Technical Brief : Boater Traffic Simulator (BoTS).

The purpose of this document is to provide a summary of the state of the BoTS project at the time I am ending my postdoc on December 1, 2024. It provides, alongside the code and documentation, the necessary information for someone to resume the project where I left it. Importantly, this document is not meant to be a research paper and is thus not written with the corresponding standards in mind.

I have initiated and worked on this project with Clément and James for about a year. This document is complementary to the code and documentation[1], raw datasets[2] and reproducibility models[3].

1 Overview of the project

1.1 High-level objective

We aim to learn a **fine-grained** and **controllable** boater traffic simulator from the AIS data and Camille Kowalski's beluga-boater interaction model.

- **Fined-grained:** we want to generate traffic at the individual agent/trajectory level (individual (x, y, t) sequences for each trip). This will enable us to compute a precise *"noise budget"* of *"which boats impact which belugas, where and when"* (once coupled with a spatio-temporal explicit agent based model of belugas, and the noise generation and propagation modules).
- **Controllable:** we want to use BoTS to investigate counterfactuals (*"what if"* questions) and asses the effect of regulation policies. This requires to be able to condition the model on contexts of interest (weather, speed limits, destination, etc.), generate coherently the remaining of the context (trip duration for instance) and generate coherent trajectories for that complete context. It also requires that a trajectory takes into account observables entities (coasts, speed limitation areas, belugas, etc.).

1.2 Overview of the data

Currently, we have two datasets:

- `ptsAISB20182019_crs32187.csv`: 312 AIS ids, 2451 trips, 2.4M points between May 2021 to October 2022.
- `ptsAISB20212022_crs32187_v3.csv`: 186 AIS ids, 1331 trips, 680k points between May 2018 to October 2019.

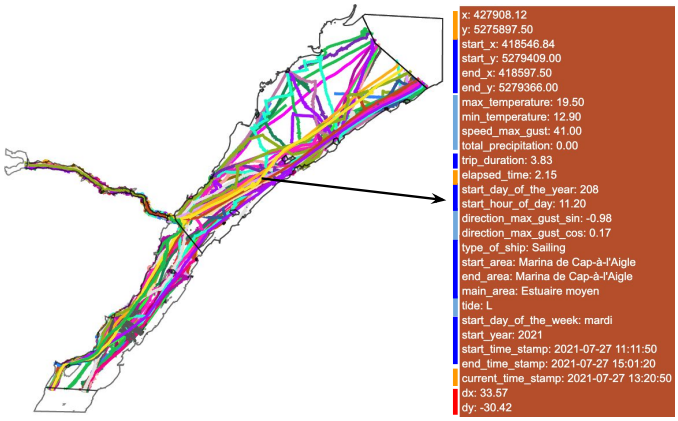


Figure 1: AIS data illustration.

An illustration of the validation dataset produced after processing `ptsAISB20182019_crs32187.csv` (i.e., 15% of the trips) is shown in Figure 1. As illustrated, each data point contains information related to:

- **navigation plan** (departure location and time, arrival location and time, boat characteristics, etc.)
- **weather** (temperature, wind, rain, tide, etc.)
- **current state** (location and time)
- **future displacement** (vector drom current location to next location).

For simplicity we will regroup **navigation plan** and **weather** into what we call **context**.

You can find and download an interactive `.html` file of this plot[4].

[1] `https://github.com/PBarde/boater_project.git`
[2] `https://huggingface.co/datasets/boater-lisse/boater_raw_data/tree/main`
[3] `https://huggingface.co/boater-lisse/reproductibility_models/tree/main`
[4] `https://drive.google.com/file/d/1UCOGEPufW9d4DwFDF9AOQ-5Wh6KhE3eA/view?usp=drive_link`

1.3 High-level methodology

The BoTS project is part of the 3MTSim overarching project (see Figure 2 (a) for illustration). Specifically, BoTS focuses on integrating the simulation of recreational boaters and their interactions with marine-mammals in 3MTSim. To do this, we propose a hierarchical two-level boater model. The trajectory level model (TLM) is a Machine Learning AIS trajectory generator, it allows capturing the complexity of boaters behaviors and efficiently producing high-resolution trajectories directly from the AIS dataset. Yet, it is not very controllable in the sense that we do not have a lot of control over the specific boater behavior (i.e., to test different reactions to regulations, whales, etc.) nor does TLM detect or react to other 3MTSim agents (such as whales); indeed, it is solely derived from recorded trajectories for which we do not have information about the corresponding whale activity. Thus, TLM will be complemented with an interaction level model (ILM) which is triggered every time we detect a co-occurrence between a boater agent and a whale agent in 3MTSim (similar locations at similar time) Note that ILM could also be triggered if the boater enters a navigation restricted zone to test different speed/acceleration reactions. ILM is a ruled-based model that follows Camille Kowalski's interaction patterns (follow whale, turn-off engine, etc.). Once the interaction phase is terminated, we transition back from ILM to TLM and TLM recomputes an adapted trajectory to meet the navigation plan from the perturbed state (due to the interaction with the whale). The two-level model is illustrated in Figure 2 (b).

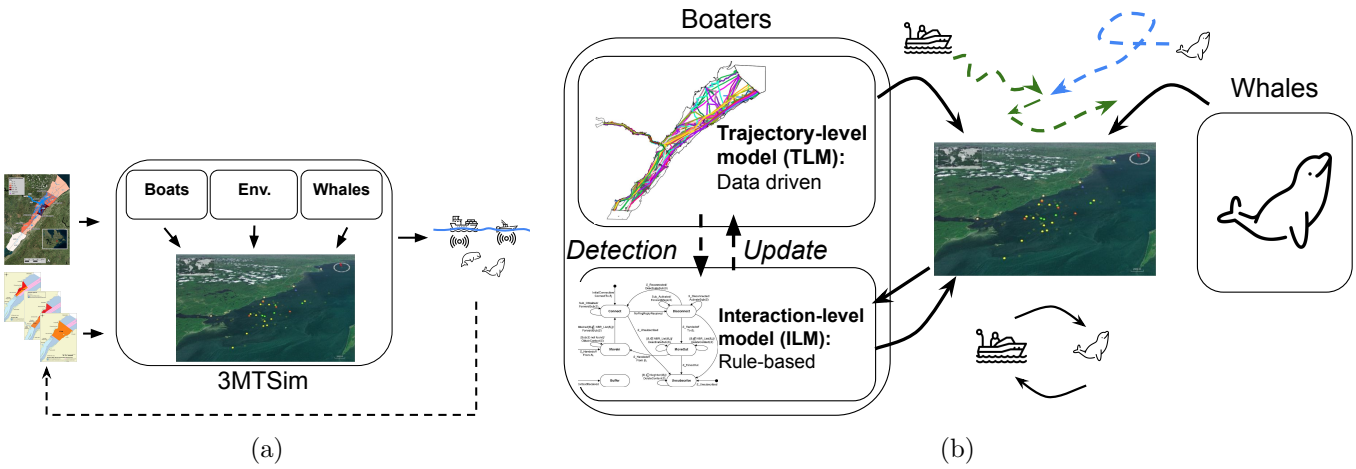


Figure 2: (a) Illustration of the simplified 3MTSim workflow. It takes as inputs some conditions (whale and ship specifications, navigation constraints, etc.) and simulates boat and whale agents in the environment to estimate the cumulative noise received by the whales. (b) BoTS two-level Boaters model. The high-level data-driven model (TLM) generates boaters trajectories. If we detect a co-ocurrence between a boater trajectory and a whale agent, we transition to the low-level rule based whale-boater interaction model (ILM). Once the interaction phase is completed, we transition back to the high-level model and update the trajectory to meet the requirements of the **navigation plan**.

1.4 Project limitations

- The trajectory-level model (TLM) is solely based on AIS data, so we do not model nor account for boats that are not equipped with AIS, such as small fishing boats.
- Since that boats were recorded likely encountered whales, AIS trajectories already incorporate whale-boat interactions. Yet, we ignore this interaction signal in the AIS data since we cannot tie it to any recorded whale activity for the moment.

1.5 Current state of the project

Currently, we have a TLM model that is able to generate trajectories from a given **context**. Some trajectories collide with the coasts, so we have to filter out approximately 7% of the trajectories generated on **context** that were never seen during training. For known **context** this proportion is close to 1%. The generated trajectories produce a traffic density that matches the ground-truth traffic densities. We will give more details on the model performance in Section 2.6.

1.6 Next steps

1.6.1 For TLM

- Check with collaborators that the generated trajectories are satisfactory (check densities, speed profiles, and ask what other metrics would be of interest).
- Make TLM robust to state perturbation and trajectory replanning (when transitioning back from ILM). To do this, I think we can simply modify TLM training such that any point on a trajectory can serve as initial **context**.
- Create a **context** generator that can produce a **context** from scratch or from a partially defined **context**. We can look into ideas from language modelling for that, for instance using a small transformer architecture to fill in the blanks in masked contexts.

1.6.2 For ILM

- See with Camille if her theoretical interaction model is ready and implement it as rule-based model inside 3MTSim.

1.6.3 Integration in 3MTSim

- I believe what would be best would be a client-server interface between a TLM server API that sends boater trajectories (**context**, list of **current state**) to the 3MTSim client. 3MTsim detects agents co-ocurrences and runs ILM. Once interactions are done, 3MTSim queries the TML API with the current perturbed (**context**, **current state**) to get an updated trajectory it can play. This will require some synchronous execution of 3MTSim code when querying TLM for trajectories (i.e., the time loop in 3MTSim will have to wait for the updated trajectories).

2 Methodology details

2.1 Data

2.1.1 Data processing

You will find all the details and processing scripts under `boater_data/ais_data/data_processing/`. In brief we do (please refer to the code for more details):

- We project the positions from `EPSG:32187` to `EPSG:32619` (I do not really remember the rationale behind this choice and it could be revisited but I believe it is because the second projection is more common and better supported in the different libraries).
- We relabel the entries for clarity and consistency.
- We remove duplicated points.
- We clean and remove outlier points with a variety of processing:
  - For consecutive points in a trajectory that are very close to each other in location (<2 m), we keep the oldest point.
  - for consecutive points in a trajectory that are very close to each other in time (<5 s), we keep the point that is better aligned with the previous and next points. For the points are at the start or end of a trajectory, we keep the point which location yields the smaller velocity. If we have multiple consecutive segments with the same time stamp, we iteratively remove the first point of the segments that yield a too big velocity (> 26 m/s, 50 knots, 100 km/h). After this, we iteratively remove the first point of segments such that there are no consecutive points with less than 5s between them. Finally, we drop the points with too big a velocity (> 26 m/s).
- We linearly interpolate the trajectories to get a point every 2 minutes. We remove the trips which interpolation leads to collision with the coasts (we could avoid this by running a path planning algorithm to avoid collisions during the interpolation).
- We preprocess the variables to get either discrete ids (like start and end area names) or continuous variable (we use a sine and cosine to express the day of the year, for instance). We also normalize the quantities either with (mean, std) or max values.

2.1.2 Additional inputs

Note that we also compute some additional inputs to the model from the data:

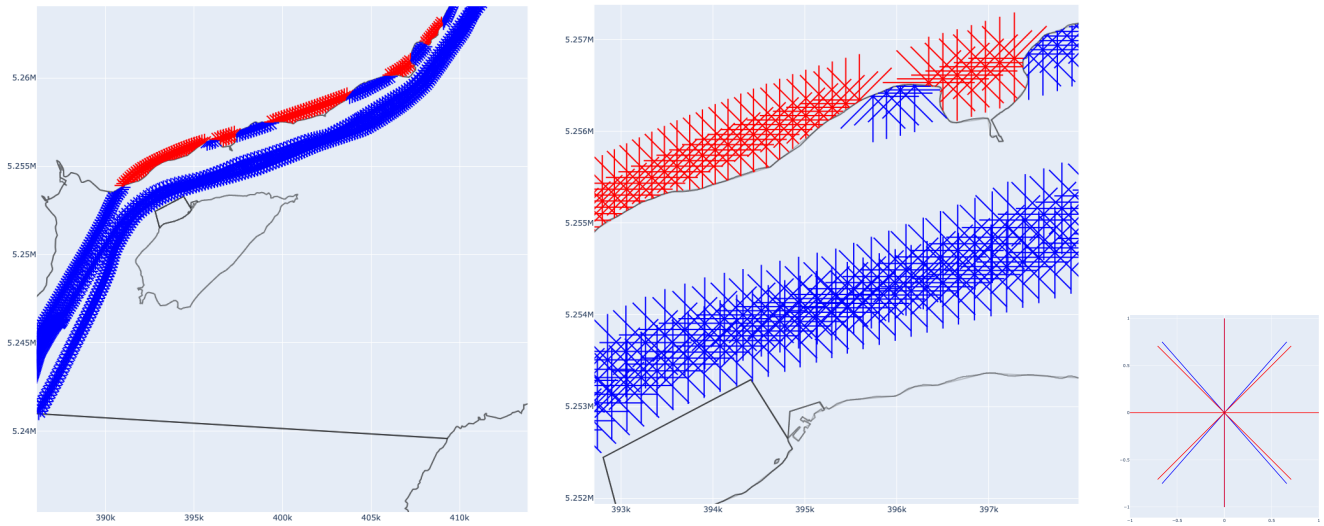
- We compute the **future displacement** and normalize it, we also create a **done** flag that indicates if a point is the last of the trajectory.
- We compute the vector from the current position to the end position in normalized position space (we call it delta to end position). We also compute the time remaining for each point using the trip duration and the elapsed time.
- We compute the **lidar distance** for each point (see description below).

Lidar distances: for each point in a trajectory we cast lidar rays at angles (0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°). For each ray, if it intersects with a coastline at a distance below 500 m we report this distance, otherwise we report 500 m. Sections of trajectories with lidar distances displayed are shown in Figure 3.

Now, because inputs to the model are in normalized position space, we have to normalize the lidar distances. Note that since the normalization is different between the x and y coordinates (the mean and std values are different) the resulting normalized rays are at different angles in normalized space than in the reference space. This is illustrated in Figure 3 (c).

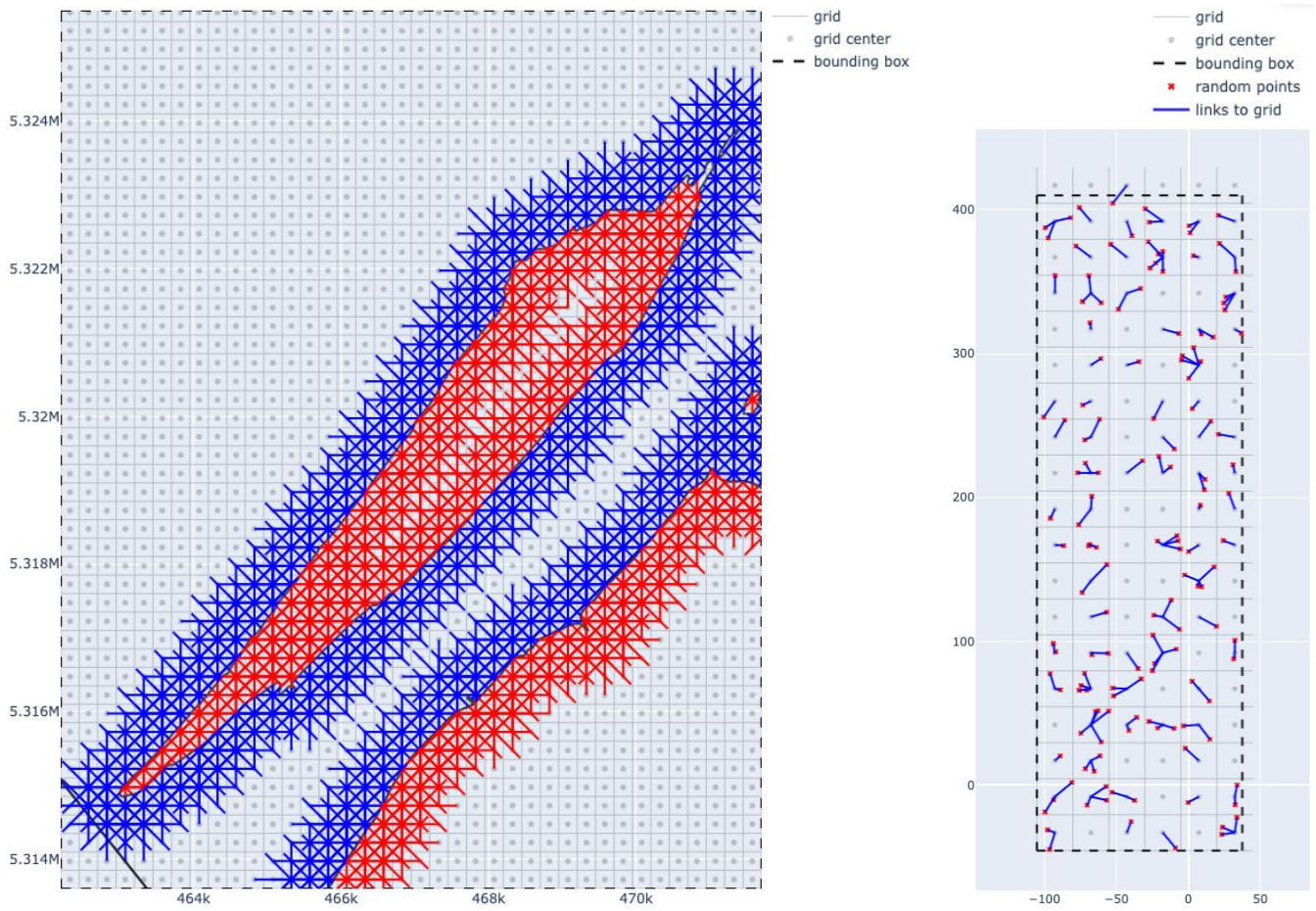
Finally, computing ray-coast intersections on the fly during training is compute-intensive and slows down training prohibitively. To palliate to this, we precompute the lidar rays distance on a grid covering the whole domain and then use nearest neighbor to estimate the lidar distances for a point on a grid. This is illustrated in Figure 4.

You will find all the details under `boater_data/ais_data/lidar/` and `boater_data/ais_data/gridmaps/distance_maps/`



(a) (b) (c)

Figure 3: Trajectories with lidar distances displayed. Distances are blue if the point is in the water and red if the point is on the land (collision). (a) Trajectories where one of them has been shifted to collide with the coastline. (b) Zoom near the coastline, note that the lidar rays capture the distance from the boat to the coastline across different angles. (c) Lidar angles are distorted in the normalized space because the scaling for the x and y coordinates differ (due to different std). Red shows the rays in reference space and blue in normalized space



(a) (b)

Figure 4: Cached lidar distances on a grid. The grid resolution is coarser than in the actual trainings to avoid visual cluttering. (a) A zoomed portion of the domain for which lidar distance are calculated. We do not compute distance for a point further than 500 m away from the coast (we will use a default value of 500 m anyway). Distances are negative and displayed in red if inside the coast. (b) Illustration of nearest neighbor on the grid, points take the value of the center point of the grid square they are in.

2.1.3 Notations

- **context** $\triangleq c$
- **current state** $\triangleq s$
- **future displacement** $\triangleq \Delta$
- **done** $\triangleq d$
- $(\cdot)_t \triangleq$ quantity $(\cdot)$ at timestep t
- $(\cdot)_{\{:t\}} \triangleq$ sequence of $(\cdot)_t$ from timestep 0 to timestep t
- $\tilde{(\cdot)}$ denotes an approximation of quantity $(\cdot)$ as produced by our model.

2.1.4 Dataset splits

We split into training/validation/test datasets while trying to maintain 70%/15%/15% ratios in terms of number of points, trips, and AIS ids.

2.1.5 Data artifacts

As discussed with Jean-François Senécal there are some artifacts in the data (probably introduced by the extraction scripts on the rawest data previous to the `.csv` files). We have for instance duplicated points, point with exact same positions but different timestamps, or point with same time stamps but different positions (infinite velocity). Also, points are removed from marinas even if the boat just passes through it without ending or starting the trip.

2.2 The model

The model is quite simple, it takes as inputs two kind of values:

- **continuous inputs**: 'positions', 'trip_weather_floats', 'start_time_floats', 'trip_duration', 'elapsed_time', 'start_position', 'end_position', 'lidar_distances', 'delta_to_end_position', 'time_remaining'
- **discrete (or categorical) inputs**: 'type_of_ship', 'start_area', 'end_area', 'main_area', 'tide', 'start_day_of_the_week'

The model is illustrated in Figure 5. The discrete inputs are passed into Embedding layers to be converted into float representations, then they are concatenated with the continuous inputs and passed to a Linear layer that is fed to the Torso module. The Torso is either:

- a sequence of Linear + ReLu + Dropout layers, in which case we call the model *"plain"* and denote it MLP.
- a sequence of GRU layers[5] (also with dropout) in which case we call it *"GRU"* and denote it GRU.

You will find all the details in `boater_algo/actors.py`. The outputs of the Torso are then processed by two independent heads (comprised of Linear, ReLu, and potentially Sigmoid layers) to produce the model outputs that are the **future displacement** and the **done** flag.

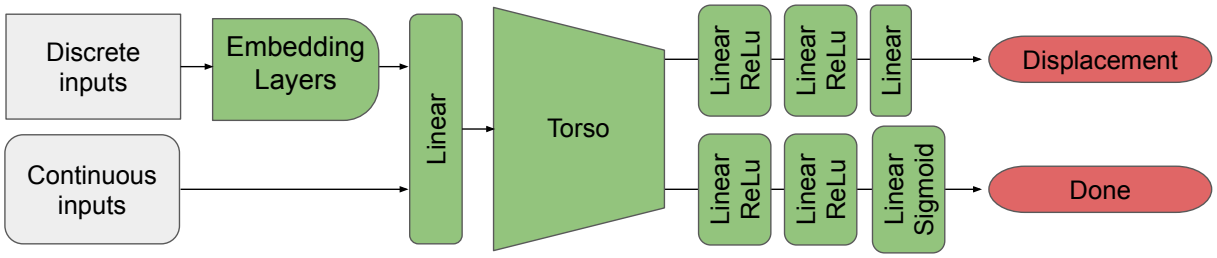


Figure 5: Model architecture. Inputs are in gray, trainable parameters in green and targets in red.

The plain model looks only at the **context** and the **current state** to produce the current **future displacement** and current **done**, or to put it mathematically:

$$\tilde{\Delta}_t, \tilde{d}_t = MLP(c_t, s_t) = MLP(c, s_t)$$

On the other hand, the GRU model looks at the whole past trajectory:

$$\tilde{\Delta}_t, \tilde{d}_t = MLP(c_{\{:t\}}, s_{\{:t\}})$$

Note that $c_t = c \ \forall t$, meaning that the **context** for a given trajectory is the same regardless of the timestep (**navigation plan** and **weather** are trip-wise quantities and do not vary during the trip).

2.3 The losses

To train the model, we use the following losses:

- L2 loss for the **future displacement**: $L_\Delta(\Delta_t, \tilde{\Delta}_t) = ||\Delta_t - \tilde{\Delta}_t||^2$
- Binary cross entropy for the **done** flag : $L_d(d_t, \tilde{d}_t) = -\Big(p_t d_t \log(\tilde{d}_t) + (1 - d_t)\log(1 - \tilde{d}_t)\Big)$, p_t is a weight to trade off recall and precision by adding weight to positive examples, indeed we only have one positive example (**done**) per trajectory so we set p_t to the trajectory length if $d_t = 1$ otherwise we set it to 1. Finally, we use label-smoothing on d_t with coefficient α: $d_t = d_t \times (1 - \alpha) + 0.5 \times \alpha$

Finally, since both losses impact the torso, we scale their magnitude to be the same before each backpropagation, see `boater_algo/actors.py` for more details.

[5] `https://pytorch.org/docs/stable/generated/torch.nn.GRU.html`

2.4 Generating trajectories

Once the model is trained we can use it to generate a trajectory autoregressively: we can get s_0 from c and use our model to get Δ_0, then adding Δ_0 to s_0 we get s_1, and so on and so forth...

$$\tilde{s}_t = \tilde{s}_{t-1} + \tilde{\Delta}_{t-1} = \tilde{s}_{t-1} + MLP(c, \tilde{s}_{t-1})$$

The above equation can be regressed all the way to s_0 which is given by the **context** c. The same can be done for the GRU model.

2.5 Teacher forcing

We see that there is a discrepancy between training and the usage we will make of the model. Indeed, during training the model always takes as inputs the *"true"* values s_t (that is, the values that are present in the dataset). On the other hand, when we use the model to generate trajectories, it has to produce $\tilde{\Delta}_t$ and $\tilde{d}_t$ by looking at the estimated quantity $\tilde{s}_t$ it just produced. This will likely result in compounding errors during generation that have not been addressed during training. Using the ground truth quantity s_{t-1} as input during training, regardless of the value of $\tilde{s}_{t-1}$, is referred to as "teacher forcing".

We investigated and implemented a version (`traj_plain`) where we do not use teacher forcing at every step, but instead we keep $\tilde{s}_{t-1}$ as input with a given probability. This resulted in a much slower training (we have to autoregressively generate trajectories during training while retaining the compute graph across time steps) without significant performance benefits (but this might be due to overfitting, see. Section 2.7).

2.6 Evaluation

The main problem I had during the evaluation was collisions with the coast. I implemented a *"bouncing"* mechanism such that boats will bounce off the coast during trajectory generation (like for elastic collisions) but it ended-up in boats getting stuck in concave coast areas (like bays). So eventually, I decided to just filter out trajectories with collisions based on a threshold (see. `boater_algo/results_vizualization`). This results in rejecting approximately 8% of trajectories for test and validation datasets.

We can then visualize representative trips on each dataset, as well as the traffic density corresponding to the trajectories (true vs. generated). Some of these visualizations are displayed in Figures 6,7 and 8, for more details go to `boater_algo/results_vizualization`.

2.7 Potential next steps to improve the model performance

- train with dropout for `traj_plain` to avoid overfitting (see Section 2.5).
- to speed up `traj_plain`, enable it only after full teacher-forcing training has converged (see Section 2.5).
- to avoid collisions, we could add an adversarial loss (i.e., GAN training) where the discriminator looks at whole trajectories and the generator backpropagates through it. In that case, the generator has to produce the trajectories without teacher forcing and retrain the computational graph.
- collaborators were suggesting using tides at start/end marina locations and times instead of mean overall tide.
- additional metrics could be used to assess the quality of the generated trajectories, for instance we could look at how the per-boat-type-speed-profiles are reproduced.

2.8 Final remarks

You will find a lot of resources and details in the code and the `boater_project/README.md`. There is also a folder `boater-figs`[6] that contain a lot of `.html` images of results and analysis that can be generated with the code. Finally, the trainings reproducing the current results can be found on HuggingFace[7]. Finally, do not hesitate to reach out to me `paul.b.barde@gmail.com`.

[6] `https://drive.google.com/file/d/1geHD9Ypqj8ZwK1X8htKMkpHihpOvG8Rz/view?usp=sharing`
[7] `https://huggingface.co/boater-lisse/reproductibility_models/tree/main`

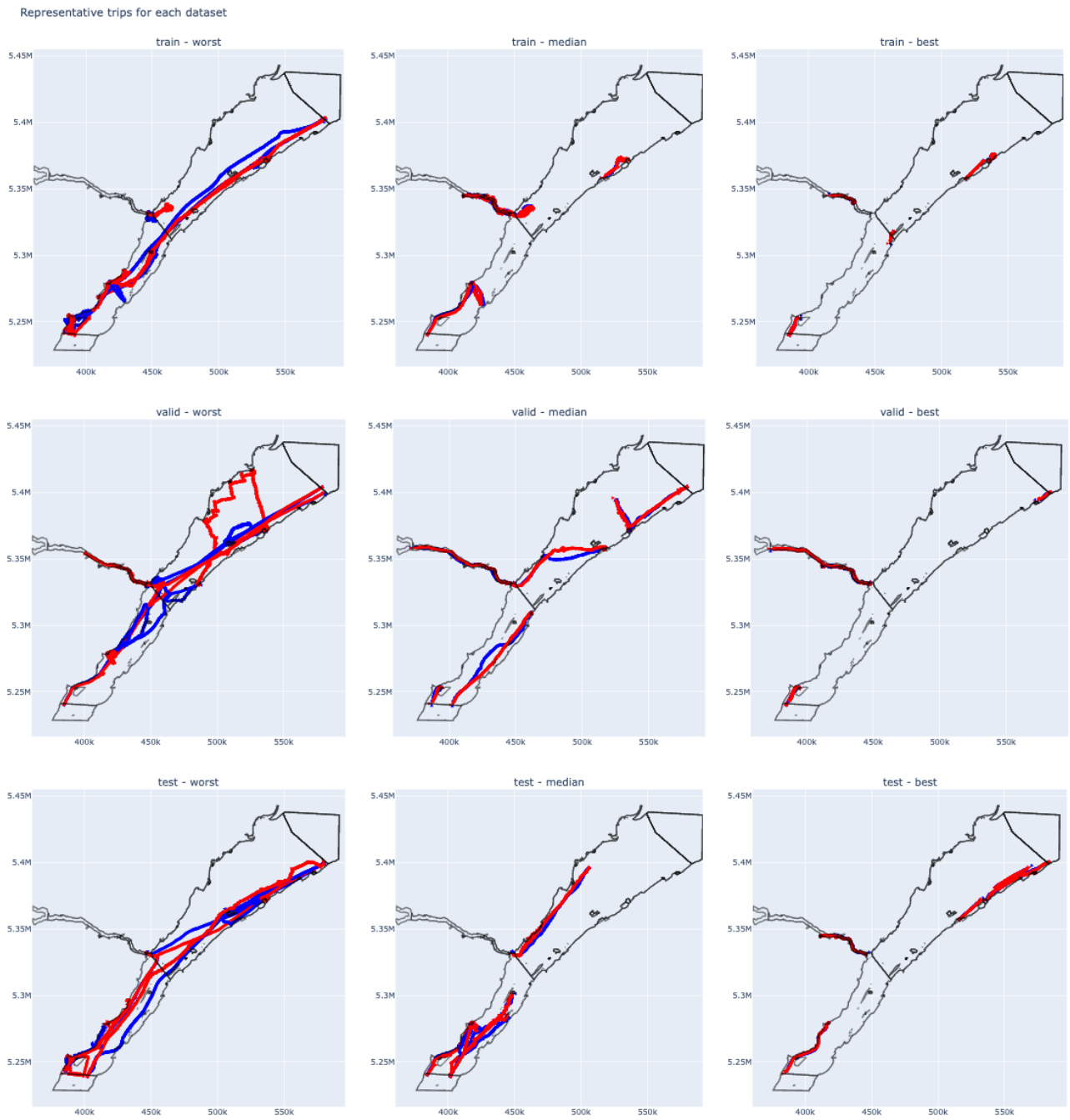


Figure 6: True trips (red) vs. generated ones (blue) for different levels of performance (worst, median, and best) for the different datasets (train, validation, and test).

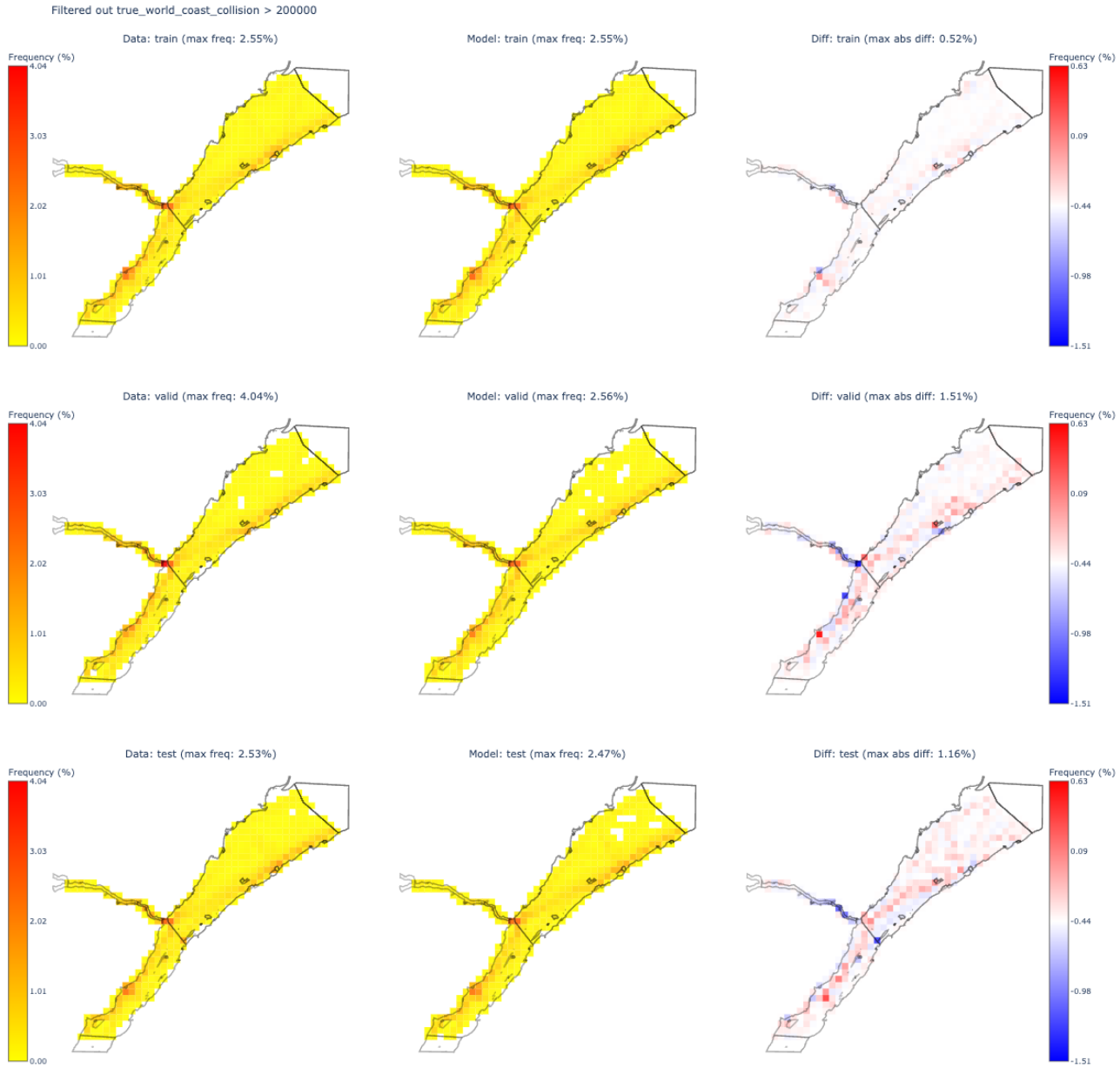


Figure 7: True traffic density (left) vs. generated (middle) vs. differences (right) for the different datasets (rows). We see that our model tend to underestimate traffic in the Saguenay fjord (probably due to filtering out colliding trajectories in this narrow canal).

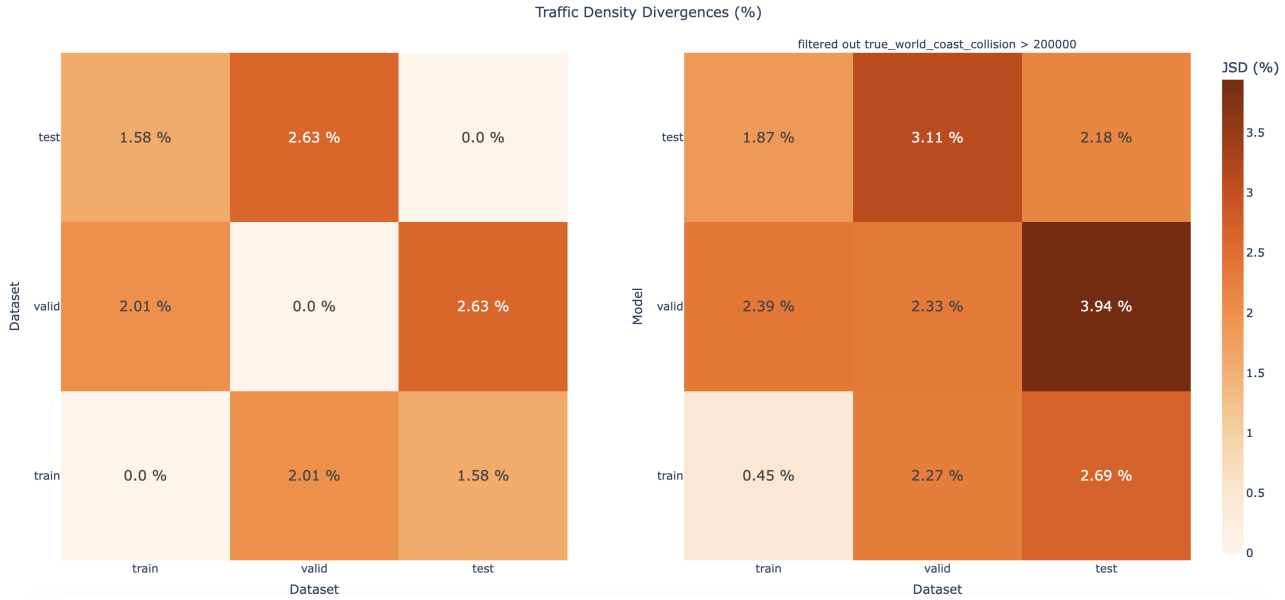


Figure 8: Jensen-Shannon Divergence (JSD) between datasets' traffic densities (left) and JSD between generated and true traffic densities (right). We see that the worst generating divergence (2.33%) is close to the worst divergence between true datasets (2.63%).