Beltone 2nd AI Hackathon

*Powered by:* ***Robin***

# **Problem Statement Overview:**

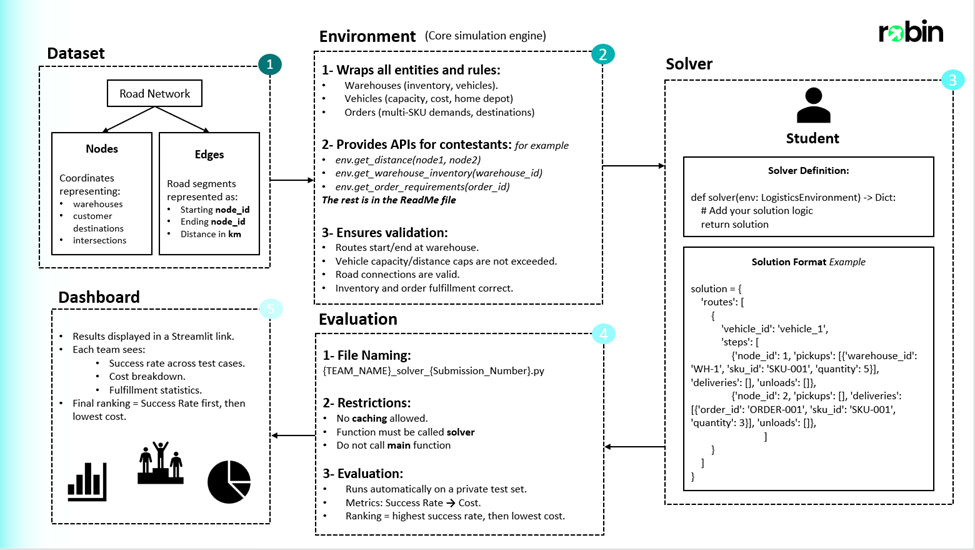
Efficient multi-warehouse vehicle routing problems **(MWVRP)** are a critical challenge in modern logistics. Traditional algorithms struggle to handle the complexity, having to account for:

* Limited inventory distributed across multiple warehouses
* Directional road network constraints
* Vehicular capacity limits (weight & volume)
* Multi-warehouse pickup coordination
* Real-time inventory tracking during routing

Participants must design an **intelligent optimization solver** that dynamically plans delivery routes which:

* Respect road network connectivity
* Allocate and track inventory across warehouses
* Ensure vehicle capacity feasibility
* Sequence pickups across multiple depots correctly
* Optimize cost, efficiency, and fulfillment rates

The solution will be evaluated on **order fulfillment, operational cost minimization, vehicle utilization, and real-world feasibility** under constrained and imperfect logistics conditions.



***and API Reference file***

# **Dataset:**

The dataset used is a collection of entries that represent locations in all of Cairo. The dataset models roads and locations in Cairo as a graph network comprised of nodes and edges. This data is represented as follows:

1. Nodes: These represent a landmark in Cairo identified as co-ordinates.
2. Edges: These represent directional links between 2 Nodes (not necessarily bidirectional), identified using the 2 nodes and the length between them.

This image shows a small portion of the dataset highlighting the area from Masr el-gedida and Madinat Nasr to the beginning of El-Rehab City.



# **Environment:**

The **Robin Logistics Environment** simulates a large-scale real-world logistics network for solving **multi-warehouse vehicle routing problems (MWVRP)** under strict operational constraints.

* **Road Network:** 332k nodes, directional/bi-directional roads, Dijkstra-based pathfinding.
* **Warehouses:** Multiple depots with limited inventories and defined vehicle fleets.
* **Vehicles:** Light vans to heavy trucks, capacity (weight & volume) limits, fixed + variable costs, max distance constraints.
* **Orders & Inventory:** Multi-SKU customer orders, distributed demand, real-time inventory tracking, scarcity-driven allocation.
* **Constraints:**
  + Routes start/end at home warehouses
  + Capacity checks at every step
  + Valid road connections only
  + Full order fulfillment (partials take points but not fully)
  + One route per vehicle
  + Running-Time of solvers submitted should **not** **exceed 30 minutes** (more than this won’t be evaluated)
  + **Environment manipulations or altercations is NOT ALLOWED** (don’t play in environment, which is robin-logistics-env)
  + **Building a string solution is not allowed, solver is being evaluated against environment and scenarios.**
  + **Has to be a Python (.py) file.**
* Allowed:
  + Multi-pickup from several warehouses
  + Multi-vehicle per order
  + Unload to any warehouse
  + Directed roads
  + Vehicles start/end at home warehouse

**Challenge Characteristics *(Variable to change according to scenario)*:** 50 orders, 12 vehicles, 3 SKUs, 2 Warehouses and constrained resources over a massive road network. Solutions must optimize **cost, capacity use, and fulfillment rates** while passing strict **validation on connectivity, inventory, and business logic compliance**.

Please use **“pip install robin-logistics-env"** to install the environment.

# **Evaluation:**

## ***Process***

* Solvers are tested on **Private Scenarios** (hidden) and **5 Public Scenarios** (for local testing).
* Each private scenario evaluates both **cost efficiency** and **fulfillment rate**.
* Presentation will be held for **top teams** after deadline, to present solutions against the panel, **affecting final ranking**

## ***Scoring Equation***

For every scenario:

Scenario Score=Your Scenario Cost + Benchmark Solver Cost × (100 − Your Fulfillment %)

* **Lower = Better**
* Missing fulfillment is heavily penalized.
* Once fulfillment is high, cost efficiency becomes the differentiator.

## ***Dynamic Ranking & Points***

1. After computing scenario scores for all solvers, they are **ranked dynamically** (lowest → highest score).
2. Points are awarded based on rank:
   * Rank 1 = 20 points
   * Rank 2 = 19 points
   * Rank 3 = 18 points
   * … and so on.
3. Final score = **sum of points across all private scenarios**.

This ensures:

* Rankings **shift dynamically** as new solvers are submitted.
* Strong, consistent performance across scenarios matters more than excelling in just one.
* Even a single weak scenario can change the leaderboard significantly.

## ***Leaderboard***

* Updated **live** throughout the hackathon.
* Displays each team’s **total points** across all scenarios.
* The leaderboard is **dynamic** — every submission can reshuffle rankings.

# **Submission:**

Link for the submission portal: [AI Hackathon Submission](http://hackathon-frontend2025.s3-website-us-east-1.amazonaws.com/)

All participating teams have to follow a strict pipeline for submissions. The submission process will be done as:

1. After testing the model locally, the model should be saved as a .py file using the following naming convention {TEAM\_NAME}\_solver\_{Submission\_Number} e.g.(Robin\_solver\_1), The submission number starts from 1 and is incremented by 1 for every submission. (**It is the team’s responsibility to ensure the correct naming for each submission, any inconsistencies may result in an unevaluated submission)**
2. The Model is then evaluated automatically using a private testing set.
3. The results will be posted on a live leaderboard hosted on Streamlit. [Results Dashboard](http://44.222.203.147:8502/)

# **Rules and Regulations:**

* The submitted file must have a main function defined as follows:

def solver(env):

* The submitted file **MUST NOT** use any caching techniques.
* **Do not import or initialize** the environment inside the solver function only in main
* **Comment out the main function** when submitting the solver file

# Starter Skeleton — baseline solver to get participants started

Below is a compact, ready-to-run baseline skeleton you can include in your repository and adapt. It is intentionally simple and **meant as a launchpad**. Use it to get fast local runs and then iterate.  
  
**Solver skeleton is included as a python file named solver.py  
  
----------------------------------------------------------------------------------------------------------------------------------**

# Local Scenario Dashboard – Statistics based on your solver (optional): run\_dashboard.py is a file that when runs a dashboard that helps you to visualize your solver – **this is an optional tool** **Dashboard is included as a python file named run\_dashboard.py**

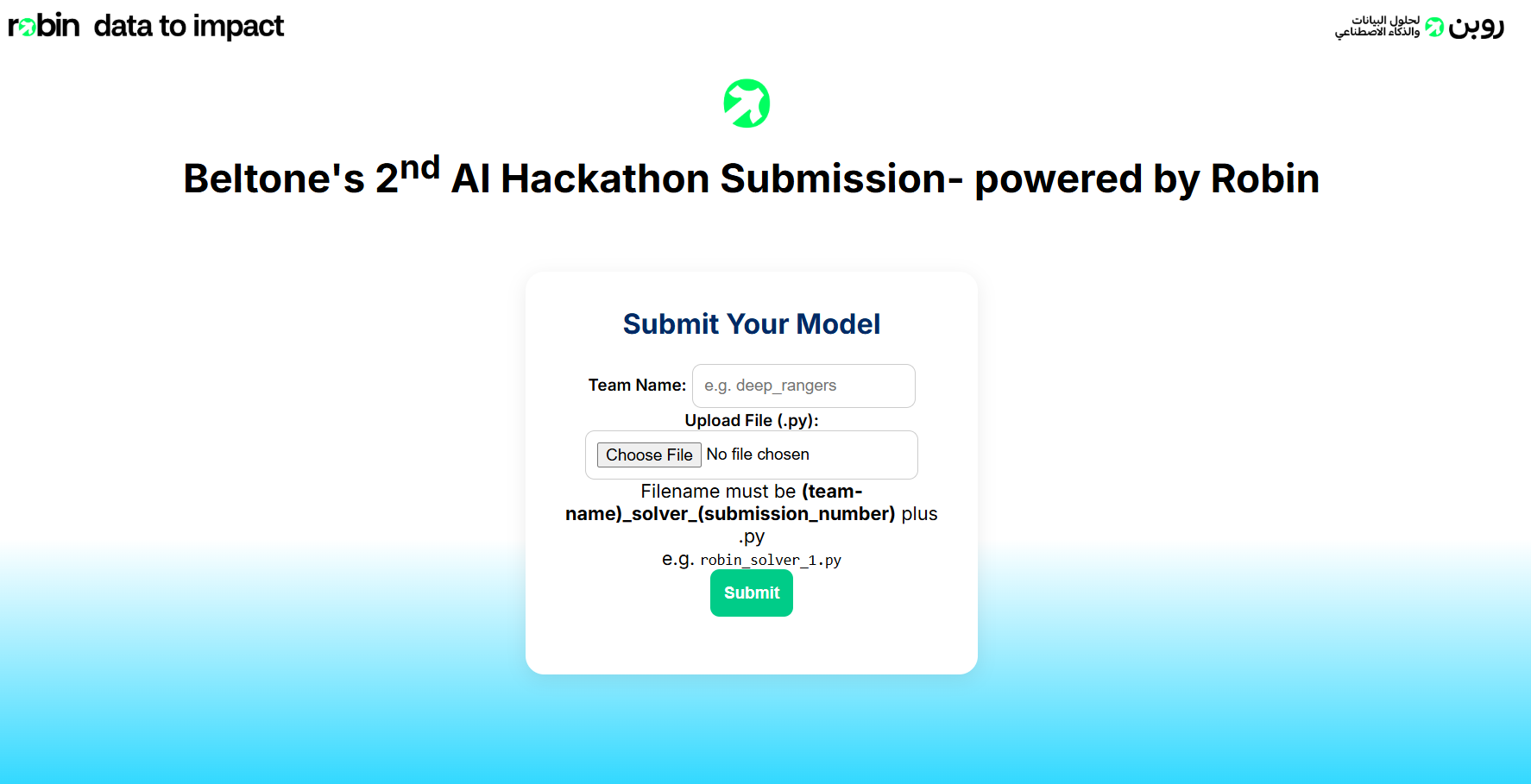
**----------------------------------------------------------------------------------------------------------------------------------**

Functions Documentation Excel file:  
Included is an excel file named functions\_exmaples\_documentation, which contains a documented rows of functions , with examples of it’s input and output used for the Hackathon Problem.

**Excel file name is functions\_examples\_documentation**

**Submission Portal Link:** [AI Hackathon Submission](http://hackathon-frontend2025.s3-website-us-east-1.amazonaws.com/)

**When Submitting the Solver file, it should be named according with your team name, like example below:  
  
You team name is : Robin**

**Then the solver file you’re submitting should be accordingly:  
Robin\_solver\_1.py (DO NOT WRITE MANULLAY .py , THIS IS FILE TYPE)  
and inside the portal in team name area you write , Robin (has to be identical to the team name naming of the solver file you submitted)  
  
like this image below:  
  
  
If you did another iteration (you can submit more than once to increase your ranking), then you should increment the number of submission id (eg: 1), to be 2 , then a 3rd iteration to be 3 , and so on…  
example : 2nd iteration: Robin\_solver\_2.py**

**example : 3nd iteration: Robin\_solver\_3.py**