

IE437 Final Project Guidelines

Solving Real-world Decision-Making Problems with AI Agent

Dept. of Industrial & Systems engineering, KAIST

Jinkyoo Park



Contents

1. Motivation of the project

2. Requirements

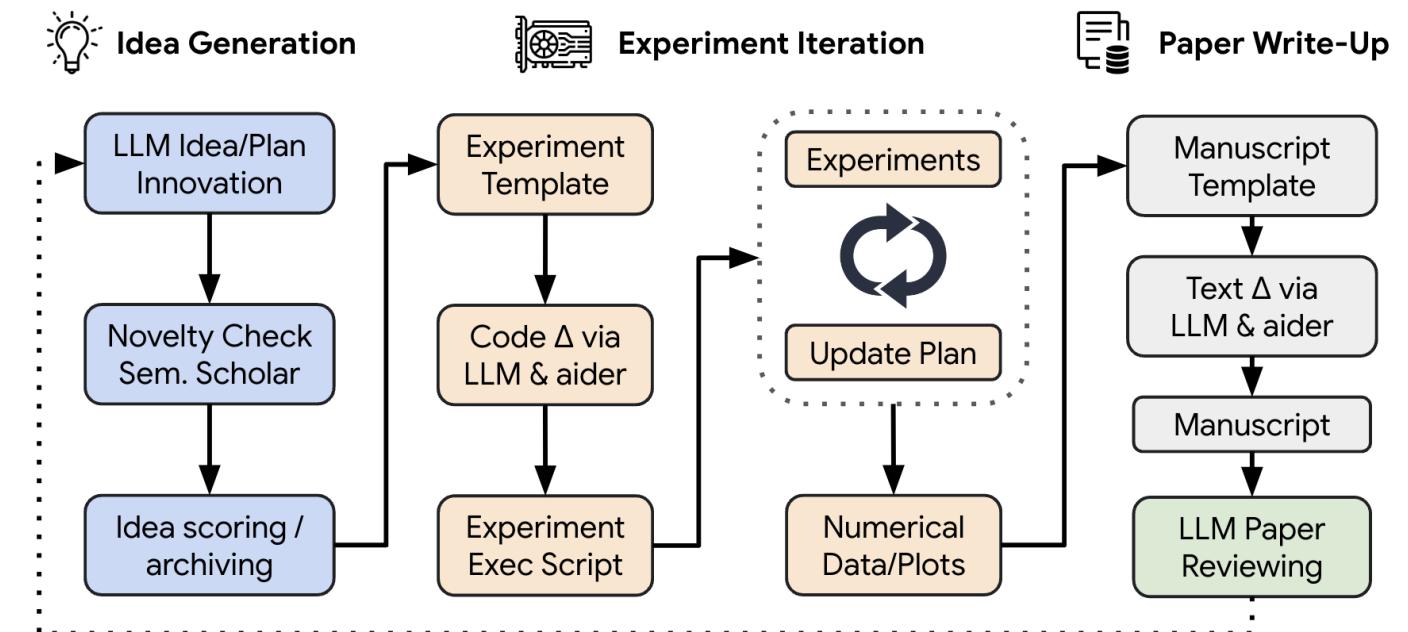
3. Demo

4. Evaluation Criteria

Motivation

Why an AI-Agent Final Project?

- AI agents are becoming a core paradigm for real-world decision making. They can plan, reason, and interact with tools in dynamic environments.
- Traditional optimization works well, but human experts should explicitly define objectives and constraints, choose appropriate algorithms, compare the results by their own.
- LLM-based agents can adapt, interpret instructions, and improve decision quality. This project lets you explore how agents solve practical decision-making problems.



Requirements

Installation)

- We need to setup the following things:
 - **uv:** <https://pydevtools.com/handbook/how-to/how-to-install-uv/>
 - **git:** <https://git-scm.com/install/windows>

How to install uv

macOS/Linux [Windows](#)

Open PowerShell and run:

```
powershell -ExecutionPolicy Bypass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

The `-ExecutionPolicy Bypass` flag allows running the installation script from the internet.

Verifying Installation

After installation, restart your terminal and verify uv is working:

```
uv --version
```

The screenshot shows the official Git website at git-scm.com. The header features the Git logo and the tagline "distributed-is-the-new-centralized". A search bar is located in the top right. The main navigation menu includes links for About, Learn, Tools, Reference, Install (which is highlighted), and Community. Below the menu, there's a sidebar with information about the Pro Git book and links to Amazon.com. The main content area is titled "Install" and shows options for Windows, macOS, Linux, and Build from Source. It includes a call-to-action button for downloading the latest version (2.51.2) and links to other download options like Standalone Installer and Git for Windows/x64 Setup.

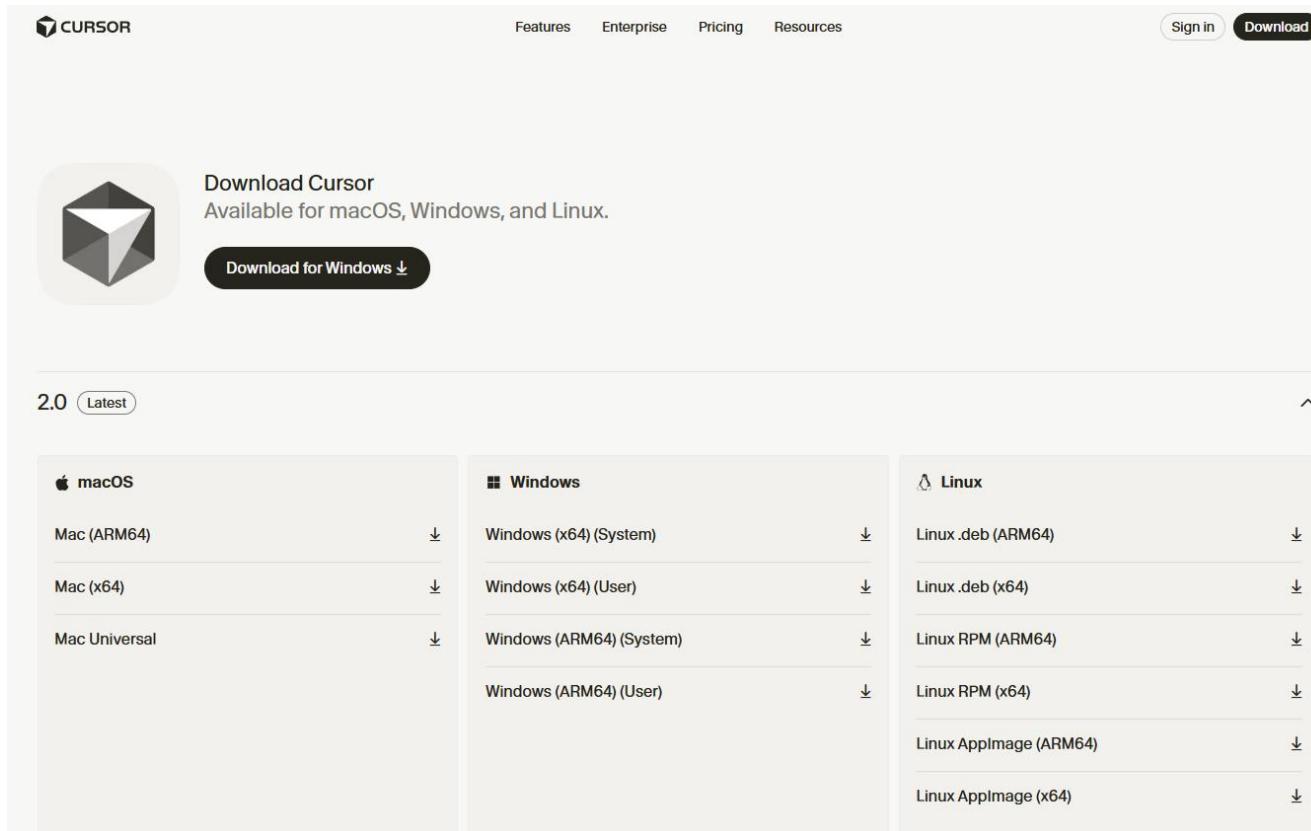
Requirements

Installation)

- We need to setup the following things:

- **MCP server**

- Step 1: Install cursor (<https://cursor.com/download>)



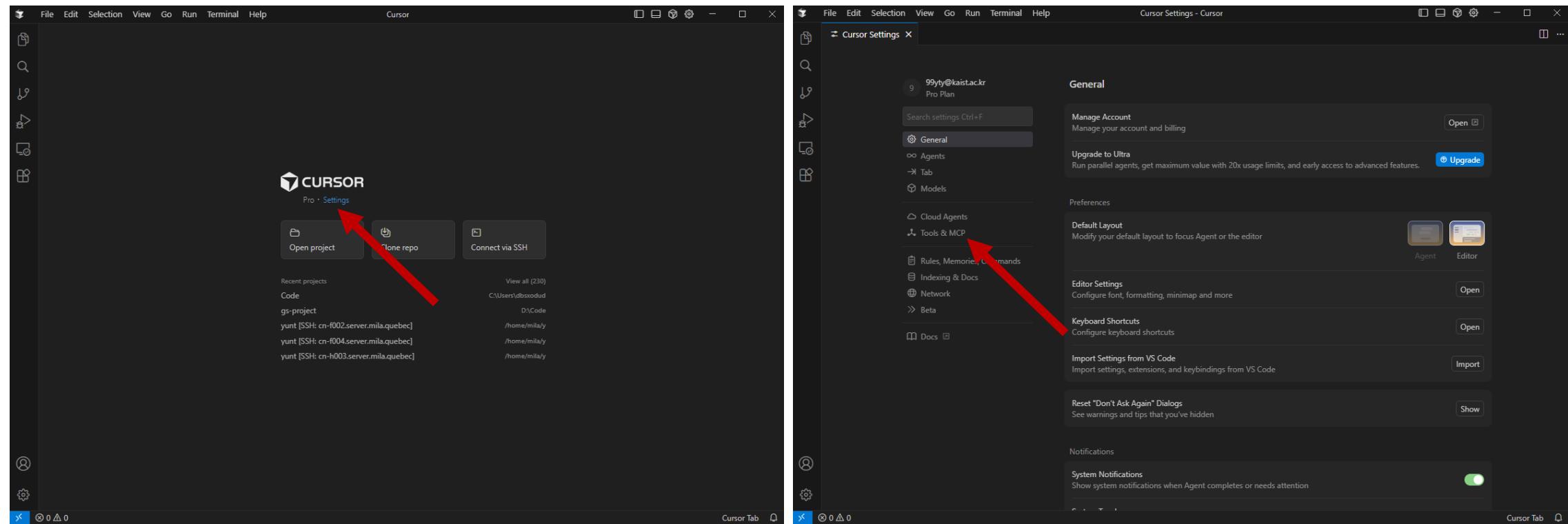
The screenshot shows the 'Download Cursor' page on cursor.com. At the top right, there are 'Sign in' and 'Download' buttons, with a red arrow pointing to the 'Download' button. Below the buttons, there's a section for 'Download Cursor' available for macOS, Windows, and Linux, featuring a large icon of a geometric cube. A 'Download for Windows' button is visible. The main content area shows download links for various operating systems and architectures. The 'Latest' version is 2.0.

macOS	Windows	Linux
Mac (ARM64)	Windows (x64) (System)	Linux .deb (ARM64)
Mac (x64)	Windows (x64) (User)	Linux .deb (x64)
Mac Universal	Windows (ARM64) (System)	Linux RPM (ARM64)
	Windows (ARM64) (User)	Linux RPM (x64)

Requirements

Installation)

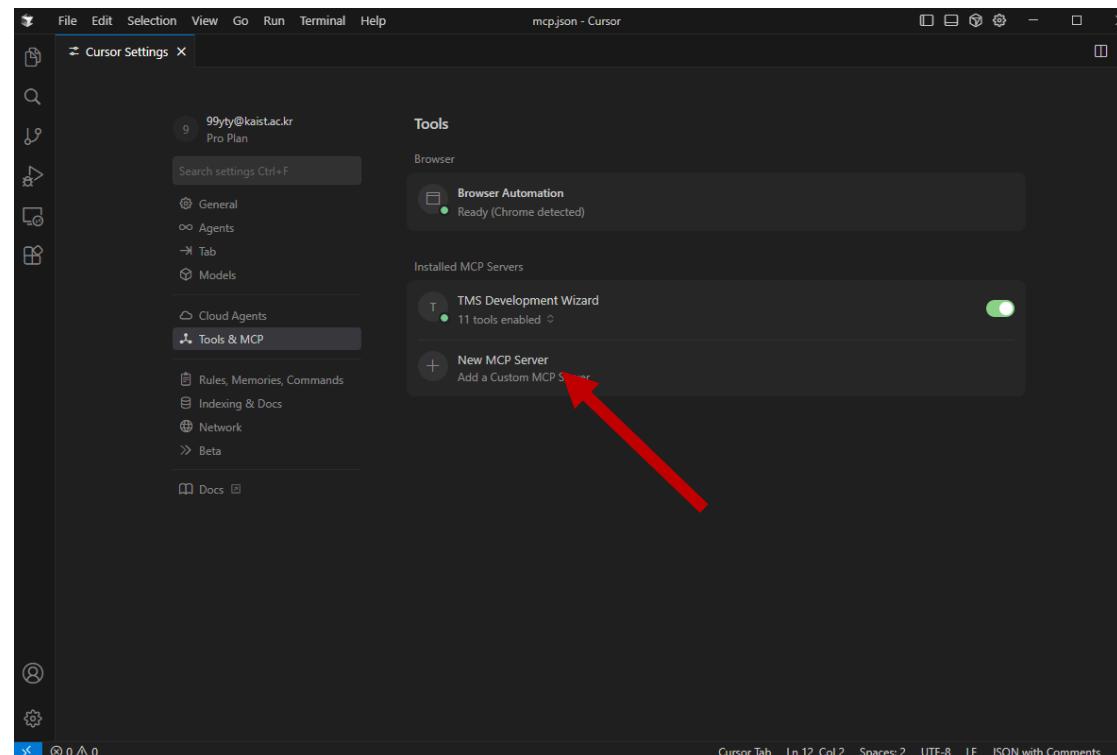
- We need to setup the following things:
 - **MCP server**
 - Step 2: Open cursor → Settings → Tools & MCP



Requirements

Installation)

- We need to setup the following things:
 - **MCP server**
 - Step 3: New MCP server → copy *mcp.json*, *.env* provided in the zip file



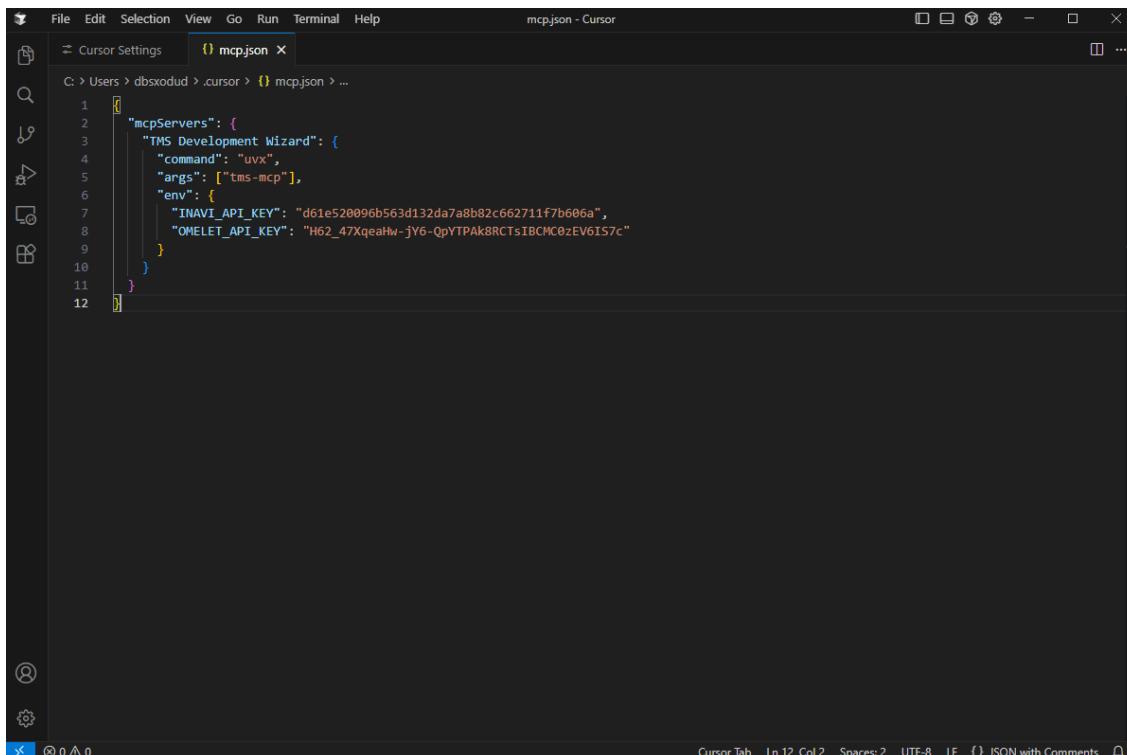
A screenshot of a code editor showing the contents of the "mcp.json" file. The file path is "C > Users > dbsxodud > .cursor > mcp.json". The code is as follows:

```
1 "mcpServers": {
2   "TMS Development Wizard": {
3     "command": "uvx",
4     "args": ["tms-mcp"],
5     "env": {
6       "INAVI_API_KEY": "d61e520096b563d132da7a8b82c662711f7b606a",
7       "OMELET_API_KEY": "H62_47XqealHw-jY6-QpYTPAK8RCTsIBCMC0zEV61S7c"
8     }
9   }
10 }
11 }
12 }
```

Requirements

Installation)

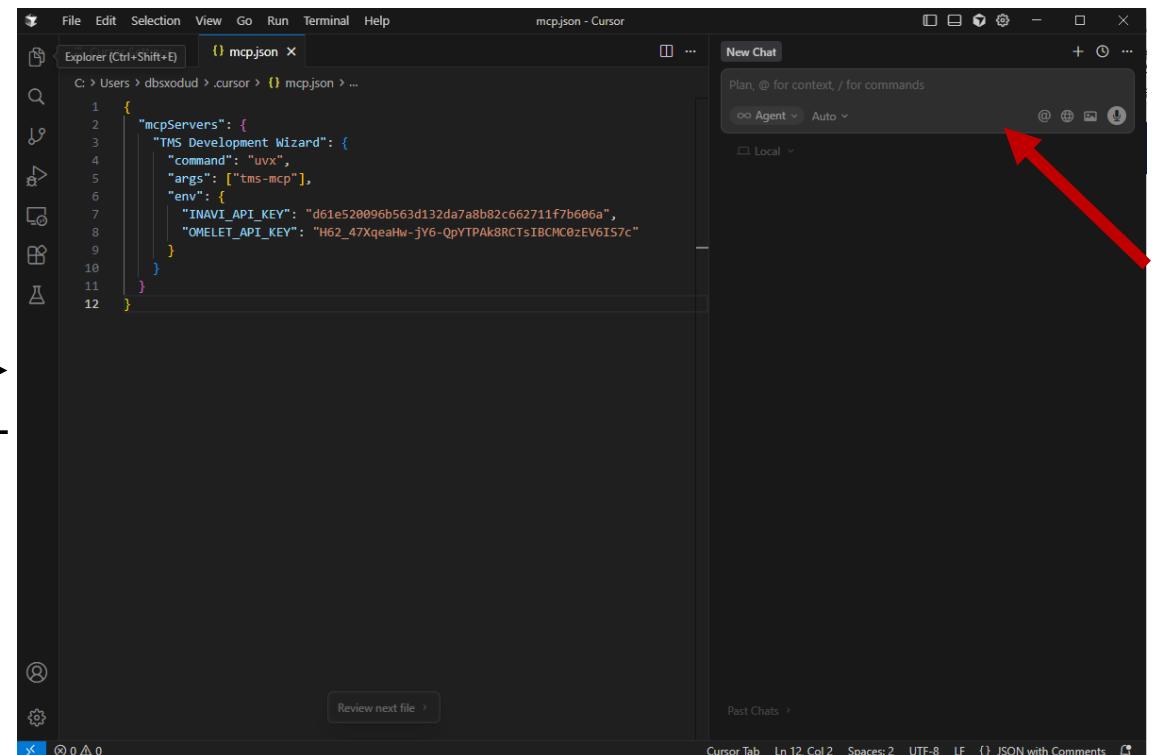
- We need to setup the following things:
 - **MCP server**
 - Step 4: Using Ctrl + L to start chatting with AI Agent



```
mcp.json - Cursor
File Edit Selection View Go Run Terminal Help
C:\Users\dbsxodud\.cursor\mcp.json

1 "mcpServers": {
2   "TMS Development Wizard": {
3     "command": "uvx",
4     "args": ["tms-mcp"],
5     "env": {
6       "INAVI_API_KEY": "d61e520096b563d132da7a8b82c662711f7b606a",
7       "OMELET_API_KEY": "H62_47XqealHw-jY6-QpYTPAk8RCTsIBCMC0zEV6IS7c"
8     }
9   }
10 }
11 }
12 }
```

Ctrl + L



```
mcp.json - Cursor
File Edit Selection View Go Run Terminal Help
C:\Users\dbsxodud\.cursor\mcp.json

1 "mcpServers": {
2   "TMS Development Wizard": {
3     "command": "uvx",
4     "args": ["tms-mcp"],
5     "env": {
6       "INAVI_API_KEY": "d61e520096b563d132da7a8b82c662711f7b606a",
7       "OMELET_API_KEY": "H62_47XqealHw-jY6-QpYTPAk8RCTsIBCMC0zEV6IS7c"
8     }
9   }
10 }
11 }
12 }
```

Demo

Tips for writing effective prompts)

- To get high-quality results from the Routing MCP Server, follow these key principles:
 - Step 1: Exploration
 - Move gradually from high-level understanding to implementation (e.g., review the API overview, check the schemas, study examples, ...)
 - Step 2: Test loop
 - Adopt an iterative development cycle: Implement → Test → Fail → Debug → Retest.
 - This loop dramatically improves reliability and solution quality.
 - Step 3: Plan Before You Execute
 - For complex tasks, always begin by sending a “planning prompt”. Let the agent outline the steps first, then proceed to execution.
 - Step 4: Provide Concrete Requirements
 - Be specific about what you want. Clear constraints leads to better results.
 - Step 5: Consider stochasticity of LLM
 - LLMs are probabilistic. The same prompt may produce different outputs, so always test multiple times when needed.

Demo

Example: Seoul Delivery VRP Demo)

- Suppose that we want to make react-based dashboard with following attributes:
 - Generate N random delivery points in Seoul
 - Let user add multiple logistics centers (DCs) by clicking on the map
 - Assign each delivery point to the nearest DC and color points and DCs by assignment
 - Draw optimal routes for delivery

Problem 9 (20 Points)

Story. A convenience-store company plans to redesign its logistics system. It must decide where to locate new regional warehouses that will supply hundreds of stores across the country. This problem involves two levels of decision-making:

- **Upper level (strategic):** Select warehouse locations.
- **Lower level (operational):** For each set of chosen warehouses, determine delivery routes to stores and estimate the resulting logistics cost.

The company's goal is to minimize the overall logistics cost while maintaining timely deliveries. However, evaluating each possible location combination requires running a complex routing simulation, which is time-consuming and uncertain. You are asked to reason through this hierarchical decision problem and propose an effective strategy for solving it.

Demo

Example: Seoul Delivery VRP Demo)

- Here is an example prompt we can use:

Below are the requirements. Please design and implement the full project based on them. I want to build a demo application with the following features: "Generate N delivery locations in Seoul → Specify logistics center locations → Assign delivery locations to centers based on road distance → Color mapping → 3-vehicle VRP optimization per center → Map visualization."

Requirements:

- When the user click Number of delivery points and write a number N, generate N random delivery points (latitude/longitude) within the boundary of Seoul and display them on a map.
- When the user clicks on the map, add that location as a logistics center (DC). Multiple DCs should be supported.
- For every delivery point, use the iNavi road-distance API to compute the actual road travel distance to each DC, and assign the point to the nearest DC based on road distance.
- Use color mapping so that all delivery points assigned to a DC share the same color as that DC's icon.
- For each DC, assign 3 vehicles and call the Omelet Routing Engine /vrp API to compute optimal routes.
- Display the resulting routes on the map as polylines. The UI should be a React-based map dashboard.
- After the design is complete and approved, implement the application according to the design.

Demo

Example: Seoul Delivery VRP Demo)

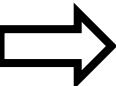
- Generating....

⌚ mcp.json Below are the requirements. Please design and implement the full project based on them. I want to build a demo application with the following features: "Generate N delivery locations in Seoul → Specify logistics center locations → Assign delivery locations to centers based on road distance → Color mapping → 3-vehicle VRP optimization per center
↳ Map visualization"

Creating the design document, then implementing.

To-dos 8

- Create design document for the demo application architecture
- Set up React project structure with dependencies
- Implement API service layer (iNavi distance matrix, Omelet VRP)
- Create map component with Leaflet integration
- Implement delivery point generation (Seoul boundaries)
- Implement DC selection and assignment logic
- Implement color mapping and visualization
- Implement VRP optimization per DC with route visualization



⌚ mcp.json Below are the requirements. Please design and implement the full project based on them. I want to build a demo application with the following features: "Generate N delivery locations in Seoul → Specify logistics center locations → Assign delivery locations to centers based on road distance → Color mapping → 3-vehicle VRP optimization per center
↳ Map visualization"

⌚ 8 of 8 To-dos Completed

- ❖ .gitignore +25 -1
- TS index.ts +91 -1
- TS constants.ts +24 -1
- TS colors.ts +20 -1
- TS inaviApi.ts +118
- TS omeletApi.ts +75 -1
- TS useDeliveryPoints.ts +32
- TS useDCAssignment.ts +87 -1
- TS useVRPOptimization.ts +140 -1
- ❖ MapView.tsx +174 -1
- ❖ ControlPanel.tsx +160 -1
- ❖ App.tsx +145
- ❖ main.tsx +11 -1
- # index.css +34 -1
- ❶ README.md +131 -1
- ⬇ QUICKSTART.md +105 -1

To run

1. Install dependencies:
`npm install`
2. Start dev server:
`npm run dev`
3. Open browser to the URL shown (typically <http://localhost:5173>)

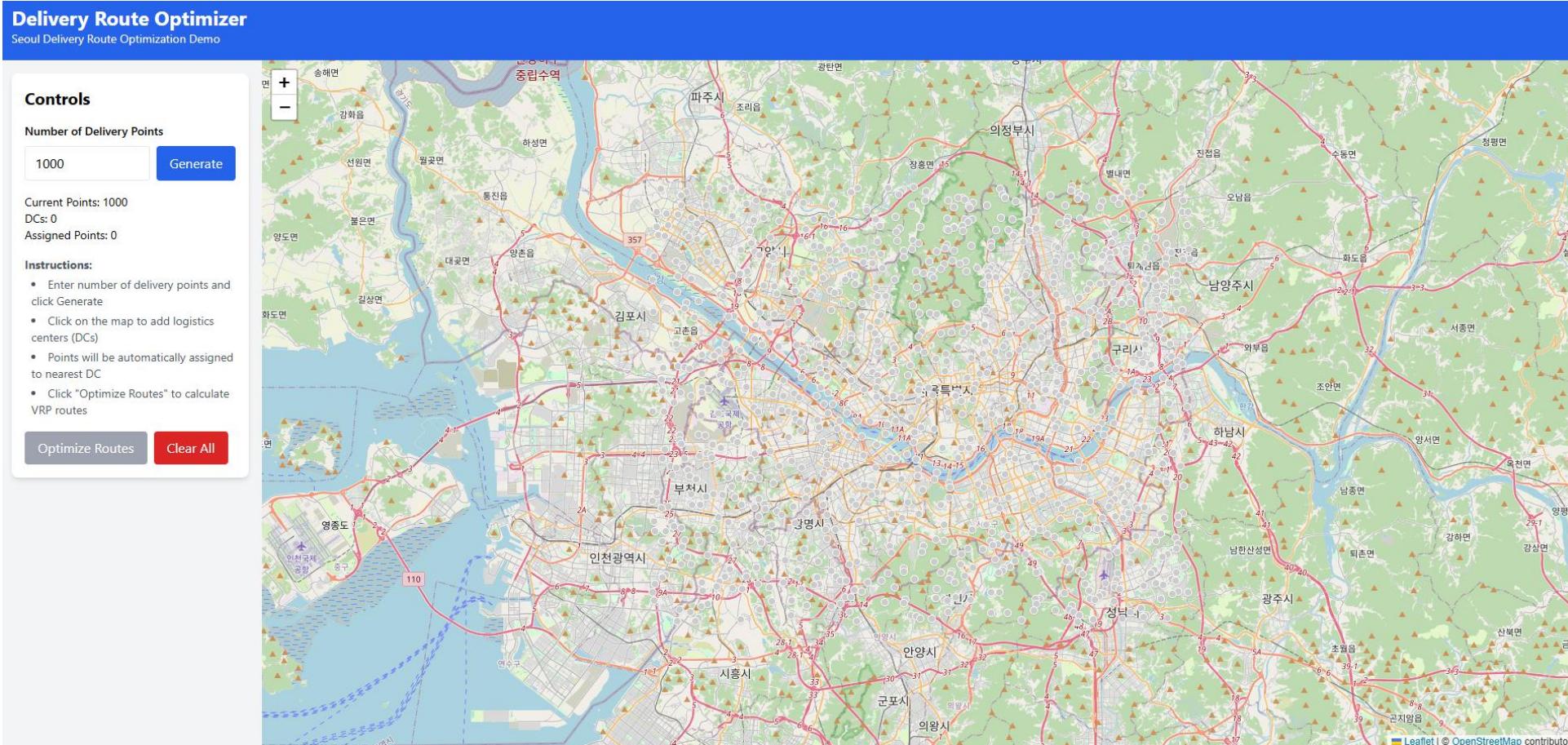
Documentation

- [DESIGN.md](#) - Architecture and design decisions
- [API_OVERVIEW.md](#) - API documentation and examples
- [README.md](#) - Project documentation
- [QUICKSTART.md](#) - Quick start guide

Demo

Example: Seoul Delivery VRP Demo)

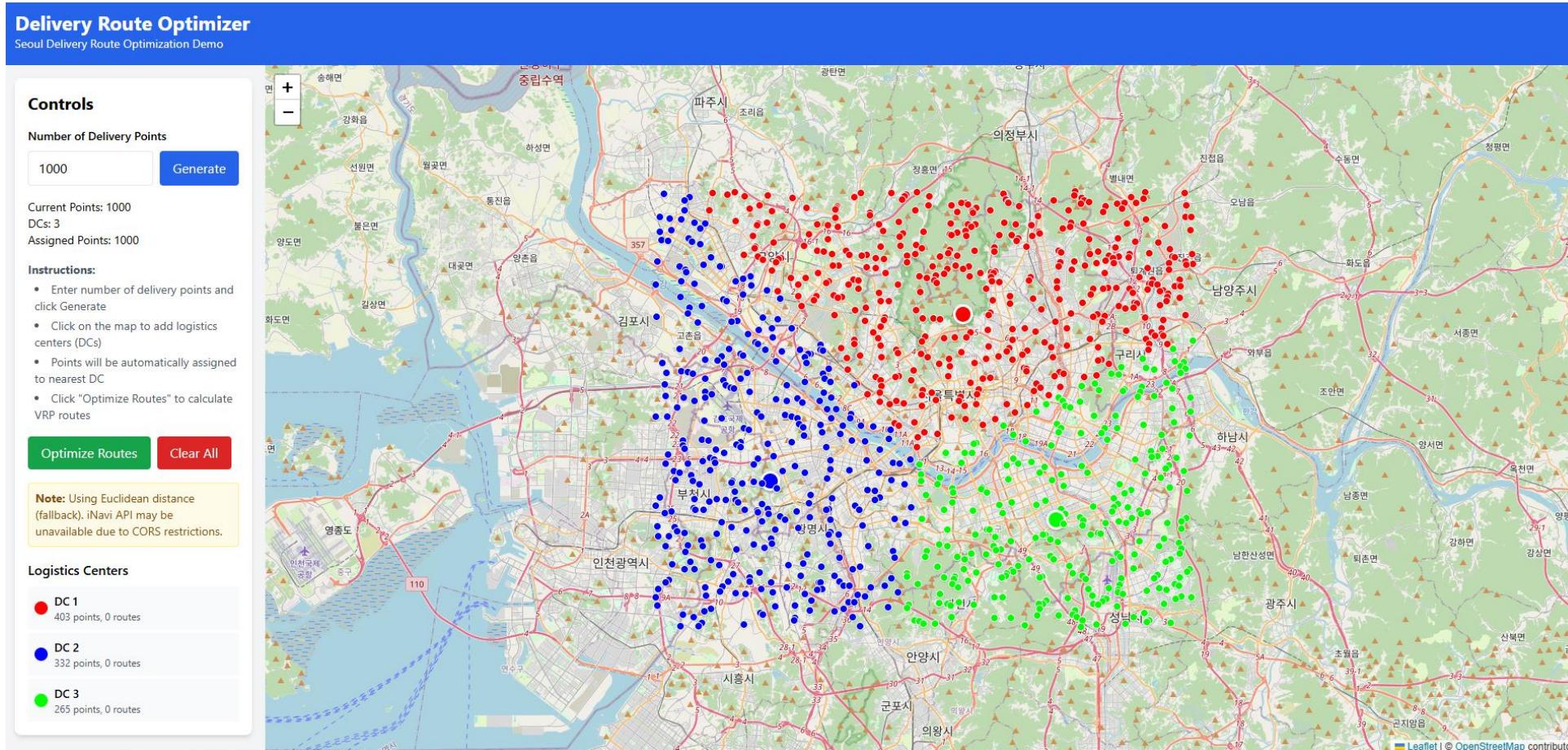
- Demo Browser



Demo

Example: Seoul Delivery VRP Demo)

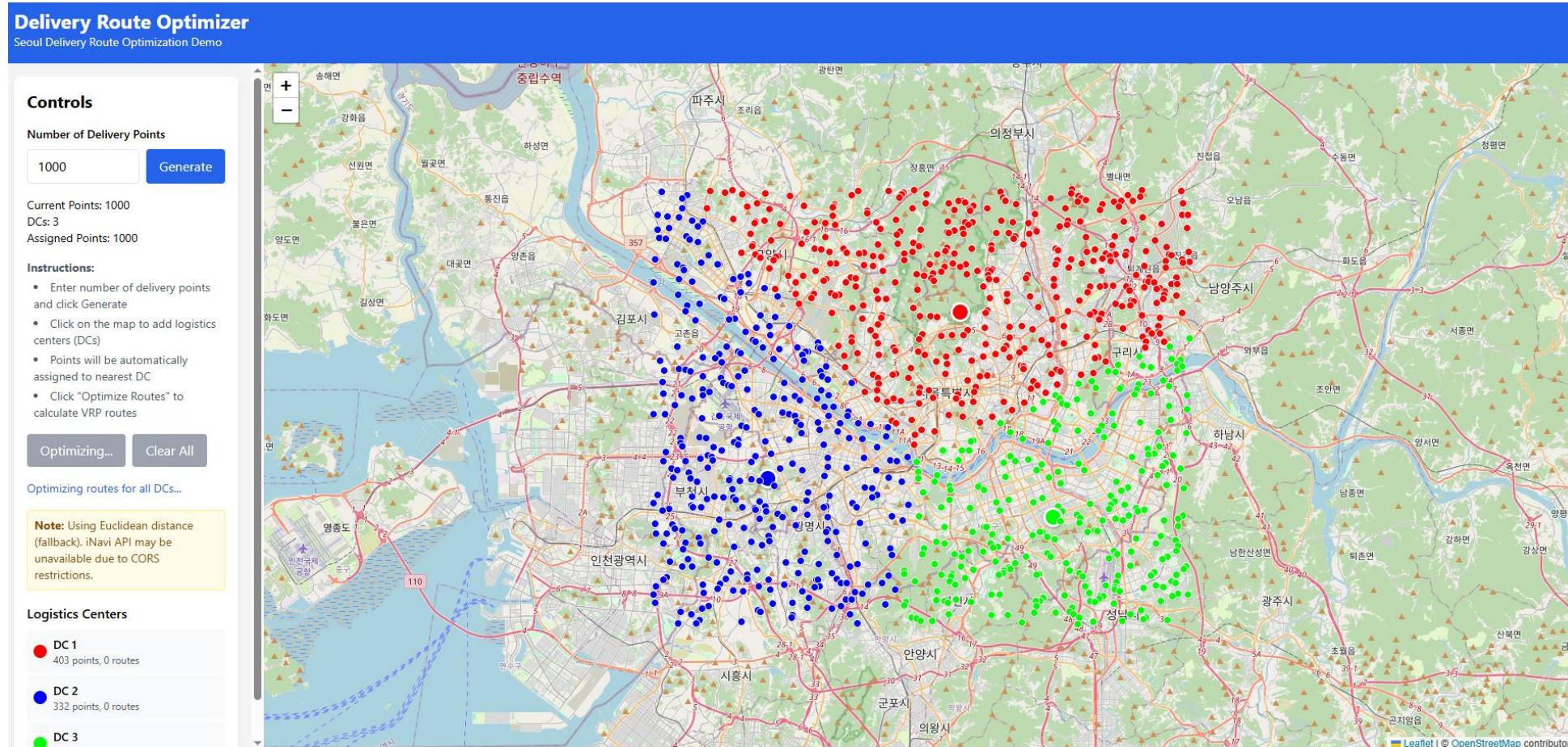
- Demo Browser



Demo

Example: Seoul Delivery VRP Demo)

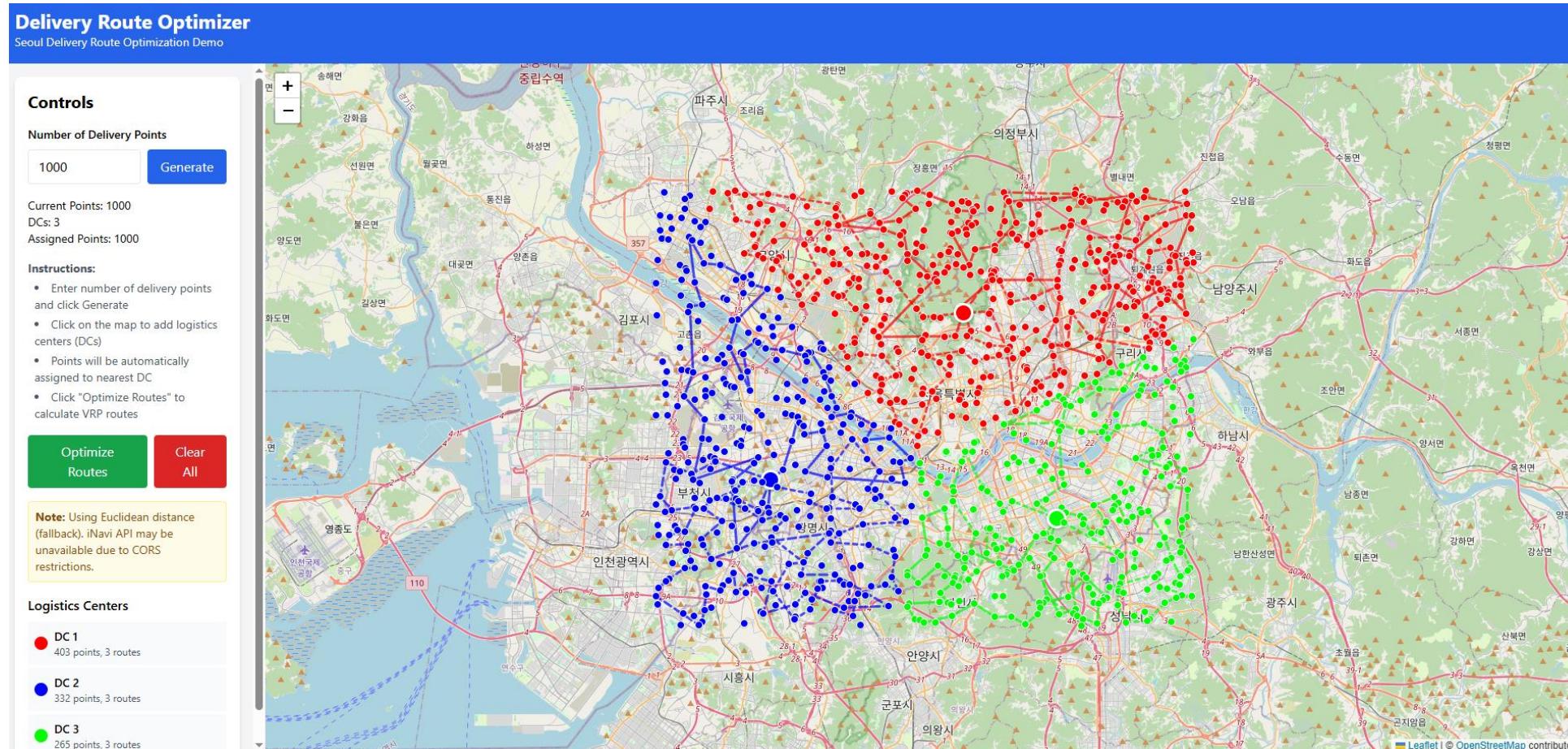
- Demo Browser



Demo

Example: Seoul Delivery VRP Demo)

- Demo Browser



Evaluation Criteria

Total: 40 points

Submission Files: Project Report (pdf) / Source Code (zip file)

Team: 1 to 4 (For more people, we anticipate more quality)

- **Problem Definition (10 pts)**

- Clarity, realism, and justification of the decision-making problem

- **Optimization Modeling (10 pts)**

- Quality of Routing API input design (vehicles, stops, constraints, parameters).

- **Demo App Quality (10 pts)**

- Functionality, UI/UX, stability, and completeness of the application.

- **Report (10 pts)**

- Organization, readability, and insights of the report.

Evaluation Criteria

You can also do final project as an algorithmic development

Total: 40 points

Submission Files: Project Report (pdf) / Source Code (zip file)

Team: 1 to 4 (For more people, we anticipate more quality)

- **Problem Definition (10 pts)**
 - Clarity, realism, and justification of the decision-making problem
- **Optimization Modeling (10 pts)**
 - Correctness and completeness of the proposed method
- **Effectiveness of the Method on the problem (10 pts)**
 - Empirical evaluation or theoretical analysis on the proposed method
- **Report (10 pts)**
 - Organization, readability, and insights of the report.