GeoInformatica (2026) 30:23
https://doi.org/10.1007/s10707-026-00580-4

RESEARCH

A scalable AIS-based model for vessel-generated underwater noise

Giulia Rovinelli[1] · Esteban Zimányi[2] · Marta Simeoni[1,3] · Davide Rocchesso[4] · Alessandra Raffaetà[1]

Received: 1 November 2025 / Revised: 20 April 2026 / Accepted: 9 June 2026

Abstract
Underwater noise pollution from shipping activities is widely recognised as a significant threat to marine life. Noise emitted by vessels can have various detrimental effects on fish and marine ecosystems. Accurately estimating and analysing vessel-generated underwater noise is therefore of critical importance for the protection and conservation of marine environments. In this paper, we present an enhanced version of our model for the spatiotemporal characterisation of vessel-generated underwater noise, with a focus on improving its scalability. The original model was limited to *fishing* vessels and relied on Automatic Identification System (AIS) data to reconstruct trajectories, as well as engine horsepower to estimate emitted noise. Here, we generalise the approach to include all vessel categories — including tankers, cruise ships, and recreational boats — still relying on AIS data, but estimating noise as a function of vessel length overall (LOA) and category, since horsepower information is not available for all vessels in the dataset. We broaden the study area to include the Central Adriatic Sea, in addition to the Northern part previously considered. The enlarged area and the substantially greater volume of AIS data introduce significant computational challenges, making scalability a primary concern. We address these challenges through a comprehensive analysis of optimisation strategies to improve query execution performance. In particular, we restructure the computational pipeline by implementing table partitioning and leveraging parallelisation techniques. Specifically, we employ PostgreSQL's native parallel query execution and implement multiple partitioning strategies, including range, hash, and list partitioning. We further explore spatial partitioning through space tiling, comparing regular, adaptive, and k-d tree-based grids. Finally, we leverage the Citus extension to distribute computation across four and eight nodes. Our approach improves computational efficiency while preserving the accuracy of noise calculation, offering a scalable solution for large datasets.

Keywords Spatiotemporal databases · Underwater noise · Parallelisation techniques

Extended author information available on the last page of the article

Published online: 21 July 2026

1 Introduction

Marine ecosystems encompass a vast diversity of habitats and species, whose preservation is vital to maintaining global biodiversity and ensuring ocean health. However, increasing anthropogenic pressures — including overexploitation, pollution, and habitat degradation — are compromising their resilience and functionality. Among the various pressures, underwater noise pollution — mainly due to vessels activities — has been shown to cause both short- and long-term effects on marine species. Vessel-generated noise, primarily originating from propeller cavitation, engine and machinery vibrations, and hydrodynamic flow around the hull, can travel great distances underwater, disrupting the natural soundscapes that marine organisms depend on for navigation, communication, feeding, and reproduction. Documented impacts include disruption of acoustic communication, behavioural alterations, increased stranding and mortality rates [1–3]. Unlike chemical or particulate contaminants, acoustic pollution does not accumulate in the marine environment; however, its spatial and temporal persistence is maintained through the continuous activity of vessel fleets. Consequently, the characterisation of underwater noise in space and time is essential for monitoring aquatic ecosystem health, evaluating potential risks, and supplying critical insights to ecologists and policy makers. Such assessments facilitate the formulation of effective management strategies aimed at sustaining productive and resilient marine ecosystems.

Measuring underwater noise is inherently complex and computationally demanding. Deploying hydrophones requires specialised expertise, proper calibration, and significant resources, while subsequent data processing also demands substantial computational capacity, particularly when covering broad spatial or temporal scales. Furthermore, estimating underwater noise in areas where empirical data collection is infeasible, or extending the covered area beyond the monitoring capabilities of existing hydrophones, remains a critical requirement. To address this limitation, many approaches in the literature have employed acoustical models derived from numerical simulations to predict sound pressure levels within the designated regions of interest, see e.g. [4–6]. These models, however, rely on extensive environmental and vessel-specific input data and involve intensive calculations. Hence, developing sound propagation models that achieve a balance between accuracy and computational efficiency is crucial. Such models must provide reliable predictions while minimising resource use, enabling broader and more scalable applications.

In this work, building upon the underwater noise model developed in [7, 8], we focus on exploring different techniques to improve its efficiency and scalability. Specifically, in [7, 8] we proposed an approach that avoids using numerical simulation and fully relies on AIS data to calculate the underwater noise generated by vessels in space and time. The proposed framework is based on semantic trajectories [9, 10]. Starting from AIS data, we reconstruct the vessel trajectories and deploy them in a spatiotemporal database. These trajectories are enhanced with semantic information, such as the characteristics of the vessels and the activities conducted along their paths, which are then used to infer how the noise spreads in the area of interest. As a case study, we considered the fishing vessels of the Northern Adriatic Sea for the year 2020. To determine the acoustic characteristics of the vessels' engines and fine tune the propagation model, we took advantage of the direct acoustic measurements produced by the Interreg project SOUNDSCAPE [11] that carried out an acoustic monitoring in the Northern Adriatic Sea from March 2020 to June 2021. We implemented the

spatiotemporal database using MobilityDB [12] and performed various analyses, including the production of cumulative noise maps of the area and period of interest. Moreover, to better demonstrate the effectiveness of our framework, in [13] we proposed some possible optimisations to the underwater noise calculation, aiming at improving efficiency and scalability of the approach.

In this study, we evaluate our framework using an expanded geographical area and a substantially larger volume of AIS data. Specifically, we extend the analysis to include the Central Adriatic Sea, in addition to the previously considered Northern region, and process AIS data for the entire vessel fleet operating during 2020, including cargo, tanker, recreational vessels, etc. On one hand, the inclusion of all vessel categories required revising the procedure used to estimate the source-level noise. When only fishing vessels were considered, source levels were derived from engine horsepower. Since this information is not available for the other vessel categories, the present study estimates noise levels based on the vessel's category and its length overall (LOA), following the model proposed by MacGillivray and de Jong [14]. On the other hand, the enlarged area and the substantially larger volume of AIS data introduce significant computational challenges, making scalability a primary concern and motivating a detailed evaluation of optimisation strategies to improve query performance.

Specifically, we restructured the underwater noise propagation pipeline with the primary objective of improving its scalability and execution efficiency. The proposed optimisations include code-level enhancements, PostgreSQL's native parallel query execution, and table partitioning using range, hash, and list strategies. Furthermore, we implemented spatial tiling with regular, adaptive, and k-d grids, and extended the framework to a distributed environment by leveraging the Citus extension [15] across four and eight nodes. The experimental evaluation demonstrated that these optimisations substantially reduced execution time while preserving the accuracy of the original underwater noise model. In particular, considering a 63 Hz frequency, for the spring season of 2020 (April–June) the execution time for computing the underwater noise propagation decreased from nearly three days to about three hours.

Summing up, the new contributions of this work w.r.t. the preliminary version presented in Rovinelli et al. [13] are the following:

- We extended the AIS dataset by considering all vessels categories (and not only the fishing vessels) for the year 2020, significantly increasing the data volume and enabling a more comprehensive evaluation of scalability;
- We also extended the basin area by considering the Northern and Central Adriatic Sea (and not only the Northern part), partitioned using an enhanced spatiotemporal grid that provides a more accurate definition of the study area. In particular, the grid was refined to better capture the actual sea boundaries and to exclude land areas;
- We refined the sound propagation model to ensure its applicability across different vessel types. Specifically, we followed the empirical model by MacGillivray and de Jong [14] to determine the source level noise emitted by vessels. We also adapted the model to overcome its limitations at low speeds and to account for vessel activity;
- We applied several optimisation techniques and compared their performance. Although some of them have been already evaluated in [13], in this paper we considered the new dataset and the refined underwater noise model for the spring season and the 63 Hz

frequency. We selected this period because it combines the highest volume of AIS data with the availability of SOUNDSCAPE measurements throughout the entire season. Moreover, we selected a low-frequency band (i.e., 63 Hz), as sound propagates over longer distances at these frequencies, thereby increasing the computational demand for underwater noise calculations. Specifically, we evaluated:

- The original and the optimised pipeline;
- PostgreSQL's native parallel query execution on the optimised pipeline;
- PostgreSQL's Range partitioning on time for increasing number of partitions: 2, 4, 8, 16;
- PostgreSQL's Hash partitioning on MMSI for increasing number of partitions: 2, 4, 8, 16;
- PostgreSQL's Space partitioning using regular and adaptive grids. We also added the use of k-d tree partitioning method for increasing number of partitions: 2, 4, 8, 12;
- Citus extension for distributed computation with four nodes (one coordinator and three workers), as well as with eight nodes (one coordinator and seven workers). We also employed Citus together with PostgreSQL range partitioning on the time dimension and k-d tree spatial tiling.

The paper is organised as follows. Section 2 briefly reviews the literature on underwater noise characterisation and on optimisation techniques. Section 3 describes the data sources used in this study. Section 4 presents our model for the underwater sound propagation and provides a preliminary evaluation against the SOUNDSCAPE measurements. Section 5 outlines the implementation of the overall process in MobilityDB and introduces some code-level optimisations for noise propagation. Section 6 focuses on data partitioning techniques using PostgreSQL and investigates the integration of the Citus extension to enable distributed processing. Finally, Section 7 draws some concluding remarks.

2 Related work

Research on characterisation of underwater noise generated by boats is wide and multifacet. Here, we focus on three main categories of contributions already explored in [8]: (*i*) works reporting direct underwater noise measurements; (*ii*) approaches based on acoustic models obtained by numerical simulation; and (*iii*) proposals based on AIS data and a sound propagation model.

Works in the first category address the issue of underwater noise data acquisition, transmission and storage. The most interesting one in our perspective is the dataset produced by the SOUNDSCAPE project [11], which carried out an acoustic monitoring of the Northern Adriatic Sea from March 2020 to June 2021. Nine monitoring stations were set up that encompass different environmental characteristics. Two datasets have been released, composed of 20 and 60 seconds averaged sound pressure level (SPL) data in a wide range of frequencies collected at the nine stations. SPL is the level of the root-mean-square sound pressure expressed in decibel, relative to a reference value of $1\mu\text{Pa}$ [16]. These datasets are available on Zenodo [17] and the whole process of data acquisition, storing and post-

processing is described by Petrizzo et al. in [18]. Furthermore, Picciulin et al. [19] perform some analyses and describe the spatial and temporal variations of the sound pressure levels recorded by the SOUNDSCAPE project during the monitoring period.

Another approach directly related with our work is the one by MacGillivray and de Jong [14], which proposes the JOMOPANS–ECHO source level model, relating ships' AIS data with their corresponding noise emissions. It calculates the ship source level spectrum, in one-third octave bands, as an explicit function of frequency, vessel speed, vessel length, and AIS ship type. In this study, we adopt the JOMOPANS–ECHO model for determining the source level noise of all vessels in our dataset.

Approaches in the second category are based on acoustic models obtained by numerical simulation to predict the sound levels in the area of interest, see, e.g., the proposals by MacGillivray et al. [4], Larayedh et al. [5] and Ghezzo et al. [6]. These proposals simulate sound propagation considering reflection, diffraction and absorption phenomena and require precise information in input. Such models usually account for many environmental variables (e.g. sea temperature, wind, waves, salinity), precise bathymetry information and sound speed profiles, possibly at different sea depths. They use AIS data to derive the vessels' trajectories and include their emitted sound into the model. The results obtained are often validated through real acoustic data measured on specific sites. A common characteristic of simulation-based approaches is that they require huge computational effort. Optimisations in this context include using high-performance computing and applying techniques such as parallel processing (see, e.g., [20]) or reducing the model dimensionality from 3D to 2D to lower the computational effort and, consequently, the execution time.

The third category of related contributions is about proposals that avoid the use of numerical simulation and give a more simplified description of the sound propagation effects. They are based on AIS data and usually adopt a sound propagation model to calculate the transmission loss in the area of interest. They are computationally efficient, even for large datasets. The main proposals in this category are the approaches by Erbe et al. [21], Neenan et al. [22] and Jallkanen et al. [23]. Their main goal is to estimate the underwater cumulative noise in the area of interest, but no computational optimisations are discussed or proposed.

Our approach fall into the third category, since we rely only on AIS data and on a sound propagation model to characterise the underwater noise of the studied area. However, our proposal is characterised by two distinguishing features: the use of semantic trajectories and their integration into a spatiotemporal database. Semantic trajectories allow linking semantic information to movement data, such as the activities carried out by a vessel during its trip. The spatiotemporal database supports queries about underwater noise across various spatial and temporal granularities, with the possibility to include any selection of vessels. However, while the use of a spatiotemporal database enhances the expressiveness and flexibility of the approach, it also demands exploring optimisation techniques to efficiently perform complex tasks, such as calculating and storing the noise propagation in space and time without loss of precision, which constitutes the main focus of this paper.

In this context, achieving efficient query execution requires the adoption of optimisation strategies such as data partitioning and parallel processing. These techniques are employed in our approach and have been extensively studied in the context of distributed systems and databases. Foundational works on distributed systems and data-intensive applications [24, 25] highlight how scalability is achieved through data distribution and parallel execution, while classical database literature formalises these principles in the context of distributed

query processing [26]. When dealing with spatial and multidimensional data, additional challenges arise, as data locality and load balancing must be carefully managed. Modern database systems, such as PostgreSQL [27], provide built-in partitioning mechanisms (e.g., range and hash partitioning), whose effectiveness depends on the underlying data distribution and workload characteristics. Multidimensional tiling has been proposed as an effective strategy to organise such data and improve query performance [28], particularly in scenarios involving large-scale spatial joins and grid-based computations.

Furthermore, distributed database systems enable horizontal scalability across multiple nodes, improving query performance for large-scale workloads. Among these, Citus[1] is an extension of PostgreSQL designed to ease horizontal scaling, making it suitable for handling large datasets across multiple machines [15]. It distributes both data and queries across a cluster, allowing users to leverage the power of a distributed system while maintaining compatibility with existing PostgreSQL tools. By using sharding and replication Citus scales PostgreSQL across several servers. *Sharding* is a method employed in distributed systems to divide data horizontally across multiple servers or nodes. It involves splitting a large dataset into smaller, more manageable pieces known as *shards*. Each shard holds a portion of the data, and, collectively, they represent the entire dataset. Citus enables timeseries data to be scaled by combining PostgreSQL single-node declarative table partitioning with its distributed sharding capabilities, creating a scalable time-series database. Recent works have also demonstrated how Citus can be effectively integrated with MobilityDB to support distributed spatiotemporal data management and query processing [29–31].

3 Data sources

In this section, we present the data sources used in this study. Specifically, six datasets are employed: AIS data, information on vessel characteristics, harbour areas, a spatiotemporal grid of the Northern and Central Adriatic Sea, environmental data, and hydrophone data from the SOUNDSCAPE project [11].

Automatic Identification System (AIS) The Automatic Identification System (AIS) was originally developed to enhance navigation safety by preventing vessel collisions. The International Maritime Organization[2] (IMO) requires AIS transmission for ships over 300 gross tons, all passenger vessels, and boats longer than 15 metres. Specifically, AIS data include several key components: the vessel's name; the Maritime Mobile Service Identity (MMSI), which uniquely identifies the vessel; its International Maritime Organization (IMO) number; the vessel's call sign; its position, given by latitude and longitude coordinates (in decimal degrees); the timestamp of the transmission (in UTC); speed over ground (SOG); course over ground (COG); heading; length overall (LOA); and the vessel type (e.g., cargo, sailing boat, fishing vessel). In some cases, AIS data may also provide details such as the starting port, destination, and estimated time of arrival.

[1] https://www.citusdata.com/

[2] https://www.imo.org/

We work on a dataset provided by the Italian Coast Guard, consisting of terrestrial AIS data for all types of vessels operating in the Northern and Central Adriatic Sea (GSA 17 area) from January 2020 to December 2020. We applied a data cleaning procedure to remove duplicated, incomplete, and out of range data. Specifically, duplicated AIS entries (identical position and timestamp) were eliminated, as they clearly represent transmission errors. As the estimation of underwater noise requires both the length overall (LOA) and vessel type, AIS records transmitted by vessels lacking either attribute were excluded. Only AIS records located within the Northern and Central Adriatic Sea were retained. For vessel types other than tankers, container ships, cruise and passenger ships, AIS points within ports with recorded speeds below 0.5 knots were removed, as they likely correspond to stationary conditions after berthing, when vessels are moored but still transmitting AIS signals. Conversely, for tankers, container ships, cruise and passenger vessels (i.e. those with AIS ship type identifiers between 60 and 89), such points were retained because these ships generally do not switch off their engines after being berthed; more precisely, their main propulsion systems are inactive, but auxiliary engines remain operational to supply electrical power and other essential onboard services. After data cleaning, approximately 32.5% of the records for the year 2020 were removed. Note that this dataset is not publicly available.

Table 1 summarises the number of vessels and AIS records (both raw and cleaned) for the year 2020. It is worth noting that, the year 2020 represents a period of restricted shipping activity due to the COVID-19 pandemic outbreak. In Italy, a lockdown was imposed from 9 March to 18 May 2020, during which many activities, including restaurants, were either closed or had to limit their operations. A second lockdown period began on 6 November 2020, when Italy adopted a new set of restrictions that remained in effect throughout December 2020.

Table 2 provides a detailed overview of the vessel types present in the dataset, including the number of unique vessels, the total number of AIS records, and the average length overall (LOA) for each category. Recreational vessels represent the largest share of the fleet, with about 2,700 vessels (38.2% of the total), followed by cargo with 1,916 units (26.9%), and tankers with 887 (12.5%). In terms of AIS data volume, however, fishing vessels overwhelmingly dominate the dataset, accounting for nearly 84 million AIS records (44.7% of

Table 1 No. of vessels and AIS records (raw and cleaned) for the year 2020

	No. Vessels		No. AIS records	
Period	Raw	Cleaned	Raw	Cleaned
January	2, 139	1, 893	25, 707, 105	20, 403, 601
February	2, 297	2, 034	25, 045, 757	19, 607, 111
March	2, 190	1, 941	21, 437, 011	15, 387, 298
April	1, 999	1, 703	21, 322, 047	13, 161, 003
May	2, 460	2, 171	24, 745, 515	15, 619, 650
June	3, 272	3, 024	31, 838, 085	20, 803, 598
July	3, 718	3, 537	34, 500, 114	21, 788, 265
August	3, 567	3, 365	20, 709, 501	13, 168, 429
September	3, 423	3, 188	19, 785, 086	12, 451, 660
October	3, 036	2, 730	18, 436, 188	12, 014, 857
November	2, 547	2, 213	17, 970, 259	12, 071, 466
December	2, 325	1, 933	16, 834, 648	11, 327, 564
Year 2020	7, 356	7, 122	278, 331, 316	187, 804, 502

Table 2 Vessel types in the cleaned dataset with the number of vessels, AIS records, and mean LOA (in metres) for 2020

Type of vessel	AIS Ship type ID	No. Vessels	No. AIS records	Mean LOA
Fishing	30	754	83, 930, 247	21.64
Tug	31, 32, 52	126	12, 434, 108	36.13
Naval	35	36	80, 903	40.85
Recreational	36, 37	2, 721	13, 902, 561	23.38
Government/Research	51, 53, 55	71	430, 002	22.69
Cruise	60-69 (LOA>100m)	62	3, 562, 093	188.08
Passenger	60-69 (LOA≤ 100m)	222	9, 989, 440	52.34
Cargo	70-79	1, 916	36, 940, 168	136.88
Tanker	80-89	887	14, 132, 712	160.54
Dredger	33	11	546, 814	39.84
Other	all other type IDs	316	11, 855, 454	40.48

all transmissions) — well above any other category. Cargo vessels follow with nearly 37 million records (19.7%), while recreational vessels contribute nearly 14 million (7.4%). This contrast suggests significant differences in operational patterns and AIS reporting intensity among vessel types.

Finally, Fig. 1 shows the distribution of AIS records by vessel type across the four seasons of 2020 (winter: January–March; spring: April–June; summer: July–September; and autumn: October–December). Fishing vessels consistently dominate AIS transmissions across all seasons, confirming their intensive and regular activity at sea. Nonetheless, their AIS volume gradually decreases from winter to autumn. This trend could be attributed to COVID-related restrictions and seasonal variability in fishing activity. Cargo, tanker, and tug vessels also display a similar downward trend, reflecting a general decline in maritime activity towards the end of the year. Conversely, passenger and cruise vessels display clear seasonal variability, with a pronounced increase during the summer months, corresponding to the peak of passenger and tourism-related traffic. Recreational vessels show an even more evident seasonal pattern, with intense activity in summer being about three times higher than in spring and more than four times higher than in autumn, followed by a further decline

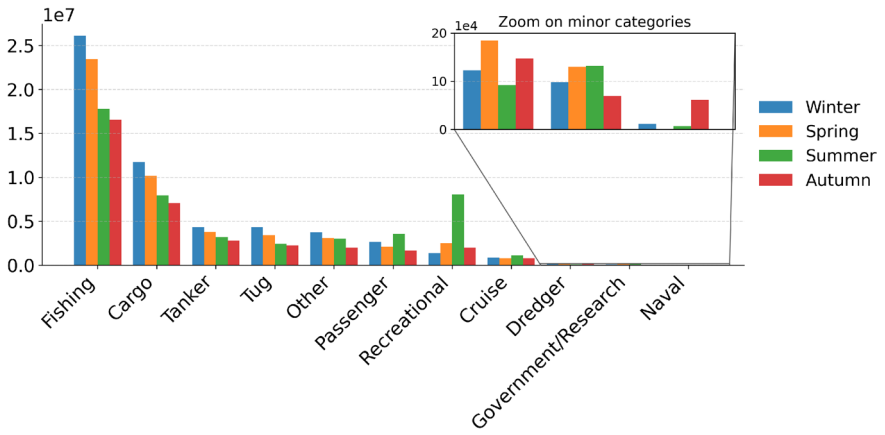


Fig. 1 Distribution of AIS data per vessel category across the four seasons of 2020

in winter. The inset highlights minor vessel categories — such as dredgers, government/research vessels, and naval units — that, despite contributing only a small portion of the total AIS data, still show noticeable seasonal variability due to their occasional or mission-oriented activities.

Vessel information In addition to AIS data, knowing the characteristics of the vessels is crucial. Specifically, for the fishing vessels, the Italian Coast Guard provided us with a dataset containing several attributes, including the MMSI of the vessel, the vessel's name, its country, the name of the port where the vessel is registered, the type of gear used, and the engine power (in horsepower). The information on the gear type is particularly relevant, as it enables to determine when a vessel is engaged in fishing activities and to assess the different impact that each fishing activity may have on the underwater noise, depending on the gear employed. More precisely, if the average speed of the fishing vessel is in the range of the fishing speed of the gear the boat is equipped with, the boat is assumed to be in a fishing phase, otherwise it is assumed to be in a navigation phase. The considered gears and their minimum and maximum speed during fishing activities are reported in Table 3.

Moreover, both the vessel category and the length overall (LOA) are key parameters for underwater noise assessment, as they are used for the estimation of the source level generated by boats (see Section 4.1). These two types of information were obtained from AIS data whenever available; when either of them was missing, they were retrieved from *MarineTraffic*[3] and *VesselFinder*[4] based on the vessel's MMSI.

Harbour areas An additional dataset provides information on the spatial extent and characteristics of harbour areas, which are essential to determine when vessels return to port and subsequently depart for a new trip. The dataset includes the geometries of the harbours, together with their names, the corresponding country, and the administrative region they belong to. Within the Northern and Central Adriatic Sea, a total of 178 harbours were identified. This area encompasses four countries, with 99 harbours located in Croatia, 72 in Italy, 4 in Montenegro, and 3 in Slovenia.

Grid of the Northern and Central Adriatic Sea In order to model the spatial distribution of vessel-generated underwater noise, we partition the Northern and Central Adriatic sea (GSA

Table 3 Gears and their minimum and maximum fishing speed (in knots)

ID	Gear description	Min speed	Max speed
PS	Purse seines	0	1.5
GNS	Set gillnets	0	1
LLS	Set longlines	0	2
DRB	Dredges	0	2
SOTB	Small bottom otter trawl	2	4.5
LOTB	Large bottom otter trawl	2	4.5
PTM	Pelagic pair trawl	2	5.5
RAP	Rapido	4	7

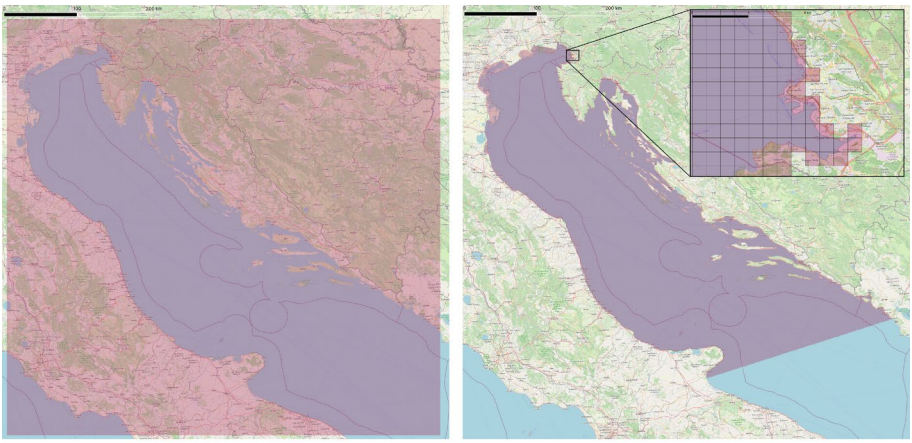
[3] https://www.marinetraffic.com/en

[4] https://www.vesselfinder.com/en

17 area) into a regular grid composed of square spatial cells ($1\text{km} \times 1\text{km}$). The cell size was defined by environmental experts to ensure an adequate representation of sound propagation while maintaining a balance between computational efficiency and spatial accuracy. The study area was delimited by the geographic boundaries of the Adriatic sea, within which a regular square grid was automatically generated to ensure uniform spatial coverage (Fig. 2a).

Subsequently, to isolate only the marine areas, we obtained the geometries of the Italian mainland from the *"Istituto Nazionale di Statistica"* (ISTAT),[5] while those of neighbouring countries intersecting the grid (e.g., Croatia and Slovenia) were derived from the *Natural Earth* dataset.[6] These geometries were employed to refine the grid boundaries and to distinguish between terrestrial and marine areas. As a result, the final grid, consisting of 101,113 cells, accurately follows the coastline and provides a detailed representation of the maritime regions. Figure 2b illustrates the resulting $1\text{km} \times 1\text{km}$ spatial grid covering the GSA 17 area.

Environmental data To calculate the propagation of underwater sound, it is essential to consider the environmental factors that play a crucial role in sound absorption. Therefore, we consider daily measurements of sea surface temperature (in degrees Celsius), daily measurements of sea salinity (in PSU, Practical Salinity Units), seawater potential of hydrogen (pH), and sea depth (in metres). The temperature dataset includes 7,651 daily measurements, while the salinity and pH datasets comprise 7,866 and 22,440 daily measurements, respectively. Temperature, salinity, and pH data are sourced from *Copernicus*,[7] whereas sea depth information, consisting of 10,387 records, comes from the *European Marine*



(a) Initial squared grid including both land and sea areas.

(b) Subset of the grid covering the area of interest (GSA 17, Northern and Central Adriatic Sea).

Fig. 2 Regular grid of $1\text{km} \times 1\text{km}$

[5] https://www.istat.it/it/archivio/222527

[6] https://www.naturalearthdata.com/

[7] https://www.copernicus.eu/en

Observation and Data Network (EMODnet).[8] Since the environmental measurements are available only at discrete locations, an *Inverse Distance Weighting* (IDW) interpolation is applied to generate continuous surfaces for each environmental variable across the Adriatic Sea grid. IDW is a deterministic spatial interpolation technique that estimates values at unsampled locations as a weighted average of nearby observations, with weights decreasing as distance increases. The resulting interpolated values are then assigned to each grid cell to ensure a complete environmental characterisation of the study area. These environmental features are used to calculate the absorption of sound in seawater (denoted as α) detailed in Section 4.2. The values of α are computed on a daily basis and used to enrich the spatiotemporal grid, thus providing a temporally varying environmental characterisation that captures daily variations in the absorption coefficient.

Hydrophone data To determine the ambient noise levels of the Northern and Central Adriatic Sea and fine tune the propagation model, we leverage direct acoustic measurements from the Interreg Project SOUNDSCAPE [11, 17]. The SOUNDSCAPE dataset is composed of 20 and 60 seconds averaged sound pressure levels (SPLs, dB ref 1μPa), recorded across a wide frequency range divided into third-octave bands. These data were collected at nine monitoring stations from March 2020 to June 2021, encompassing both COVID-19 lockdown periods: the first from March to May 2020 and the second from November 2020 to March 2021.

Figure 3 illustrates the positions and names of the nine monitoring stations set up by the project SOUNDSCAPE. In this work, we use the 60-second interval dataset and we focus exclusively on the 63 Hz frequency, which is part of the standards defined by the *European Marine Strategy Framework Directive* (MSFD). We select this low-frequency band because underwater sound at 63 Hz propagates over longer distances due to its lower absorption coefficient, thereby increasing the computational cost of underwater noise modelling compared with higher frequencies.

Table 4 presents, for each hydrophone, its location (latitude and longitude) and the number of records in the dataset of 60-second averaged SPLs (one-third octave, base 10), covering the period from March to December 2020. The acoustic measurements collected at the nine hydrophone stations are used to derive a continuous representation of ambient noise levels across the Northern and Central Adriatic Sea through an IDW interpolation. The interpolated data are then aggregated to compute the monthly ambient noise (see Section 4.3), which is subsequently assigned to each cell of the spatiotemporal grid.

4 Underwater noise model

In this section, we describe the model for underwater sound propagation based on our previous works [7, 8], focusing on the differences w.r.t. [8].

The basic objective of noise modelling is to assess how much noise a particular activity generates in the surrounding area. Specifically, the aim is to model the received noise level (RL) at a given point (or points), based on the sound source level (SL) of the noise source, and the amount of sound energy which is lost as the sound wave propagates from the source

[8] https://emodnet.ec.europa.eu/en

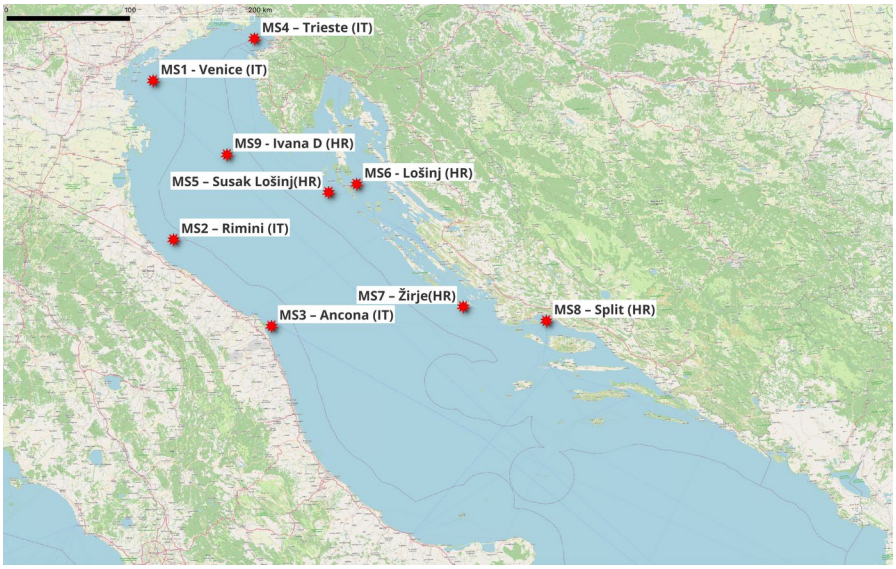


Fig. 3 SOUNDSCAPE hydrophones in the Northern Adriatic Sea

Table 4 Location and number of records for each hydrophone in the 60-second averaged SPL dataset from the SOUNDSCAPE project (March–December 2020)

Monitoring station	Longitude	Latitude	Mar–Dec 2020
MS1 - Venice (IT)	12°30.883′	45°19.383′	387, 432
MS2 - Rimini (IT)	12°42.656′	44°10.254′	340, 928
MS3 - Ancona (IT)	13°40.932′	43°31.954′	343, 479
MS4 - Trieste (IT)	13°33.917′	45°37.095′	318, 533
MS5 - Susak Lošinj (HR)	14°17.293′	44°29.545′	418, 330
MS6 - Lošinj (HR)	14°34.510′	44°32.747′	387, 488
MS7 - Žirje (HR)	15°36.020′	43°37.788′	339, 706
MS8 - Split (HR)	16°25.336′	43°29.895′	358, 889
MS9 - Ivana D (HR)	13°15.720′	44°46.953′	217, 928

to the receiver (transmission loss or propagation loss, TL). The conceptual workflow of the underwater noise model, connecting data inputs, physical assumptions, and database operations, is represented in Fig. 4. In a given frequency band, the source noise level depends on the characteristics of the vessel and its trajectory, while the spatial attenuation depends on morphological and environmental variables. A position-dependent ambient noise is interpolated based on available measurements at a small number of points, so to consider only vessel-produced noise that exceeds ambient noise, which is then aggregated with the noise of other independent vessels in the same area to output a cumulative soundscape. The corresponding database operations apply the formulas to determine the underwater noise and store the obtained results.

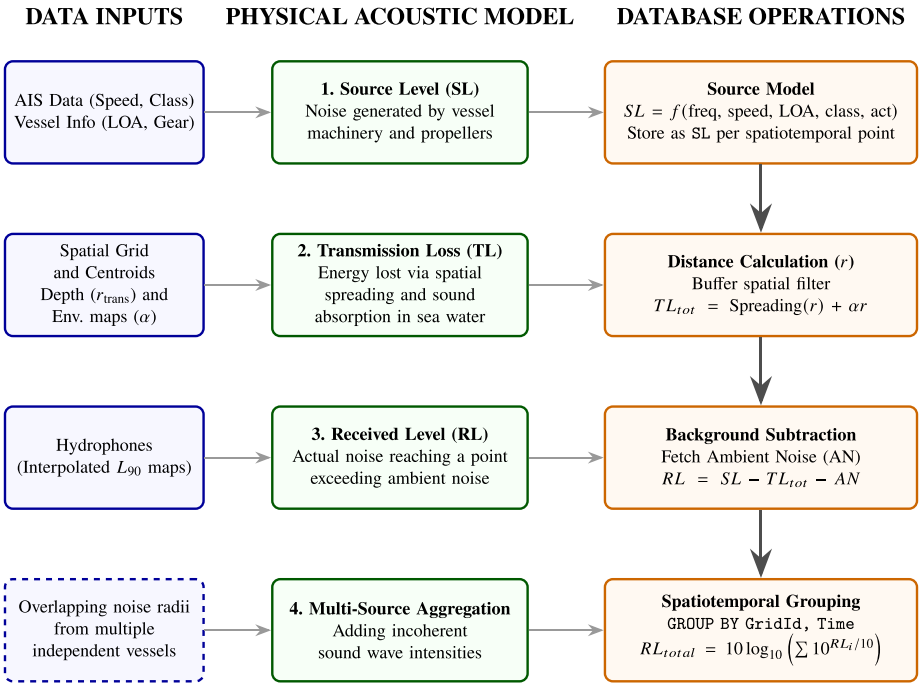


Fig. 4 Conceptual workflow of the underwater noise model

4.1 Source level

The principal sources of underwater noise are machinery, propellers, and cavitation. Our AIS dataset and the vessel information dataset contain details about boats, including their length overall (LOA), and the fishing gear in case of fishing vessels. However, these datasets do not include direct measurements of the sound pressure levels of the vessels, and do not report, for all vessel types, other key parameters, such as engine horsepower, which we used in our previous work [8] to estimate source levels for fishing vessels. Consequently, source level values are inferred from models and empirical data reported in the literature on underwater noise.

Specifically, we relied on the JOMOPANS–ECHO source level model proposed by MacGillivray and de Jong [14], which is widely accepted in the literature (see e.g. [5, 6]). This model was developed within the framework of the Joint Monitoring Programme for Ambient Noise in the North Sea (JOMOPANS),[9] aiming at improving underwater acoustic mapping derived from AIS data. The model was designed to address the significant uncertainties involved in relating the limited information available from AIS data to vessel noise emissions.

The JOMOPANS-ECHO model was developed as an updated reference spectrum model, designed to preserve the accurate speed and length dependencies of the RANDI 3.1 model [32] while incorporating critical modifications to enhance its alignment with the measured source levels (validation data) from the ECHO dataset [33]. The resulting model

[9] https://northsearegion.eu/jomopans/

calculates the ship source level spectrum, in one-third octave bands, as an explicit function of frequency (*f*), vessel speed (*V*), vessel length (LOA), and AIS ship type.

One of the most significant adjustments introduced by the JOMOPANS–ECHO model concerns the vessel speed: the generic reference speed $V_0 = 12$ knots in RANDI 3.1 is replaced by a reference speed per vessel class (V_C). This new reference speed V_C was determined by minimising the mean residual difference between model predictions and measured broadband source levels per vessel class, leading, for example, to V_C set to 18.0 kn for container ships and 6.4 kn for fishing vessels (see Table 5).

The model also incorporates an updated, class-specific reference spectrum developed individually for each vessel type by minimising model-data residuals across one-third octave bands. For certain vessel categories such as container ships, bulkers, and tankers, the revised spectrum includes an additional peak below 100 Hz, reflecting low-frequency energy components observed in the measurements. The classification of vessels is explicitly handled by integrating AIS information. The vessel class (*C*) is assigned on the basis of the AIS ship type ID and, where necessary, refined using additional criteria such as vessel length or operational speed.

In line with MacGillivray and de Jong [14], the source level SL (in dB re $1\ \mu\mathrm{Pa}\cdot\mathrm{m}$) is a function of frequency, vessel speed, length and class, as well as vessel activity, and is given by (1).

$$\begin{aligned} \mathrm{SL} = {} & K - 10\cdot(B+2)\cdot\log_{10}(f_1) \\ & + 5\cdot B\cdot\log_{10}(f) \\ & - 10\cdot\log_{10}\left[\left(1-\left(\frac{f}{f_1}\right)^{0.5\cdot(B+2)}\right)^2 + D^2\right] \\ & + 60\cdot\log_{10}\left(\frac{V}{V_\mathrm{C}}\right) + 20\cdot\log_{10}\left(\frac{\mathrm{LOA}}{\mathrm{LOA_C}}\right) \\ & + 10\cdot\log_{10}(0.231\cdot f) \\ & + G\,. \end{aligned} \tag{1}$$

Table 5 Classification of vessel classes (C) with corresponding AIS ship type IDs, and reference speedV_C (in knots), as in Table 1 of [14]

Vessel Class (C)	AIS Ship Type ID	V_C(kn)
Fishing	30	6.4
Tug	31, 32, 52	3.7
Naval	35	11.1
Recreational	36, 37	10.6
Government/Research	51, 53, 55	8.0
Cruise	60-69 (length$LOA_\mathrm{ref} > 100$ m)	17.1
Passenger	60-69 (length$LOA_\mathrm{ref} \leq 100$ m)	9.7
Bulker (Cargo)	70, 75-79 (speed$V_\mathrm{ref} \leq 16$ kn)	13.9
Containership (Cargo)	71-74 (all speed); 70, 75-79 (speed$V_\mathrm{ref} > 16$ kn)	18.0
Tanker	80-89	12.4
Dredger	33	9.5
Other	all other type IDs	7.4

Equation (1) incorporates refinements to the JOMOPANS-ECHO model; a detailed description of the parameters and the applied refinements is provided in Appendix A.

4.2 Transmission loss

To account for transmission loss, we adopt a combination of spherical propagation and mode stripping [34]. The resulting formula is indeed a function of distance r from source to receiver, at a given centre frequency:

$$TL = \begin{cases} 20 \cdot \log_{10}(r) & \text{if } r \leq r_{\text{trans}} \\ 15 \cdot \log_{10}(r) + 5 \cdot \log_{10}(r_{\text{trans}}) & \text{if } r > r_{\text{trans}} \end{cases} . \tag{2}$$

The $15 \cdot \log_{10}(r)$ dependence on range is known as mode stripping because it results from the gradual erosion of steep ray paths (high-order modes) after multiple bottom reflections. The transition region where neither spherical nor cylindrical spreading accurately describes the sound propagation could be approximated by assuming a sudden shift from spherical to cylindrical spreading at a *transition range* $\mathrm{r}_{\text{trans}}$. At 63 Hz this transition is expected to occur at around 400 metres, approximately 10 times the water depth. This indicates that $\mathrm{r}_{\text{trans}}$ is parametrically dependent on depth through a multiplicative factor of 10.

Environmental absorption features may affect the transmission loss, especially for large distances and high frequencies. To take into account all the environmental aspects that influence the sound propagation underwater, we add a sound absorption term, which is frequency-dependent and proportional to the distance from the source [35]:

$$TL_{tot} = TL + \alpha \cdot r \, . \tag{3}$$

In the literature there are several models for predicting the absorption of sound α in sea water which retain the essential dependence on temperature, pressure, salinity, acidity and other environmental features. In the Francois and Garrison model [36] the general equation for the absorption of sound in sea water, at a given frequency, is given as the sum of contributions from boric acid, magnesium sulfate, and pure water. Specifically, at frequency 63 Hz, α is on the order of 10^{-6} $\mathrm{dB/m}$.

4.3 Received noise and ambient noise

The classic sonar equation [37] provides an estimation of the received noise level (*RL*) by subtracting the transmission loss (*TL*) from the sound source level (*SL*). However, it does not consider ambient (or background) noise, which is present in the marine environment and varies with frequency. The *RL* exceeding the ambient noise (*AN*) is the following:

$$RL = SL - TL_{tot} - AN \, . \tag{4}$$

The received noise level is a function of frequency, vessel speed, length, class and activity, and distance from source to receiver.

The SOUNDSCAPE measurements [18, 19] are used to estimate the ambient noise. In particular, we employed the exceedance level L_{90} [38], which indicates the sound level

that is exceeded 90% of the time. As mentioned in [19], L_{90} can be referred to as common natural acoustic conditions. To account for spatial and temporal variability, we partitioned the Northern and Central Adriatic Sea into a regular grid composed of square spatial cells (1 km $\times$ 1 km). Given a certain frequency (63 Hz for this work), for each month of the year, we compute the *ambient noise map* in the following way. We assign an ambient noise value to each cell of our grid, starting from the cells where hydrophones are located: we associated the exceedance level L_{90} of the hydrophone at the corresponding cell. Then we applied an Inverse Distance Weighting (IDW) interpolation using QGIS,[10] producing maps that capture the heterogeneous underwater acoustic environment. For instance, for the year 2020 at 63 Hz, the most silent area is Ancona (Italy), with an ambient noise level of 61.5 dB, whereas the loudest zone is Ivana D (Croatia), with a value of 87 dB (both recorded in March).

4.4 Aggregating received noise levels

At a measurement point that is equally distant from two equally-powerful sound sources, the two contributions would add up in magnitude and phase. However, distinct and independent sources, such as two vessels, can be treated as incoherent sources. Even in a narrow frequency band, there will be a random phase difference between the two sources. Therefore, the noise in a 1/3 octave band around 63 Hz (or in any other band) gets increased by 3 dB if there are two equal contributions, by 6 dB if there are four equal contribution, etc. [16]. More precisely, what is added are the intensities, after inversion of the logarithmic function that defines the decibel. Generally, if we have *n* sources reaching a cell with *n* different values of RL, the total noise level is

$$RL_{total} = 10 \cdot \log_{10}(10^{RL_1/10} + \ldots + 10^{RL_n/10}) \,. \tag{5}$$

4.5 Model results

Having a set of vessel trajectories extracted from AIS data, in a given water basin in a certain period of time, the model produces a two-dimensional distribution of Sound Pressure Level (SPL) data. To illustrate the model behaviour in a real setting, we consider the data framework described in Section 3 for the Northern and Central Adriatic Sea. Specifically, we compare model data — generated on a $1\mathrm{km} \times 1\mathrm{km}$ spatial grid and a $60\mathrm{s}$ time grid — with the measurements of the SOUNDSCAPE project [18] for the spring 2020 season. The statistical comparison, based on the Cumulative Distribution Functions (CDFs), is performed at the positions of nine hydrophone measurement points. We notice that, at the modelling stage, the same measurement dataset has been used only to compute the interpolated ambient noise that is employed in (4). All other model parameters are extracted from the trajectories of the spatiotemporal database. Still, we register a misalignment (bias) of the CDFs at some measurement stations. The removal of such bias can be interpreted as a post-hoc calibration of the model, in line with [6, 39], to account for neglected variables that affect local propagation and ambient noise (e.g., sediment, masking islands, wind and waves), for local characteristics of vessel classes, and even for possible differences in measurement setups. The bias values at the nine measurement stations are listed in Table 6. We

[10] https://qgis.org/en/site/

Table 6 Bias (dB) and RMSE (dB) between the model-based and the observed CDFs values at the measurement stations

	MS1	MS2	MS3	MS4	MS5	MS6	MS7	MS8	MS9
Bias	19.46	-8.03	16.66	12.51	0.97	13.13	-0.33	-1.52	1.99
RMSE	0.44	2.00	5.42	5.62	2.11	5.52	1.05	5.17	1.36

note that the model-based CDFs show little bias at stations MS5, MS7, MS8, and MS9, the bias is relatively large and positive at stations MS1, MS3, MS4 and MS6, and it is moderate and negative at station MS2.

In a comparison of model data with observations, we are mostly interested in the shape of the statistical distributions of the measured and model-based sound pressure levels, aligned after bias removal. Figure 5 compares the observed (in dashed-blue) and model-based (in continuous-orange) cumulative distribution functions at the nine monitoring stations for the spring season. Overall, the model-based CDFs reproduce the general shape and range of the observed ones, indicating that the model captures the main statistical features of the measurements. In particular, for hydrophones MS1, MS2, MS5, MS7, and MS9, the model-based curves closely follow the observed CDFs. For MS4 and MS8, some discrepancies appear in the tails, with model based probability distributions that look narrower than the

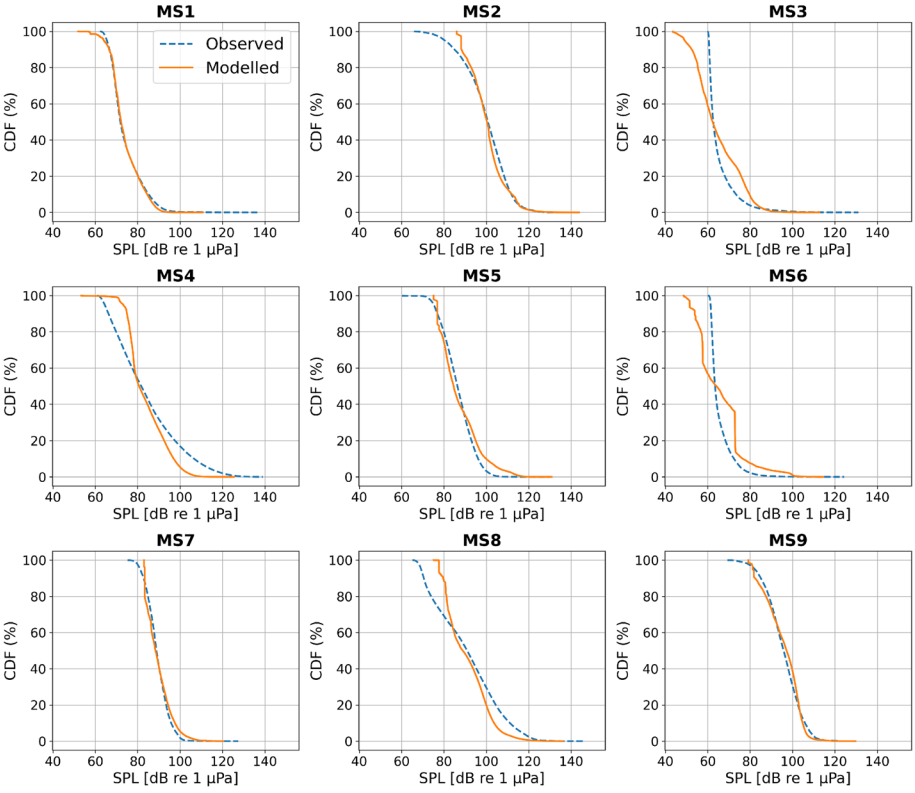


Fig. 5 Comparison between the observed and model-based Cumulative Density Functions (CDFs) at the nine monitoring stations during spring season

measured ones. MS3 and MS6 display more pronounced differences in CDF shapes. Overall, the agreement between the observed and model-based CDFs is fairly good.

The Root Mean Square Error (RMSE) of the CDFs, expressed in decibel, is computed as the mean squared deviation between the two CDFs for percentiles ranging from 5 to 95, to exclude extreme and rare events from computation [6]. In Table 6, the stations MS1, MS2, MS5, MS7, and MS9 show the lowest RMSE values, all remaining below or around 2 dB, indicating a very good statistical agreement between model and observations. Conversely, stations MS3, MS4, MS6, and MS8 exhibit higher RMSE values, close to 5 dB. These discrepancies may arise from local morphological and environmental factors not explicitly accounted for in the model, such as surface conditions, or from extraneous noise sources, such as non-tracked AIS-free boats.

5 Implementation of the underwater noise model

We aim to assess the performance of the proposed model and introduce optimisations to improve efficiency, enhance scalability, and reduce computational overhead. These improvements are essential, as the volume of AIS data has already increased compared to our previous work [8, 13] and is expected to continue growing. Section 5.1 outlines the workflow implementing the underwater noise model described in Section 4. To evaluate performance, we conducted experiments on data from spring 2020, extending the study presented in [13]. Section 5.2 presents some key implementation aspects, including the main database tables and operations involved in underwater noise computation. In particular, we focus on one of the most computationally expensive operations — the query that identifies the cells affected by noise propagation — which is the primary target of our optimisations. Section 5.3 introduces code-level optimisations for noise propagation, while Section 6 explores database-level improvements.

5.1 Framework overview

The overall process to calculate the underwater noise generated by vessels is illustrated in Fig. 6. It is organised into three main stages: (*i*) *Data Sources*, (*ii*) *Data Fusion and Trajectory Modelling*, and (*iii*) *Underwater Noise*.

The *Data Sources* considered in this process include AIS data, harbour areas, vessel information, hydrophone measurements, and environmental factors. These heterogeneous data are integrated within the *Data Fusion and Trajectory Modelling* stage to support the estimation of underwater noise. To this end, the Northern and Central Adriatic Sea is partitioned into a regular grid of square spatial cells ($1\text{km} \times 1\text{km}$). This grid, consisting of 101,113 cells, is enriched with ambient noise, estimated using the SOUNDSCAPE measurements, sea depth and the coefficient of absorption, which are essential for noise calculation. In parallel, vessel trajectories are reconstructed from AIS data and deployed in a spatiotemporal database [40]. These trajectories are equipped with semantic information, such as vessel length overall (LOA), vessel class, and the activities performed along their paths, which are used to infer how the noise propagates in the area of interest. The reconstruction of the trajectories and their semantic enrichment leverage the temporal and spatiotemporal types of MobilityDB, as well as the functions provided by this spatiotemporal database.

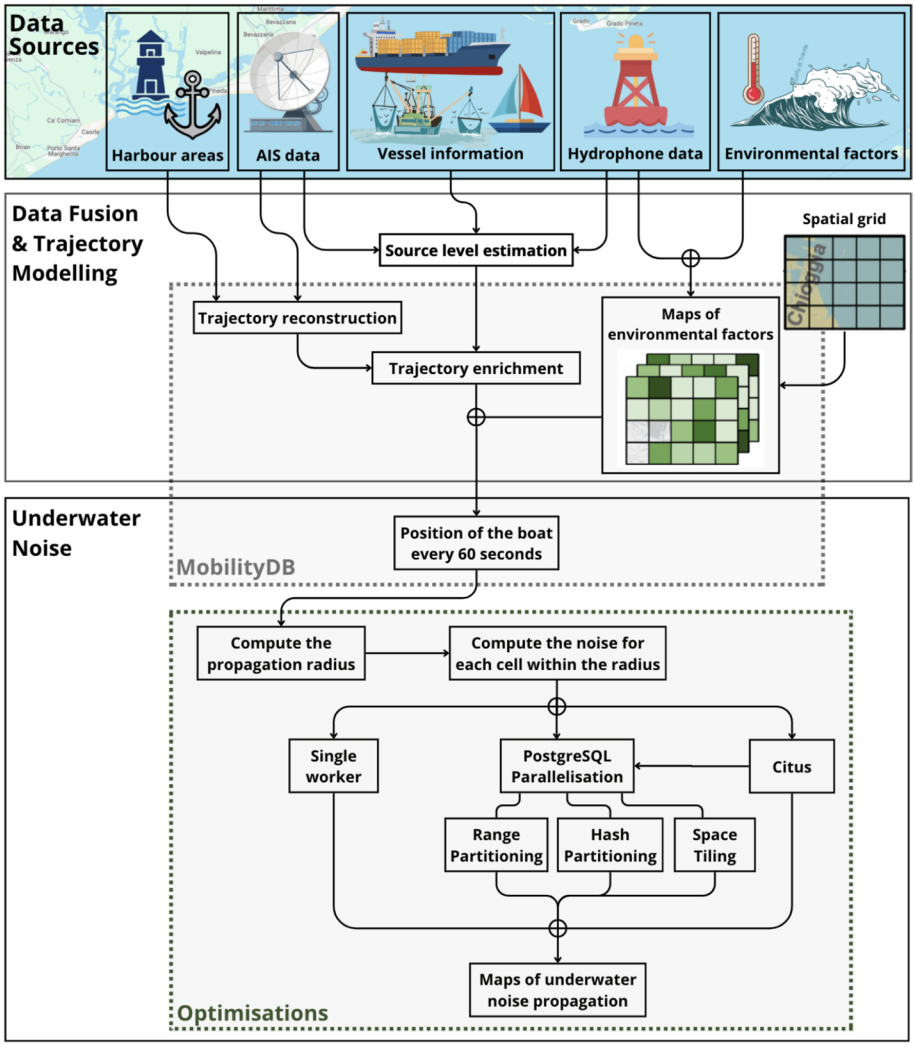


Fig. 6 Overview of the workflow divided into three stages, with MobilityDB handling spatiotemporal processing and PostgreSQL/Citus managing optimisation and parallelisation

The following stage, i.e., the *Underwater Noise* in Fig. 6, describes the main operations involved in computing the underwater noise maps, as well as the proposed optimisations. First, we use the spatiotemporal functions of MobilityDB to sample vessel trajectories at one-minute intervals, in order to determine vessel positions at specific temporal instants. For each position p, we estimate, for a considered frequency, the decibels produced by the vessel, based on its LOA, its class, its activity and speed. Next, we calculate the propagation radius r, i.e. the distance at which the noise generated by the vessel gets drowned into ambient noise, and we construct a buffer b with radius r around p. We select all the grid cells whose centroids fall within b and compute the distance between the sampled point p and these centroids. This distance is used to determine the received noise in the selected cells.

Finally, by grouping by cell id and time, we combine all the received sound levels to obtain the total noise level to be associated with the cell. All these database operations are also illustrated on the right side of Fig. 4.

The core computational contribution of this work lies in this stage, particularly in the optimisation of the noise propagation over the spatial grid. This is addressed through a progressive set of optimisation strategies, highlighted in the *Optimisations* module in Fig. 6. We first consider query-level optimisations executed in a single-worker setting, as described in Section 5.3. We then exploit PostgreSQL partitioning techniques (range, hash, and list) to enable parallel execution across multiple workers on a single machine. Finally, we extend this approach to a distributed setting by leveraging data distribution and sharding mechanisms provided by Citus, as discussed in Section 6.

For our performance study, we focus on the period from April to June 2020, which covers the spring season. This period was selected because it contains the largest volume of AIS data, and, additionally, SOUNDSCAPE measurements are available throughout the entire interval. Specifically, there are 3,778 vessels of different types, generating approximately 49.6 million AIS data points. Since the AIS data are limited to the Northern and Central Adriatic Sea, we consider the projected coordinate system for Italy, specifically the spatial reference identifier (SRID) 6876.

To process this data and build our model, we used a distributed infrastructure composed of eight virtual machines, each equipped with an AMD EPYC 7413 24-Core Processor (16 active cores running at 2.00 GHz), 32 GB of DDR4 RAM, and 150 GB of local storage, running Ubuntu 24.04 LTS. An additional 5 TB storage volume is provided for database hosting. The system hosts PostgreSQL 16.6, PostGIS 3.5.2, and MobilityDB 1.3.0, compiled with GEOS 3.12.1, PROJ 9.4.0, JSON-C 0.17, and GSL 2.7.1. Note that Citus was deployed in two distributed configurations, using either four or eight machines. In both cases, one machine acted as the coordinator, while the remaining machines served as workers, enabling distributed query execution. For all other configurations, including those involving code-level optimisation, PostgreSQL-based partitioning, and space tiling, only a single machine (the coordinator) was used.

5.2 Original pipeline

We now provide implementation details of the original pipeline to calculate sound propagation, which are also useful to better illustrate the query optimisations presented in the following sections. The schemas of the main database tables, produced by the *Data Fusion & Trajectory Modelling* stage, are created by the commands in Listing 1. The `Grid` table contains the spatial grid cells and is structured as follows: the unique grid identifier, the centroid geometry, the cell depth, the ambient noise level (which varies monthly), and the absorption coefficient α (which varies daily). The `VesselTrip` table stores the reconstructed vessel trajectories together with their semantic information. In particular, it includes the following attributes: the trip identifier, the vessel MMSI, and the vessel type. Additional attributes are the propagation radius, vessel speed, emitted noise (Source Level), vessel activity, and the spatial coordinates of the movement followed by the vessel. The last six attributes are modelled using temporal types, allowing for the representation of the variation in time of these values starting from the AIS data and appropriately interpolated according to their semantic

```
CREATE TABLE Grid(GridId integer PRIMARY KEY, GeomCentroid geometry,
  Depth real, AmbientNoise tfloat, Alpha tfloat);
CREATE INDEX IdxGridGeomCentroid ON Grid USING Gist(GeomCentroid);

CREATE TABLE VesselTrip(TripId integer, MMSI bigint,
  ShipTypeCode smallint, Radius tfloat, Speed tfloat,
  SL tfloat, Activity tint, Trip tgeompoint,
  PRIMARY KEY(TripId, MMSI));
```

Listing 1 Schema of the Grid and VesselTrip tables

characteristics (see [8] for further details). Note that each vessel may perform multiple trips, therefore the primary key is defined by the pair `TripId` and `MMSI`.

The next command (Listing 2) creates two tables. The first, `UnnestTrip`, contains the positions of vessel trajectories sampled at one-minute intervals, together with the vessel information and the values for activity, speed, emitted noise, propagation radius computed at the selected time instants. The second table, `Buffer`, is derived from `UnnestTrip` by associating each point with a circular buffer whose radius corresponds to the associated propagation radius.

One of the most computationally expensive steps in the workflow for the underwater noise computation is the selection of grid cells affected by noise propagation, performed through Listing 3. This query executes a `JOIN` between the `Buffer` table, which stores each point (`Point`) together with its associated buffer (`PropagationArea`), and the `Grid` table. The `ST_Intersects` operation identifies the grid cells whose centroids fall within the buffer associated with `Point`, thereby determining which areas are influenced by the noise generated at `Point`. For each point, the query returns the vessel position, the `GridId` of the corresponding grid cell, and the data required to compute the received noise level (RL) at the cell centroid, as defined in (4).

The query shown in Listing 3 is the target of our optimisations. The execution is dominated by the spatial join between the `Buffer` and `Grid` tables. Since the buffer geometries are generated dynamically for each sampled vessel position, no indexes are defined on the `Buffer` table. As a result, the query plan, shown at the bottom of Listing 3, adopts a nested-loop strategy, in which each buffer is processed sequentially and candidate grid cells are retrieved from the `Grid` table. To support this operation, the geometry attribute of the `Grid` table is indexed using a GiST spatial index (see Listing 1). This index is exploited through a bounding-box condition, which enables the efficient identification of candidate cells whose geometries may intersect the buffer. More precisely, the index performs a bounding-box intersection test, which serves as a coarse filtering step. The exact spatial predicate `ST_Intersects` is then applied to confirm the actual geometric intersection, discarding non-intersecting candidate cells. This two-stage evaluation significantly reduces the number of exact geometric comparisons; however, the spatial join remains the dominant component of the query cost.

For the spring season, trajectory reconstruction and enrichment take approximately 38 minutes. In contrast, the pipeline for computing underwater noise propagation, which spans from the sampling of vessel positions at one-minute intervals to the computation of received noise in each cell (the *Underwater Noise* stage in Fig. 6), is considerably more time-consuming, requiring about 2 days, 21 hours, and 49 minutes (i.e., 69 hours and 49 minutes). In Fig. 13, this configuration is referred to as *Original Pipeline*.

```
CREATE TABLE UnnestTrip(TripId, MMSI, ShipTypeCode, Longitude, Latitude,
  Point, Time, Activity, Speed, SL, Radius) AS
WITH PointMinutes (TripId, MMSI, ShipTypeCode, Time, Point, Activity,
  Speed, SL, Radius) AS (
  SELECT TripId, MMSI, ShipTypeCode,
    getTimestamp(unnest(instants(Trip))),
    getValue(unnest(instants(Trip))),
    getValue(unnest(instants(Activity))),
    getValue(unnest(instants(Speed))),
    getValue(unnest(instants(SL))),
    getValue(unnest(instants(Radius)))
  FROM VesselTrip )
SELECT TripId, MMSI, ShipTypeCode, ST_X(Point), ST_Y(Point),
  Point, Time, Activity, Speed, SL, Radius
FROM PointMinutes;

CREATE TABLE Buffer AS
SELECT TripId, MMSI, ShipTypeCode, Point, Time, Activity, Speed, SL,
  Radius, ST_Buffer(Point, Radius) AS PropagationArea
FROM UnnestTrip;
```

Listing 2 Creation of the UnnestTrip and Buffer tables

```
INSERT INTO PointGridDistance(TripId, MMSI, Time, ShipTypeCode, Point,
  SL, GridId, Depth, Alpha, AmbientNoise, Distance)
SELECT b.TripId, b.MMSI, b.Time, b.ShipTypeCode, b.Point, b.SL,
  g.GridId, g.Depth, valueAtTimestamp(g.Alpha, b.Time::DATE) AS Alpha,
  valueAtTimestamp(g.AmbientNoise, b.Time::DATE) AS AmbientNoise,
  ST_Distance(b.Point, g.GeomCentroid) AS Distance
FROM Buffer b, Grid g
WHERE ST_Intersects(b.PropagationArea, g.GeomCentroid);
/* Insert on pointgriddistance
    ->  Nested Loop
     ->  Seq Scan on buffer b
     ->  Index Scan using idxgridgeomcentroid on grid g
      Index Cond: (geomcentroid && b.propagationarea)
      Filter: st_intersects(b.propagationarea, geomcentroid) */
```

Listing 3 Original pipeline: returns the cells affected by noise propagation for each point

5.3 Code-level optimisation

The first improvement to the original pipeline involves a code optimisation that removes the `ST_Intersects` operation from Listing 3. Since `ST_Intersects` relies on geometry types and requires computationally intensive spatial operations, its removal enhances model performance. To avoid this computational overhead, we make two significant changes: (*i*) restructure the table `Grid` and (*ii*) use a bounding box instead of a buffer in noise propagation.

Grid table restructuring We add two new attributes to the cells of the grid: `GridR` and `GridC`, which indicate the row and column numbers within the grid. Hence, starting from the lower-left corner, the grid cells are numbered sequentially, so they are identified as (1, 1), (1, 2) and so on. This grid-based system allows for an efficient identification of the cells within a bounding box, without the need for costly spatial operations. The schema of the restructured table `Grid` is created by Listing 4. Note that two indexes are defined on the `GridR` and `GridC` columns to improve spatial query performance.

Bounding box for noise propagation To compute the total received noise level for each cell of our grid, we proceed as illustrated in Fig. 7. After reconstructing the vessel trajectories from the AIS data, we get the positions of all the vessels at the same time instants, i.e., every 60 seconds (Step 1 in Fig. 7). For each point p, we determine the cell c it belongs to, by comparing the coordinates of p with the grid cell boundaries which are computed by adding or subtracting 500 metres from the coordinates of the cell centroid. We calculate the noise generated by the vessel according to (1). Then, we compute the propagation radius r (in metres) and we build the *sound propagation bounding box* (Step 2 in Fig. 7), defined by the minimum and maximum row and column identifiers that enclose all the cells affected by the noise generated by the vessel at p. We store the points with their bounding box in the table `PointBoundingBox` created by Listing 5. The attribute `MMSI` is the identifier of the vessel the point belongs to, `PointId` identifies the spatiotemporal point, `Longitude` and `Latitude` are the coordinates of the point, `Time` specifies the date and hour of the point, `SL` denotes the decibel level generated by the vessel at that point, and `ShipTypeCode` represents the type of vessel according to the AIS classification. The remaining attributes represent the row and column identifiers used to construct the sound propagation bounding box. Moreover, B-tree indexes are built on `RowMin`, `RowMax`, `ColMin`, and `ColMax` to support the subsequent join operation for retrieving the grid cells intersecting each bounding box.

Notice that the boundaries of the bounding box are obtained simply by adding or subtracting the propagation radius (`Radius`) from the row and columns identifiers of the cell c, `GridR` and `GridC`. Hence, while in [8] we used a circular buffer around the point p, here we build a bounding box starting from the cell the point p belongs to. This can be computed easily by using the row and column identifiers of the grid cell, avoiding the use of the `ST_Intersects` operation, which is very time consuming. Indeed, with this approach, we retrieve the cells involved in the noise calculation in just 3 minutes for the entire dataset of the spring season.Next, we select all grid cells inside the bounding box and compute the distance between the point p and the cell centroids (Step 3 in Fig. 7). We use this distance to estimate the transmission loss, which allows us to determine the received noise level in the selected cells. Listing 3 is reformulated as shown in Listing 6.

```
CREATE TABLE Grid(GridId integer PRIMARY KEY,
  GridR integer NOT NULL, GridC integer NOT NULL,
  CentroidLong double precision, CentroidLat double precision,
  Depth real, AmbientNoise tfloat, Alpha tfloat);
CREATE INDEX IdxGridR ON Grid(GridR);
CREATE INDEX IdxGridC ON Grid(GridC);
```

Listing 4 Schema of the restructured Grid table

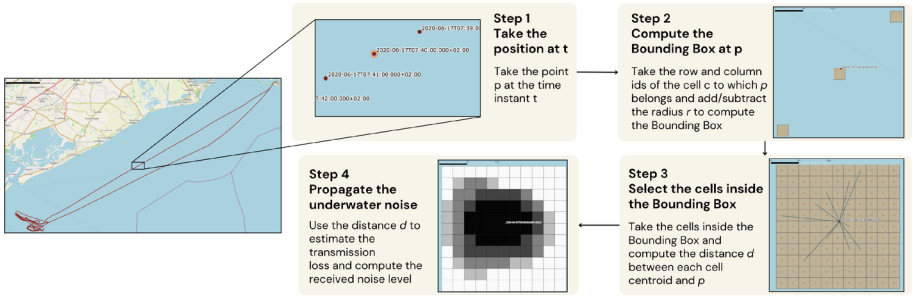


Fig. 7 Main steps in the calculation of the noise maps

```
CREATE TABLE PointBoundingBox AS
WITH UnnestTripWithCell AS (
  SELECT MMSI, TripId, ShipTypeCode, Longitude, Latitude, Time, SL,
  row_number() OVER (ORDER BY Time, TripId) AS PointId,
  (Radius/1000.) AS Radius, GridR, GridC
  FROM UnnestTrip JOIN Grid
    ON Longitude >= CentroidLong - 500 AND
    Longitude <= CentroidLong + 500 AND
    Latitude >= CentroidLat - 500 AND
    Latitude <= CentroidLat + 500 )
SELECT MMSI, PointId, Longitude, Latitude, Time, SL, ShipTypeCode,
  (GridR - Radius)::integer AS RowMin,
  (GridR + Radius)::integer AS RowMax,
  (GridC - Radius)::integer AS ColMin,
  (GridC + Radius)::integer AS ColMax
FROM UnnestTripWithCell;
CREATE INDEX IdxPtBboxRowMin ON PointBoundingBox (RowMin);
CREATE INDEX IdxPtBboxRowMax ON PointBoundingBox (RowMax);
CREATE INDEX IdxPtBboxColMin ON PointBoundingBox (ColMin);
CREATE INDEX IdxPtBboxColMax ON PointBoundingBox (ColMax);
```

Listing 5 Creation of the PointBoundingBox table

It is worth noting that we avoid the `ST_Intersects` operation by checking whether a cell falls within a point's bounding box through a simple comparison of the cell's row and column identifiers with those of the bounding box. The query plan, shown at the bottom of Listing 6, relies on a nested-loop join in which, for each tuple of `PointBoundingBox`, the candidate grid cells are retrieved through two bitmap index scans on `GridR` and `GridC`. These scans are combined through a bitmap-and operation before accessing the corresponding tuples via a bitmap heap scan. This execution strategy confirms that PostgreSQL effectively exploits both indexes on the `Grid` table to reduce the search space of the join. In contrast, the `PointBoundingBox` table is accessed through a sequential scan in this execution plan.

```
INSERT INTO PointBoundingBoxCells(MMSI, PointID, GridR, GridC, Time, SL,
  ShipTypeCode, Dist, Depth, AmbientNoise, Alpha)
SELECT p.MMSI, p.PointId, g.GridR, g.GridC, p.Time, p.SL,
  p.ShipTypeCode, SQRT(POWER(p.Longitude-g.CentroidLong,2) +
    POWER(p.Latitude-g.CentroidLat,2)) AS Dist, g.Depth,
  valueAtTimestamp(g.AmbientNoise,time::DATE) AS AmbientNoise,
  valueAtTimestamp(g.Alpha,time::DATE) AS Alpha
FROM PointBoundingBox p JOIN Grid g
  ON g.GridR >= p.RowMin AND g.GridC >= p.ColMin
  AND g.GridR <= p.RowMax AND g.GridC <= p.ColMax;
/* Insert on pointboundingboxcells
    ->  Nested Loop
     ->  Seq Scan on pointboundingbox p
     ->  Bitmap Heap Scan on grid g
      Recheck Cond: ((gridr >= p.rowmin) AND (gridr <= p.rowmax)
       AND (gridc >= p.colmin) AND (gridc <= p.colmax))
      ->  BitmapAnd
       ->  Bitmap Index Scan on idxgridr
        Index Cond: ((gridr >= p.rowmin) AND (gridr <= p.rowmax))
       ->  Bitmap Index Scan on idxgridc
        Index Cond: ((gridc >= p.colmin) AND (gridc <= p.colmax)) */
```

Listing 6 Optimised pipeline: reformulation of Listing 3

The last step, Step 4 in Fig. 7, combines all the contributions of the points of the different trajectories, and it is implemented by grouping by `GridR`, `GridC` and `Time`. We thus obtain for each cell the received noise level (RL).

The optimised pipeline produces results identical to those of the original version, but with notably improved performance. In the experiment for spring 2020, it completes in 2 days, 13 hours, and 6 minutes (61 hours and 6 minutes), which is approximately nine hours faster than the original pipeline (see Fig. 13, labelled *Optimised Pipeline*).

6 Partitioning and parallelisation

To further enhance the performance of the optimised pipeline, we present an analysis of various partitioning and parallelisation techniques. In Section 6.1, we examine PostgreSQL's native parallel query execution, which is also exploited in the subsequent optimisations. Section 6.2 explores table partitioning techniques in PostgreSQL, applying both range and hash partitioning strategies. In Section 6.3, we extend the previous approach by combining PostgreSQL partitioning with multidimensional tiling, focusing on the spatial dimension. Finally, in Section 6.4, we leverage the Citus extension of PostgreSQL to apply sharding and take advantage of its parallel query execution capabilities.

6.1 PostgreSQL parallelisation

The execution of Listing 6, selecting the cells affected by noise propagation for each point p (Step 3 in Fig. 7), remains a computationally expensive operation. This complexity arises from

the `JOIN` operation between the table `PointBoundingBox`, which contains each vessel position along with its sound propagation bounding box, and the table `Grid`, which consists of 101,113 cells. In our experiment, the table `PointBoundingBox` stores more than 58 million points. Consequently, the `JOIN` command involves a computational effort equivalent to approximately 58 million $\times$ 101 thousand operations, making it inherently costly.

To mitigate this computational cost, we first rely on PostgreSQL's native parallel query execution. PostgreSQL supports parallel query processing, a feature that enables the database engine to exploit multiple CPU cores when executing a single query. This mechanism allows certain operations to be distributed among several worker processes. While not all queries can benefit from parallel query processing, when applicable, it can yield substantial performance improvements, often achieving two- to fourfold speed-ups over single-core execution [27]. We evaluate the impact of PostgreSQL's native parallel query mechanism on the optimised pipeline described in Section 5.3. In this configuration, no additional partitioning was applied. The query planner automatically allocated six parallel workers to the task, resulting in a total execution time of approximately 12 hours and 24 minutes (see Fig. 13, labelled as *Optimised + Parallelisation*). Compared to the single-core execution time of about 2 days and 13 hours, this represents a fivefold improvement, confirming the effectiveness of parallel query execution in enhancing the efficiency of large-scale computations.

Note that all the optimisations described in the following sections take advantage of the PostgreSQL's native parallel query mechanism.

6.2 PostgreSQL partitioning

In this section, we explore the *table partitioning* technique in PostgreSQL to enhance the execution time of the code of the optimised pipeline. This method consists in dividing a logically large table into smaller physical segments, with each partition being an independent table that stores a specific subset of the original data. PostgreSQL natively supports three forms of partitioning [27]: (*i*) *Range partitioning*, where the table is divided into *ranges* based on a key column or set of columns, with each partition containing non-overlapping ranges of values; (*ii*) *List partitioning*, which explicitly assigns specific key value(s) to each partition, allowing precise control over data distribution; and (*iii*) *Hash partitioning*, where the table is divided by applying a hash function to the partition key.

Table partitioning offers several advantages that significantly improve both performance and data management. It enhances query execution by allowing the database management system to filter out irrelevant partitions, thus speeding up query processing, especially for large datasets. Additionally, partitioning simplifies data management tasks such as archiving, purging, backup and restore operations. Furthermore, indexing becomes faster as partitions reduce the scope of the data being indexed [41].

6.2.1 Range partitioning on time

To enhance the performance of our pipeline, as we have already remarked, we can improve the execution time of the `JOIN` operation between the table `PointBoundingBox` and the table `Grid`. To accomplish this task we partition the table `PointBoundingBox`. We apply range partitioning to the table `PointBoundingBox` to evaluate its impact on performance, focusing on the trade-off between execution time and parallelisation overhead. To

this end, we divide the dataset into 2, 4, 8, and 16 temporal partitions, each evenly spanning the period from 1 April to 30 June 2020. We create the partitioned table and define time-based partitions, each corresponding to a specific time interval. For instance, dividing the dataset into two temporal partitions results in the creation of the following tables, as shown in Listing 7.

After inserting the data into the partitioned table, the entries are automatically routed to the appropriate partition. Some statistics regarding the number of rows and disk usage for the configuration with two partitions are reported in Table 7.

Listing 6 is the query to optimise. We replace the table `PointBoundingBox` with the partitioned table `PtBboxRangePart`. The query plan involves a combination of parallel and sequential scans to optimise the data retrieval process. The first step is a parallel append operation, which processes multiple partitions of the table `PtBboxRangePart` in parallel. Each partition (corresponding to a different time range) is accessed through a parallel sequential scan. The second part consists of a bitmap heap scan on `Grid`, where candidate cells are efficiently filtered through bitmap index scans on `IdxGridR` and `IdxGridC` based on the bounding box constraints, following the same strategy illustrated in Listing 6. In essence, the query plan performs a parallel scan of partitioned data, followed by an efficient indexed search of the grid, ensuring faster query execution by narrowing down the relevant data points through partitioning and indexing.

By partitioning the table `PointBoundingBox` with range partitioning on time while leaving the rest of the code of the optimised pipeline unchanged, the entire process completes in 12 hours and 21 minutes with two partitions, 14 hours and 11 minutes with four partitions, 13 hours and 33 minutes with eight partitions, and 24 hours with sixteen partitions. Figure 8 illustrates the execution times for each configuration. As shown in the figure, performance does not scale linearly with the number of partitions. Although the system is configured to support up to eight parallel workers, the PostgreSQL planner automatically determines the number of workers based on cost estimates. In our experiments, it allocates six workers for the configuration with two partitions and only five for the other configurations. This indicates that increasing the number of partitions does not lead to a higher degree of parallelism. Consequently, the better performance obtained with two partitions suggests that this configuration achieves the optimal trade-off between minimising partition-management overhead and maximising parallel execution efficiency. It is also important to note that the `JOIN` operation produces a massive output expansion, from approximately 60 million input tuples to more than 12 billion result tuples. As a consequence, the dominant cost of the pipeline is not only the join itself but also the subsequent insertion of this large volume of data, which further limits the benefits of partitioning.

```
CREATE TABLE PtBboxRangePart(LIKE PointBoundingBox)
  PARTITION BY RANGE(Time);
CREATE TABLE PtBboxRangePart1 PARTITION OF PtBboxRangePart
  FOR VALUES FROM ('2020-04-01') TO ('2020-05-20');
CREATE TABLE PtBboxRangePart2 PARTITION OF PtBboxRangePart
  FOR VALUES FROM ('2020-05-20') TO ('2020-07-01');
```

Listing 7 Range-based partitioning of the PointBouningBox table on the Time attribute

Table 7 Statistics for the partitions by range on the `time` column

No. Partitions	Period	Disk Usage	Rows
1	1 April – 19 May	1, 908 MB	26, 118, 413
2	20 May – 30 June	2, 353 MB	32, 218, 155

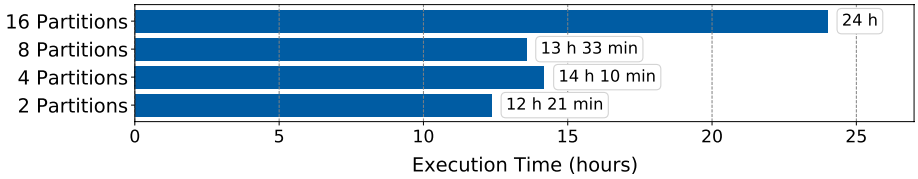


Fig. 8 Execution times (in hours) for the range partitioning configurations with two, four, eight, and sixteen partitions

6.2.2 Hash partitioning on MMSI

As a second partitioning experiment, we apply *hash partitioning* on the `MMSI` column of the `PointBoundingBox` table. As in the range partitioning configuration, we analyse the trade-off between execution time and parallelisation overhead by dividing the dataset into 2, 4, 8, and 16 hash partitions. The partitioned table for the case of two partitions is shown in Listing 8.

Next, we insert the values into the table `PtBboxHashPart`, which are automatically distributed across the partitions. Table 8 presents some statistics on the number of rows and the disk usage of each partition in the configuration with two partitions. In this case, we can observe that the data distribution across the two partitions is more balanced compared to the partitions obtained through time-based range partitioning.

We now employ the `PtBboxHashPart` table in Listing 6 in order to optimise its performance. The query plan is the same as that described for range partitioning and consists of a Parallel Seq Scan across the two partitions of the hash-partitioned table and a Bitmap Heap Scan on the table `Grid`. The number of workers assigned in the query plan varies according to the partitioning configuration: PostgreSQL allocates six workers when using two partitions, five workers when using four or sixteen partitions, and four workers when using eight partitions. This behaviour reflects PostgreSQL's adaptive cost-based strategy, which adjusts the number of workers according to workload expectations and resource availability.

By partitioning the table `PointBoundingBox` with hash partitioning on MMSI while leaving the rest of the code of the optimised pipeline unchanged, the entire process completes in 12 hours and 22 minutes with two partitions, 13 hours and 36 minutes with four partitions, 15 hours and 44 minutes with eight partitions, and 24 hours and 3 minutes with sixteen partitions. Figure 9 illustrates the execution times for each configuration. As shown in the figure, similarly to the range partitioning results, the configuration with two partitions proves to be the most efficient, suggesting that this setup provides the right balance between partition-management overhead and parallel processing efficiency.

```
CREATE TABLE PtBboxHashPart (LIKE PointBoundingBox)
  PARTITION BY HASH(MMSI);
CREATE TABLE PtBboxHashPart1 PARTITION OF PtBboxHashPart
  FOR VALUES WITH (MODULUS 2, REMAINDER 0);
CREATE TABLE PtBboxHashPart2 PARTITION OF PtBboxHashPart
  FOR VALUES WITH (MODULUS 2, REMAINDER 1);
```

Listing 8 Hash-based partitioning of the PointBoundingBox table on the MMSI attribute

Table 8 Statistics for the partitions by hash on the `MMSI` column

No. Partitions	Disk Usage	Rows
1	2, 438 MB	30, 260, 981
2	2, 262 MB	28, 075, 587

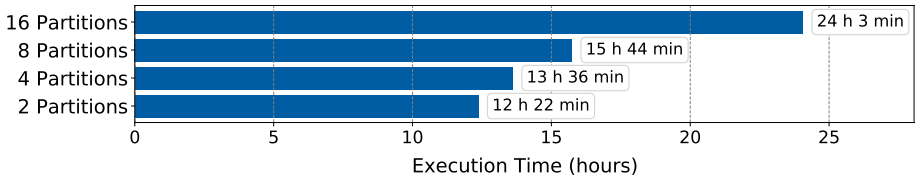


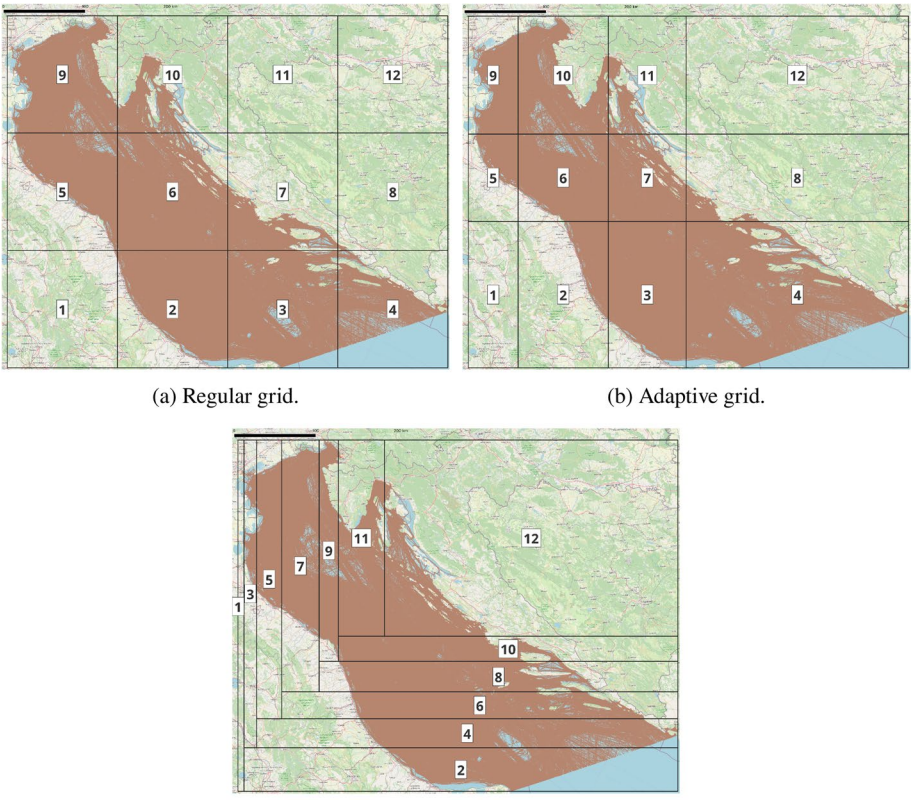
Fig. 9 Execution times (in hours) for the hash partitioning configurations with two, four, eight, and sixteen partitions

6.3 Space tiling and partitioning

Multidimensional tiling is a technique that partitions an *n*-dimensional domain into tiles of varying dimensions. This approach has several applications. For instance, multidimensional tiling can be applied to partition and/or distribute datasets across a cluster of servers. One key advantage of this partitioning mechanism is that it preserves spatial and temporal proximity, unlike traditional hash-based partitioning methods. This distribution reduces the amount of data that needs to be exchanged between nodes during query processing, a process commonly known as *reshuffling* [41].

In our work, we focus on tiling with respect to the spatial dimension. Specifically, we partition the positions of vessels based on their spatial locations. We consider three different types of tiling: (*i*) *regular*, where all tiles are of equal size in each dimension; (*ii*) *adaptive*, where the size of the cells adapts to the distribution of vessel points (spatial density); (*iii*) *k-d tree* (or k-dimensional tree), which also adapts to spatial density but divides the space hierarchically, alternating between the *X* and *Y* dimensions at each recursive step.

In the first case, we employ a regular tiling, constructing a uniform grid consisting of 4×3 cells, as shown in Fig. 10a. To generate this grid, we used the MobilityDB function `spaceTiles`. The grid size was manually tuned to balance the trade-off between the number of partitions and the data distribution within each partition. Then, we create the partitioned table along with the corresponding tables for the space tiles, by using the *list partitioning* technique, as shown in Listing 9.



(a) Regular grid.

(b) Adaptive grid.

(c) K-d tiling grid.

Fig. 10 Partitioning of vessel data with a regular, an adaptive, and a k-d tiling grids

Only the creation of the first tile is specified. Once the data is inserted into the partitioned table, the entries are automatically directed to their corresponding partitions. The limitation of this type of tiling is that it does not ensure balanced workload distribution across the tiles.

A possible solution to this issue is to use an *adaptive* grid, as illustrated in Fig. 10b. In this case, we create a grid that divides the region based on the distribution of vessel points in the Northern and Central Adriatic Sea. Specifically, the coordinates of all vessel locations are ordered along each spatial dimension and divided into a predefined number of bins using PostgreSQL's native `NTILE()` window function. This function ensures that each bin contains approximately the same number of points, thereby adapting the grid resolution to the actual spatial density of the data. The resulting X and Y bins are then combined to form spatial tiles, each defined by its lower and upper coordinates and represented as a geometry.

```
CREATE TABLE PtBboxRegGrid(LIKE PointBoundingBox)
  PARTITION BY LIST(TileId);
CREATE TABLE PtBboxRegGrid1 PARTITION OF PtBboxRegGrid
  FOR VALUES IN (1);
```

Listing 9 List-based partitioning of the PointBoundingBox table on the TileId attribute

The final `AdaptiveGrid` table therefore defines a grid whose cell size varies with data density, providing a more accurate and data-driven representation of the study area compared with the regular grid. Then, we partition the table `PointBoundingBox` according to the adaptive grid structure. The process of creating the partitioned table, along with the corresponding tables for the spatial tiles, follows the same steps as for the regular grid.

An alternative way to achieve a better balance among partitions is to employ the idea of *k-d tiling*, performing six splitting steps per dimension, as shown in Fig. 10c. To construct the tiles, we use the `kdtree_space_partition` function following the procedure described by Sakr et al. [41]. This approach does not construct a classical k-d tree data structure. Instead, it adopts the idea of recursive splitting along one dimension at a time, alternating between the X and Y dimensions, to generate spatial partitions with a more balanced data distribution. Specifically, the first split point is computed along the X dimension over the entire spatial domain. Once the first tile is created, its area is excluded from the remaining region to be partitioned. The next tile is then obtained by computing the split point along the Y dimension, and the process continues alternately for the subsequent tiles. As a result, the generated tiles may appear elongated and the resulting structure does not form a balanced binary tree. The objective is to generate partitions with comparable data volumes, thereby improving workload distribution across parallel workers.

After generating all tiles, we follow the same procedure as for the regular grid: we create the table `PtBboxKdGrid` and the associated tile tables using list partitioning. Table 9 presents statistics on the number of rows in each tile, as well as their respective disk usage, for the regular, adaptive, and k-d tree grids. The table shows that the data partitioned according to the adaptive grid exhibits a more balanced distribution across the tiles compared to the regular tiling. However, certain tiles (specifically, tiles 1, 11, and 12) contain few or even no data points, because they mostly cover the mainland. The k-d tree tiling further improves the balance, achieving the most homogeneous distribution of both data volume and disk usage across tiles, with only minimal variation between partitions.

To optimise Listing 6, we replace the `PointBoundingBox` table with the appropriate partitioned version — `PtBboxRegGrid`, `PtBboxAdGrid`, or `PtBboxKdGrid` — according to the chosen spatial tiling. The query plan, like the previous ones described in Section 6.2, combines parallel and sequential scans to optimise data retrieval. The first step is a parallel append operation, which processes multiple partitions of the specific partitioned table concurrently. This is followed by a bitmap heap scan on the table `Grid`, where rows are selected based on conditions that compare the grid's row and column identifiers with the corresponding bounding box identifiers from the partitions.

By tiling the space with the regular grid, the full pipeline is executed in 21 hours 30 minutes, while using the adaptive grid it completes in 22 hours and 14 minutes. Both configurations are slower than the range and hash partitioning approaches (approximately 10 hours), probably because the computational cost associated with handling a large number of spatial tiles outweighs the advantage of enhanced spatial selectivity. In contrast, when using the k-d tree tiling, the pipeline completes in approximately 15 hours, yielding a gain of about 7 hours with respect to the regular and adaptive grid configurations. The number of workers assigned in the query plan is four for all tiling configurations with twelve partitions.

Figure 11 compares the execution times of the regular, adaptive, and k-d tree tiling configurations, each evaluated with 12 partitions. The improved performance of the latter approach is due to the more balanced spatial distribution of data across tiles: unlike the other grids,

Table 9 Statistics for the partitions by list on the `TileId` column, obtained using the regular, adaptive, and k-d tree tiling approaches

	Regular Grid		Adaptive Grid		K-d tree Grid	
Tile	Disk Usage	Rows	Disk Usage	Rows	Disk Usage	Rows
1	7, 840 kB	94, 729	0 bytes	0	440 MB	5, 455, 702
2	622 MB	7, 723, 244	161 MB	2, 003, 973	296 MB	3, 676, 667
3	344 MB	4, 268, 358	616 MB	7, 648, 291	369 MB	4, 574, 386
4	234 MB	2, 898, 165	789 MB	9, 792, 606	330 MB	4, 101, 677
5	700 MB	8, 694, 421	376 MB	4, 664, 978	316 MB	3, 919, 916
6	666 MB	8, 263, 615	399 MB	4, 956, 881	329 MB	4, 080, 463
7	569 MB	7, 058, 812	406 MB	5, 033, 523	303 MB	3, 765, 680
8	9, 136 kB	110, 420	386 MB	4, 790, 590	384 MB	4, 761, 643
9	1, 324 MB	16, 437, 041	799 MB	9, 919, 706	400 MB	4, 965, 459
10	225 MB	2, 787, 763	614 MB	7, 623, 476	516 MB	6, 407, 275
11	0 bytes	0	153 MB	1, 902, 544	403 MB	5, 003, 764
12	0 bytes	0	0 bytes	0	614 MB	7, 623, 936

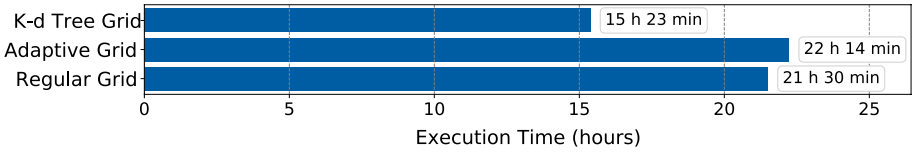


Fig. 11 Execution times (in hours) for the regular, adaptive, and k-d tree tiling configurations with twelve partitions

the k-d tree tiling produces no empty partitions and maintains a relatively uniform number of records per tile. As a result, the workload is more evenly distributed among parallel tasks, reducing the time spent on partitions with sparse data and improving overall query efficiency.

Since spatial partitioning with the k-d tiling balances the data more effectively across partitions, resulting in the best computational performance among the tiling methods, we also investigate the trade-off between execution time and parallelisation overhead, as in the case of range and hash partitioning. In particular, we divide the dataset into 2, 4, and 8 k-d-based partitions, in addition to the 12-partition configuration already evaluated. Also in this case, the number of workers assigned in the query plan is automatically determined by PostgreSQL based on cost estimates. Specifically, six workers are assigned for the k-d tiling with two partitions, five for the configurations with four and eight partitions, and four for the configuration with twelve partitions. This confirms that increasing the number of partitions does not result in a higher degree of parallelism. The total execution times are 12 hours and 22 minutes for the two-partition configuration, 13 hours and 40 minutes for the four-partition one, and 13 hours and 39 minutes for the eight-partition configuration. Figure 12 shows the execution times obtained using spatial partitioning with k-d tiling for the four configurations analysed. As can be observed, increasing the number of partitions results in longer execution times. The configuration with two partitions achieves the best performance, while additional partitions introduce computational overhead, reducing the overall efficiency of the model.

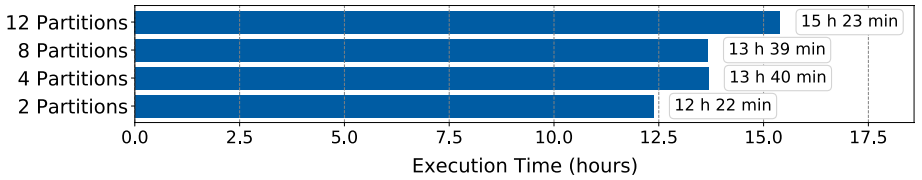


Fig. 12 Execution times (in hours) for the k-d tree configurations with two, four, eight, and twelve partitions

6.4 Using Citus for parallelisation

To further optimise our pipeline, we first distributed the data partitions using the Citus sharding mechanism. The implementation was deployed on the machine described in Section 5.1, running Citus 12.1.9. We evaluated two distributed configurations, using either four or eight machines, as also considered in Cubukcu et al. [15]. In both cases, one machine acted as the coordinator, while the remaining machines served as workers, enabling distributed query execution without applying any additional partitioning strategies. We adopted four- and eight-node configurations because, as detailed in Cubukcu et al. [15], a single-node Citus setup does not yield immediate performance improvements. This behaviour was also observed in [13], where a single-node Citus configuration performed slightly worse than parallel execution with PostgreSQL. The tables `PointBoundingBox` and `Grid` were distributed using Citus functions as shown in Listing 10.The first function distributes the table `PointBoundingBox` into multiple horizontal shards based on the `PointId` column. The second function distributes the table `Grid` into a single shard and replicates this shard to every worker node. Tables distributed in this way are called *reference tables* and are used to store data that require frequent access by multiple nodes within the cluster.

When executed using Citus, the query plan reveals that the workload is divided into multiple tasks, with a total of 32 tasks created. Each task is assigned to a specific execution node, ensuring efficient parallel processing. Within each task, a gathering operation takes place, using multiple worker threads to further parallelise the workload. The query plan performs two main operations: the *Parallel Append* retrieves data from multiple partitioned tables, and the *Bitmap Heap Scan* identifies the relevant grid cells by verifying that their positions fall within the bounding box. This step is optimised through index-based filtering on the row and column attributes, further enhancing performance.

Using Citus alone with four machines, the entire pipeline is executed in 7 hours. The computation of sound propagation is about eight times faster than the optimised pipeline without partitioning (Section 5.3) and roughly twice as fast as the partitioned PostgreSQL version (Section 6.2). When scaling to eight machines, the total execution time is reduced to approximately 2 hours and 56 minutes, corresponding to a speed-up of about 2.4 times compared to the four-node configuration.

We then adopted the same Citus configuration while also exploiting PostgreSQL range partitioning on the time dimension. As outlined in Section 6.2, the table `PointBoundingBox` was partitioned based on time ranges. In this case, we use the best-performing partitioning from our experiments, which divides the data into two partitions — one spanning 49 days and the other 42 days. The partitions can be created using the Citus function shown in Listing 11. Furthermore, the tables `PointBoundingBox` and `Grid` were distributed

```
SELECT create_distributed_table('PointBoundingBox', 'PointId');
SELECT create_reference_table('Grid');
```

Listing 10 Definition of the distributed PointBoundingBox and the reference Grid tables

```
SELECT create_time_partitions (
  table_name := 'PtBboxRangePart', partition_interval := '49 days',
  start_from := '2020-04-01 00:00:00', end_at := '2020-06-30 23:59:59');
```

Listing 11 Partition of the PtBboxRangePart table using Citus

using Citus, as described previously. Table 7 reports statistics on the number of rows in each partition and their corresponding disk usage. Using both Citus and PostgreSQL range partitioning on time with four machines, the entire pipeline is executed in 6 hours and 44 minutes, which is slightly faster (about 20 minutes) than the configuration using Citus alone. Using eight machines, the pipeline is executed in 3 hours and 25 minutes, corresponding to a speed-up of about 2 times compared to the four-node configuration.

We also employ Citus for the spatial tiling described in Section 6.3. Specifically, the `PointBoundingBox` table is partitioned according to the k-d tree grid structure with two partitions, as this tiling yielded the best performance in our experiments. The table is then distributed using the Citus function introduced earlier. The query plan is clearly similar to the case described above, with the workload distributed across multiple tasks. With four machines, the execution time for the entire pipeline, using Citus and distributing the points according to the two-partition k-d tree grid, is 6 hours and 59 minutes. This configuration is slightly slower than the range-partitioned configuration, yet marginally faster than using Citus alone. With eight machines, the execution time decreases to 3 hours and 8 minutes, corresponding to a speed-up of about 2.2 times compared to the four-node configuration. Figure 13 illustrates the execution times obtained using Citus for each configuration.

As detailed in Cubukcu et al. [15], the use of Citus with a four-node cluster can significantly improve query performance through parallel task execution and distributed data processing. Overall, the execution with Citus using four nodes is approximately eight times faster than the optimised pipeline, more than twice as fast as the k-d tree tiling, and around twice as fast as the range and hash partitioning strategies. When scaling to eight nodes, the execution time is further reduced, achieving an improvement of over 20 times compared to the optimised pipeline and about 4 times compared to the k-d tree, range, and hash partitioning strategies.

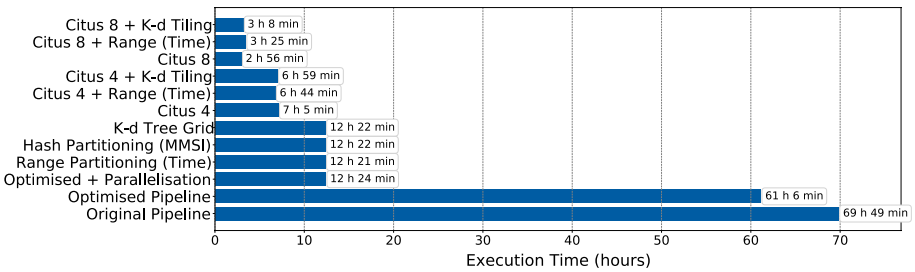


Fig. 13 Execution times (in hours) of the different implementations

6.5 Final considerations

A performance evaluation is conducted to quantify the impact of the proposed data partitioning and distribution strategies on execution time. Figure 13 summarises the results of our experiments conducted on the spring season of 2020, reporting the best performance achieved by each technique.

We start from the baseline configuration (*Original Pipeline*), which includes the computationally expensive `ST_Intersects` operation used to identify the grid cells affected by the noise emitted by each vessel. This model requires 2 days 21 hours and 49 minutes (69 hours and 49 minutes) to complete. Then, we consider the optimised version (*Optimised Pipeline*), which eliminates the costly geometric intersection by using the grid row and column identifiers to determine the affected cells. This configuration completes in 2 days 13 hours and 6 minutes (61 hours and 6 minutes), achieving a reduction of approximately 9 hours compared to the original pipeline.

Next, we evaluate the impact of PostgreSQL's native parallel query mechanism on the optimised model, without applying any partitioning technique (*Optimised + Parallelisation*). This configuration achieves a total execution time of 12 hours and 24 minutes. Compared to the single-core execution, this corresponds to a fivefold improvement in performance. Similarly, applying range partitioning on time and hash partitioning on MMSI within PostgreSQL yields comparable results, with execution times slightly exceeding 12 hours. The use of the k-d tiling (spatial tiling) with two partitions leads to an execution time of 12 hours and 22 minutes, which is similar to the execution times observed for range and hash partitioning with two partitions.

A significant improvement is achieved with Citus, deployed on four virtual machines (one coordinator and three workers), reducing the execution time to approximately 7 hours. This corresponds to an almost tenfold speed-up with respect to the original pipeline, a ninefold improvement over the optimised version, and a twofold gain compared to PostgreSQL with range or hash partitioning. Moreover, integrating Citus with range partitioning on time or with k-d tiling further reduces the execution time to slightly less than 7 hours, yielding marginal improvements over the configuration using Citus alone.

Finally, we evaluate the same configurations using a larger Citus cluster with eight nodes. In this setting, execution times decrease to approximately 3 hours, with slight variations depending on the partitioning strategy: Citus alone completes in 2 hours and 56 minutes, Citus with k-d tiling in 3 hours and 8 minutes, and Citus with range partitioning in 3 hours and 25 minutes. Compared to the four-node deployment, this results in an additional speed-up of more than twofold, further highlighting the strong scalability of the distributed solution. These findings confirm the scalability and efficiency of the distributed approach, showing that the adoption of Citus with multiple nodes significantly enhances the performance of our model.

7 Concluding remarks

Monitoring underwater noise pollution caused by human activities is crucial for preserving a healthy marine ecosystem. For this reason, in [7, 8] we developed a model for underwater noise propagation based on AIS data, leveraging semantic trajectories to describe

vessel movements and estimate their acoustic impact over space and time. In the present work, we build upon and extend the optimisations proposed in [13], introducing additional enhancements to improve efficiency, scalability, and overall computational performance. Specifically, we expanded the AIS dataset to include all vessel categories (rather than only fishing vessels) for the year 2020, and we enlarged the study area to encompass both the Northern and Central Adriatic Sea. We also refined the sound propagation model to ensure its applicability across different vessel types, and we validated the resulting model w.r.t. the SOUNDSCAPE measurements. After extending the dataset and refining the model, we further focused on improving the computational efficiency of the underwater noise model. The proposed optimisations include code-level improvements, table partitioning (using range, hash, and list strategies), spatial tiling (with regular, adaptive, and k-d grids), and distributed processing with Citus deployed across four and eight servers. The experimental results demonstrate that these strategies substantially improve performance while preserving the accuracy of the sound propagation model, reducing the overall execution time from nearly three days to about three hours — thus confirming the effectiveness of parallel and distributed processing for large-scale underwater noise modelling.

Beyond the specific application addressed in this work, the proposed optimisation strategies are applicable to a broader class of data-intensive spatial and spatiotemporal workflows. In particular, they are relevant for scenarios characterised by large intermediate result expansion, where join operations and subsequent insertion phases dominate the overall execution cost. In such cases, our study provides practical guidance for selecting appropriate partitioning schemes, spatial tiling strategies, and distributed configurations, reducing the need for extensive preliminary analysis.

As future work, we plan to extend the current analysis to cover the entire year 2020, rather than focusing solely on the spring season. Once the dataset is expanded, we will evaluate the model at yearly scale and conduct a more comprehensive investigation of underwater noise pollution across seasons and vessel categories (e.g. fishing, passenger, or cargo vessels). In addition, we intend to investigate further performance optimisation strategies for large-scale spatiotemporal processing. In particular, we will explore the adoption of GPU-accelerated database extensions, such as PG-Strom, which may significantly reduce the execution time of computationally intensive operations, including large joins within our pipeline.

Overall, this work enhances our original underwater sound propagation model with greater computational efficiency, offering a scalable solution for modelling underwater noise. By balancing estimation accuracy with computational effort, it can provide a convenient alternative to existing approaches, which often rely on hydrophone measurements or acoustic simulations and require extensive input data along with significant computational resources to manage complex calculations.

Appendix A: Source level model: parameters and details

In this appendix, we provide a detailed description of the source level formula presented in Section 4.1 (see (1)) and reported here as (A1).

The source level SL (in dB re $1\ \mu\mathrm{Pa}\cdot\mathrm{m}$) is a function of the centre frequency of the one-third octave bands (f, in Hz), the vessel speed (V, in metres), the vessel class (C, see Table 5), the vessel length overall (LOA, in metres), and the vessel activity (act). To ease the

reading, in the formula we omit the parameters of the main function $SL(f, V, \mathrm{LOA}, C, \mathrm{act})$ as well as of the auxiliary functions:

$$\begin{aligned}\mathrm{SL} = {} & K - 10 \cdot (B+2) \cdot \log_{10}(f_1) \\ & + 5 \cdot B \cdot \log_{10}(f) \\ & - 10 \cdot \log_{10}\left[\left(1 - \left(\frac{f}{f_1}\right)^{0.5\cdot(B+2)}\right)^2 + D^2\right] \\ & + 60 \cdot \log_{10}\left(\frac{\underline{V}}{V_\mathrm{C}}\right) + 20 \cdot \log_{10}\left(\frac{\mathrm{LOA}}{\mathrm{LOA_C}}\right) \\ & + 10 \cdot \log_{10}(0.231 \cdot f) \\ & + G\,.\end{aligned} \tag{A1}$$

In (A1), V_C (knots) is the reference speed associated with the vessel class (see Table 5) and $\mathrm{LOA_C}$ (in metres) denotes the reference length overall, set to 91.44 metres for all vessel classes. Moreover, the auxiliary functions $\underline{V}$, K, B, D, and G are defined as follows.

$$\underline{V}(V) = \begin{cases} V & \text{if } V \geq V_{\min}, \\ V_{\min} & \text{otherwise.} \end{cases} \tag{A2}$$

$$K(f, C) = \begin{cases} 208 & \text{if } f < 100 \text{ Hz and } C \in \{\text{Cargo, Tanker}\}, \\ 191 & \text{otherwise.} \end{cases} \tag{A3}$$

$$B(f, C) = \begin{cases} 2 & \text{if } f < 100 \text{ Hz and } C \in \{\text{Cargo, Tanker}\}, \\ 0 & \text{otherwise.} \end{cases} \tag{A4}$$

$$f_1(f, C) = \begin{cases} 600 \cdot \dfrac{1}{V_\mathrm{C}} & f < 100 \text{ Hz and } C \in \{\text{Cargo, Tanker}\}, \\ 480 \cdot \dfrac{1}{V_\mathrm{C}} & \text{otherwise.} \end{cases} \tag{A5}$$

$$D(f, C) = \begin{cases} 0.8 & \text{if } f < 100 \text{ Hz and } C = \text{Cargo}, \\ 1 & \text{if } f < 100 \text{ Hz and } C = \text{Tanker}, \\ 4 & \text{if } C = \text{Cruise}, \\ 3 & \text{otherwise.} \end{cases} \tag{A6}$$

$$G(f, V, C, \mathrm{act}) = \begin{cases} g(f) & \text{if } C = \text{Fishing and act} = \text{fishing}, \\ -15 & \text{if } C \in \{\text{Cruise, Passenger, Cargo, Tanker}\} \\ & \quad \text{and act} = \text{in port and } V \leq 0.5, \\ 0 & \text{otherwise.} \end{cases} \tag{A7}$$

Equation (A1) extends the formula presented in [14] in three ways. First of all, to avoid an excessively-negative contribution of the speed term in the JOMOPANS–ECHO model at very low vessel speeds, a lower bound needs to be imposed. The fact that the JOMOPANS-ECHO model as such cannot be applied at very-low values of speed is evident from the form of the speed term $60 \cdot \log_{10}(V/V_\mathrm{C})$, that diverges to $-\infty$ as V goes down to zero. Such model limitation is also evident in Fig. 5 of MacGillivray and de Jong [14], reporting predicted and observed source levels versus speed, in the 63 Hz third-octave band. In the present study,

the source model is adjusted by adopting, for all vessel classes, a floor speed value $V_{\min}$ of 5 knots. For instance, for fishing vessels $V_C = 6.4\,\text{kn}$, the maximum decibel reduction due to the speed correction, ΔS, is computed as $\Delta S = 60 \cdot \log_{10}(V_{\min}/V_C) = -6.43\ \text{dB}$. This implies that, for fishing vessels operating below $5\ \text{kn}$, a constant correction of $-6.43\ \text{dB}$ is applied, whereas above this threshold the original formulation is used. The introduction of the function $\underline{V}$ (A2) ensures that the model remains physically consistent at low speeds, preventing unrealistic reductions in source level.

Second, if the ship is of class *Fishing*, when it is trawling, it typically produces higher levels of radiated noise compared to free-running vessels operating under the same machinery settings. Although published data are limited, some studies report noise increases of 5-15 dB during trawling [42, 43], with minimal effects below 100 Hz and greater increases at higher frequencies. In (A7), such increase is described by the function $g(f)$, whose value is set to $+5$ dB in the specific case of $f = 63$ Hz.

Finally, for container ships, tankers, cruise and passenger vessels (i.e. those with AIS ship type identifiers between 60 and 89), we retained the points located within ports even when the recorded speed was below 0.5 knots (as described in Section 3). This choice was motivated by the fact that these vessels usually do not switch off their engines after being berthed; more precisely, their main propulsion systems are inactive, but the auxiliary engines remain operational to supply electrical power and other essential onboard services. For this reason, the noise was computed as the source level (SL) reduced by 15 dB [44], to account for the fact that these vessels do not have active propellers when stationary, and the generated noise originates primarily from auxiliary machinery rather than propulsion activity (Second case of (A7)).

Acknowledgements This study was funded by the European Union – NextGenerationEU, in the framework of the iNEST – *Interconnected Northeast Innovation Ecosystem* (iNEST ECS_00000043 - CUP H43C22000540006). The fourth author was supported by the project "Multiscale Analysis of Human and Artificial Trajectories: Models and Applications", funded by the European Union - NextGenerationEU, Mission 4 Component 1 (MUR PRIN 2022RB939W, CUP B53D23013150006). This work took place within the framework of the DoE 2023-2027 (MUR, AIS.DIP.ECCELLENZA2023_27.FF project). The views and opinions expressed are solely those of the authors and do not necessarily reflect those of the European Union, nor can the European Union be held responsible for them.

Author Contributions GR, MS and AR developed the initial idea. DR and GR refined the underwater noise propagation model and performed its validation. EZ and GR implemented all model optimisations. GR, MS, and AR performed the various optimisations and interpreted the results. All authors contributed to the manuscript writing and approved the submitted version.

Data Availability The datasets presented in this article are not available because the raw AIS data are protected by confidentiality.

Code Availability The code is publicly available on GitHub. (https://github.com/giuRov/underwaterNoiseAdriaticSea). The repository includes the full pipeline, covering data ingestion, the construction of the Adriatic Sea grid enriched with environmental features, trajectory reconstruction and semantic enrichment from AIS data, as well as the sound propagation model. In addition, the code for the sound propagation component is provided for a selection of the partitioning and parallelisation strategies evaluated in this work, including range partitioning in PostgreSQL, adaptive grid-based approaches, the use of Citus for distributed parallelisation across multiple nodes, and its integration with k-d tree spatial partitioning.

Declarations

Competing interests The authors declare no competing interests.

References

1. Slabbekoorn H, Bouton N, Opzeeland I, Coers A, Cate C, Popper AN (2010) A noisy spring: the impact of globally rising underwater sound levels on fish. Trends in ecology & evolution 25(7):419–427. https://doi.org/10.1016/j.tree.2010.04.005
2. Williams R, Wright AJ, Ashe E, Blight LK, Bruintjes R, Canessa R, Clark CW, Cullis-Suzuki S, Dakin DT, Erbe C, Hammond PS, Merchant ND, O'Hara PD, Purser J, Radford AN, Simpson SD, Thomas L, Wale MA (2015) Impacts of anthropogenic noise on marine life: Publication patterns, new discoveries, and future directions in research and management. Ocean & Coastal Management 115:17–24. https://doi.org/10.1016/j.ocecoaman.2015.05.021
3. Duarte CM, Chapuis L, Collin SP, Costa DP, Devassy RP, Eguiluz VM, Erbe C, Gordon TAC, Halpern BS, Harding HR, Havlik MN, Meekan M, Merchant ND, Miksis-Olds JL, Parsons M, Predragovic M, Radford AN, Radford CA, Simpson SD, Slabbekoorn H, Staaterman E, Opzeeland ICV, Winderen J, Zhang X, Juanes F (2021) The soundscape of the Anthropocene ocean. Science 371(6529):4658. https://doi.org/10.1126/science.aba4658, https://www.science.org/doi/pdf/10.1126/science.aba4658
4. MacGillivray A, McPherson C, McPherson G, Izett J, Gosselin J, Li Z, Hannay D (2014) Modelling underwater shipping noise in the Great Barrier Reef Marine Park with AIS vessel track data. In: INTER-NOISE and NOISE-CON Congress and Conference Proceedings, pp 4467–4476. Institute of Noise Control Engineering, Melbourne, Australia. https://www.acoustics.asn.au/conference_proceedings/INTERNOISE2014/papers/p671.pdf
5. Larayedh R, Cornuelle BD, Krokos G, Hoteit I (2024) Numerical investigation of shipping noise in the Red Sea. Sci Rep 14(1):5851. https://doi.org/10.1038/s41598-024-56523-2
6. Ghezzo M, Petrizzo A, Madricardo F, Folegot T, Gallou R, Clorennec D, Chavanne R, Hemon E, Ferrarin C, Mihanović H, Pikelj K, Bastianini M, Pari A, Pari S, Menegon S, McKiver WJ, Farella G, Bosi S, Barbanti A, Picciulin M (2024) Natural and shipping underwater sound distribution in the Northern Adriatic Sea basin and possible application on target areas. Mar Pollut Bull 207:116852. https://doi.org/10.1016/j.marpolbul.2024.116852
7. Rovinelli G, Rocchesso D, Simeoni M, Raffaetà A (2024) Using semantic trajectories for spatio-temporal characterisation of underwater noise. In: Proceedings of the 6th International Workshop on Big Mobility Data Analytics (BMDA 2024) - EDBT/ICDT Workshops, vol 3651. https://ceur-ws.org/Vol-3651/BMDA-5.pdf
8. Rovinelli G, Rocchesso D, Simeoni M, Zimányi E, Raffaetà A (2025) Spatiotemporal characterisation of underwater noise through semantic trajectories. GeoInformatica 29:845–876. https://doi.org/10.1007/s10707-025-00547-x
9. Parent C, Spaccapietra S, Renso C, Andrienko G, Andrienko N, Bogorny V, Damiani ML, Gkoulalas-Divanis A, Macêdo JAF, Pelekis N, Theodoridis Y, Yan Z (2013) Semantic trajectories modeling and analysis. ACM Computing Surveys (CSUR) 45(4):1–32. https://doi.org/10.1145/2501654.2501656
10. Mello RS, Bogorny V, Alvares LO, Santana LHZ, Ferrero CA, Frozza AA, Schreiner GA, Renso C (2019) MASTER: A multiple aspect view on trajectories. Trans GIS 23(4):805–822. https://doi.org/10.1111/tgis.12526
11. SOUNDSCAPE Project. https://www.italy-croatia.eu/web/soundscape (2019-2021)

12. Zimányi E, Sakr M, Lesuisse A (2020) MobilityDB: A Mobility Database Based on PostgreSQL and PostGIS. ACM Trans Database Syst 45(4). https://doi.org/10.1145/3406534
13. Rovinelli G, Zimányi E, Simeoni M, Rocchesso D, Raffaetà A (2025) A Scalable Model for Vessel-Generated Underwater Noise: Enhancing Efficiency through Parallelisation. In: Proceedings of the 7th International Workshop on Big Mobility Data Analytics (BMDA 2025) - EDBT/ICDT Workshops. https://ceur-ws.org/Vol-3946/BMDA-5.pdf
14. MacGillivray A, Jong C (2021) A Reference Spectrum Model for Estimating Source Levels of Marine Shipping Based on Automated Identification System Data. J Marine Sci Eng 9(4):369. https://doi.org/10.3390/jmse9040369
15. Cubukcu U, Erdogan O, Pathak S, Sannakkayala S, Slot M (2021) Citus: Distributed PostgreSQL for data-intensive applications. In: Proceedings of the 2021 International Conference on Management of Data. SIGMOD '21, pp 2490–2502. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3448016.3457551
16. Erbe C, Duncan A, Hawkins L, Terhune JM, Thomas JA (2022) In: Erbe C, Thomas JA (eds) Introduction to Acoustic Terminology and Signal Processing, pp 111–152. Springer, Cham. https://doi.org/10.1007/978-3-030-97540-1_4
17. Petrizzo A, Barfucci G, Bastianini M, Centurelli M, Codarin A, Cukrov Car M, Dadić V, Falkner R, Ghezzo M, Leonori I, Mihanović H, Muslim S, Picciulin M, Radulović M, Rako-Gospić N, Sabbatini D, VučurBlazinić T, Vukadin P, Zdroik J, Madricardo F (2022) SOUNDSCAPE North Adriatic Underwater Noise Sound Pressure Levels. https://zenodo.org/records/7472152
18. Petrizzo A, Barbanti A, Barfucci G, Bastianini M, Biagiotti I, Bosi S, Centurelli M, Chavanne R, Codarin A, Costantini I, Cukrov Car M, Dadić V, Falcieri FM, Falkner R, Farella G, Felli M, Ferrarin C, Folegot T, Gallou R, Galvez D, Ghezzo M, Kruss A, Leonori I, Menegon S, Mihanović H, Muslim S, Pari A, Pari S, Picciulin M, Pleslić G, Radulović M, Rako-Gospić N, Sabbatini D, Soldano G, Tęgowski J, Vučur-Blazinić T, Vukadin P, Zdroik J, Madricardo F (2023) First assessment of underwater sound levels in the Northern Adriatic Sea at the basin scale. Scientific Data 10(1):137. https://doi.org/10.1038/s41597-023-02033-1
19. Picciulin M, Petrizzo A, Madricardo F, Barbanti A, Bastianini M, Biagiotti I, Bosi S, Centurelli M, Codarin A, Costantini I, Dadić V, Falkner R, Folegot T, Galvez D, Leonori I, Menegon S, Mihanović H, Muslim S, Pari A, Pari S, Pleslić G, Radulović M, Rako-Gospić N, Sabbatini D, Tegowski J, Vukadin P, Ghezzo M (2023) First basin scale spatial-temporal characterization of underwater sound in the Mediterranean Sea. Sci Rep 13(1):22799. https://doi.org/10.1038/s41598-023-49567-3
20. Zhu Z, Ma S, Zhu X-Q, Lan Q, Piao S-C, Cheng Y-S (2022) Parallel optimization of underwater acoustic models: A survey. Chin Phys B 31(10):104301. https://doi.org/10.1088/1674-1056/ac7ccc
21. Erbe C, MacGillivray A, Williams R (2012) Mapping cumulative noise from shipping to inform marine spatial planning. J Acoust Soc Am 132(5):423–428. https://doi.org/10.1121/1.4758779
22. Neenan ST, White PR, Leighton TG, Shaw PJ (2016) Modeling vessel noise emissions through the accumulation and propagation of Automatic Identification System data. In: Proceedings of meetings on acoustics 27(1). https://doi.org/10.1121/2.0000338
23. Jalkanen J-P, Johansson L, Andersson MH, Majam aki E, Sigray P (2022) Underwater noise emissions from ships during 2014-2020. Environmental Pollution 311:119766. https://doi.org/10.1016/j.envpol.2022.119766
24. Kleppmann M (2017) Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, Sebastopol, CA
25. Steen M, Tanenbaum AS (2023) Distributed Systems, 4th edn. Maarten van Steen, Amsterdam, The Netherlands
26. Özsu MT, Valduriez P (2020) Principles of Distributed Database Systems, 4th edn. Springer, Cham
27. The PostgreSQL Global Development Group: PostgreSQL 16.6 Documentation (2024). https://www.postgresql.org/files/documentation/pdf/16/postgresql-16-A4.pdf
28. Furtado P, Baumann P (1999) Storage of Multidimensional Arrays Based on Arbitrary Tiling. In: Proceedings of the 15th International Conference on Data Engineering (ICDE), pp. 480–489. IEEE Computer Society, Sydney, NSW, Australia. https://doi.org/10.1109/ICDE.1999.754964
29. Bakli M, Sakr M, Zimányi E (2020) Distributed Spatiotemporal Trajectory Query Processing in SQL. In: Proceedings of the 28th International Conference on Advances in Geographic Information Systems. SIGSPATIAL '20, pp 87–98. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3397536.3422262
30. Bakli MS, Sakr MA, Zimányi E (2020) Distributed Mobility Data Management in MobilityDB. In: Proceedings of the 21st International Conference on Mobile Data Management (MDM), pp 238–239. IEEE, Versailles, France. https://doi.org/10.1109/MDM48529.2020.00052

31. Bakli M, Sakr M, Zimányi E, Dijk N, Slot M (2025) Distributed MobilityDB: A Scalable Moving Object Database Management System. ACM Trans Spatial Algorithms Syst 11(2). https://doi.org/10.1145/3719202
32. Breeding, JE, Pflug LA (1996) Research Ambient Noise Directionality (RANDI) 3.1 Physics Description. Technical Report NRL/FS/7176–95-9628, Naval Research Laboratory, Stennis Space Center, MS, USA
33. Siceloff L, Howell WH (2013) Fine-scale temporal and spatial distributions of Atlantic cod (Gadus morhua) on a western Gulf of Maine spawning ground. Fisheries Research 141:31–43. https://doi.org/10.1016/j.fishres.2012.04.001. Reconciling Spatial Scales and Stock Structures
34. Ainslie M (2010) Principles of Sonar Performance Modelling. Springer Praxis Books. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-87662-5
35. Erbe C, Duncan A, Vigness-Raposa KJ (2022) In: Erbe C, Thomas JA (eds) Introduction to Sound Propagation Under Water, pp 185–216. Springer, Cham. https://doi.org/10.1007/978-3-030-97540-1_6
36. Francois R, Garrison G (1982) Sound absorption based on ocean measurements: Part I: Pure water and magnesium sulfate contributions. J Acoust Soc Am 72(3):896–907. https://doi.org/10.1121/1.388170
37. Urick RJ (1983) Principles of underwater sound 3rd edition. Peninsula Publising Los Atlos, California 22:23–24
38. Geel NCF, Risch D, Wittich A (2022) A brief overview of current approaches for underwater sound analysis and reporting. Mar Pollut Bull 178:113610. https://doi.org/10.1016/j.marpolbul.2022.113610
39. Folegot T, Hemon E, Nehls G, Schmiing M, Bräger S, Bellmann M, Gerlach S, Matuschek R, Flamme J (2024) In: Popper AN, Sisneros J, Hawkins AD, Thomsen F (eds) Near Real-Time Underwater Sound Modeling of Dredging Noise to Meet Regulatory Noise Thresholds, pp 1–19. Springer, Cham. https://doi.org/10.1007/978-3-031-10417-6_52-1
40. Brandoli B, Raffaetà A, Simeoni M, Adibi P, Bappee FK, Pranovi F, Rovinelli G, Russo E, Silvestri C, Soares A, Matwin S (2022) From multiple aspect trajectories to predictive analysis: a case study on fishing vessels in the Northern Adriatic sea. GeoInformatica 26:551–579. https://doi.org/10.1007/s10707-022-00463-4
41. Sakr M, Vaisman A, Zimányi E (2025) Mobility Data Science, 1st edn. Data-Centric Systems and Applications, p 592. Springer, Cham. https://doi.org/10.1007/978-3-031-82636-8
42. De Robertis A, Wilson CD (2006) Walleye pollock respond to trawling vessels. ICES J Mar Sci 63(3):514–522. https://doi.org/10.1016/j.icesjms.2005.08.014
43. Mitson R (1993) Underwater noise radiated by research vessels. In: ICES Marine Science Symposium, vol 196, pp 147–152
44. Wittekind D (2014) A Simple Model for the Underwater Noise Source Level of Ships. Journal of Ship Production and Design 30:7–14

Giulia Rovinelli received her M.Sc. (2022) and Ph.D. (2026) in Computer Science from Ca' Foscari University of Venice, Italy. She is currently a researcher in Computer Science at the same university. Her research interests include data mining, spatiotemporal and mobility databases, and semantic trajectory modelling and analysis.

Esteban Zimányi is a professor at the Université libre de Bruxelles (ULB). He started his studies at the Universidad Autónoma de Centro América, Costa Rica, and received a Master in Computer Science degree and a doctorate in computer science from ULB. His current research interests include spatiotemporal and mobility databases, data warehouses, and geographic in formation systems. He has coauthored and coedited several books and published many papers on these topics. He was editor-in-chief of the Journal on Data Semantics (JoDS) published by Springer from 2012 to 2020. He is the coordinator of the Erasmus Mundus master's and doctorate programs "Information Technologies for Business Intelligence" (IT4BI) and "Big Data Management and Analytics" (BDMA), and the Marie Skłodowska-Curie doctorate program "Data Engineering for Data Science" (DEDS). He is a main contributor and a co-founder of MobilityDB.

Marta Simeoni received her PhD in Computer Science in 2000 from the University of Rome "La Sapienza". Since 2000 she is assistant professor at Ca' Foscari University of Venice. Her current research interests include bioinformatics and modelling and analysis of biological and ecological systems. Recent research interests include also modelling and analysing semantic trajectories.

Davide Rocchesso received the Ph.D. degree from the University of Padova, Italy in 1996. He is professor of computer science at the University of Milan, Italy. He was the coordinator of EU projects SOb (the Sounding Object) and SkAT-VG (Sketching Audio Technologies using Vocalizations and Gestures). He had been chairing the COST Action on Sonic Interaction Design. His main research interests are sound modelling and synthesis, interaction design, evaluation of multisensory interactions.

Alessandra Raffaetà MSc (1994) and PhD (2000) in Computer Science, University of Pisa - is an Associate Professor at Ca' Foscari University of Venice (Italy). Her research interests include Data warehouses, GISs, spatiotemporal reasoning, design and formal semantics of programming languages and mobility data analytics. She published over 70 papers on international journals and conferences. She was member of the program committee of several international conferences and she participated to several national and international research projects.

Authors and Affiliations

Giulia Rovinelli[1] · Esteban Zimányi[2] · Marta Simeoni[1,3] · Davide Rocchesso[4] · Alessandra Raffaetà[1]

✉ Giulia Rovinelli
giulia.rovinelli@unive.it

Esteban Zimányi
esteban.zimanyi@ulb.be

Marta Simeoni
simeoni@unive.it

Davide Rocchesso
davide.rocchesso@unimi.it

Alessandra Raffaetà
raffaeta@unive.it

1 Ca' Foscari University of Venice, Venice, Italy

2 Université libre de Bruxelles, Brussels, Belgium

3 European Centre for Living Technology (ECLT), Venice, Italy

4 Università degli studi di Milano Statale, Milano, Italy