Intelligent Onboard Routing in Stochastic Dynamic Environments using Transformers

Rohit Chowdhury
Indian Institute of Science, Bangalore
Bangalore, India
rohitc1@iisc.ac.in

Raswanth Murugan
Indian Institute of Technology, Palakkad
Palakkad, India
raswanth99@gmail.com

Deepak Subramani
Indian Institute of Science, Bangalore
Bangalore, India
deepakns@iisc.ac.in

ABSTRACT

Autonomous marine agents find extensive applications in environmental data collection, naval security, and exploration of harsh ocean regions. As intelligent agents, they must perform onboard routing, collect data about their surroundings and update their route to minimize mission travel time, energy, or data collection. While Markov Decision Processes (MDPs) and Reinforcement Learning (RL) are often used for path planning, they are computationally expensive for onboard routing as they need in-mission re-planning. In the present paper, we develop a novel, deep learning method based on the decision transformers for optimal path planning and onboard routing of autonomous marine agents. The transformer architectures convert the RL-based optimal path planning problem into a supervised learning problem via sequence modeling. Before the mission, during the offline planning phase, the environment is first modeled as a stochastic dynamic ocean flow with dynamically orthogonal flow equations. A training dataset for the transformer model is created by solving the stochastic dynamically orthogonal Hamilton-Jacobi level set partial differential equations or a dynamic programming solution for MDPs. These paths are then processed to obtain sequences of states, actions and returns for our transformer models, where the agent's state is typically its spatio-temporal coordinate and other collectible data. We propose and analyze multiple state modeling choices against the agent's state estimation capabilities and scenarios with multiple target locations. We demonstrate that *(i)* a trained agent learns to infer the surrounding flow and perform optimal onboard routing when the agent's state estimation is accurate,*(ii)* specifying the target locations (in case of multiple targets) as a part of the state enables a trained agent to route itself to the correct destination, and *(iii)* a trained agent is robust to limited noise in state transitions and is capable of reaching target locations in completely new flow scenarios. We extensively showcase end-to-end planning and onboard routing in various canonical and idealized ocean flow scenarios. We analyze the predictions of the transformer models and explain the inner mechanics of learning through a novel visualization of self-attention of actions and states on the trajectories.

KEYWORDS

Path Planning; Onboard Routing; Decision Transformers; Reinforcement Learning; Supervised Learning; Robot Planning and Control

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

ACM Reference Format:
Rohit Chowdhury, Raswanth Murugan, and Deepak Subramani. 2023. Intelligent Onboard Routing in Stochastic Dynamic Environments using Transformers. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 9 pages.

1 INTRODUCTION

Autonomous marine vehicles are often tasked with critical missions in harsh and unknown ocean regions. For tasks such as ocean data collection, monitoring underwater assets, and tracking the movement of adversaries in naval security, optimal operation of the autonomous agents is necessary to reduce costs. In some missions, it is sufficient for the agent to follow a pre-determined sequence of waypoints if an accurate deterministic forecast of the environment is available. For stochastic situations, when the uncertainty of the ocean is well quantified, the agent can be made to follow a pre-computed risk optimal path or an optimal in expectation policy that is computed from a model of the environment and the Hamilton Jacobi equations [18, 26, 27] or a Reinforcement Learning algorithm [7, 34], or a Markov Decision Process algorithm [5, 6, 16, 20]. However, in most applications, autonomous agents are required to follow pre-computed paths and perform intelligent onboard routing by taking decisions based on real-time in-mission measurements of the environment or their position on the fly. A key challenge for such agents is that they are easily advected by strong, dynamic, and uncertain ocean currents. Therefore, fast and scalable onboard routing algorithms for real-time applications and accurate environment modeling techniques are imperative for successful path-planning operations.

This paper considers time-optimal path planning and onboard routing of an autonomous marine agent in a stochastic, dynamic ocean environment. Various approaches can tackle this problem (see a detailed survey in [29]). For example, there are path planners based on Dijkstra's algorithm [25], variants of A* [24, 31] and Delayed D* [10]. Despite working well in deterministic settings, their Monte Carlo versions are computationally inefficient. Stochastic Hamilton-Jacobi equation-based methods that give a distribution of paths have been developed for path planning for single objective missions in uncertain environments [17]. Recently, path planners based on Markov Decision Process (MDP) [16] and Reinforcement Learning (RL) [1, 7] have become popular, albeit slow and computationally expensive for large-scale, realistic and real-time applications. However, onboard routing with all the above planners is still computationally intensive, requiring efficient data

assimilation schemes and re-computing optimal policies during the mission.

Deep learning-based approaches with transformers [4, 13] have yielded impressive results in learning policies in various deterministic but complex environments. In the present work, we propose an onboard routing algorithm that exploits some of the important properties of these transformer-based deep learning architectures and develop a novel framework for using them in stochastic dynamic environments. In addition, we develop a framework that enables the agent to make intelligent real-time decisions based on its state estimate.

1.1 Problem Statement

Let us consider a spatiotemporal domain as shown in Figure 1, where $\mathbf{x} \in \mathbb{R}^n$ denotes space ($n = 2, 3$ for 2, 3-D space) and $t \in [0, \infty)$ denotes time. Let $\mathbf{V}(\mathbf{x}, t; \omega)$ be a stochastic, dynamic flow in the domain that strongly advects the agent, where ω represents a realization of the uncertain flow field. At $t = 0$, the autonomous agent is at its initial position $\mathbf{x}_0$ and must travel to a pre-specified target location $\mathbf{x}_f$ among multiple possible targets. At any given time t, the agent must move relative to the flow by taking action a according to an optimal policy $\pi(\mathbf{x}, t|\mathbf{x}_f, y(t))$, while simultaneously being advected by the instantaneous flow velocity $\mathbf{V}$. Here $y(t)$ denotes the agent's trajectory up to time t. However, the true realization of the velocity field is unknown to the agent before the mission and can potentially be estimated during the mission from its trajectory history. Hence, the autonomous agent must intelligently utilize its traversed trajectory and route itself optimally to reach the pre-specified target location $\mathbf{x}_f$ while minimizing travel time.

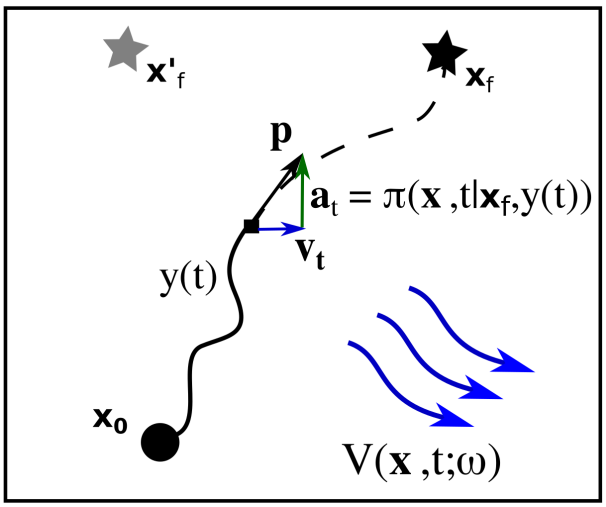

Figure 1: Schematic of the onboard routing problem: The autonomous agent must travel from $\mathbf{x}_0$ to $\mathbf{x}_f$ in a stochastic dynamic velocity field $\mathbf{V}(\mathbf{x}, t; \omega)$ in the shortest time possible. Let $\mathbf{y(t)}$ be the path traveled. The agent must take optimal actions $a_t = \pi(\mathbf{x}, t|\mathbf{x}_f, \mathbf{y(t)})$ while simultaneously being advected by the flow $\mathbf{v} = \mathbf{V}(\mathbf{x}, t; \omega)$. The agent's resultant motion is along $\mathbf{p} = \mathbf{v} + \mathbf{a}_t$

.

1.2 Previous Progress on Deep Learning-based Optimal Path Planning

Over the recent years, deep learning methods have gained popularity for solving path planning problems [33, 35]. Specifically, transformers [30] have been very successful in multiple supervised learning applications, especially sequence modeling tasks such as natural language translation. Recently, transformer-based architectures have been successfully used for motion planning applications. A motion planning transformer that uses the attention mechanism to predict regions in a static, deterministic environment where valid paths may exist has been proposed [14]. However, the actual path planning is done using a sample-based planner in the predicted region. Transformers have been used to learn and predict optimal value functions given a static map with obstacles and a goal point [3]. The actions are then computed using the predicted value function for a new planning problem. However, the environment here is deterministic, and creating training data for large problems with stochastic fields is intractable. [15] used transformers to solve the traveling salesman problem, where it learns to predict the sequence of nodes that must be visited as a stochastic policy. However, it is trained using a reinforcement learning algorithm, which is a non-trivial exercise. Transformers have also been used as a policy network to learn heuristics for routing problems [2, 32]. However, these methods are better suited to combinatorial optimization problems with deterministic environments.

The scene memory transformer [9] stores the observations in memory, embeds them, and feeds the embeddings into the transformer encoder. The decoder outputs actions. This transformer, which is effectively a memory-based policy network, is trained using deep Q-learning [19] and is therefore limited by the traditional RL paradigm. Recently, decision transformers [4] and trajectory transformers [13] have been proposed to model the offline RL problem as a sequence modeling problem. The key idea in this paradigm is to view an agent's experiences, which is a sequence of states, actions, and rewards, as the transformer's input sequence, and return action sequences as output. This framework uses well-studied, high-capacity sequence model architectures like GPT-2 [22] and may benefit from their good scaling and generalizing capabilities.

The main contribution of the paper is the development of a novel transformer-based planning and onboard routing algorithm by *(i)* creating expert datasets that embody optimal or near-optimal behavior in stochastic, dynamic oceanic environments, thereby enabling the attention mechanism in the decision transformer to learn significant associations across states, actions and returns; *(ii)* training a decision transformer model for path planning and onboard routing with capabilities to learn the optimal actions to be performed in completely unseen scenarios; and *(iii)* a novel visualization of the attention mechanism on the trajectory itself giving insights into the representation learning.

In what follows, we discuss individual components of our modelling framework and necessary theoretical background (Sec. 2). Then, we develop our transformer-based planning and onboard routing algorithm (Sec. 3). In Sec. 4, we demonstrate our algorithm on canonical and idealized ocean flow scenarios and discuss the reasons behind its effective performance. Finally, Sec. 5 discusses future research directions.

2 MODELLING FRAMEWORK AND BACKGROUND THEORY

Our onboard routing algorithm for autonomous marine vehicles requires a stochastic ocean flow modeling system and an algorithm to provide an ensemble of exact time-optimal paths in an ensemble of flow scenarios for the supervised training of the transformer model. The following subsections discuss these two systems. We also describe the background theory of transformer models.

2.1 Probabilistic Ocean Modeling and Simulation System

We use the Dynamically Orthogonal (DO) barotropic QuasiGeostrophic (QG) stochastic flow equations to forecast an idealized ocean velocity field [17, 28]. Further, realistic 4D (3D in space and 1D in time) stochastic primitive equation simulations can also be utilized. In the Langevin form, the stochastic barotropic QG dynamics may be represented as a set of equations describing the conservation of mass, momentum, and energy [27]. We decompose the stochastic dynamic velocity field using the DO expansion and get explicit equations for its DO mean, modes, and coefficients (see, e.g., [23]). The DO method offers a significant computational advantage (2–4 orders of magnitude) over naive Monte Carlo simulations by carrying the uncertainty evolution in an adaptive and dynamic stochastic subspace that is solved using the DO mode equations. Alternatively, Polynomial Chose Expansion [21] or simple ensemble modeling can also be used to model the probabilistic ocean flow.

2.2 Hamilton-Jacobi level set PDE

To obtain a set of exact time-optimal paths in the stochastic flow field, we employ the DO stochastic scalar Hamilton-Jacobi-Bellman (HJB) level-set partial differential equations (PDEs) [27] that govern the stochastic reachability fronts. Here, the reachability front is defined as the set of all points the agent p can reach when starting from $\mathbf{x}_0$ at $t = 0$ in the flow $\mathbf{v}(\mathbf{x}, \mathbf{t}; \omega)$. The optimal travel time $T(\mathbf{x}_f; \omega)$ is the first time instance that the reachability front reaches $\mathbf{x}_f$ and the optimal path $\mathbf{X}_P(\mathbf{x}_0, t; \omega)$ is computed for every ω by solving the particle backtracking equation.

2.3 Modelling the underlying MDP

The underlying MDP of the offline path planning RL problem is defined by the tuple $< \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma >$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $\mathcal{P}$ is the matrix of state transition probabilities, $\mathcal{R}$ is the one-step rewards, and γ is the discount factor. For our problem, we choose to model the state $s \in \mathcal{S}$ as $s = (t, x, y)$ (or alternatively $s = (t, \mathbf{x})$, where $t \in \{0, 1, 2, .., T\}$ denotes the time step , $\mathbf{x} = (x, y) \in \mathbb{R}^2$ denote the spatial coordinate. Additionally, the local velocity measurement $\mathbf{v} = (u, v) \in \mathbb{R}^2$ observed at the spatio-temporal coordinate (t, x, y) may also be included in the state if required. The action space $\mathcal{A} = [0, 2\pi)$, where an action $a \in \mathcal{A}$ denotes the agent's heading, as it moves with a constant speed F, relative to the flow. While the state transition probabilities $\mathcal{P}$ are induced by the stochastic flow field, we need not compute it explicitly. The rewards $\mathcal{R}$ are defined to minimize travel time, with a one-step reward r_t that gives the agent a large positive reward r_{term} if it reaches the target state and a large negative penalty $r_{outbound}$ if it goes out of bounds of the domain. Mathematically,

$$r_t(s, a, s') = \begin{cases} r_{term} & s' \in \mathcal{S}_{targ} \\ r_{outbound} & s' \in \mathcal{S}_{penalty} \\ -\Delta t & o.w. \end{cases} \tag{1}$$

where s' is the next state to which the agent transitions, based on the environment dynamics captured in the state transition probabilities $\mathcal{P}$, $\mathcal{S}_{targ} = \{s = (t, \mathbf{x}) | \quad s \in \mathcal{S}, \quad ||\mathbf{x} - \mathbf{x_f}||^2 < \epsilon\}$, $\mathcal{S}_{penalty}$ is the set of all states, whose coordinates lie outside the planning domain. Lastly, the discount factor $\gamma = 1$, since we are modeling the problem as a finite-horizon MDP.

2.4 Transformers for offline Reinforcement Learning

The causally masked self-attention mechanism is the key ingredient of a transformer that allows it to attend to the most important parts of the input and output sequences and draw dependencies between them. A simplified summary is as follows. Let a sequence consists of K tokens, which are first projected to an embedding space of a fixed dimension through multiple transformations. Let these transformed tokens be $(w_1, w_2, ..w_K)$. Then, the similarity score α_{ij} of each transformed token w_i for $i = 1, 2, ..K$ with the previous tokens $(w_1, w_2, ..w_i)$ is computed through a scaled dot product operation, leading to a lower triangular matrix of attention weights. The output of the attention mechanism is a sequence of context-aware tokens (or vectors) $(w'_1, w'_2, ..w'_K)$, where $w'_i = \sum_{j=1:i} \alpha_{ij} w_j$. The attention mechanism is causally masked because each token attends only to the previous tokens in the sequence. Finally, the context-aware sequence is passed through a feed-forward layer to get the desired output. We refer the reader to [30] for further details.

Offline RL is a paradigm that involves learning optimal policies using data from previously collected interactions with the environment or a simulator. Hence, policies are learned from a fixed dataset rather than online experiences. Recently, decision transformers [4] were introduced, which allows solving offline RL problems as a sequence modeling problem using the transformer architecture. The key idea is to treat trajectories of experience from a given dataset as an input sequence of the form $(R_1, s_1, a_1, R_2, s_2, a_2, ...R_T, s_T)$, where s_i, a_i and R_i are the state, action, and returns-to-go at the ith timestep and converted to embeddings, which in turn are processed by an autoregressive transformer model. The model is trained to predict the next action given the previous tokens. During inference, a target return value and start state are specified, and the model iteratively predicts an action given all the previous returns-to-go, states, and actions. Fig 2 shows a schematic of the decision transformer architecture with the causal self-attention mechanism. We refer the reader to [4] for further details. Decision transformers have proven to be very successful in various environments, capable of learning better policies than even the behavior policy which was used to collect the data.

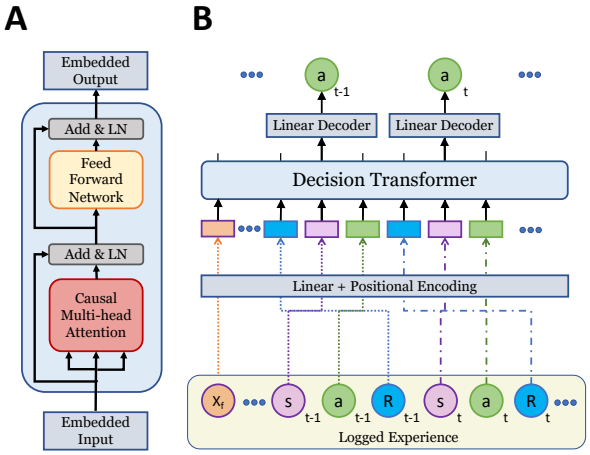

Figure 2: (A) The transformer encoder (B) The decision transformer architecture with an initial target token

3 PLANNING AND ONBOARD ROUTING WITH TRANSFORMERS

Development and training of our planning algorithm involve the following steps: *(i)* Obtaining the stochastic dynamic environment flow forecast with an ensemble of form N_r realizations of the flow field; *(ii)* computation of the distribution of N_r exact time-optimal paths in this N_r realizations; *(iii)* extracting the training sequences for the decision transformer; *(iv)* training the decision transformer with hyper-parameter tuning and *(v)* inference in new flow situations.

The sequence modeling nature of the decision transformer provides a natural framework for onboard routing as the agent's choice of action is conditioned on all previous states and actions, where the state consists of the trajectory so far. Since the agent knows its speed, this information effectively encompasses the local velocity field information as well. This allows the agent to infer the most likely realization of the environment and act accordingly without the need for expensive onboard computations of data assimilation schemes.

3.1 Creating the database

To train the decision transformer, we first need to prepare appropriate datasets. The dataset preparation involves the following phases:

3.1.1 Log data from a behavioral policy. To learn without exploring the environment, offline RL algorithms need logged data of environment interactions previously collected by the agent (or other similar agents) executing a behavior policy. If an environment model is available, these interactions may also be logged from simulations. We efficiently solve the exact time-optimal paths in the flow using its forecast and the HJB level set PDEs discussed (Sec. 2.2) and obtain a distribution of trajectories as shown in Fig. 3. Note that the data can be logged from any other behavioral policy as well, and the decision transformer can be trained. A novelty of the current work is the use of a distribution of the exact time-optimal paths obtained from the HJB equations for training.

We also note that the HJB time-optimal path is obtained for a given realization of the stochastic velocity field. Hence, theoretically, if the knowledge of the realization is available to the agent, then it can simply follow the sequence of waypoints that constitute the time-optimal path. However, in reality, the agent rarely has prior knowledge of the *true* realization of the velocity field and can only gather this knowledge through in-mission environment observations. Hence, optimal onboard routing in such scenarios is a complex, non-trivial task that requires efficient, intelligent, and generalizable algorithms.

3.1.2 Extract experiences from the logged data. First, the time-optimal trajectories from the HJB PDEs, which are in the form of a sequence of waypoints, are converted to trajectories of experiences in the MDP framework. The time-optimal PDE path for realization j is given by the sequence $\tau_{HJ}^{(j)} = (\mathbf{x_k}^{(j)})_{k=1:n_d}$, where $\mathbf{x_k}^{(j)} = (x_k, y_k)$ denotes the spatial coordinates in the defined domain and n_d is the number of waypoints that constitute the path $\tau^{(j)}$. For each $j \in \{1, 2, .., N_r\}$, we initialize the simulator environment with the agent at the start location in the field $\mathbf{v_0^{(j)}} = V(0, \mathbf{x_0}; \omega_j)$. Next we compute the action a_0 that the agent must perform at s_0 such that it reaches the furthermost possible point $\mathbf{x_{k'}}^{(j)}$ $(k' > k)$ in the trajectory $\tau_{HJ}^{(j)}$. In doing so, the agent transitions to the state $s_1 = (\Delta t, \mathbf{x_1})$ and collects a one-step reward r_0. This process continues until the agent reaches a terminal state, and the episode terminates. Consequently, for each j, we obtain the experience trajectory $\tau^{(j)} = (s_0, a_0, r_0, s_1, a_1, r_1, ...s_T)$.

3.1.3 Create a training dataset for the decision transformer. Next, the experience trajectories are converted into a dataset of sequences as expected by the decision transformer. First, for each $\tau^{(j)}$, the returns-to-go at each timestep are computed as $R_t = \sum_{k=t:T} r_k$. This associates states and actions with long-term returns instead of the one-step reward. Then the states are normalized to have zero mean and unit standard deviation. Additionally, the sequence length of the data fed to the transformers, called the context length, must be fixed. For our problem, we set the context length for the decision transformer to be greater than the longest trajectory in the dataset. Consequently, the input sequence to the decision transformer is defined as $\tau_{DT,ip}^{(j)} = (\mathbf{x_f}, R_0, s_0, a_0, R_1, s_1, a_1, ..R_T, s_T, tok_p, tok_p, ..tok_p)$, where tok_p are padding tokens, $\mathbf{x_f}$ is the target location added as the first token in the sequence. Since the decision transformer learns autoregressively, the target sequence for training is a masked version of the input sequence $\tau_{DT,targ} = (m, m, a_0, m, m, a_1, m, m, a_2, ..a_{T-1}...)$. Hence, the agent learns to predict the action a_i given all previous returns, states, and actions in the sequence. Finally, the dataset $(\tau_{DT,ip}^{(j)}, \tau_{DT,targ}^{(j)})_{j=1:N_r}$ is split into training and testing datasets.

3.2 Training and inference with the decision transformer

The model is trained on the training dataset with the loss function as the mean squared error between the predicted and target actions,

$$\mathcal{L}(\theta) = \frac{1}{n_m} \sum_{j=1:n_m} \sum_{i=0:T-1} ||a_{i,targ}^{(j)} - a_{i,pred}^{(j)}(\theta)||^2, \qquad (2)$$

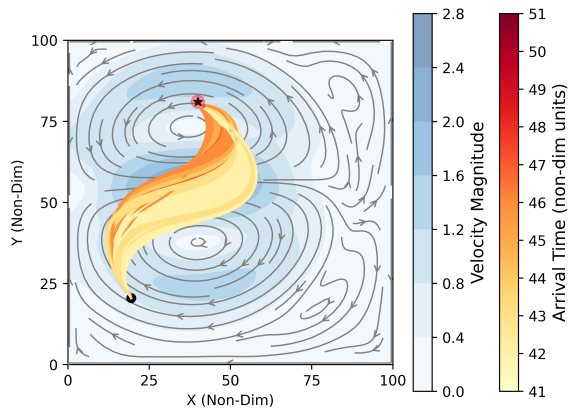

Figure 3: Distribution of time-optimal paths from Hamilton Jacobi Bellman level set PDEs: The dataset contains trajectories corresponding to $N_r = 5000$ realization of the flow field, colored by its arrival time. These trajectories are further processed to extract experience sequences in the MDP framework. The background shows the flow field (vector and magnitude) of a typical realization as an illustration of the flowfield encountered by the agent.

where θ denotes the network weights, n_m is the batch size, $a^{(j)}_{i,targ}$ is the sequence of actions extracted from $\tau^{(j)}_{DT,targ}$ and $a^{(j)}_{i,pred}$ is the sequence of actions predicted by the model. Note that we are using a continuous action space. During training, its performance is evaluated by monitoring the average returns across the test set at regular intervals. The model state is saved at the point where the average return across the test set is maximum for future inference. The model's learnable weights (θ) are updated using the AdamW optimizer [12]. Hyper-parameters like embedding dimension, number of blocks, number of attention heads etc., are tuned via a grid search method. Once we have a trained model, the agent has effectively learned a policy (or plan) π, which can be readily executed in new environment realizations, making the agent intelligent and capable of navigating autonomously to the target location.

The agent performs inference when it is set into an environment realization about which it has no prior knowledge. This may happen either in simulation or during real-time missions. During inference, the first action predicted by the model is conditioned on the target state, target returns-to-go, and initial state s_0. During the mission, the agent may utilize onboard sensors like an Acoustic Doppler Current Profiler (ADCP) or another sensor and a dead-reckoning system to log its location. The agent uses this information and takes action a_0 predicted by running the sequence $(\mathbf{x_f}, R_0, s_0)$ through a forward pass of the trained model. Consequently, the agent transitions to the next state, adds the information to the state definition of s_1, and notes the one-step reward based on Eq. 1. The target return at the next time-step is simply obtained by subtracting the one-step reward from the previous return, i.e. $R_1 = R_0 - r_0$. The action a_0 is now appended to the input sequence and the model can now predict a_1 conditioned on the sequence $(\mathbf{x_f}, R_0, s_0, a_0, R_1, s_1)$. This process allows the agent to perform actions based on prior trajectory observations. Algorithm 1 summarizes the overall procedure discussed in this section.

Algorithm 1: Onboard routing algorithm

Input: *env_data, prob_params*
Output: π

/* Create dataset */
$\{\mathbf{V}(t, \mathbf{x}; \omega_j)\}_{j=1:N_r} \leftarrow$ DO_SPDE_solve();
for j in range$(0, N_r)$ **do**
  $\tau^{(j)}_{HJ} \leftarrow$ HJ_level_set_solve$(\mathbf{V}(t, \mathbf{x}; \omega_j), \mathbf{x_0}, \mathbf{x_f})$;
  env $\leftarrow$ initialize_MDP_env$(\mathbf{V}(t, \mathbf{x}; \omega_j), \mathbf{x_0}, \mathbf{x_f})$;
  $\tau^{(j)} \leftarrow$ env.extract_experience$(\tau^{(j)}_{HJ})$;
  $(\tau^{(j)}_{DT,ip}, \tau^{(j)}_{DT,targ}) \leftarrow$ create_DT_dataset$(\tau^{(j)})$;
end for
/* Train model */
for $(\tau_{DT,ip}, \tau_{DT,targ})$ in dataloader **do**
  $a_{preds} \leftarrow$ decision_transformer$(\tau_{DT,ip})$;
  $a_{targ} \leftarrow$ extract_actions$(\tau_{DT,targ})$;
  $\mathcal{L} \leftarrow$ MSE(a_{preds}, a_{targ});
  optimizer.update_weights$(\mathcal{L})$;
end for
/* Inference or model evaluation */
$R \leftarrow R_0$;
done $\leftarrow$ False
$\tau_{DT,in} \leftarrow (R, s_0)$;
while not done **do**
  $a \leftarrow$ decision_transformer$(\tau_{DT,in})$;
  $s', r,$ done $\leftarrow$ step(a);
  $R \leftarrow R - r$;
  $\tau_{DT,in}$.append(a, R, s');
end while

4 APPLICATIONS

We demonstrate our path planning and onboard routing algorithm with various flow scenarios derived from a canonical flow called the highway (HW), and an idealized ocean flow called the double gyre (DG) flow field. The spatial extent is considered on a square-shaped domain of size 100x100 non-dimensional units. The highway is confined to exist between 40 and 60 in the y-direction. The HW flow field is a stochastic static flow field consisting of a band or highway of ocean current flowing left (LHW; east to west) or right (RHW; west to east). The double-gyre (DG) flow field is a stochastic dynamic flow field obtained by solving the DO quasi-geostrophic equations [23]. Such wind-driven double-gyre flows are frequently observed in mid-latitude oceanic regions such as the Northwest Atlantic Ocean (Gulf Stream and eddies) [8, 11]. These flow fields are typically used to demonstrate path planning algorithms [17]. See Fig. 4 to visualize the flow field.

The planning is performed on a spatio-temporal grid which is continuous in space and discrete in time. The spatial domain is bounded within the space $x \in [0, 100], y \in [0, 100]$. Based on the MDP formulation of the problem (see Sec. 2.3), the temporal coordinate $t \in \{0, 1, 2..., 120\}$, with $\Delta t = 1$, $r_{term} = 100, r_{outbound} = -100$, and $\epsilon = 2$. Here, the ranges of t, x, y and Δt are in non-dimensional units. The decision transformer is trained on the training set described in Sec. 3.2. The model is evaluated at regular intervals of

training iterations by performing inference with it on the 500 unseen realizations of the validation set while monitoring the average return obtained by the agent across the validation set.

We performed multiple experiments with results shown in Fig. 4 and summarized in Table 1. For each scenario, the flow is visualized in the background with blue colour and grey streamlines. The starting location is marked with a black circular marker and the target locations are marked with a star marker within a red circle. Each scenario shows two panels, the left panel shows the trajectories from the test set and the right panel shows the trajectories predicted on this test set by our trained model. Table 1 provides numerical details for the experiments conducted in each scenario. It also provides information on the training and testing scenarios and the expected arrival times (EAT) with 1 standard deviation for the validations trajectories and our models.

In scenario S1, the agent was trained and tested on a double gyre (DG) flow scenario. Although small errors in predicted actions can significantly affect arrival times due to the strong flow field, the trained model performs nearly as well as the ground truth trajectories.

In scenario S2, we have a mixed highway (mixed HW) flow field. Half the realizations of this flow is LHW, and the other half is RHW. Consequently, the trajectories also initially lean towards the left or right depending on the particular realization. The agent has been trained and tested on mixed HW flow distribution samples in this scenario. Here we see that our model achieves the same EAT (up to round-off errors) as the ground truth.

It is clear from scenarios S1 and S2 that the transformer models have learned good policies and can predict impressive action sequences across different flow realizations. We believe this performance is because the model infers the correct flow realization from the history of states and action sequences. In other words, if an agent performs an action $\mathbf{a}$ in a state $\mathbf{s}$ and reaches the state $\mathbf{s}'$, then it is straightforward to compute the local flow velocity $\mathbf{v}$ that led to the transition, assuming the agent's state estimation is accurate. Since the transformer models learn to predict actions conditioned on the trajectory history, they are essentially learning how to behave in a given realization. Consequently, our transformer-based agent has performed well in these scenarios.

In scenarios S3, we introduce 2 target locations, $\mathbf{x_{fl}}$ and $\mathbf{x_{fr}}$. The agent is put in a mixed HW flow such that it must reach $\mathbf{x_{fl}}$ in RHW realizations and $\mathbf{x_{fr}}$ in LHW realizations. For each realization, the agent has a-priori knowledge of its specific target location, given to it as the target token. The agent is trained and tested on samples from the same distribution. We observe that the agent does just as well as the ground truth on the test set.

In scenario S4, we have 2 target locations corresponding to two identical sets of RHW flows. Each flow realization leads to two different trajectories, one for each target. Hence, the agent's ability to infer the flow realization is insufficient to reach the desired target unless it is explicitly provided to the agent. We observe that a trained agent can navigate correctly and optimally to the desired target location when given a target token (the first token in our input sequence). In the ablation study, we used a sequence without the target token, and found that model did not do well in this task.

For both scenarios S3 and S4, we have conducted experiments using a model without a target token and observed that the agent learns to proceed to the closest target, i.e., the one which minimizes the travel time. The corresponding plots for an agent without the target token are not shown here for brevity.

In scenario S5, we have a mixed HW flow with one target location. However, here the agent starts its mission outside the highway region. This makes an interesting case because HJB-level solutions that constitute the dataset were computed assuming full knowledge of the flow field. Hence, these trajectories either move left or right from the initial timestep itself to reach the target in minimum time. However, our transformer-based agent relies on the executed trajectory to infer the realization of the flow field, which is initially unavailable in this scenario. Moreover, it has been trained on roughly an equal number of RHW and LHW realizations. As a result, the agent takes an average action and goes straight up and reaches the highway. Another interesting point is that by the time the agent reaches the target, its actions are now conditioned on a trajectory it has never seen in the training dataset. Despite this, the agent manages to predict an impressive sequence of actions and reaches the target. It becomes visually evident when we compare the predicted trajectories post-highway impact with trajectories in scenario S2.

In scenario S6, we present a mixed flow condition containing an equal number of realizations from DG and LHW. Moreover, to demonstrate that our transformer-based agent can learn from various kinds of expert datasets, the trajectories for the DG realizations have been computed by solving the formulated MDP through dynamic programming (DP) with an off-the-shelf end-to-end GPU accelerated solver [6]. The trajectories for the LHW realizations were derived from the HJB level set solver. We observe that our transformer model outperforms the expert trajectories. A possible reason for the same in this scenario is that DP for DG realizations are optimal only in expectation and not for each realization exclusively. Hence, there is potential for the transformer-based model to better than the DP paths.

In scenario S7, we present a mixed HW flow scenario with 1 target. To analyze the agent's performance in significantly different unseen environments, the agent has been trained only on expert trajectories corresponding to the LHW realizations and is tested exclusively on RHW realizations. Since the test dataset only contains RHW realizations, Fig. 4-S7 may appear like an RHW flow scenario but is indeed a mixed HW flow. We observe that our transformer-based agent successfully reaches the target location for all realizations, albeit with a larger EAT, even though it was never trained on such realizations. Our observations in scenarios S2 and S7 imply that transformer-based agent has the potential to extrapolate reasonably well to unseen environments.

Finally, in scenario S8, we analyze the robustness of the transformer's learned representation by adding noise to the state transitions during testing. This is equivalent to having a noisy flow field that deviates from the modeled flow field during the mission. We simulate the noisy flow field by adding Gaussian noise with mean=0 and variance=0.2 in the state transition. The flow used for the simulation is a mixed HW flow with one target. The agent is trained on non-noisy expert trajectories for the mixed flow and tested on a noisy mixed flow. Consequently, the agent is conditioned on relatively new trajectories during testing and is still capable of reaching the target with a slightly higher EAT. Multiple experiments were

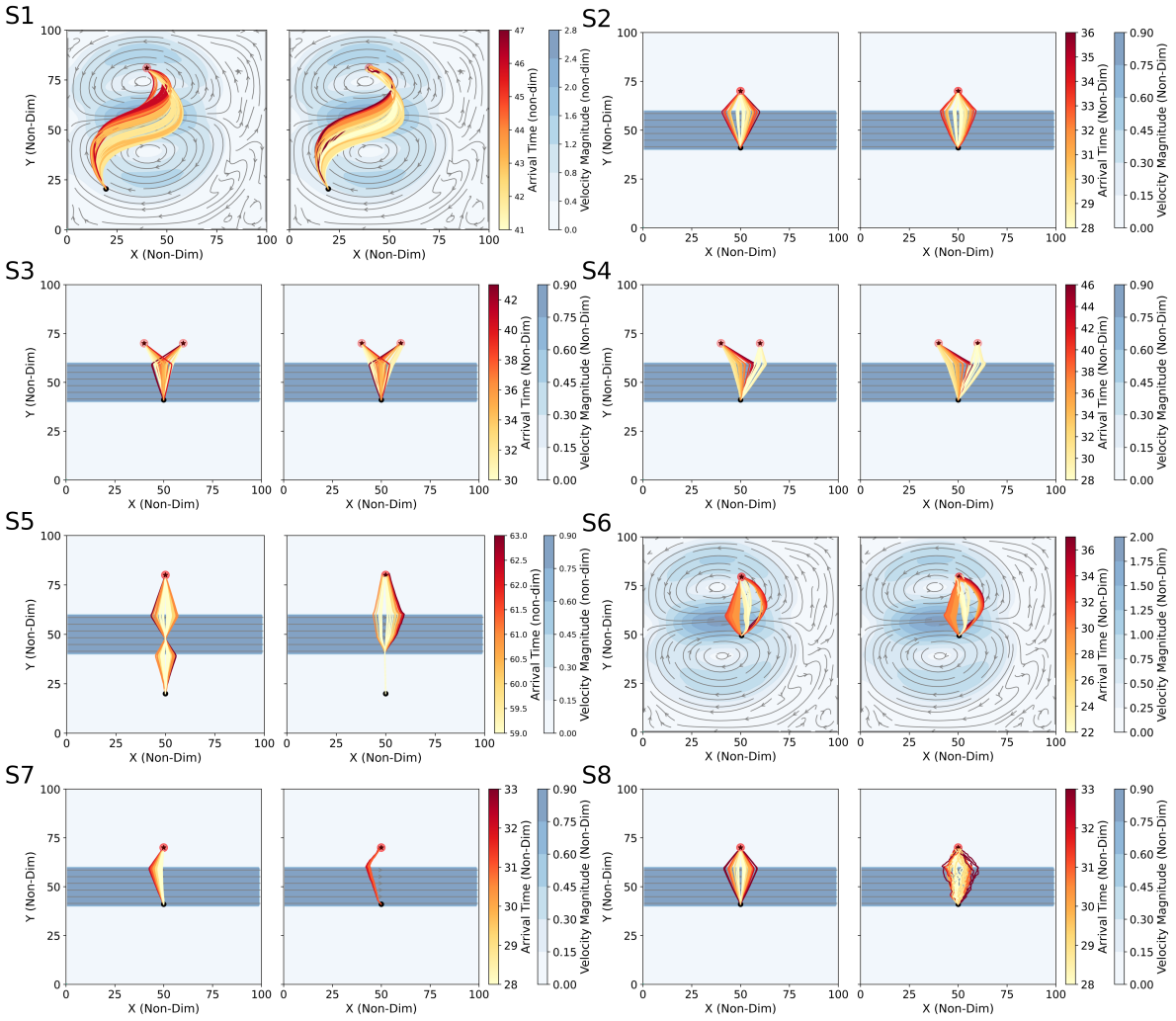

Figure 4: Inference on the test set for eight different scenarios summarized in Table. 1. The ground truth paths are on the left and the predicted paths are on the right. Each path is colored by its arrival time and the background shows a typical test flow field as an illustration.

done in this scenario with varying amounts of noise. As expected, with increasing noise, the EAT increases, and the success ratio (percentage of target hits) decreases.

We also note that our proposed method is computationally cheaper during inference compared to solving a HJ level-set PDE. A forward pass on our trained transformer takes just 1 ms, whereas solving the HJ level-set PDE for a given environment realization takes 4s. The final transformer architecture after hyper-parameter tuning has one encoder block, context length = 70, number of attention heads = 4, embedding dimension = 8, with the transformer's feedforward block defined as a sequence of $\langle \text{Linear}(C, 4C), \text{GELU}(), \text{Linear}(4C, C), \text{Dropout}(p) \rangle$ layers, where $C = 32$, projection dropout, $p = 0.1$ and GELU stands for Gaussian Error Linear Unit. The learning rate is 10^{-4} and batch size is 64.

4.1 Visualizing attention weights

To explain why our algorithm performs well, we analyzed the attention weights of a trained decision transformer model. We investigate which prior actions $(a_i)_{i=0:t-1}$ and states $(s_i)_{i=0:t-1}$ in the sequence receive relatively higher attention scores when predicting action a_t. We do so by visualizing the attention scores in two ways: first, through a heatmap of the attention matrix; second, by projecting these attention weights as colors on the predicted trajectories at different times as shown in Fig. 5 for Scenario 1.

Fig. 5 left panel shows the attention scores that constitute the attention matrix of the top-most encoder block of a trained decision transformer when used for inference on a given test realization. Since we have set the context length to 60 and there are 3 tokens (R_t, s_t, a_t) per timestep in addition to the target token, the matrix size is (181×181). Hence, rows and columns with index of the form $3t + 1$, $3t + 2$ and $3t + 3$, correspond to R_t, s_t, a_t, respectively,

Table 1: Summary of experiments

Scenario	Flow		Locations		EAT	
	Train	Test	$\mathbf{x_0}$	$\mathbf{x_f}$	π_{expert}	DT
S1	DG	DG	(20,20)	(40,80)	43.7 ± 1.5	44.4 ± 1.7
S2	mixed HW	mixed HW	(50,41)	(50,70)	30.2 ± 2.3	30.1 ± 2.1
S3	mixed HW	mixed HW	(50,41)	[(40,70),(60,70)]	34.1 ± 3.4	34.1 ± 3.4
S4	RHW	RHW	(50,41)	[(40,70),(60,70)]	32.0 ± 4.6	32.1 ± 4.7
S5	mixed HW	mixed HW	(50,20)	(50,80)	59.0 ± 0.4	61.0 ± 2.0
S6	DG(DP)+ LHW	DG(DP) + LHW	(50,41)	(40,80)	28.5 ± 3.7	28.4 ± 3.7
S7	LHW	RHW	(50,41)	(50,70)	29.3 ± 1.3	31.5 ± 0.5
S8	mixed HW	mixed HW + noise	(50,41)	(50,70)	29.6 ± 1.6	30.0 ± 1.9

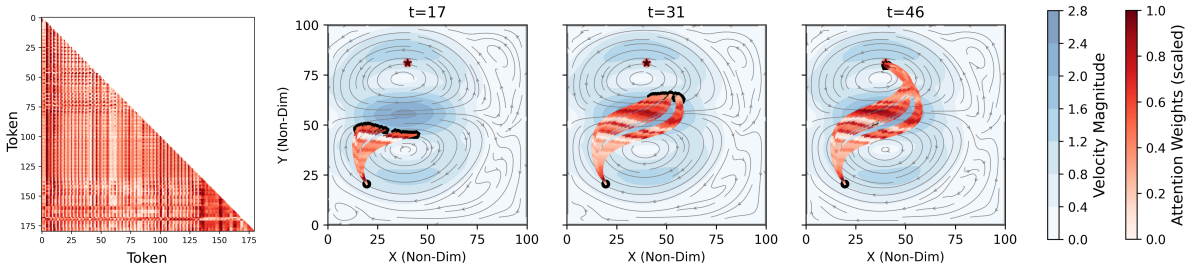

Figure 5: Self-attention matrix and visualization of instantaneous attention on the trajectory at three snapshots for scenario S1.

for $t = [0, 1, ..59]$. The element α_{ij} for $j <= i$, is the attention score between the token i and some previous token j. For example, $\alpha_{9,5}$ is the attention score between a_2 and s_1. A lower diagonal matrix (with masked upper diagonal elements)is used to impose a causal relationship in the attention mechanism, allowing the model to only attend to past information in the sequence for predicting the current action We observe that the heatmap appears to be more structured till approximately the 126^{th} row and relatively less structured below. This is because the corresponding predicted trajectory reaches the target location at the 42^{nd} timestep. We also observe dark vertical and horizontal bands in the top-middle and middle-left regions of the matrix. This observation is better visualized in the next three panels of Fig. 5. These panels show the evolution of all the predicted trajectories from the test set with the attention weights computed with a_t and $(a_k)_{k=0:t}$ at three snapshots. Each state transition of a trajectory up to the given timestep, $(\tau^{(j)}_{DT,pred}[0:3t])$, has been colored with a shade proportional to the action-action $((\alpha_{3t+3,3k+3})^{(j)}_{k=1:t})$ attention weights of that trajectory. In other words, for a trajectory j at time t, we are laying down every $(3k+3)^{th}$ element of the $(3t+3)^{th}$ row of the attention matrix j along the t transitions of the j^{th} trajectory. Simply put, we visualize which prior states and actions were relatively more important for the model to predict the current action.

We observe that the model attends more to states and actions in regions with the strongest flow magnitude, as shown in Fig. 5, where the darkest parts of the trajectories align with the darkest (strongest flow) regions of the velocity field at times $t = 17, 31, 46$. This observation aligns with intuition, as the agent is advected the most in these regions, and a small change in its heading or action could significantly affect its travel time and returns. To aid visualization, a row-wise scaled version of the attention matrix is shown, where the lower diagonal part of each row is linearly scaled independently to have a minimum and maximum element of 0 and 1, respectively.

5 CONCLUSION

We developed and trained a decision transformer-based deep learning model for the onboard routing of autonomous marine vehicles. Notably, the use of the solution of exact time-optimal paths computed as a solution of the stochastic Hamilton Jacobi Bellman level set partial differential equations makes it a first-ever application of the transformer architecture for optimal planning of autonomous marine vehicles advected by the strong currents in the environment they operate. We showed that the transformer could learn representations and reliably execute optimal paths in flow fields different from the training set. Additionally, the model shows strong generalization capabilities even in the presence of noisy fields during inference. Further, training on a few target locations enables it to learn about navigating to other target locations also. In the next steps, we plan to develop a large trajectory model using a single trained model for various flow fields and start-target combinations similar to large language models. We also plan to use multiple agents and learn in a coordinated or adversarial fashion.

ACKNOWLEDGMENTS

The authors would like to thank the Prime Minister's Research Fellowship (PMRF) for supporting the graduate studies of R.C. We also acknowledge the partial support received through research grant No. CRG/2021/004595 from SERB, DST, Govt. of India.

REFERENCES

[1] Enrico Anderlini, Gordon G. Parker, and Giles Thomas. 2019. Docking Control of an Autonomous Underwater Vehicle Using Reinforcement Learning. *Applied Sciences* 9, 17 (2019). https://doi.org/10.3390/app9173456

[2] Xavier Bresson and Thomas Laurent. 2021. The transformer network for the traveling salesman problem. *arXiv preprint arXiv:2103.03012* (2021).

[3] Devendra Singh Chaplot, Deepak Pathak, and Jitendra Malik. 2021. Differentiable Spatial Planning using Transformers. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 1484–1495. https://proceedings.mlr.press/v139/chaplot21a.html

[4] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. 2021. Decision Transformer: Reinforcement Learning via Sequence Modeling. https://doi.org/10.48550/ARXIV.2106.01345

[5] Rohit Chowdhury, Atharva Navsalkar, and Deepak Subramani. 2022. GPU-Accelerated Multi-Objective Optimal Planning in Stochastic Dynamic Environments. *Journal of Marine Science and Engineering* 10, 4 (2022). https://doi.org/10.3390/jmse10040533

[6] Rohit Chowdhury and Deepak Subramani. 2022. Optimal Path Planning of Autonomous Marine Vehicles in Stochastic Dynamic Ocean Flows Using a GPU-Accelerated Algorithm. *IEEE Journal of Oceanic Engineering* (2022), 1–16. https://doi.org/10.1109/JOE.2022.3152514

[7] Rohit Chowdhury and Deepak N Subramani. 2020. Physics-Driven Machine Learning for Time-Optimal Path Planning in Stochastic Dynamic Flows. In *International Conference on Dynamic Data Driven Application Systems*. Springer, 293–301.

[8] Benoit Cushman-Roisin and Jean-Marie Beckers. 2011. *Introduction to geophysical fluid dynamics: physical and numerical aspects*. Academic press.

[9] Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. 2019. Scene Memory Transformer for Embodied Agents in Long-Horizon Tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[10] D. Ferguson and A. Stentz. 2005. The Delayed D* Algorithm for Efficient Path Replanning. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. 2045–2050. https://doi.org/10.1109/ROBOT.2005.1570414

[11] Avijit Gangopadhyay. 2022. *Introduction to Ocean Circulation and Modeling*. CRC Press.

[12] Loshchilov Ilya and Hutter Frank. 2019. Decoupled weight decay regularization. In *Proceedings of ICLR*.

[13] Michael Janner, Qiyang Li, and Sergey Levine. 2021. Offline Reinforcement Learning as One Big Sequence Modeling Problem. https://doi.org/10.48550/ARXIV.2106.02039

[14] Jacob J Johnson, Linjun Li, Ahmed H Qureshi, and Michael C Yip. 2021. Motion planning transformers: One model to plan them all. *arXiv preprint arXiv:2106.02791* (2021).

[15] Wouter Kool, Herke Van Hoof, and Max Welling. 2018. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475* (2018).

[16] Dhanushka Kularatne, Hadi Hajieghrary, and M Ani Hsieh. 2018. Optimal Path Planning in Time-Varying Flows with Forecasting Uncertainties. In *2018 IEEE ICRA*. 1–8.

[17] T. Lolla, P. F. J. Lermusiaux, M. P. Ueckermann, and P. J. Haley, Jr. 2014. Time-Optimal Path Planning in Dynamic Flows using Level Set Equations: Theory and Schemes. *Ocean Dynamics* 64, 10 (2014), 1373–1397.

[18] Tapovan Lolla, Mattheus P. Ueckermann, Konur Yiğit, Patrick J. Haley, Jr., and Pierre F. J. Lermusiaux. 2012. Path planning in time dependent flow fields using level set methods. In *IEEE International Conference on Robotics and Automation (ICRA), 14-18 May 2012*. 166–173. https://doi.org/10.1109/ICRA.2012.6225364

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[20] Arvind A. Pereira, Jonathan Binney, Geoffrey A. Hollinger, and Gaurav S. Sukhatme. 2013. Risk-aware Path Planning for Autonomous Underwater Vehicles using Predictive Ocean Models. *Journal of Field Robotics* 30, 5 (2013), 741–762. https://doi.org/10.1002/rob.21472

[21] Mass Per Pettersson, Gianluca Iaccarino, and Jan Nordström. 2015. Polynomial Chaos Methods. In *Polynomial Chaos Methods for Hyperbolic Partial Differential Equations*. Springer, 23–29.

[22] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[23] Themistoklis P. Sapsis and Pierre F. J. Lermusiaux. 2009. Dynamically orthogonal field equations for continuous stochastic dynamical systems. *Physica D: Nonlinear Phenomena* 238, 23–24 (Dec. 2009), 2347–2360. https://doi.org/10.1016/j.physd.2009.09.017

[24] Yogang Singh, Sanjay Sharma, Robert Sutton, Daniel Hatton, and Asiya Khan. 2018. A constrained A* approach towards optimal path planning for an unmanned surface vehicle in a maritime environment containing dynamic obstacles and ocean currents. *Ocean Engineering* 169 (2018), 187–201. https://doi.org/10.1016/j.oceaneng.2018.09.016

[25] Yogang Singh, Sanjay Sharma, Robert Sutton, Daniel Hatton, and Asiya Khan. 2018. Feasibility study of a constrained Dijkstra approach for optimal path planning of an unmanned surface vehicle in a dynamic maritime environment. In *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. 117–122. https://doi.org/10.1109/ICARSC.2018.8374170

[26] Deepak N. Subramani and Pierre F. J. Lermusiaux. 2019. Risk-Optimal Path Planning in Stochastic Dynamic Environments. *CMAME* 353 (2019), 391–415.

[27] D. N. Subramani, Q. J. Wei, and P. F. J. Lermusiaux. 2018. Stochastic Time-Optimal Path-Planning in Uncertain, Strong, and Dynamic Flows. *CMAME* 333 (2018), 218–237.

[28] M. P. Ueckermann, P. F. J. Lermusiaux, and T. P. Sapsis. 2013. Numerical schemes for dynamically orthogonal equations of stochastic fluid and ocean flows. *J. Comput. Phys.* 233 (Jan. 2013), 272–294. https://doi.org/10.1016/j.jcp.2012.08.041

[29] Anete Vagale, Rachid Oucheikh, Robin T Bye, Ottar L Osen, and Thor I Fossen. 2021. Path planning and collision avoidance for autonomous surface vehicles I: a review. *Journal of Marine Science and Technology* (2021), 1–15.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[31] Zhao Wang and Xianbo Xiang. 2018. Improved Astar Algorithm for Path Planning of Marine Robot. In *2018 37th Chinese Control Conference (CCC)*. 5410–5414. https://doi.org/10.23919/ChiCC.2018.8483946

[32] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. 2022. Learning Improvement Heuristics for Solving Routing Problems. *IEEE Transactions on Neural Networks and Learning Systems* 33, 9 (2022), 5057–5069. https://doi.org/10.1109/TNNLS.2021.3068828

[33] Jing Xin, Huan Zhao, Ding Liu, and Minqi Li. 2017. Application of deep reinforcement learning in mobile robot path planning. In *2017 Chinese Automation Congress (CAC)*. IEEE, 7112–7116.

[34] Byunghyun Yoo and Jinwhan Kim. 2016. Path optimization for marine vehicles in ocean currents using reinforcement learning. *JMST* 21, 2 (2016), 334–343.

[35] Lin Zhang, Yingjie Zhang, and Yangfan Li. 2020. Path planning for indoor mobile robot based on deep learning. *Optik* 219 (2020), 165096.