-exponential and Pareto distributions have higher standard deviations than the reference exponential distribution. The empirical values obtained correctly align with the theoretical results, except for the Pareto that, since $\alpha \leq 2$, should have infinite variance, but still, this is somehow reflected in the extremely high std dev values.

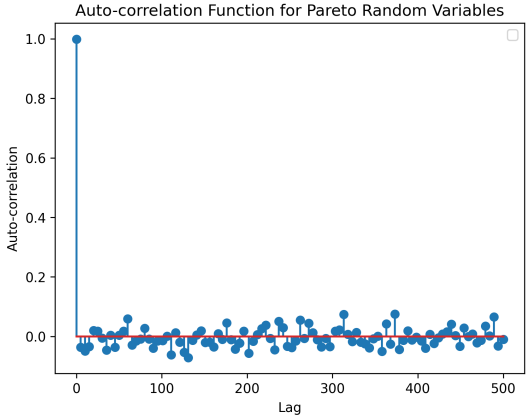
In particular, Figure 3 shows how high-variance distributions, both for inter-arrival times and service times, lead to



(a) Hyper-exponential samples.



(b) Erlang-k samples.



(c) Pareto samples.

Fig. 2: Estimated autocorrelation functions for random samples.

poor system performance, increasing queue length and waiting time. The best performance is achieved using the low-variance Erlang-k distribution for both inter-arrival and service times; on the other hand, when using the Pareto distribution, the system performs over 30 times worse, reaching an average waiting time of almost 30 seconds. Note that system utilization remains constant across all configurations (80%), as the average inter-arrival time and service time are always the same.

TABLE I: Simulation Configurations

Inter-arrival Dist.	Service Dist.	Inter-arrival Parameters	Service Parameters
Exponential	Exponential	$\lambda = 1$	$\lambda = 1.25$
Erlang-k	Exponential	$k = 3, \lambda = 3$	$\lambda = 1.25$
Hyper-exponential	Exponential	$\mathbf{p} = (0.5, 0.5)$ $\boldsymbol{\lambda} = (2.0, 0.666)$	$\lambda = 1.25$
Pareto	Exponential	$\alpha = 1.5, x_m = 0.3333$	$\lambda = 1.25$
Exponential	Erlang-k	$\lambda = 1$	$k = 4, \lambda = 5$
Erlang-k	Erlang-k	$k = 3, \lambda = 3$	$k = 4, \lambda = 5$
Exponential	Hyper-exponential	$\lambda = 1$	$\mathbf{p} = (0.5, 0.5)$ $\boldsymbol{\lambda} = (2.0, 0.909)$
Hyper-exponential	Hyper-exponential	$\mathbf{p} = (0.5, 0.5)$ $\boldsymbol{\lambda} = (2.0, 0.666)$	$\mathbf{p} = (0.5, 0.5)$ $\boldsymbol{\lambda} = (2.0, 0.909)$
Exponential	Pareto	$\lambda = 1$	$\alpha = 1.714, x_m = 0.3333$
Pareto	Pareto	$\alpha = 1.5, x_m = 0.3333$	$\alpha = 1.714, x_m = 0.3333$

TABLE II: Comparative Results Summary (Averaged over 10 runs)

Configuration	Std Inter	Std Service	Queue Length	Waiting Time	Util
Exponential/Exponential	1.0018	0.7979	3.1787	3.1795	0.7993
Erlang-3/Exponential	0.5784	0.7992	1.9432	1.9437	0.7996
Hyperexp/Exponential	1.2246	0.7999	4.1038	4.1049	0.7987
Pareto/Exponential	7.8076	0.7992	9.3156	9.3258	0.7983
Exponential/Erlang-4	1.0012	0.4005	2.0451	2.0420	0.8019
Erlang-3/Erlang-4	0.5769	0.4009	0.8018	0.8022	0.7999
Exponential/Hyperexp	0.9979	0.9086	3.7909	3.7881	0.8021
Hyperexp/Hyperexp	1.2257	0.9083	4.5229	4.5239	0.8008
Exponential/Pareto	0.9997	2.4456	17.3452	17.3116	0.7978
Pareto/Pareto	6.2460	2.6524	29.6118	29.4071	0.8050

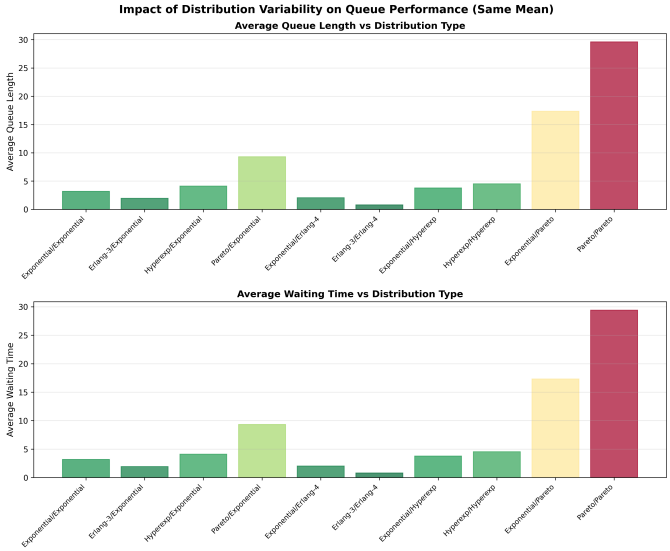


Fig. 3: Average queue length and waiting time across different configuration of the queuing system.

E. Performance Evaluation - Car-sharing System

Lastly, these random variables are used to generate inter-arrival times in a car-sharing system. Also in this case, all the different distributions have the same mean of 4 requests per minute.

The tested configurations are summarized in Table III. To highlight the differences between the distributions, the number of requests/arrivals per hour is monitored in each run. At the end of the simulation, the mean and standard deviation of hourly arrivals are calculated. The results are summarized in Table IV and plotted in Figure 4.

TABLE III: Car-sharing Configurations

Inter-arrival Distribution	**Parameter Values**
Exponential	$\lambda = 4.0$
Hyper-exponential	$\boldsymbol{\lambda} = (6.0, 4.0, 2.0)$ $\mathbf{p} = (0.6, 0.2, 0.2)$
Erlang-k	$k = 3, \lambda = 12.0$
Pareto	$\alpha = 5/3 \approx 1.67, \quad x_m = 0.1$

TABLE IV: Arrival Pattern Statistics (Bin width: 60 min)

Distribution	**Mean**	**Std Dev**	**Min**	**Max**
Hyper-exponential	237.94	19.72	183	309
Erlang-k	239.39	9.48	215	272
Pareto	244.43	32.07	76	303

Like for the basic queuing system, the results show that the arrivals are more consistent when they are generated by a low-variance distribution like the Erlang-k, resulting in a more stable system. On the contrary, the Pareto distribution, with its infinite variance, cause the system to behave in a very unsteady way. These differences in the behavior of the system are clearly shown in Figure 5.

II. Airport Departure Lounge

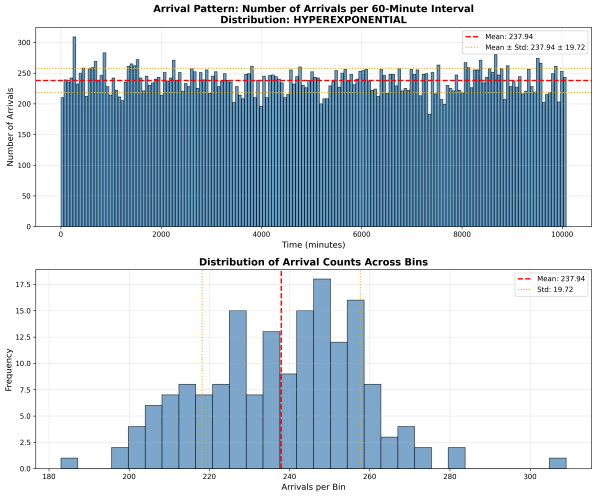
The second task of the delivery concerns the design and development of a passenger-flow simulation model for a medium size, non-hub, airport.

A. Model Design

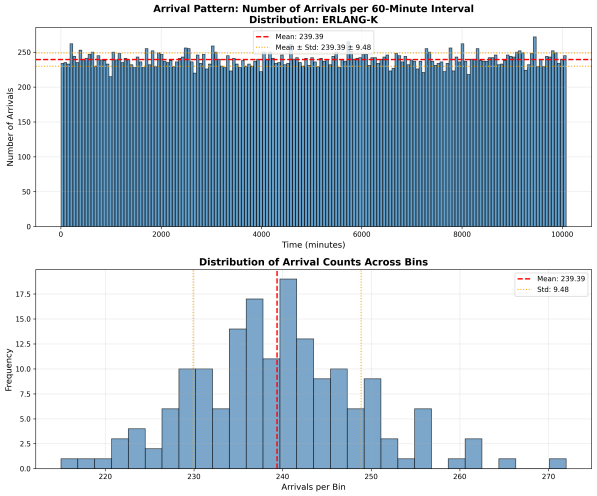
The simulation models the entire duration of passengers' stay, from their arrival at the airport to their flight departure.

Passengers, together with accompanying friends or relatives (a.k.a. companions), arrive at the airport a few hours before their flight. In particular, the arrival time of passengers is determined by a normal distribution that precedes the take-off time: if t_0 is the take-off time of flight F_0, for all the passengers in $\mathcal{P}_{F_0}$ the arrival time a_p is calculated as shown in Equation 5.

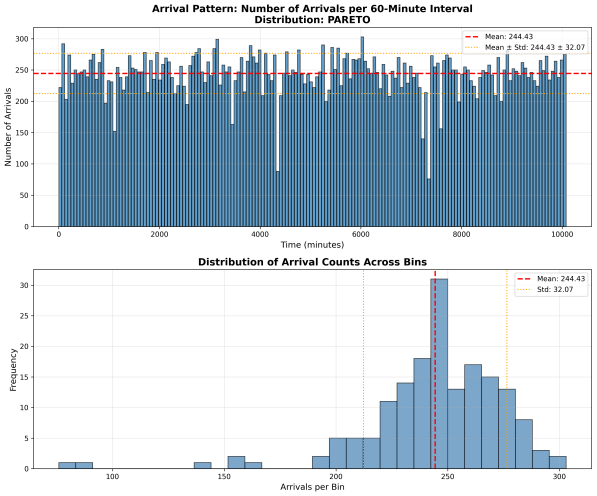
$$\begin{aligned} a_p &= t_0 - \tau_p, \quad \forall p \in \mathcal{P}_{F_0} \\ \text{where:} \quad \tau_p &\sim \mathcal{N}(\mu_A, \sigma_A^2) \end{aligned} \tag{5}$$



(a) Hyper-exponential inter-arrival times.

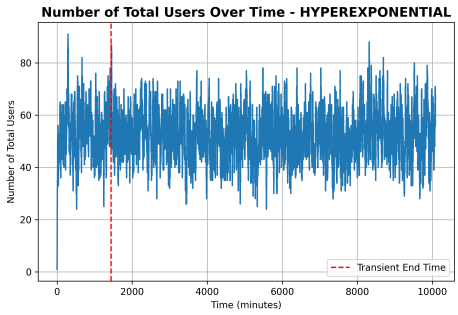


(b) Erlang-k inter-arrival times.

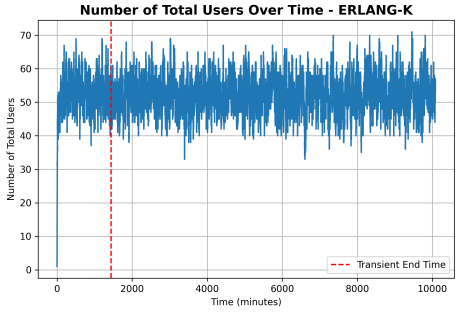


(c) Pareto inter-arrival times.

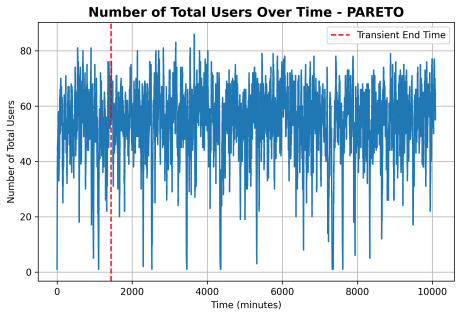
Fig. 4: Hourly arrivals statistics for different inter-arrival time distributions.



(a) Hyper-exponential



(b) Erlang-k



(c) Pareto

Fig. 5: Comparison of system behavior under different inter-arrival time distributions.

The number of companions for each passenger is drawn from a Poisson distribution with parameter λ_C (average number of companions).

After entering the airport, each group (the passenger and its companions), wanders around the landside area for an amount of time, drawn from an exponential distribution Exp(β_{LS}). Then, with probability P_B, they may purchase something in a shop.

The buying process, for the sake of simplicity, is modeled by a single queue that encompasses all the shops, with a number of servers equal to S_{LS} (number of shops in the landside area). The queue length considers the number of groups, not the people, but still, the service time of each group depends on its size. Once the purchase is accomplished, the companions leave the airport and the passengers go to the security check.

The security check is modeled as a priority queue, where passengers belonging to more expensive classes (e.g., first class) are processed before lower class passengers. The number of servers is equal to the number of security gates S_{SG} and the service time is drawn from an exponential distribution.

After passing through security, passengers enter the reserved area (airside) where, as before, they can wander around the shops for a certain period of time (derived from Exp(β_{AS})), after which they may purchase something.

The purchasing is handled in the same way as in the landside area, with the only difference that in the airside there are no more companions, so each client's service time is generated according to the same distribution. The number of servers in this case is S_{AS}.

Once purchases are complete or boarding begins, passengers enter the boarding queue. Again, the queue takes into account each passenger's priority, with first-class passengers boarding first, followed by business and economy class passengers. Unlike before, when there was a single queue for all passengers, each flight has its own boarding queue. Hence, passengers are sorted into different queues depending on their flight.

When the boarding process terminates, the flight takes off and passengers leave the airport.

Both security checks and boarding have a deadline, after which passengers are no longer allowed to enter the queues. Therefore, if a passenger arrives late at the airport or if the shopping line is too long, they will inevitably skip their purchases and go straight to security/boarding.

The all process is detailed in Figure 6.

B. Hyperparameters and Key Performance Indicators

The behavior of the system depends on several hyperparameters, categorized as follows:

Passenger Generation and Behavior

- μ_A: Mean passenger lead time (time of arrival before departure).
- σ_A^2: Variance of the passenger lead time.
- λ_P: Average number of passengers per flight.
- Δt_{flight}: Time interval between each flight departure.
- λ_C: Average number of companions per passenger.
- P_B: Probability that a passenger makes a purchase.

Resource Capacities (Number of Servers)

- S_{LS}: Number of shops available in the landside area.
- S_{AS}: Number of shops available in the airside area.
- S_{SG}: Number of open security screening gates.
- S_{BG}: Number of boarding gates assigned per flight.

Service and Dwell Times

- β_{LS}: Average dwell time in landside area (browsing/waiting).
- β_{AS}: Average dwell time in airside area.
- β_C: Average service time at a shop cashier.
- β_{SG}: Average processing time at security check.
- β_{BG}: Average boarding time per passenger.

Operational Constraints (Deadlines)

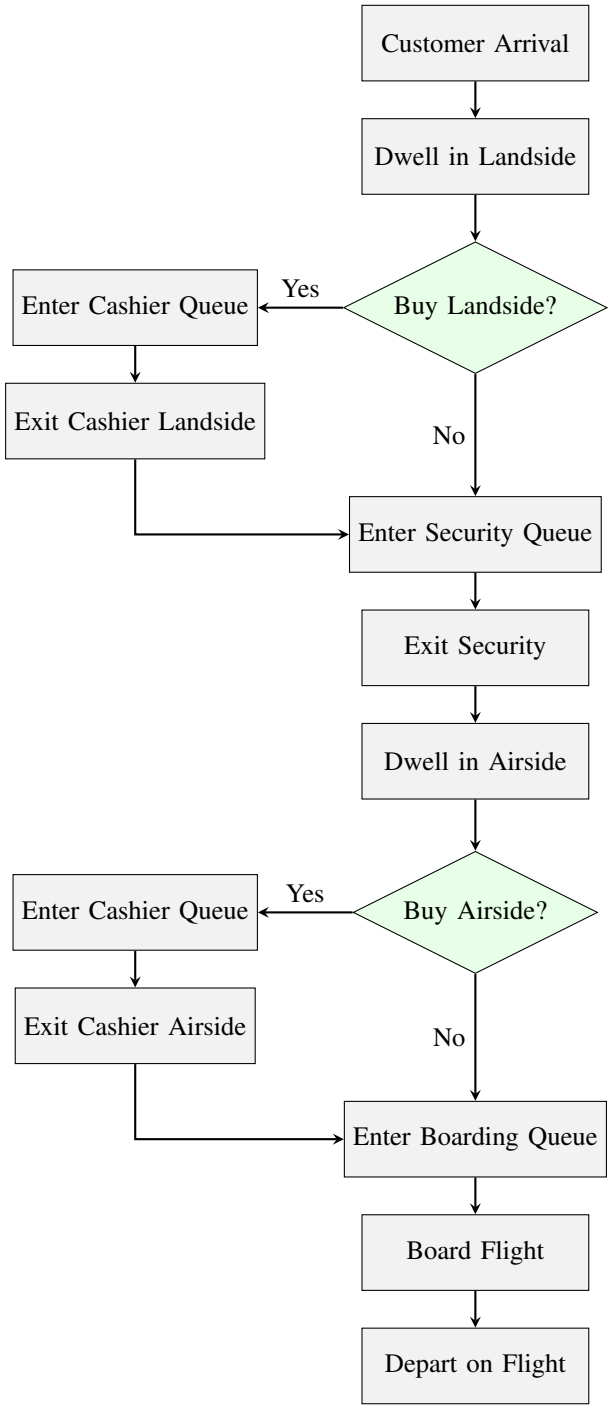


Fig. 6: Passenger Flow Simulation Logic

- $t_{security-end}$: Time cutoff for security checks (minutes before departure).
- $t_{boarding-end}$: Time cutoff for boarding (minutes before departure).

To evaluate the quality and efficiency of the simulator, several key performance indicators (KPIs) are monitored. These metrics rely on two fundamental statistical definitions calculated over the total simulation duration T.

First, for state variables that persist over time (such as queue lengths or occupancy), we calculate the **Time-Weighted Average** ($\bar{X}$):

$$\bar{X} = \frac{1}{T} \int_0^T X(t)\, dt \tag{6}$$

where $X(t)$ is the value of the variable at time t.

Second, to measure server efficiency, we calculate **Resource Utilization** (U):

$$U = \frac{\int_0^T N_{busy}(t)\, dt}{T \times S} \tag{7}$$

where $N_{busy}(t)$ is the number of active servers at time t, and S is the total capacity (number of available servers).

The specific KPIs monitored are:

Throughput and Reliability

- N_P: Total passenger arrivals (throughput).
- N_M: Number of passengers who missed their flight.

Occupancy Levels (Calculated using Eq. 6)

- $\bar{N}_A$: Average number of people in the airport (total system occupancy).
- $\bar{N}_{LS}$: Average number of people in the landside area.
- $\bar{N}_{AS}$: Average number of passengers in the airside area.

Queue Lengths (Calculated using Eq. 6)

- $\bar{L}_{LS}$: Average queue length for landside shops.
- $\bar{L}_{AS}$: Average queue length for airside shops.
- $\bar{L}_{SG}$: Average queue length for security checks.
- $\bar{L}_{BG}$: Average queue length for boarding gates.

Waiting Times (Discrete average per passenger)

- $\bar{W}_{LS}$: Average waiting time before purchasing in the landside area.
- $\bar{W}_{AS}$: Average waiting time before purchasing in the airside area.
- $\bar{W}_{SG}$: Average waiting time before the security check (stratified by passenger class).
- $\bar{W}_{BG}$: Average waiting time before boarding (stratified by passenger class).

Resource Utilization (Calculated using Eq. 7)

- U_{LS}: Utilization of landside shops (where $S = S_{LS}$).
- U_{AS}: Utilization of airside shops (where $S = S_{AS}$).
- U_{SG}: Utilization of security gates (where $S = S_{SG}$).
- U_{BG}: Utilization of boarding gates (averaged across all flights).

C. Events and System Structure

The system is modeled as a **discrete-event simulation** (DES). The various events to be handled are stored in a **Future Event Set** (FES), implemented as a priority queue, which processes the events in chronological order.

The types of event designed for the simulation are detailed in Table V.

The overall simulation relies on 2 main objects:

- **Passenger**: Associated with a specific flight, with a randomly generated number of companions and stochastically assigned to a priority class (Economy $\approx 70\%$,

TABLE V: Airport Event Details

Event	Description	Triggered By	Triggers	General Behavior
FLIGHT SCHEDULED	Flight scheduled; passengers generated.	Simulation setup (recurring).	• CUSTOMER ARRIVAL • FLIGHT DEPARTURE	Creates flight object. Samples passengers (Poisson) with random arrival times, priority classes, and companions.
CUSTOMER ARRIVAL	Passenger, along with companions, arrives at airport.	FLIGHT SCHEDULED	• ENTER CASHIER QUEUE (if buying) • ENTER SECURITY QUEUE (if not)	Decides purchase intent. Checks if time permits purchase. If buying risks missing deadline, skips to security.
ENTER CASHIER QUEUE	Enters shop queue (Land/Air).	• CUSTOMER ARRIVAL • EXIT SECURITY QUEUE	• EXIT CASHIER QUEUE (if free) • Waits (if busy)	Joins FIFO queue. If server free, schedules exit immediately (service time scaled by group size). Otherwise tracks wait time.
EXIT CASHIER QUEUE	Finishes shopping.	Previous queue logic.	• ENTER SECURITY QUEUE (Landside) • ENTER BOARDING QUEUE (Airside)	Records metrics. Frees server. Directs passenger to next stage (Security or Boarding). Processes next in queue.
ENTER SECURITY QUEUE	Enters security check.	• CUSTOMER ARRIVAL • EXIT CASHIER QUEUE	• EXIT SECURITY QUEUE (if free) • Waits (if busy)	Joins priority queue (First $>$ Business $>$ Economy). **Companions leave**.
EXIT SECURITY QUEUE	Finishes security check.	Previous queue logic.	• ENTER CASHIER QUEUE (if buying) • ENTER BOARDING QUEUE (if not)	Enters airside. Decides on airside purchase based on time remaining.
ENTER BOARDING QUEUE	Enters boarding gate queue.	• EXIT SECURITY QUEUE • EXIT CASHIER QUEUE	• EXIT BOARDING QUEUE (if free) • Waits (if busy)	Joins priority queue. Cannot board before the boarding window opens. Waits if server busy or window closed.
EXIT BOARDING QUEUE	Boards aircraft.	Previous queue logic.	• Passenger removed (Success)	Records total time. Updates flight count. Frees server. Passenger leaves system successfully.
FLIGHT DEPARTURE	Aircraft takes off.	Scheduled takeoff time.	• None (Terminal)	Removes all passengers still in queues as "Missed Flight". Cleans up flight object.

Business $\approx 20\%$, or First Class $\approx 10\%$). To better track the state of the system, each passenger has a status that can be either:
  - ARRIVED
  - IN SHOP LANDSIDE
  - IN SECURITY
  - IN AIRSIDE
  - IN SHOP AIRSIDE
  - IN BOARDING
  - LEFT AIRPORT
- **Flight**: Associated with the total number of passengers and their list, with a take-off time and a boarding window duration (i.e., how many minutes before take-off the boarding gates open).

Since the airport is simplistically modeled as a set of queues (for shopping, for security and for boarding), the code implementation defines the `QueueSystem` class, with all the functions needed to manage a generic FIFO queue with priority. For the simulation, three instances of this class are initialized to model the landside and airside shopping queues (both without priority) and the security check queue (prioritized). Additionally, for each flight, once boarding begins, another instance of the `QueueSystem` class is created with priority. According to Kendall's notation, all the queues of the airport system are **G/M/c queues** (with infinite capacity), where:

- **G** means that the customer arrival process is not a specific distribution, but rather a general distribution. This is because a passenger's arrival in the queue depends on numerous factors: e.g., the flight departure time, whether they made a purchase in a store, the number of companions (longer service time), and the time spent wandering around.
- **M** means that the service time is memoryless, i.e., exponentially distributed. That is the case for every queue in the system.
- **c** is the number of servers for the queue. This is a predefined hyperparameter.

The key performance indicators for the simulation are computed and stored within the class `Metrics`, that encapsulates all the utility functions to keep track of sampled statistics (e.g., number of passengers in airport, landside or airside area), time-weighted statistics (e.g., average queue lengths and average occupancy) and cumulative statistics (e.g., total flights and passengers, resources utilization time, number of passengers that missed their flight). Some metrics are calculated class-wise, such as security and boarding waiting times.

The core of the simulation is implemented in the `SimulationEngine` class, in which the hyperparameters are set, the FES and the queues are initialized, and the first flight is scheduled. Then, the event loop method handles all the events that appear in the FES, until the simulation time expires. The simulation lasts a full working day, with flights departing from 6:00 am to 11:00 pm. The `SimulationEngine` class also contains some methods to retrieve, print, and plot statistics for the simulation.

Lastly, to compute **confidence intervals** for the various output metrics (KPIs), the `ConfidenceInterval` class is used. This class is the same as the previous delivery: it stores a list of data points concerning the measure for which the C.I. has to be calculated, then different methods are defined to compute the sample mean and standard deviation used to retrieve the interval according to Equation 8.

$$C.I. = \bar{x} \pm Z_{1-\frac{\alpha}{2}} \left(\frac{\sigma}{\sqrt{n}} \right) \tag{8}$$

Multiple **independent runs**, with different seeds, are performed to collect the statistics on which to calculate the confidence intervals.[2]

D. Experiments and Results

To evaluate the performance and resilience of the airport simulator, different scenarios are tested, with an increasing level of stress on the system. In particular four different configurations of hyperparameters are selected:

1) **Low stress**: Low flight frequency (2.7 flights/hour), over-staffed resources, and widely dispersed passenger arrivals.
2) **Medium stress**: Normal operations baseline, balanced 3 flights/hour with standard staffing levels.
3) **High stress**: Elevated frequency (3.2 flights/hour), slightly under-staffed, and moderately synchronized arrivals.
4) **Extreme stress**: Critical scenario with maximum frequency (3.4 flights/hour), severely under-staffed, and highly synchronized arrivals.

The detailed configurations are shown in Table VI.

To compare the results of the different scenarios, the most critical performance indicators are selected, namely the average waiting time at security and boarding, the dropping rate of passengers and the utilization of both security and boarding gates. The obtained results are plotted in Figure 7.

The image shows a clear difference between low/medium stress and high/extreme stress, where the former have zero passenger abandonment rates and short wait times at security checkpoints, while the latter have very high passenger abandonment rates and long queues at security checkpoints. On the contrary, the waiting time for boarding is significantly lower in the high and extreme stress scenarios. What may seem paradoxical at first glance is actually easily explained by the fact that, in these scenarios, most passenger losses occur at security checkpoints; passengers, therefore, don't have time to board before the cut-off time, reducing congestion at the next stage. The fact that boarding wait times are so short is therefore not a positive sign, but rather an indication that many passengers are being lost before even beginning boarding, highlighting a **lack of resources** at security checkpoints. The total time passengers spend in the system is similar in all scenarios, but still increases with the level of stress. The same happens with the utlization of security checks, while for boarding gates the utilization reaches its maximum for the medium stress scenario and then decreases, for the reasons mentioned above.

To assess whether the system operates under steady-state conditions, the realistic, high-performance **medium-stress**

[2]Additional information about the code implementation can be found in the documentation of `airport_departure.py`.

TABLE VI: Stress Test Configurations

Parameter	Low	Med	High	Ext
Passenger Generation & Behavior				
Δt_{flight} (min)	**24**	**20**	**19**	**18**
λ_P (pax/flight)	100	100	100	100
λ_C (companions)	1.5	1.5	1.5	1.5
P_B (buy prob)	0.8	0.8	0.8	0.8
μ_A (lead time min)	120	120	120	120
σ_A (std dev min)	**35**	**30**	**27**	**26**
Resource Capacities (S)				
S_{LS} (Landside)	**19**	**17**	**16**	**16**
S_{AS} (Airside)	**10**	**8**	**7**	**7**
S_{SG} (Security)	**10**	**8**	**7**	**7**
S_{BG} (Boarding)	**3**	**2**	**2**	**2**
Service & Dwell Times (β)				
β_C (Cashier min)	2.0	2.0	2.0	2.0
β_{SG} (Security min)	1.5	1.5	1.5	1.5
β_{BG} (Boarding min)	0.5	0.5	0.5	0.5
β_{LS} (Landside min)	30	30	30	30
β_{AS} (Airside min)	15	15	15	15
Operational Constraints (t)				
$t_{security-end}$ (min)	40	40	40	40
$t_{boarding-end}$ (min)	20	20	20	20

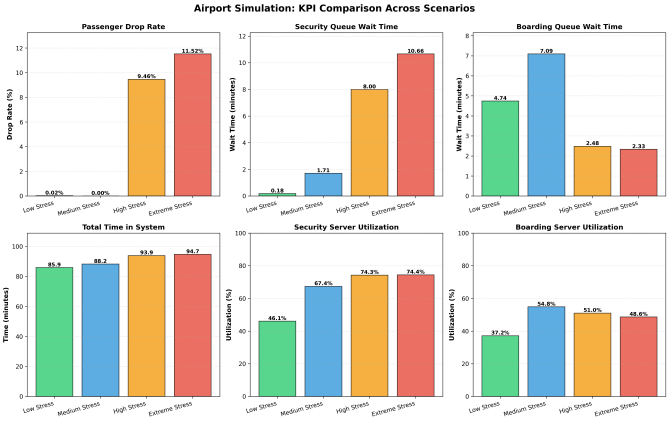


Fig. 7: Comparison of the results obtained by the different stress tests.

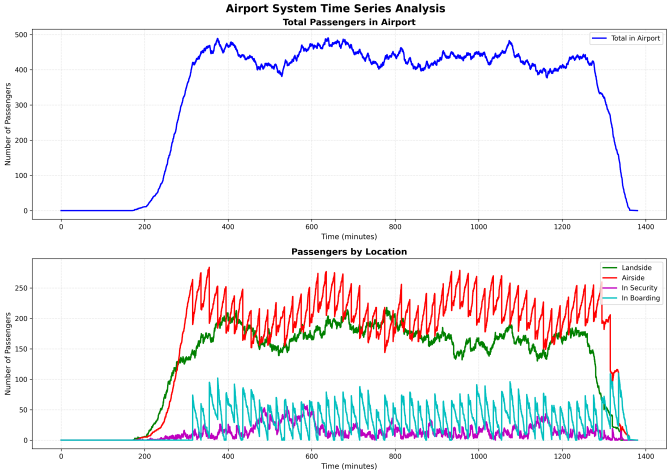


Fig. 8: Number of passengers in the airport over time, also divided according to the specific area.

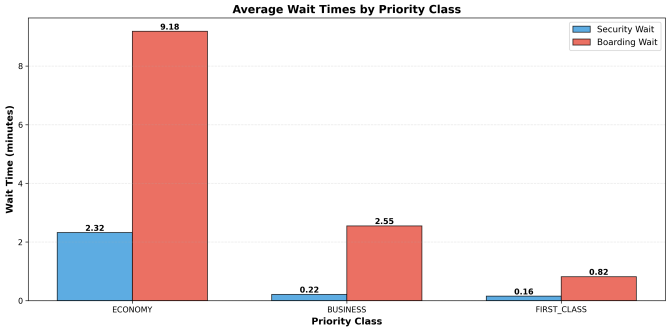


Fig. 9: Comparison of security and boarding waiting times across the different priority classes.

configuration is selected, which is then executed in multiple independent runs, over which confidence intervals of the KPIs are also calculated. The results for a single run are shown in Figures 8 and 9.

Although the total number of customers at the airport quickly reaches a nearly stable value (around 450), at the start of daily departures (6:00 a.m.), the system composition is highly unstable, as shown by the colored lines, particularly with regard to the number of passengers in the airside area and at the boarding gate. These latter two time series are directly related to flight departures: passengers in the reserved area cannot begin the boarding process until the boarding window for their flight opens (45 minutes before takeoff). This mechanism causes both the airside and boarding time series to peak at the opening of the boarding window, when all waiting passengers enter the boarding queue. For this reason, the system does not operate under constant steady-state conditions, but rather in an approximate **cyclic steady-state**, which repeats for each flight departure until the end of the day (11:00 p.m.).

Finally, Table VII reports the confidence intervals for the most critical performance indicators, calculated from 20 independent runs. The results obtained are consistent with those calculated for the stress test. For some metrics (e.g., security checkpoint waiting time), the intervals did not converge despite the large number of independent executions, indicating that the system is still moderately susceptible to initial conditions.

TABLE VII: Confidence Interval Results (95% Confidence, $n = 20$ runs)

Metric	Status	Mean	95% C.I.	Width
$\bar{W}_{SG}$ (Security Wait)	○	2.145	$[1.793, 2.497]$	0.704
$\bar{W}_{BG}$ (Boarding Wait)	✓	7.026	$[6.862, 7.191]$	0.329
$\bar{T}_{sys}$ (Total Time)	✓	88.186	$[87.897, 88.475]$	0.579
P_{miss} (Drop %)	○	0.023	$[0.011, 0.035]$	0.024
U_{SG} (Security Util)	✓	0.674	$[0.669, 0.680]$	0.011
U_{BG} (Boarding Util)	✓	0.554	$[0.551, 0.558]$	0.008

Legend: ✓= Converged, ○ = More data needed

III. Two Species Co-Evolution

The third task of the delivery concerns the design and development of a **Markov model** for the dynamics of two interacting species sharing one environment: **carnivores** (C) and **herbivores** (H). The system is modeled as a **Continuous Time Markov Chain (CTMC)**. This stochastic framework represents the evolution of populations as a sequence of discrete events occurring at random intervals, where the probability of future states depends solely on the current state configuration.

A. Model Design

The simulation tracks the exact number of individuals within the environment. To satisfy the requirement for realistic reproductive dynamics—specifically the inclusion of gestation periods—the state space is structured by **sex** and **reproductive status**. Consequently, the system state vector $\mathbf{S}(t)$ at any time t is defined as:

$$\mathbf{S}(t) = [H_M, H_F, H_{F_{preg}}, C_M, C_F, C_{F_{preg}}] \tag{9}$$

where M, F, and F_{preg} denote males, non-pregnant females, and pregnant females, respectively. The logic of state transitions involving these variables is illustrated in Figure 10.

Reproduction is implemented as a two-stage process to account for biological delays. Initially, a conception event transitions a non-pregnant female to the pregnant state ($H_F \to H_F - 1, H_{F_{preg}} \to H_{F_{preg}} + 1$). This event is stochastic and conditional on the presence of at least one male of the same species. Subsequently, a birth event occurs after a random gestation period. This returns the female to the fertile pool and increments the population count of either males or females, chosen with equal probability.

Mortality encompasses natural death, which removes individuals from any state in the system. A critical feature of the model is that pregnant females are subject to the same mortality risks as the rest of the population. If a pregnant female dies, the unborn offspring is simultaneously removed from the system, effectively cancelling the potential birth event.

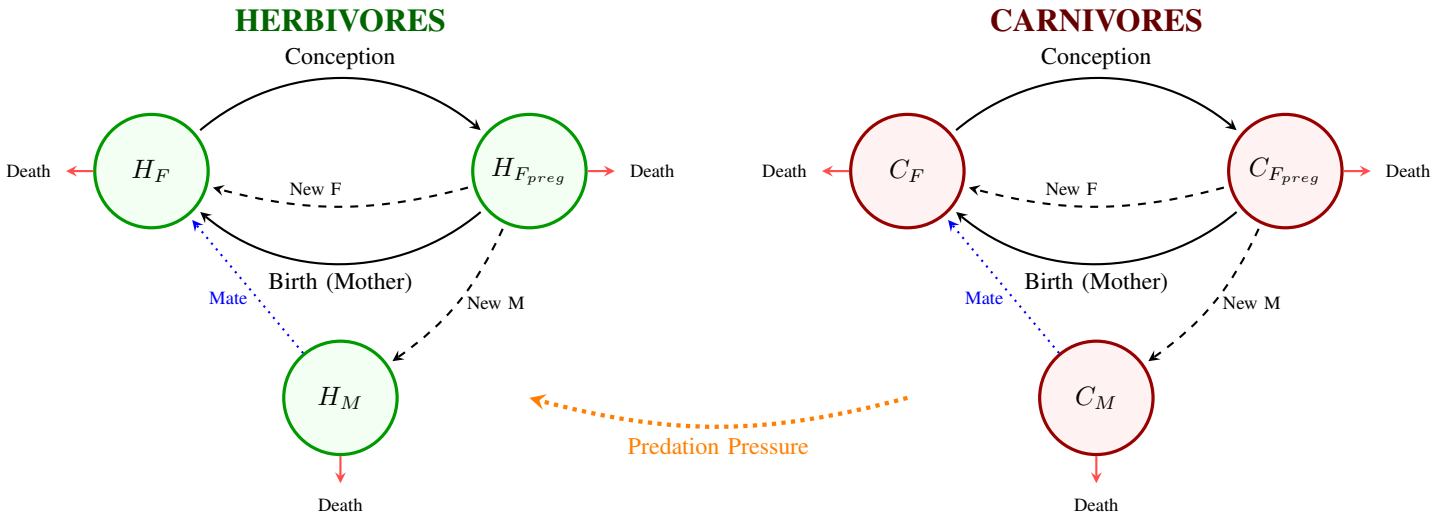


Fig. 10: Compartmental representation of the Markov Chain state transitions. Solid loops represent the reproductive cycle of females; dashed arrows represent the generation of new offspring (Male or Female). Blue dotted arrows indicate the mating requirement. The Orange dotted arrow represents the predation pressure exerted by Carnivores on Herbivores.

Predation models the trophic interaction where carnivores consume herbivores. This is treated as a mass-action process, where the rate of interaction is proportional to the product of the predator and prey populations. When a predation event occurs, a herbivore is removed from the system; the specific state (Male, Female, or Pregnant) of the victim is selected probabilistically based on their relative abundance.

Intraspecific Conflict represents competition for resources or territory. These conflict events occur exclusively between individuals of the same species and same sex. Such interactions result in the death of one individual and occur with a frequency that scales quadratically with the density of that specific sub-population (e.g., proportional to $H_M \times (H_M - 1)$).

To represent the finiteness of the environment, the model implements density-dependent feedback loops that dynamically adjust fertility and mortality rates using **logistic functions**. For **Herbivores**, parameters are scaled based on the total population H_{total} relative to a carrying capacity H_{thresh}:

$$\begin{aligned} \phi_{fert}^{(H)} &= \frac{1}{1 + e^{k_H(H_{total} - H_{thresh})}} \\ \phi_{mort}^{(H)} &= 1 + \frac{1}{1 + e^{-k_H(H_{total} - H_{thresh})}} \end{aligned} \tag{10}$$

These factors reduce the birth rate and increase the death rate as the population grows. For **Carnivores**, parameters depend on the prey-to-predator ratio $R = H_{total}/C_{total}$ relative to a viability threshold R_{thresh}:

$$\begin{aligned} \phi_{fert}^{(C)} &= \frac{1}{1 + e^{-k_C(R - R_{thresh})}} \\ \phi_{mort}^{(C)} &= 1 + \frac{1}{1 + e^{k_C(R - R_{thresh})}} \end{aligned} \tag{11}$$

This models starvation, severely penalizing the carnivore population when food sources are insufficient.

B. Hyperparameters and Key Performance Indicators

The behavior of the co-evolutionary system is governed by the following hyperparameters:

Vital Dynamics

- r_H, r_C: Base conception rate per non-pregnant female.
- m_H, m_C: Base natural mortality rate per individual.
- τ_H, τ_C: Average gestation period (mean time to birth).

Interactions and Thresholds

- α_{pred}: Predation efficiency (interaction rate).
- γ_H, γ_C: Intraspecific conflict rates.
- H_{thresh}: Herbivore carrying capacity.
- R_{thresh}: Minimum prey-to-predator ratio for carnivore viability.

To evaluate stability and ecological balance, the following **Key Performance Indicators (KPIs)** are monitored:

- **Average Populations** ($\bar{N}_H, \bar{N}_C$): The time-weighted average number of individuals.
- **Sampled Populations**: The trajectory of population counts over time.
- **Ecological Balance**: The average ratio of herbivores to carnivores.
- **Event Counts**: Cumulative totals of births, deaths, predations, and conflicts.

Standard arithmetic means are avoided due to the variable time steps of the CTMC. Instead, **Time-Weighted Statistics** are computed using the following formulation:

$$\hat{L} = \frac{1}{T_{max}} \sum_{n=0}^{N-1} \theta(S_n) \cdot W_n \tag{12}$$

where T_{max} is the total simulation duration, N is the total number of events, S_n is the state of the system at step n, $\theta(S_n)$ is the value of the metric in that state (e.g., the number

of herbivores), and W_n is the sojourn time (duration) of that state. This formulation ensures that states which persist for longer periods contribute proportionally more to the average.

C. Events and System Structure

The simulation engine relies on the theoretical properties of Continuous Time Markov Chains, specifically the distribution of sojourn times and the competition between concurrent exponential processes. The core logic is encapsulated in the `SimulationEngine` class, which executes the simulation through a `step` method. This method advances the system state by state according to the following procedure:

1) **Rate Calculation**: The propensity (instantaneous rate) λ_k for every possible event is calculated based on the current population vector $\mathbf{S}(t)$ and the logistic adjustment factors (Eq. 10, 11).
2) **Total Propensity**: The sum of all individual rates is computed as $\Lambda = \sum \lambda_k$. This represents the parameter for the exponential distribution of the sojourn time.
3) **Time Sampling**: The time to the next event is generated by sampling from an exponential distribution with rate Λ. The global simulation clock is advanced by this increment.
4) **Event Selection**: A specific event type is chosen probabilistically. The probability of selecting event k is proportional to its rate: $p_k = \lambda_k / \Lambda$.
5) **State Update**: The system state vector is updated according to the logic of the selected event (e.g., incrementing/decrementing specific population counters).

The `SimulationEngine` class provides also methods to print and plot the statistics calculated for the simulation.

The model defines **17 discrete event types** that cover the complete lifecycle of both species. These are explicitly listed in Table VIII.

Data collection is managed by the `Metrics` class. This component calculates the area under the population curve incrementally at each step, avoiding the overhead of storing the complete event history. It also records the state of the system at each step, allowing, at the end of the simulation, to track the trend of the populations of the different species over time.

Finally, the `ConfidenceInterval` class aggregates the results. It receives the final KPIs from multiple independent simulation runs and computes the 95% confidence intervals, ensuring the statistical reliability of the observations.[3]

D. Experiments and Results

To evaluate the stability, resilience, and long-term behavior of the co-evolutionary model, extensive simulations are conducted under various environmental conditions. Five distinct configurations of hyperparameters are designed to test the system's response to specific stress factors. The specific parameters for each scenario are detailed in Table IX.

[3]Additional information about the code implementation can be found in the documentation of `carnivore_herbivore.py`.

1) **Baseline**: Represents a balanced ecosystem where parameters are tuned for coexistence. This serves as the reference for stability analysis.
2) **High Predation**: Simulates an aggressive predator species (α_{pred} increased 7x) with lower food requirements. This tests the risk of prey depletion and subsequent predator collapse.
3) **Low Predation**: Represents a weak predator (α_{pred} decreased 10x). This scenario investigates the potential for herbivore unchecked growth and carnivore starvation.
4) **High Conflict**: Simulates an environment where individuals are highly aggressive towards members of their own species (conflict rates increased 5-10x). This stresses the internal population regulation mechanisms independent of resource availability.
5) **Extreme Reproduction**: Models species with very fast reproductive cycles (gestation reduced by 33%, conception rates increased by 50%). This configuration tests the system's **density-dependent feedback loops**: as the population explodes rapidly, the logistic factors must react aggressively to prevent an overflow.

To ensure the statistical robustness of the results and verify independence from starting conditions, a rigorous testing protocol is applied to **all five scenarios**. Specifically, each configuration is tested against 5 different **Initial Conditions (ICs)**, detailed in Table X. These conditions range from predator-dominated states to massive population overshoots. For each unique combination of configuration and initial condition, 5 independent simulations are executed with different random seeds, resulting in a total of 25 runs per scenario.

To assess the system's dynamics, the sample trajectories of the populations over time are analyzed. Figure 11 illustrates the behavior of each configuration.

The results reveal a fundamental dichotomy. The **Baseline**, **High Conflict**, and **Extreme Reproduction** configurations exhibit stable, long-term coexistence, oscillating around a dynamic equilibrium. Conversely, the **High Predation** scenario consistently leads to the extinction of the herbivore population due to over-predation, inevitably followed by the collapse of the carnivore population due to starvation. Similarly, the **Low Predation** scenario results in the extinction of carnivores, who fail to secure enough resources to sustain their population.

Focusing on the surviving configurations, Figure 12 compares the average population levels and the ratio between herbivores and carnivores. The **Extreme Reproduction** scenario sustains the highest number of individuals (189 herbivores). This happens because the high birth rate constantly pushes the population upwards, effectively putting pressure on the limits of carrying capacity. In the **High Conflict** scenario, while the herbivore population is similar to the baseline, the ecological balance shifts significantly. The ratio of herbivores to carnivores increases to 3.61 (compared to 3.22 in the baseline). This indicates that when predators face intense intraspecific competition (fighting among themselves), their ability to control the prey population diminishes, allowing herbivores to become relatively more numerous.

TABLE VIII: Discrete Events in the Co-Evolution Model

Event Name	Description	Rate Dependency
Reproduction		
REPRODUCTION_H	Herbivore conception	$\propto H_F \cdot \mathbb{I}(H_M > 0) \cdot \phi_{fert}^{(H)}$
REPRODUCTION_C	Carnivore conception	$\propto C_F \cdot \mathbb{I}(C_M > 0) \cdot \phi_{fert}^{(C)}$
BIRTH_H_M	Herbivore birth (Male)	$\propto 0.5 \cdot (1/\tau_H) \cdot H_{F_{preg}}$
BIRTH_H_F	Herbivore birth (Female)	$\propto 0.5 \cdot (1/\tau_H) \cdot H_{F_{preg}}$
BIRTH_C_M	Carnivore birth (Male)	$\propto 0.5 \cdot (1/\tau_C) \cdot C_{F_{preg}}$
BIRTH_C_F	Carnivore birth (Female)	$\propto 0.5 \cdot (1/\tau_C) \cdot C_{F_{preg}}$
Mortality		
DEATH_H_M	Herbivore male death	$\propto m_H \cdot H_M \cdot \phi_{mort}^{(H)}$
DEATH_H_F	Herbivore female death	$\propto m_H \cdot H_F \cdot \phi_{mort}^{(H)}$
DEATH_H_F_PREG	Pregnant herbivore death	$\propto m_H \cdot H_{F_{preg}} \cdot \phi_{mort}^{(H)}$
DEATH_C_M	Carnivore male death	$\propto m_C \cdot C_M \cdot \phi_{mort}^{(C)}$
DEATH_C_F	Carnivore female death	$\propto m_C \cdot C_F \cdot \phi_{mort}^{(C)}$
DEATH_C_F_PREG	Pregnant carnivore death	$\propto m_C \cdot C_{F_{preg}} \cdot \phi_{mort}^{(C)}$
Interactions		
PREDATION	Predation event	$\propto \alpha_{pred} \cdot H_{total} \cdot C_{total}$
CONFLICT_H_M	Herbivore male conflict	$\propto \gamma_H \cdot H_M \cdot (H_M - 1)$
CONFLICT_H_F	Herbivore female conflict	$\propto \gamma_H \cdot H_F \cdot (H_F - 1)$
CONFLICT_C_M	Carnivore male conflict	$\propto \gamma_C \cdot C_M \cdot (C_M - 1)$
CONFLICT_C_F	Carnivore female conflict	$\propto \gamma_C \cdot C_F \cdot (C_F - 1)$

TABLE IX: Co-Evolution Test Configurations

Parameter	Base	Hi-Pred	Lo-Pred	Conflict	Reprod
Vital Rates					
r_H (Herb. Conception)	0.5	0.5	0.5	0.5	**0.75**
r_C (Carn. Conception)	0.4	**0.5**	**0.3**	0.4	**0.6**
τ_H (Herb. Gestation)	6.0	6.0	6.0	6.0	**4.0**
τ_C (Carn. Gestation)	5.0	5.0	5.0	5.0	**3.3**
Interactions					
α_{pred} (Predation)	$7e^{-4}$	$\mathbf{5e^{-3}}$	$\mathbf{7e^{-5}}$	$7e^{-4}$	$7e^{-4}$
γ_H (Herb. Conflict)	$5e^{-5}$	$5e^{-5}$	$5e^{-5}$	$\mathbf{5e^{-4}}$	$5e^{-5}$
γ_C (Carn. Conflict)	$5e^{-4}$	$5e^{-4}$	$5e^{-4}$	$\mathbf{2.5e^{-3}}$	$5e^{-4}$
R_{thresh} (Ratio Limit)	3.0	**1.0**	3.0	3.0	3.0

TABLE X: Initial Conditions for Ergodicity Test

ID	Initial Herbivores (H_0)	Initial Carnivores (C_0)	Description
IC1	150	50	Balanced (3:1)
IC2	100	100	Equal (1:1)
IC3	50	70	Predator Dominated
IC4	100	10	Prey Dominated
IC5	400	50	Overshoot (8:1)

The contrast between stable coexistence and extinction motivates a formal analysis of **Ergodicity and Absorbing States**. Mechanically, the simulation contains several specific absorbing scenarios from which the system cannot fully recover:

1) **Reproductive Failure (Gender Extinction)**: If the population of Males or Females of a species drops to zero (e.g., $H_M = 0$), the conception rate becomes zero. The remaining individuals will eventually die via natural mortality, leading to total species extinction.
2) **Trophic Collapse (Total Extinction)**: If Herbivores go extinct ($H_{total} = 0$), the Carnivore population loses its food source. Due to the logistic starvation factor, the Carnivore mortality rate becomes maximal, leading to rapid secondary extinction ($\mathbf{S = 0}$).
3) **Predator Extinction**: If Carnivores go extinct ($C_{total} = 0$), the Predation rate becomes zero. The Herbivores are released from predation pressure and settle into a single-species equilibrium governed solely by their carrying capacity ($H \approx H_{thresh}$). While the Herbivores survive, the system as a whole has entered an absorbing subspace where Carnivores can never reappear.

Because there is always a non-zero probability of a random sequence of death events triggering one of these scenarios, the system is **theoretically not ergodic** (limit probability concentrates on extinction). However, experimentally, the system exhibits **Quasi-Stationary** behavior. In the stable configurations, the time required to reach these absorbing states is astronomically long, far exceeding the simulation duration. Therefore, for practical purposes, the system behaves as if it were ergodic. To verify this, the Baseline configuration results are plotted against the varying initial conditions in Figure 13. The plot demonstrates that regardless of whether the simulation starts with a massive overpopulation (IC5) or a predator-heavy state (IC3), all surviving runs converge to the same narrow band ($H \approx 160$, $C \approx 50$). This convergence confirms that the system's observed long-term behavior is effectively independent of the initial conditions.

To provide a rigorous statistical quantification of the equilibrium state, a final analysis is conducted focusing specifically on the **Baseline** configuration under **Initial Condition 1** ($H_0 = 150, C_0 = 50$). Twenty independent simulations are executed to compute the 95% Confidence Intervals for the

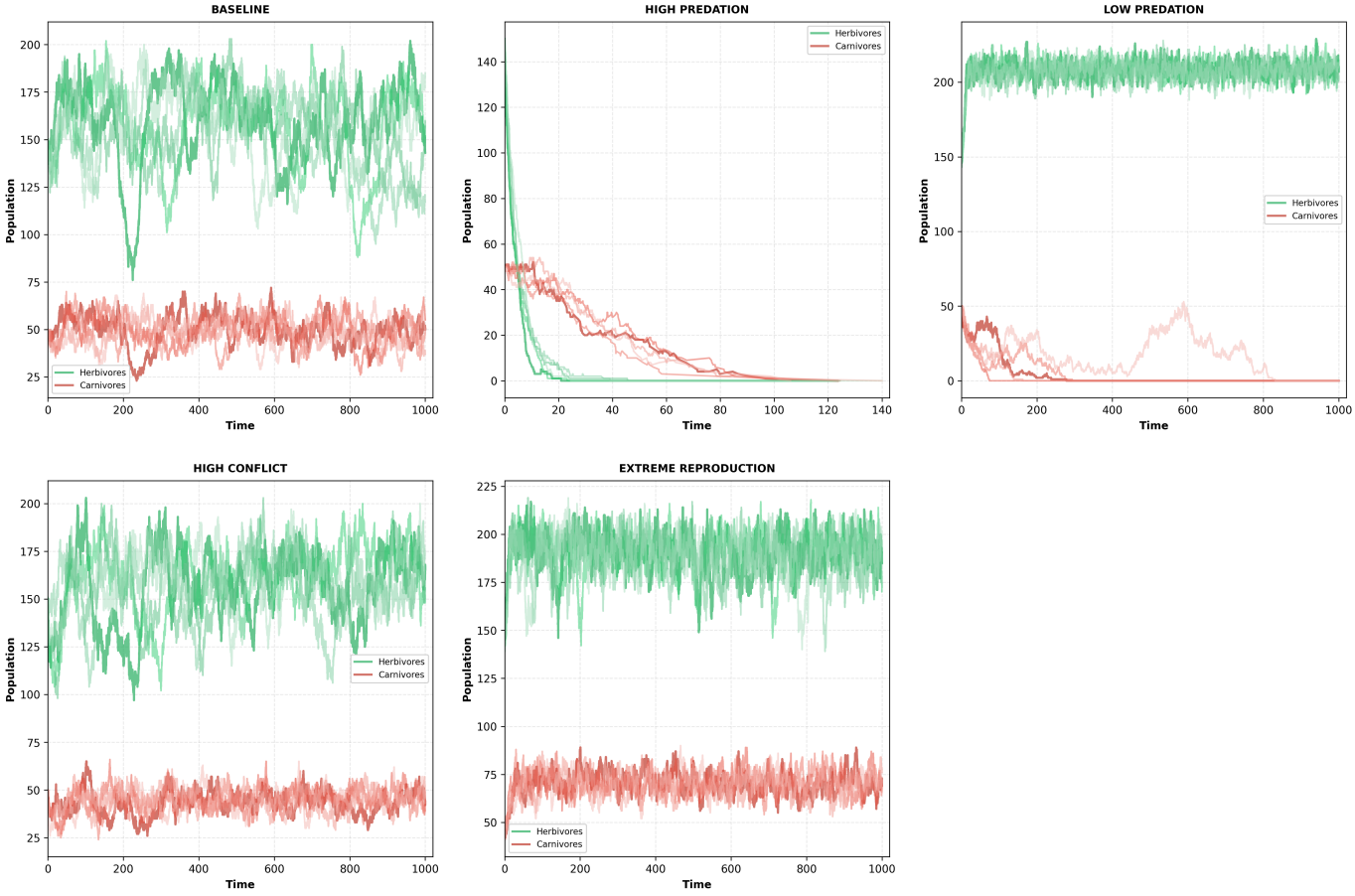


Fig. 11: Population trajectories for different configurations over time. The Baseline, High Conflict, and Extreme Reproduction scenarios show stable oscillations. High Predation leads to rapid extinction of both species (Green then Red drop to zero), while Low Predation results in Carnivore extinction (Red drops to zero).

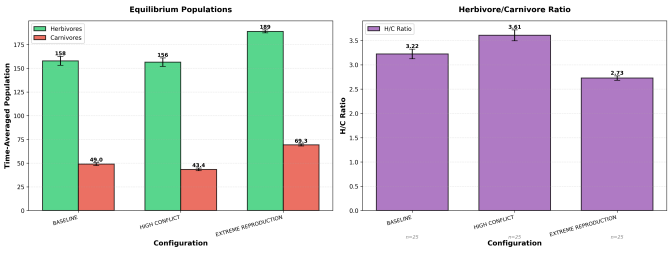


Fig. 12: Comparison of time-averaged equilibrium populations (left) and species ratios (right) for the surviving configurations. Error bars represent the standard error of the mean ($n = 25$).

time-weighted average populations. The results are visualized in Figure 14 and detailed in Table XI.

The statistical analysis confirms that the system reaches a stable equilibrium. With only 20 independent runs, the confidence intervals for all population metrics already have a relative width of less than 5% of their means. This indicates that the stochastic variability between different realizations is low once the steady state is achieved.

TABLE XI: Confidence Interval Results (Baseline, 95% Confidence, $n = 20$)

Metric	Mean	95% C.I.	Width
Herbivores			
Avg H_M	76.58	$[75.86, 77.30]$	1.45
Avg H_F	25.90	$[25.28, 26.53]$	1.25
Avg $H_{F_{preg}}$	54.67	$[53.70, 55.63]$	1.94
Avg H_{Total}	**157.15**	$[154.92, 159.38]$	**4.46**
Carnivores			
Avg C_M	23.03	$[22.62, 23.43]$	0.80
Avg C_F	14.16	$[13.91, 14.41]$	0.50
Avg $C_{F_{preg}}$	12.44	$[12.28, 12.60]$	0.33
Avg C_{Total}	**49.63**	$[49.08, 50.17]$	**1.09**

E. Generalizations beyond Markov Models

The model presented relies strictly on the Markov property, which implies that the time between events is exponentially

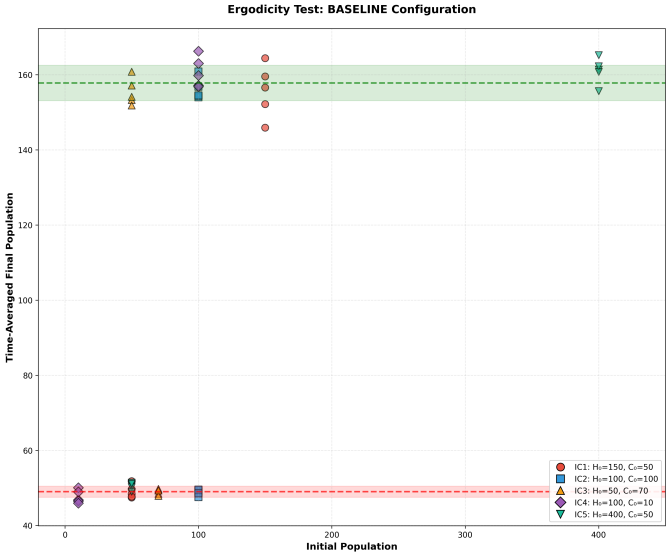


Fig. 13: Ergodicity test for the Baseline configuration. Different shapes represent the different Initial Conditions listed in Table X. All surviving runs converge to the same time-averaged equilibrium (green and red bands), confirming practical independence from the starting state.

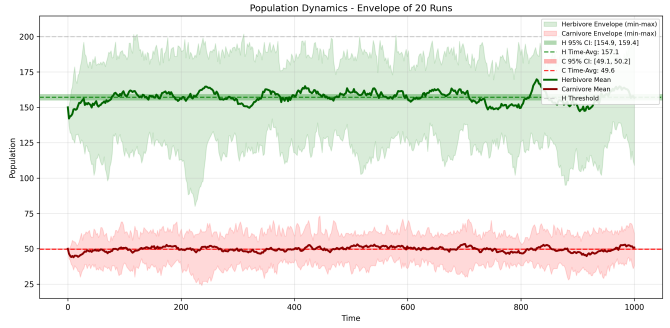


Fig. 14: Population dynamics for the Baseline configuration (IC1) showing the envelope of 20 runs. The shaded areas represent the 95% confidence intervals around the mean equilibrium values.

distributed and depends only on the current state configuration (memorylessness). While this assumption simplifies the mathematical framework and simulation logic, it introduces biological simplifications that could be addressed by adopting a more general, non-Markovian approach.

The most significant limitation concerns the **gestation period**. In the current CTMC model, the time from conception to birth follows an exponential distribution. This implies a non-zero probability of extremely short pregnancies or unrealistically long ones, as the ”countdown” to birth has no memory of how long the female has already been pregnant. A more realistic model would relax the memoryless assumption to support deterministic or low-variance distributions. In this context, the birth event would occur after a fixed duration following conception, thereby eliminating the possibility of immediate births.

A second critical relaxation involves the introduction of **age-dependent dynamics**. Currently, all individuals within a specific state (e.g., Herbivore Males) are stochastically identical: a newborn has the same mortality risk and reproductive potential as a mature adult. In biological reality, vital rates are strongly correlated with age: fertility is zero before sexual maturity, and mortality typically increases with senility. Generalizing the model to track the age of individuals—or at least distinguishing between juvenile and adult stages—would violate the Markov property (since the future depends on the time of birth) but would significantly enhance the accuracy of the population dynamics.

Finally, the assumption of time-homogeneity could be relaxed to simulate **seasonality**. The current transition rates depend only on population density, assuming a constant environment. However, real ecosystems are subject to cyclic variations where resource availability, and consequently birth and death rates, fluctuate deterministically with time (e.g., seasonal breeding seasons or winter scarcity). Integrating these time-dependent functions would transform the system into a non-homogeneous process, allowing for the study of how populations synchronize with environmental cycles.