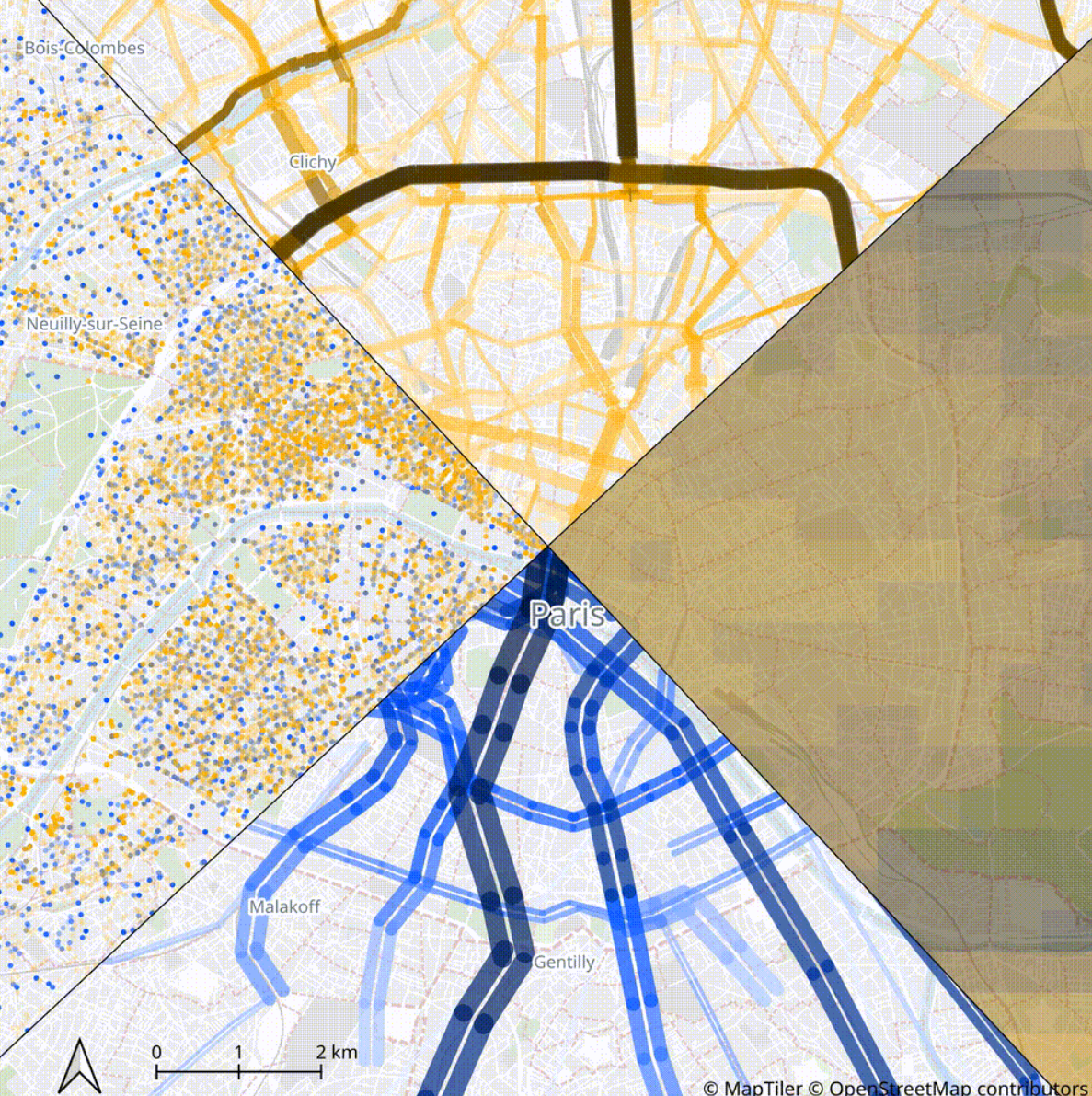




METROPOLIS2 : Technical Presentation

Lucas Javaudin

LVMT, Institut Polytechnique de
Paris

METROPOLIS2 in brief

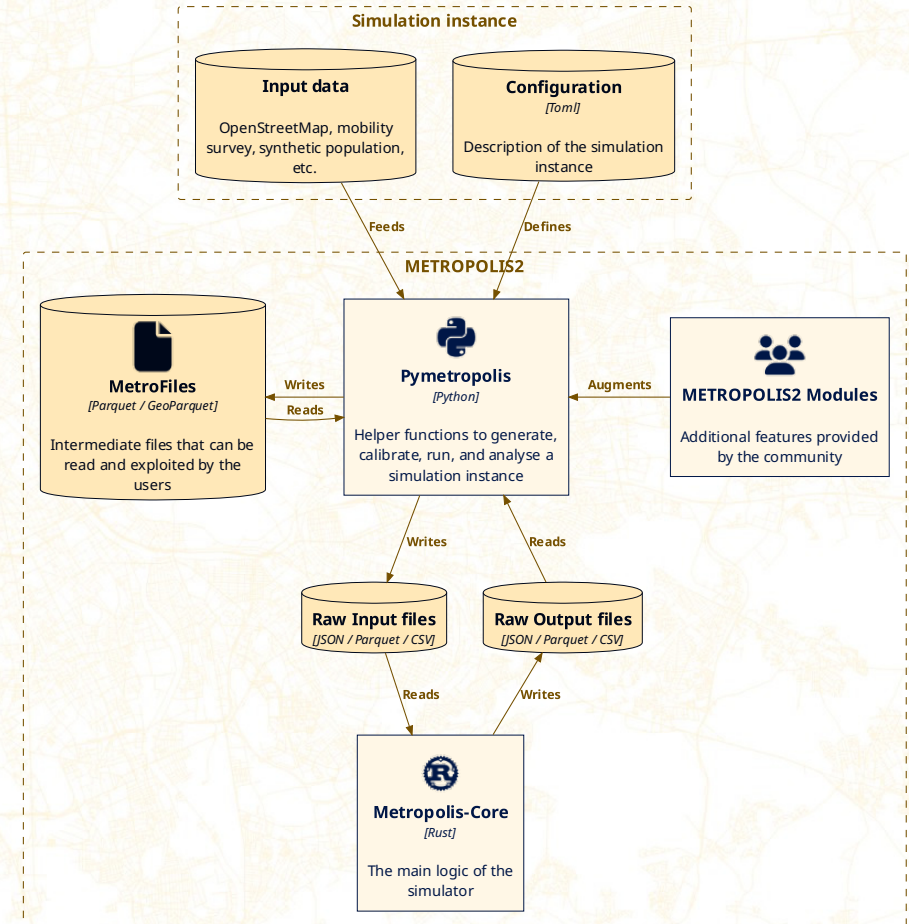
- Development started in 2021 (PhD thesis of Lucas Javaudin, supervised by André de Palma)
- Successor to METROPOLIS(1)
- Objective: evaluation of public policies
- Open source: github.com/Metropolis2

Main features:

- Multi-agent
- Mesoscopic
- Dynamic
- Discrete-choice modeling
- Bottleneck congestion
- Iterative convergence process

METROPOLIS2 Architecture

- **Metropolis-Core** [Rust]: the simulator itself; designed to be flexible
- **Pymetropolis** [Python]: pipeline to generate, run, and analyze a simulation; designed to be user-friendly





Metropolis-Core

Input data

- Parameters (JSON)
- Agents (Parquet)
- Alternatives (Parquet)
- Trips (Parquet)
- Edges (Parquet)
- Vehicle types (Parquet)

```
1 {
2   "departure_time_interval": 60.0,
3   "input_files": {
4     "agents": "input/agents.parquet",
5     "alternatives": "input/alts.parquet",
6     "edges": "input/edges.parquet",
7     "trips": "input/trips.parquet",
8     "vehicle_types": "input/vehicle_types.parquet"
9   },
10  "learning_model": {
11    "type": "Exponential",
12    "value": 0.01
13  },
14  "max_iterations": 100,
15  "nb_threads": 0,
16  "output_directory": "output",
17  "period": [
18    10800.0,
19    97200.0
20  ],
21  "road_network": {
22    "algorithm_type": "Best",
23    "max_pending_duration": 60.0,
24    "recording_interval": 900.0,
25    "spillback": true
26  },
27  "saving_format": "Parquet"
28 }
```

Trips				
Column	Data type	Conditions	Constraints	Description
agent_id	Integer	Mandatory	Value must exist in the agents file	Identifier of the agent
alt_id	Integer	Mandatory	Value must exist in the alternatives file	Identifier of the alternative
trip_id	Integer	Mandatory	No duplicate value over alternatives, no negative value	Identifier of the alternative's trip (used in the output)
class.type	String	Mandatory	Possible values: "Road", "Virtual"	Type of the trip (See Trip types)
class.origin	Integer	Mandatory if class.type is "Road", ignored otherwise	Value must match the id of a node in the road network	Origin node of the trip.
class.destination	Integer	Mandatory if class.type is "Road", ignored otherwise	Value must match the id of a node in the road network	Destination node of the trip.

Output data

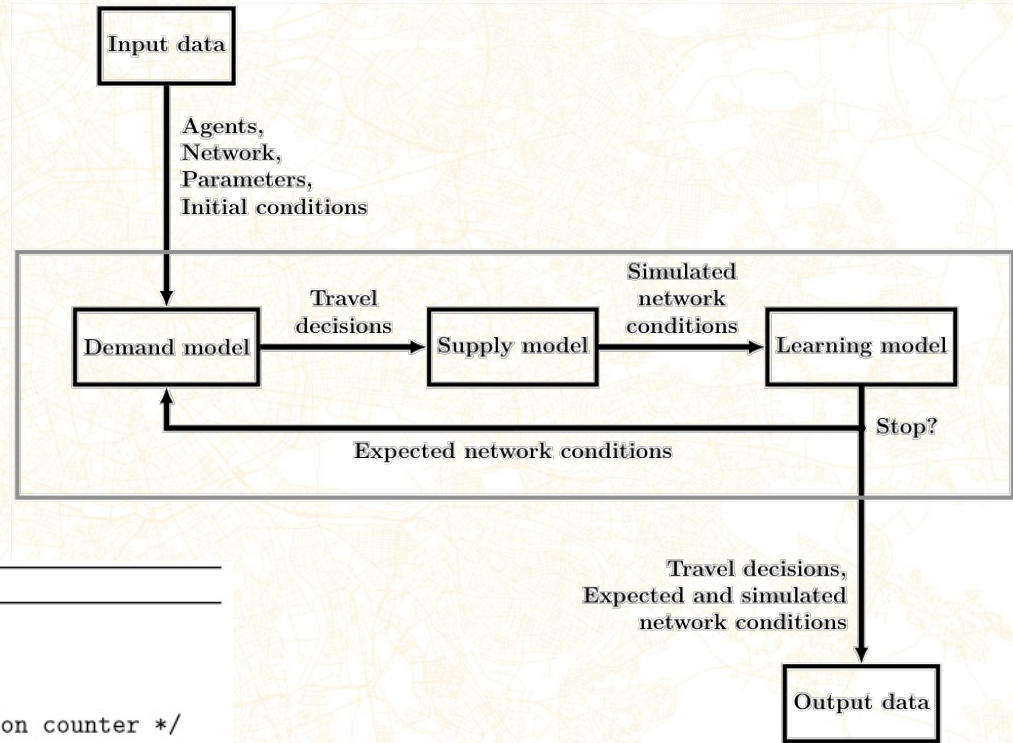
- Iteration results (Parquet)
- Agent results (Parquet)
- Trip results (Parquet)
- Route results (Parquet)
- Edge travel times (Parquet)
- Log (txt)
- Running times (JSON)

Trip results

Column	Data type	Nullable	Description
agent_id	Integer	No	Identifier of the agent (given in the input files).
trip_id	Integer	No	Identifier of the trip (given in the input files).
trip_index	Integer	No	Index of the trip in the selected alternative's trip list (starting at 0).
departure_time	Float	No	Departure time of the agent from origin for this trip, in number of seconds after midnight. For the first trip, this is the departure time after the <code>origin_delay</code> .
arrival_time	Float	No	Arrival time of the agent at destination for this trip (before the trip's <code>stopping_time</code>), in number of seconds after midnight.
travel_utility	Float	No	Simulated travel utility of the agent for this trip.
schedule_utility	Float	No	Simulated schedule utility of the agent for this trip.
departure_time_shift	Float	Yes	By how much the departure time from origin for this trip shifted from the penultimate iteration to the last iteration. If there is no previous iteration, the value is <code>null</code> .
road_time	Float	Yes	For road trips, the time spent on the road, excluding the time spent in bottleneck queues, in seconds. For virtual trips, the value is <code>null</code> .
in_bottleneck_time	Float	Yes	For road trips, the time spent waiting in a queue at the entry bottleneck of an edge, in seconds. For virtual trips, the value is <code>null</code> .
out_bottleneck_time	Float	Yes	For road trips, the time spent waiting in a queue at the exit bottleneck of an edge, in seconds. For virtual trips, the value is <code>null</code> .
route_free_flow_travel_time	Float	Yes	For road trips, the travel time of the <i>selected route</i> under free-flow conditions, in seconds. For virtual trips, the value is <code>null</code> .
global_free_flow_travel_time	Float	Yes	For road trips, the travel time of the <i>fastest route</i> under free-flow conditions, in seconds. For virtual trips, the value is <code>null</code> .
length	Float	Yes	For road trips, the length of the route selected, in meters. For virtual trips, the value is <code>null</code> .
length_diff	Float	Yes	For road trips, the total length of the edges taken during the last iteration and that were not taken during the previous iteration, in meters. For virtual trips, the value is <code>null</code> .
nb_edges	Integer	Yes	For road trips, the number of edges on the selected route. For virtual trips, the value is <code>null</code> .

Convergence process

An **iterative process** is used to adjust travel decisions until expected congestion matches simulated congestion (Nash equilibrium)



Algorithm 3: Pseudo-code of the METROPOLIS2 algorithm

Input: $D, \hat{T}_1$

```
1  $\kappa \leftarrow 0$ ;
2 repeat
3    $\kappa \leftarrow \kappa + 1$ ;           /* Update the iteration counter */
4    $\mathbf{z}_\kappa = f^{\text{demand}}(\hat{T}_\kappa; D)$ ;      /* Run the demand model */
5    $\mathbf{T}_\kappa = f^{\text{supply}}(\mathbf{z}_\kappa; D)$ ;      /* Run the supply model */
6    $\hat{T}_{\kappa+1} = f^{\text{learning}}(\kappa, \mathbf{T}_\kappa, \hat{T}_\kappa; D)$ ; /* Run the learning model */
7 until One of the stopping criteria is met;
```

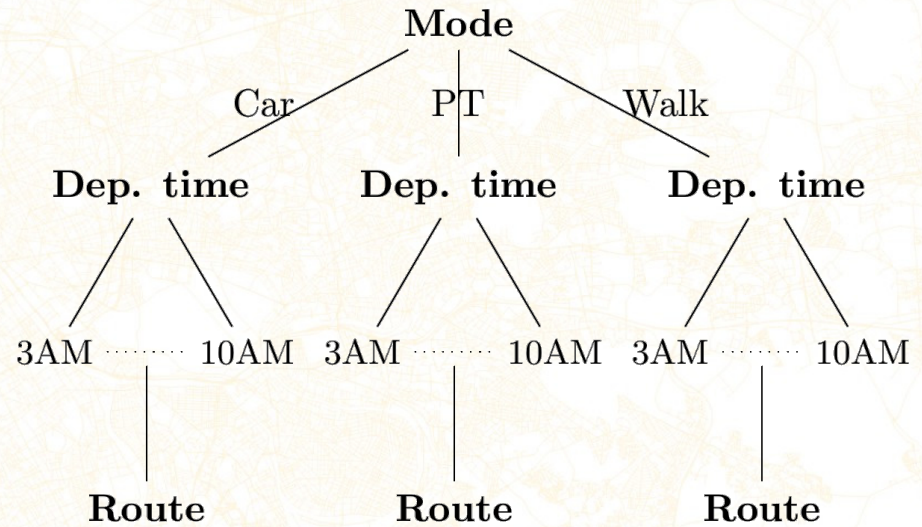
Output: $\mathbf{z}_\kappa, \mathbf{T}_\kappa, \hat{T}_\kappa$

Demand: Trips and trip chains

- Trip types:
 - **Road** (e.g., car driver, car passenger, truck)
 - **Constant travel time** (e.g., walk, bicycle, simplified public transit)
 - **Public transit** [WIP]
- **Trip chaining:** each agent is following a sequence of trips
- **Home-based tours:** mode choice is done at the home-based tour level

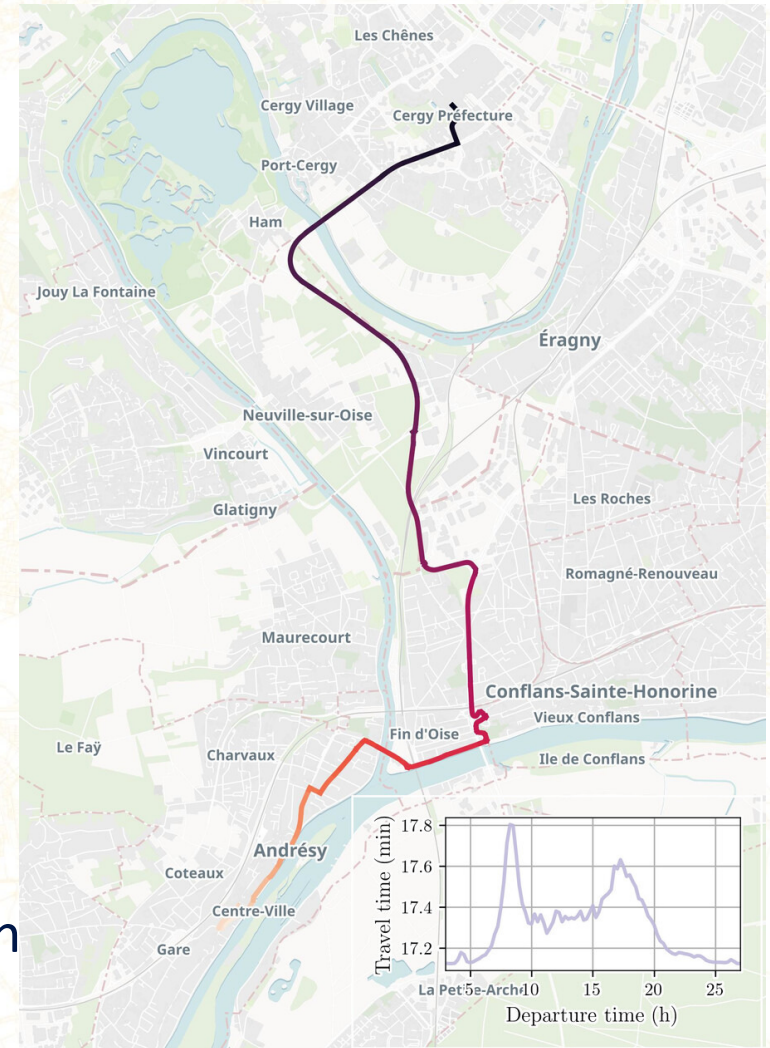
Demand: Decision process

- Level 1: Agents choose a **mode** (Multinomial Logit, Nested Logit, deterministic model, etc.)
- Level 2: Agents choose a **departure time** for their first trip (Continuous Logit, Multinomial Logit, exogenous)
- Level 3: Agents choose their **route**



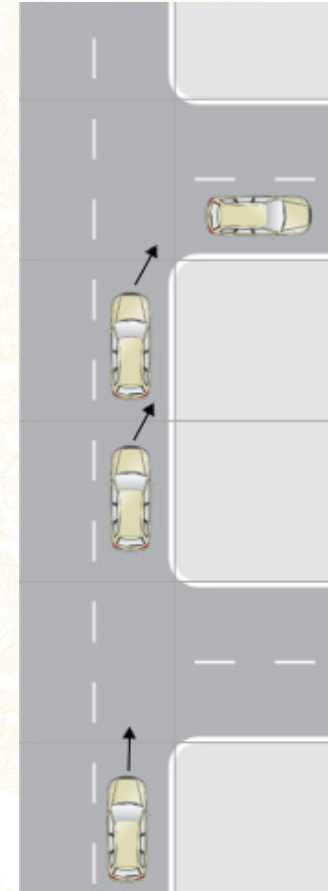
Demand: Routing

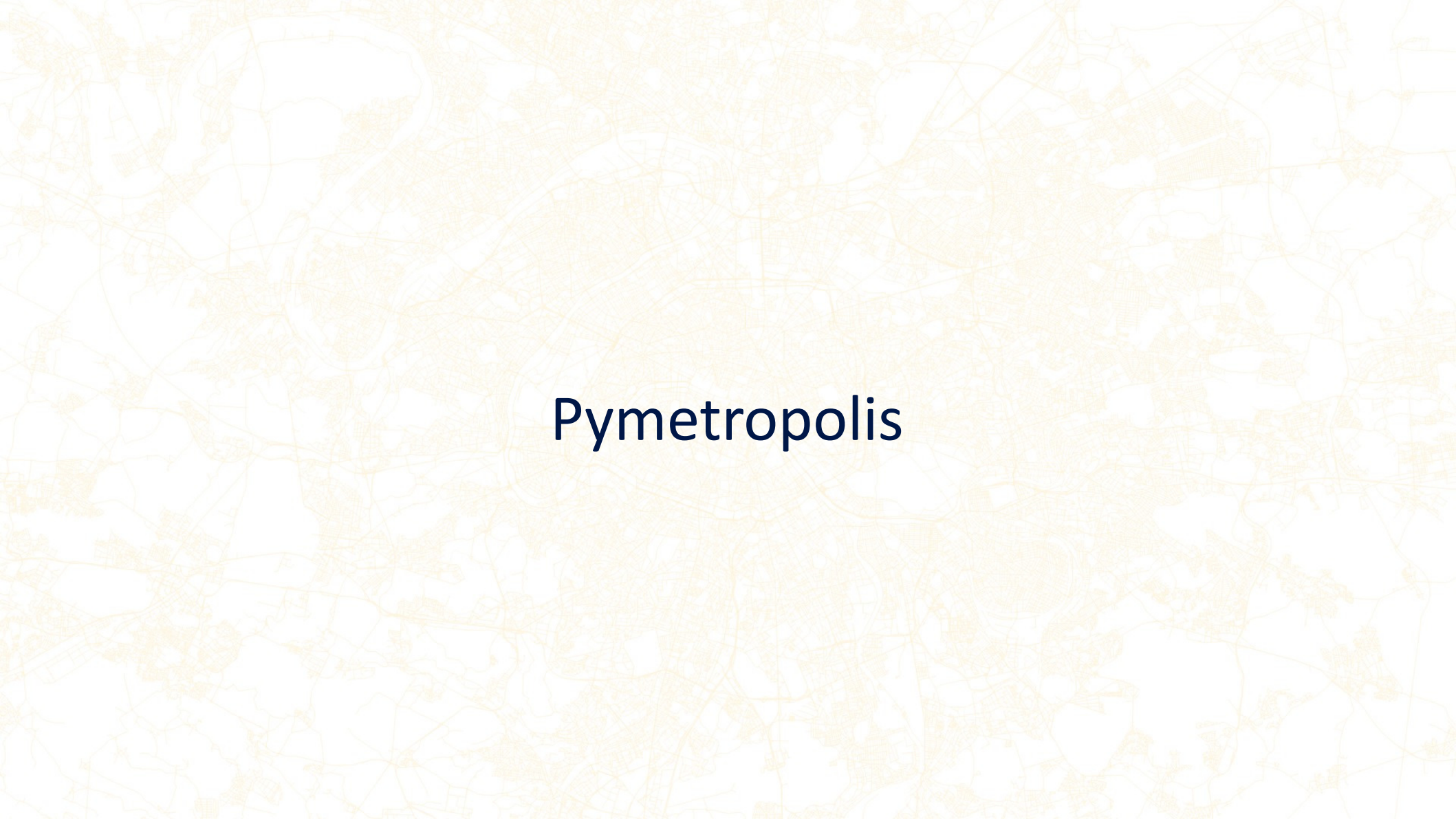
- **Routing algorithms** are used for departure-time and route choice: **(time-dependent) Contraction Hierarchies**
- Two query types:
 - Earliest arrival query: fastest travel time and route, given an origin, destination, and departure time
 - Profile query: fastest travel time for any departure time, given an origin and destination
- **OpenTripPlanner** is used for public-transit routing



Supply: Congestion modeling

- **Event-based model** (continuous time)
- **Bottleneck:** at the link level, vehicle inflow / outflow is limited by the capacity (dynamically)
- **Queue propagation (spillback):** vehicle inflow is limited by the stock capacity of each link (# vehicles to occupy full link length)





Pymetropolis

Pipeline Pymetropolis

1. Collect data
2. Write a configuration file [TOML]
3. Run Pymetropolis
 - Based on the config, various processing Steps are run
 - The running order of Steps is computed dynamically
 - When the config changes, only impacted Steps are re-run
 - Processing data / results are stored in well-formated MetroFiles

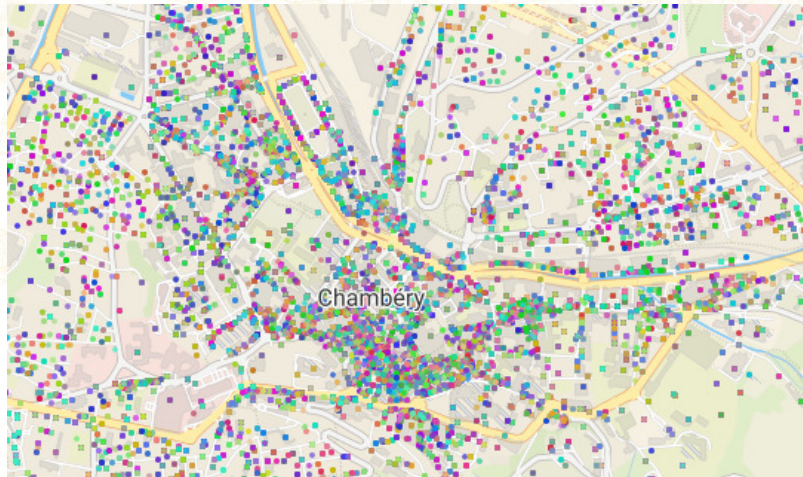
Input data: Synthetic populations

- **Fictitious population** of households and persons, generated to be representative of the actual population (Hörl & Balac, 2021)
- **Data combination:** census, household travel survey, buildings
- **Example** of household:
 - ID: 19
 - Number of cars: 1
 - Number of bicycles: 3
 - Income: €11,865

Input data: Synthetic populations

Persons

ID	Age	Gender	Socioprofessional class
34	51	Man	Employee
35	48	Woman	Executive
36	17	Man	Student
37	14	Woman	Student

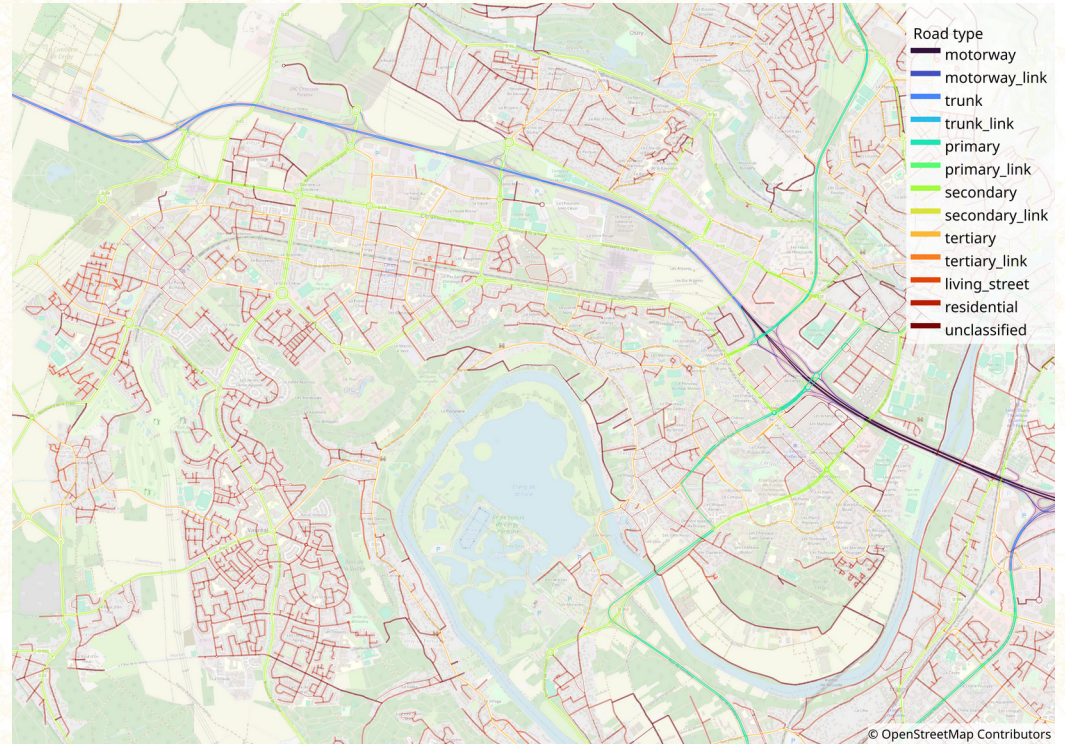


Trips

ID	Person	Preceding purpose	Following purpose	Activity duration
133	34	Home	Work	06:35:00
134	34	Work	Home	
135	35	Home	Work	08:03:00
136	35	Work	Home	
137	36	Home	Education	06:55:00
138	36	Education	Home	
139	36	Home	Leisure	01:25:00
140	36	Leisure	Home	
141	37	Home	Education	07:10:00
142	37	Education	Home	

Input data: OpenStreetMap & GTFS

- Road network is imported from **OpenStreetMap**
- Public-transit schedule is read from **GTFS files**



Pymetropolis case studies

1. Single-road bottleneck (~1 minute)
2. Circular city (~15 minutes)
3. Small-size French city (~2+2 hours)
4. Paris (~4+2 days with 20% downscaling)



Extensions and applications

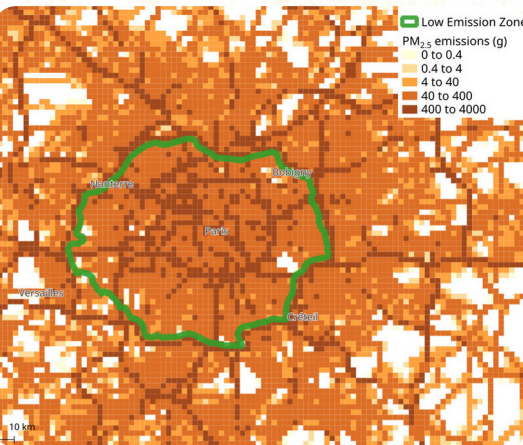
Calibration

Semi-automatic calibration of simulation parameters with *Machine Learning* models

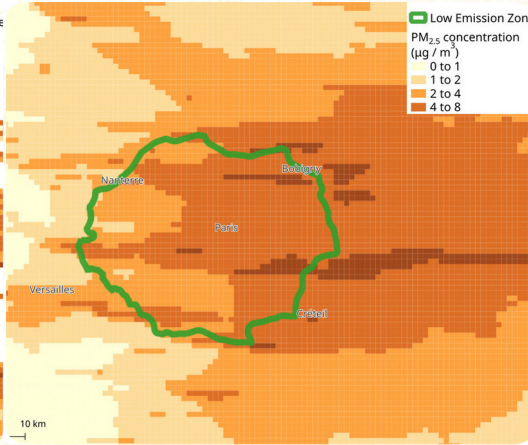
Step	Calibrated parameters	Target values	Data	Methodology
1	Free-flow speed and time penalty by link	Free-flow travel time (OD level)	TomTom API	LASSO regression
2	Capacity by link type	Congested travel time (OD level)	TomTom API	OLS regression
3	Schedule-delay penalties by purpose	Departure-time distribution by cluster	Household travel survey	Bayesian opti. with gaussian processes
4	Alternative-specific constant and value of time by mode x gender	Mode shares by cluster	Household travel survey	Bayesian opti. with random forest regression

METRO-TRACE

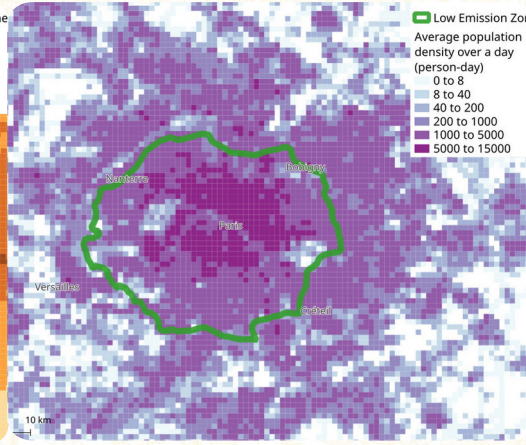
1. Emissions



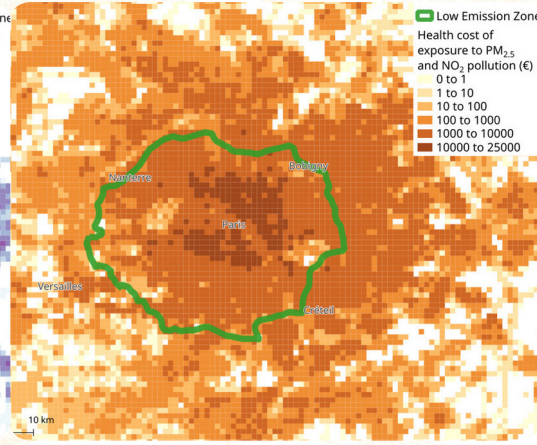
2. Dispersion



3. Population density



4. Exposure



Scenarios

- Paris (next slide)
- Quebec city (new bridge, new tramway line)
- Riyadh (traffic emissions)
- North of France [WIP] (public transit expansion)
- East of France [WIP] (public transit expansion)
- Alger [WIP]

Policy evaluation in Paris

- Low emission zone
- Subway expansion
- Ridesharing
- Speed-limit change
- High-occupancy vehicle (HOV) lanes

Comments?

Lucas Javaudin

lucas.javaudin@enpc.fr

<https://lucasjavaudin.com>

<https://metropolis2.org>