Context-Aware Multi-Agent Trajectory Transformer

Jeongho Park and Songhwai Oh

Abstract— **Transformer-based sequence models have proven effective in offline reinforcement learning for modeling agent trajectories using large-scale datasets. However, applying these models directly to multi-agent offline reinforcement learning introduces additional challenges, especially in managing complex inter-agent dynamics that arise as multiple agents interact with both their environment and each other. To overcome these issues, we propose the context-aware multi-agent trajectory transformer (COMAT), a novel model designed for offline multi-agent reinforcement learning tasks which predicts the future trajectory of each agent by incorporating the history of adjacent agents—referred to as *context*—into its sequence modeling. COMAT consists of three key modules: the transformer module to process input trajectories, the context encoder to extract relevant information from adjacent agents' histories, and the context aggregator to integrate this information into the agent's trajectory prediction process. Built upon these modules, COMAT predicts the agents' future trajectories and actively leverages this capability as a tool for planning, enabling the search for optimal actions in multi-agent environments. We evaluate COMAT on multi-agent MuJoCo and StarCraft Multi-Agent Challenge tasks, on which it demonstrates superior performance compared to existing baselines.**

I. INTRODUCTION

In single-agent offline reinforcement learning (RL), trajectory optimization-based methods that utilize transformer-based sequence models have emerged as effective tools for leveraging the large demonstration datasets [1]–[5]. These methods represent the agent's trajectory as a sequence of states, actions, and rewards, and learn their distribution from the offline dataset. A key strength lies in the attention mechanism of the transformer architecture [6], which excels at handling sequential data by capturing its long-range dependencies. With the help of large-scale offline datasets, this strength has led to strong performance on diverse offline RL tasks [1]–[5].

Recent approaches have applied trajectory optimization-based methods to offline multi-agent RL (MARL) tasks [7], [8]. However, directly extending the transformer-based sequence models to multi-agent systems introduces another challenge. In multi-agent systems, the effect of an agent's actions on the environment cannot be determined independently, since it also depends on the actions of other agents [9]. Consequently, an agent's trajectory distribution depends on the trajectories of other agents. Moreover, in many real-world multi-agent problems, decision-making is decentralized, and agents must operate in partially observable environments [10]. This lack of complete information further complicates accurate prediction of future trajectory. One potential approach to addressing this challenge is to incorporate contextual information from adjacent agents. If partial information about adjacent agents' trajectory histories—what we refer to as *contexts*—is available in such cases, incorporating them can enhance the sequence model's prediction performance by leveraging implicit information about the current state and other agents' actions. However, no prior work has modeled trajectory sequences with multi-agent *contexts* in offline MARL, leaving this key challenge open.

In this paper, we propose a *context-aware multi-agent trajectory transformer* (COMAT), a trajectory sequence model for offline MARL that explicitly incorporates *contexts* to predict the trajectories of individual agents. An overview of COMAT is provided in Figure 1. Based on the design of the Trajectory Transformer [1], our model introduces three key modules: (1) a transformer module, (2) a context encoder, and (3) a context aggregator. The transformer module serves as the core sequence model, processing each agent's trajectory sequence tokens into trajectory embeddings. The context encoder is designed to extract relevant information from the trajectory histories and effectively encode it into context embeddings. Finally, the context aggregator integrates the context embeddings from adjacent agents into the trajectory embeddings, which are then processed by MLPs to predict the distribution of the next token, enabling sequential trajectory generation. All three modules are learned end-to-end by optimizing a joint objective function, ensuring accurate trajectory prediction while incorporating the *context* information. During planning, COMAT leverages its ability to predict future trajectories as a key component for searching for each agent's action that maximizes the expected return. By incorporating *contexts*, COMAT not only considers the current environment but also takes other agents into account, enabling more informed decision-making in multi-agent settings.

We evaluate COMAT on multi-agent MuJoCo [11] and StarCraft Multi-Agent Challenge [12] tasks with offline benchmark datasets [13], [14]. Our experiments demonstrate that COMAT outperforms the baseline offline MARL algorithms in the majority of tasks across both environments.

J. Park and S. Oh are with the Department of Electrical and Computer Engineering and ASRI, Seoul National University, Seoul, Korea (email: jeongho.park@rllab.snu.ac.kr, songhwai@snu.ac.kr). *(Corresponding author: Songhwai Oh.)*

This work was supported by Korea Evaluation Institute of Industrial Technology (KEIT) grant funded by the Korea government (MOTIE) (No.20024368, Development of Integrated Control System for Multi-mobility and Autonomous Delivery Service Demonstration, 50%) and the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00336738, Development of Complex Task Planning Technologies for Autonomous Agents, 50%).

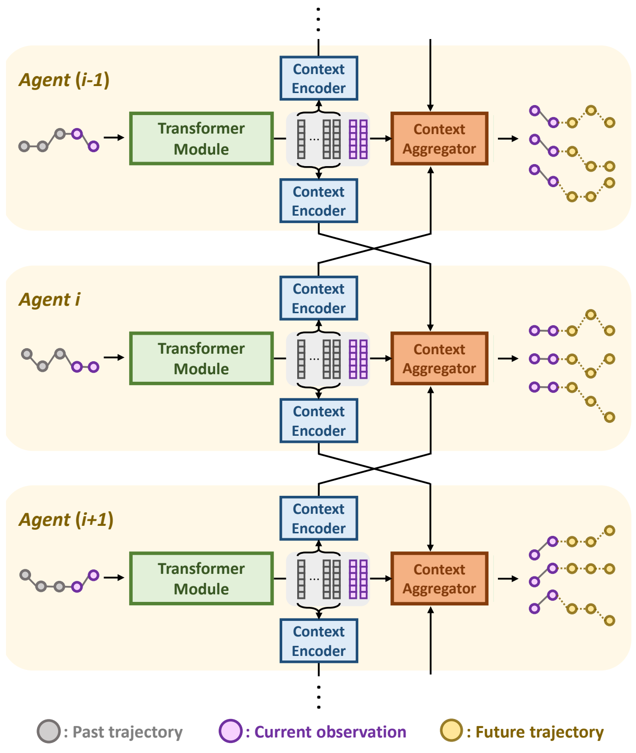


Fig. 1: **Overview of COMAT.**

Additionally, we conduct ablation studies to confirm the importance of COMAT's key components. These results highlight its effectiveness in utilizing *context* for multi-agent trajectory prediction and planning in offline MARL tasks.

II. RELATED WORK

A. *Offline multi-agent reinforcement learning*

MARL [15]–[21] has shown significant advancements in solving complex tasks by leveraging techniques from single-agent RL methods. Building on these successes, recent work has extended MARL to offline settings and has gained increasing attention [22]–[26]. OMAR [22] finds that applying conservative offline RL algorithms does not work well in the multi-agent setting due to the non-concavity of the value function, and proposes a method that combines the first-order policy gradients and zeroth-order optimization methods to optimize the value function better. OMIGA [23] introduces a framework that converts global value regularization into local-level, effectively linking multi-agent value decomposition and the regularization techniques in offline RL. These approaches aim to bridge the gap between single-agent and multi-agent offline RL settings, demonstrating improved performance across various tasks.

B. *Sequence model for reinforcement learning*

Trajectory Transformer [1] proposes a trajectory optimization-based method that uses a transformer architecture [6] to model trajectory distributions. Concurrently, Decision Transformer [2] has also been proposed using a similar architecture, and utilizes the target return as a condition to generate expert behavior. RT-X [3] introduces a transformer-based model for imitation learning that is specially designed for inferring robotic actions, along with the large-scale robot motion dataset.

In the multi-agent domain, MADT [7] extends Decision Transformer to MARL via offline pre-training and online fine-tuning. [8] also proposes applying Decision Transformer on the multi-agent domain via policy distillation from the teacher policy that is trained using global state information. Similar to our model, both approaches apply trajectory optimization-based methods to offline MARL leveraging the transformer design. However, none of them focus on utilizing information from adjacent agents. Multi-Agent Transformer [27] proposes an online MARL framework that treats agents as a sequence. The policy generates their actions in a specific order, directly modeling inter-agent dependencies through autoregressive generation. AgentFormer [28] introduces a multi-agent trajectory prediction model that aggregates information from multiple agents through a specially designed attention mechanism. However, it is primarily designed for state prediction tasks and requires further extension to be applicable to reinforcement learning.

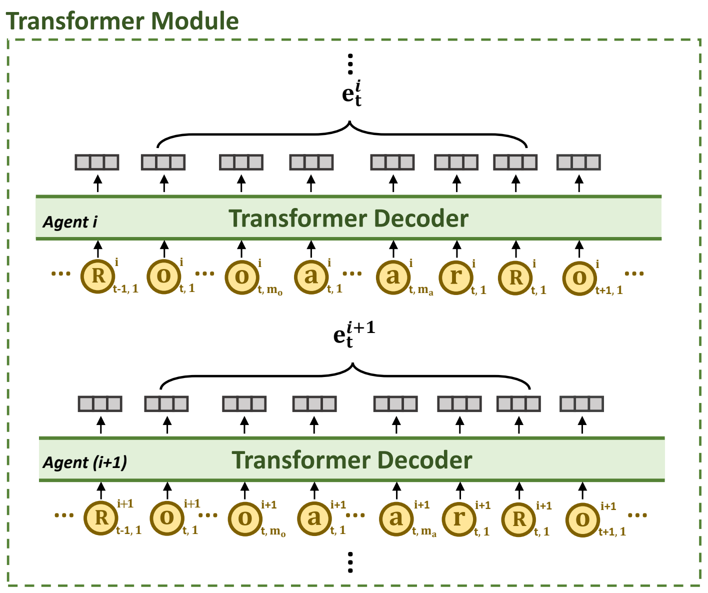
III. METHOD

COMAT, a context-aware multi-agent trajectory transformer, is a transformer-based sequence model for offline MARL that generates future trajectories for each agent conditioned on its own observation and history, as well as the histories of its adjacent agents, which we refer to as *contexts*.

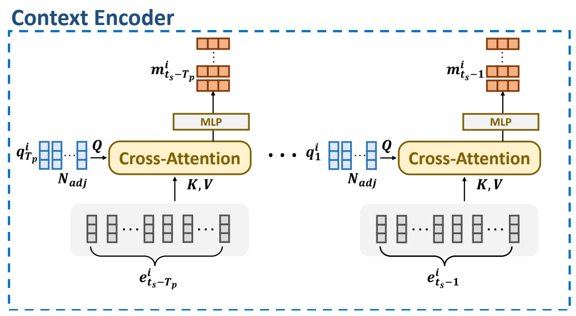
COMAT employs three core modules: the transformer module, the context encoder, and the context aggregator. These modules perform the following key tasks: (1) the transformer module processes the input trajectory sequence into trajectory embeddings, (2) the context encoder extracts context embeddings from the trajectory history and shares them with adjacent agents, and (3) the context aggregator aggregates the context embeddings from adjacent agents and integrates them into the trajectory embeddings to generate the future trajectory sequence. The model architecture is illustrated in Figure 2. In the following sections, we provide detailed descriptions of our model's formulation and the architectural design of its core modules. Also, we present the training details and explain how it is utilized in planning for offline MARL tasks.

A. *Formulation of COMAT*

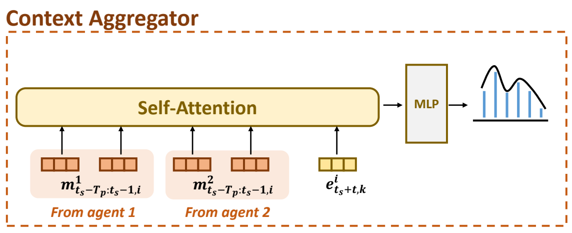
We first explain how the multi-agent system in our problem is defined and how trajectory sequence data is represented. We consider a multi-agent system consisting of N_{ag} agents, where each agent $i \in \{1, \dots, N_{\text{ag}}\}$ receives an individual partial observation $\mathbf{o}_t^i \in \mathbb{R}^{M_o}$ at each time step t. Each agent selects an action $\mathbf{a}_t^i \in \mathbb{R}^{M_a}$ and receives a joint reward r_t which is shared across all agents. M_o and M_a denote the dimensionality of a single agent's observation and action, respectively. Then, an episode trajectory $\boldsymbol{\tau}^i$ of the



(a) Transformer module.



(b) Context encoder module.



(c) Context aggregator module.

Fig. 2: **Three core modules of COMAT.** (a) Transformer module processes the input trajectory sequence of each agent into a sequence of trajectory embeddings. (b) Context encoder extracts important information from the trajectory and summarizes it into context embeddings. (c) Context aggregator integrates the context embeddings and trajectory embeddings to predict the next trajectory token distribution.

agent i is represented as a sequence of observations, actions, and rewards of T time steps:

$$\boldsymbol{\tau}^i = (\mathbf{o}_1^i, \mathbf{a}_1^i, r_1, \mathbf{o}_2^i, \mathbf{a}_2^i, r_2, \dots, \mathbf{o}_T^i, \mathbf{a}_T^i, r_T), \quad i = 1, 2 \dots N_{\text{ag}}. \tag{1}$$

We first separate the observation and action vectors and flatten them into a long sequence of scalar values:

$$\boldsymbol{\tau}^i = (\dots, o_{t,1}^i, o_{t,2}^i, \dots, o_{t,M_o}^i, a_{t,1}^i, a_{t,2}^i, \dots, a_{t,M_a}^i, r_t, \dots), \quad i = 1, 2 \dots N_{\text{ag}}, \qquad t = 1, \dots, T, \tag{2}$$

where $o_{t,k}^i$ represents the k-th dimension of the observation vector $\mathbf{o}_t^i$, and $a_{t,k}^i$ represents the k-th dimension of the action vector $\mathbf{a}_t^i$. $\boldsymbol{\tau}$ refers to the joint trajectory of all agents. Then, we discretize trajectory data into token sequences. Our approach for discretization follows that of Trajectory Transformer [1], but with per-agent discretization. Continuous values are mapped to tokens using quantile-based intervals, allowing balanced coverage of each agent's data range. Since discretization is done independently, interval boundaries may differ across agents, enabling flexible representation of heterogeneous observation and action distributions of the multi-agent system. This discretization process enables the transformer module of COMAT to adopt the standard transformer architecture while effectively modeling multi-agent trajectory sequences.

COMAT is a sequence model designed to generate each agent's next action and reward, as well as the next T_h steps of the agent's future trajectory given its current observation, past T_p steps of trajectories and the past T_p steps of adjacent agents' trajectories. Let t_s denote the time step at which COMAT receives the observation of the agent i. We define *contexts* as the trajectory history of T_p time steps from the adjacent agents represented as:

$$\mathbf{c}_{t_s}^i = \{\mathbf{o}_{t_s - T_p : t_s - 1}^j, \mathbf{a}_{t_s - T_p : t_s - 1}^j \mid j \in \mathcal{N}(i)\}, \tag{3}$$

where $\mathcal{N}(i)$ denotes the pre-defined set of adjacent agents of the agent i. We assume that the adjacency of agents is symmetric, meaning if $i \in \mathcal{N}(j)$, then $j \in \mathcal{N}(i)$ for all $i \neq j, \;\; i, j = 1, 2, \dots, N_{\text{ag}}$. We specify N_{adj} as the pre-defined number of adjacent agents for each agent, equal to the size of $\mathcal{N}(i)$. Note that *contexts* do not include the rewards, since they are identical across all agents, and therefore not needed.

Based on the formulation, we propose the sequence model for predicting the probability distribution of the next token in the trajectory sequence data for each agent. Let θ_i represent the parameters of the i-th agent's trajectory sequence model, and P_{θ_i} represent the induced conditional probabilities. We formulate the objective function to optimize our model for predicting the next observation, action, reward tokens, as well as the tokens for reward-to-go, $R_t = \sum_{t'=t}^{T} r_{t'}$. The reward-to-go tokens are placed next to the reward tokens, and they represent the cumulative rewards from the current time step to the end of the episode. These tokens are added to evaluate how well the trajectories lead to higher returns. The objective function to be maximized for time step t_s is:

$$\mathcal{L}_{t_s}(\boldsymbol{\tau}^i) = \sum_{t=t_s+1}^{T_h+t_s} \sum_{j=1}^{M_o} \log P_{\theta_i}(o^i_{t,j} \mid \mathbf{o}^i_{t,<j}, \mathbf{h}^i_{t_s,t}) + \sum_{t=t_s}^{T_h+t_s} \Bigg[\sum_{k=1}^{M_a} \log P_{\theta_i}(a^i_{t,k} \mid \boldsymbol{a}^i_{t,<k}, \boldsymbol{o}^i_t, \mathbf{h}^i_{t_s,t}) + \log P_{\theta_i}(r_t \mid \boldsymbol{a}^i_t, \boldsymbol{o}^i_t, \mathbf{h}^i_{t_s,t}) + \log P_{\theta_i}(R_t \mid \mathbf{h}^i_{t_s,t+1}) \Bigg], \tag{4}$$

where $\mathbf{h}^i_{t_s,t} = \{\mathbf{c}^i_{t_s}, \boldsymbol{\tau}^i_{t_s-T_p:t-1}\}$. The symbol $\mathbf{h}^i_{t_s,t}$ is used to simplify notation, representing the pre-given conditions provided to the model in a more compact form. Each term of the objective represents the log probability of predicting the next tokens corresponding to observations, actions, rewards, and reward-to-go values. We expand this objective by summing over all time steps t_s and agent indices i, resulting in a total objective for the joint trajectory $\boldsymbol{\tau}$:

$$\mathcal{L}(\boldsymbol{\tau}) = \sum_{i=1}^{N_{\text{ag}}} \sum_{t_s=T_p}^{T-T_h} \mathcal{L}_{t_s}(\boldsymbol{\tau}^i). \tag{5}$$

By maximizing this objective, COMAT can predict the probability distribution of the tokens corresponding to the next T_h steps of the future trajectory for each agent.

B. COMAT core modules

In this section, we provide detailed explanations of COMAT's three core modules.

Transformer Module. The transformer module processes the input trajectory sequence by passing it through a series of causal self-attention layers, which generate the trajectory embedding that corresponds to each token in the sequence (Figure 2(a)). This design follows a transformer decoder architecture [6]. The generated trajectory embeddings are later used to generate the next token in the sequence. We denote the trajectory embedding corresponding to the k-th token at the t-th time step as $\mathbf{e}^i_{t,k} \in \mathbb{R}^{d_e}$, where d_e is the dimension of the embedding vector. Since these embeddings are computed using self-attention with causal masking, each embedding can only capture the information from the token itself and previous tokens in the sequence.

Context Encoder. The context encoder of agent i encodes the trajectory embeddings generated from the agent's trajectory history into context embedding vectors. In detail, a different context embedding vector is generated for each time step and each adjacent agent receiving it, resulting in total $T_p \times N_{\text{adj}}$ context embeddings per agent (Figure 2(b)). For each time step t and for each adjacent agent $j \in \mathcal{N}(i)$, the encoder generates the following context embeddings:

$$\begin{aligned}\{\mathbf{m}^i_{t_s-t,j} \mid t = 1, 2, \dots, T_p,\ j \in \mathcal{N}(i)\} \\ = \mathit{ContextEnc}_i(\mathbf{e}^i_{t_s-T_p:t_s-1}),\end{aligned} \tag{6}$$

where $\mathbf{e}^i_{t_s-T_p:t_s-1}$ denotes the sequence of trajectory embedding vectors generated by the transformer module. Note that these vectors contain information from the history trajectory tokens corresponding to the time steps from t_s-T_p to t_s-1. For simplicity, $\mathbf{m}^i_{t_s}$ refers to the set of all context embeddings processed from the agent i for its adjacent agents.

To leverage the history effectively, we employ the cross-attention mechanism in the context encoder. The queries are learnable parameters $\mathbf{q}^i_{t,j}$, one for each time step t and adjacent agent j, allowing the model to learn how to extract relevant information. The keys and values are the trajectory embeddings obtained by processing the trajectory history through the transformer module. Each $\mathbf{q}^i_{t,j}$ attends only to $\mathbf{e}^i_t$, restricting attention to the corresponding time step. By maximizing the objective (5), each query $\mathbf{q}^i_{t,j}$ is trained to extract information that is specifically relevant to the needs of the adjacent agent j. After passing through an additional MLP layer, $T_p \times N_{\text{adj}}$ different context embeddings are generated for agent i. These context embeddings are then distributed to the adjacent agents. As previously mentioned, the rewards and reward-to-go values are already shared among agents, so there is no need to include them in the *context*. Therefore, we mask them out during cross-attention, focusing only on the observation and action tokens.

Context Aggregator. The context aggregator module of the agent i gathers the context embeddings from the adjacent agents and integrates them into the trajectory embedding of the current token through a self-attention layer. The updated embedding is then passed through an MLP decoder to produce the logits for the next token (Figure 2(c)). This process actively determines how much information from the adjacent agents' histories should be incorporated.

To clarify the process, let us assume that at time step t_s, agent i is tasked with predicting the $(k+1)$-th token at future time step t_s+t. Specifically, the k-th token's trajectory embedding $\mathbf{e}^i_{t_s+t,k}$ serves as the query, while $\mathbf{e}^i_{t_s+t,k}$ along with the context embeddings $\mathbf{m}^j_{t_s-T_p:t_s-1,i}$ from the adjacent agents $j \in \mathcal{N}(i)$ are used as the keys and values in the self-attention layer. The resulting output embedding, $\tilde{\mathbf{e}}^i_{t_s+t,k} \in \mathbb{R}^{d_e}$, is taken as the final aggregated embedding:

$$\tilde{\mathbf{e}}^i_{t_s+t,k} = \mathit{ContextAgg}(\{\mathbf{m}^j_{t_s-T:t_s-1,i} \mid j \in \mathcal{N}(i)\}, \mathbf{e}^i_{t_s+t,k}). \tag{7}$$

$\tilde{\mathbf{e}}^i_{t_s+t,k}$ is then processed by the MLP decoder to produce the logits for predicting the $(k+1)$-th token.

C. Training

All modules of COMAT are trained jointly to maximize the objective (5). While all agents share most of the parameters of the modules in COMAT, distinct input token embeddings are used to ensure that each agent maintains its unique trajectory representation. For implementation, N_{adj} is set to 2, and the adjacent agents of agent i are predefined as the agents with indices $i-1$ and $i+1$, with cyclic selection applied when the index goes below 1 or exceeds N_{ag} (i.e., 0 wraps to N_{ag} and $N_{\text{ag}}+1$ wraps to 1). However, the selection of adjacent agents is flexible and can be customized accordingly. Across all tasks, COMAT is trained for 50 epochs with a batch size of 256 and a learning rate of 6.0×10^{-4}. The number of bins used for discretizing

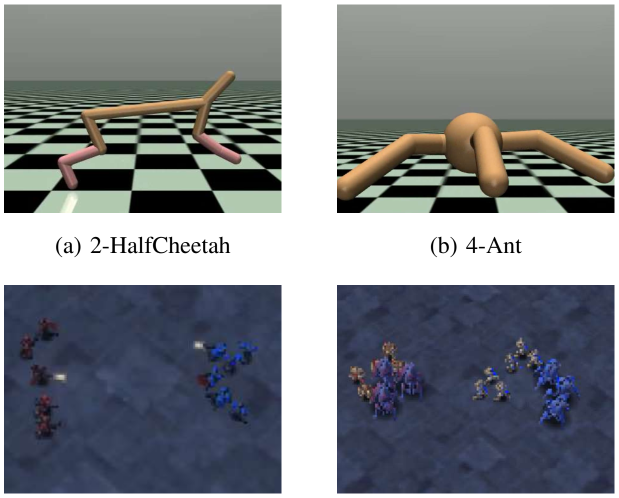

(a) 2-HalfCheetah (b) 4-Ant (c) $5m\_vs\_6m$ (d) $3s5z\_vs\_3s6z$

Fig. 3: **Task environments.**. Rendered images of the task environments used in our evaluation. (a)-(b) Two tasks from Multi-agent MuJoCo, and (c)-(d) two tasks from SMAC.

each dimension of the trajectory is set to 100, while d_e is set to 128.

D. Planning with COMAT

At inference time step t_s, COMAT samples the next tokens from categorical distributions over the output logits. By sequentially and in parallel sampling, COMAT generates diverse future trajectories per agent conditioned on its current observation, own history, and adjacent agents' *contexts* at t_s. Each generated trajectory is scored by summing its estimated rewards and the estimated reward-to-go at the final step. We then select the highest-scoring trajectories and use their first actions as the agents' next actions. We use the beam-search method from [1] for efficient trajectory selection. Note that no information is exchanged during beam search, as agents rely solely on pre-shared *contexts*, preserving decentralization and parallelism. In the implementation, we use five step history as the *context* ($T_p = 5$), set the prediction horizon to three steps ($T_h = 3$), and set the beam search width to 64.

IV. EXPERIMENTS

A. Tasks

We evaluate COMAT on two different tasks, which are multi-agent MuJoCo [11] and StarCraft Multi-Agent Challenge (SMAC) tasks [12] (Figure 3).

Multi-Agent MuJoCo. This benchmark, built on MuJoCo, enables continuous multi-agent robotic control by splitting a single robot into independent sub-bodies, each assigned to an agent [11]. We adapt the D4RL MuJoCo dataset [13] into a multi-agent format following [30]–[32], where observations and actions are organized per agent. Each agent receives joint angles and velocities of its own body as local observations, and the torso's position and velocity as globally shared observations. Agents can only control the joints that are included in their own bodies. We evaluate on 2-Halfcheetah and 4-Ant, where the original robots are divided into two and four agents respectively. We conduct evaluations on four different data quality levels: expert, medium, medium-replay and medium-expert. Medium-replay contains all samples in the replay buffer observed during training the medium-level policy. Medium-expert is a mixture of the expert and medium datasets, combining demonstrations of varying performance levels.

SMAC. SMAC focuses on micromanagement challenges where each unit in the StarCraft game is controlled by an independent agent that acts based on local observations [12]. Unlike the multi-agent MuJoCo tasks, SMAC tasks involve discrete action spaces that include actions such as moving, attacking, and more. We choose two scenarios named $5m\_vs\_6m$ and $3s5z\_vs\_3s6z$, and both present a high level of difficulty. We train our method and the baselines using the SMAC dataset proposed from OG-MARL [14] that has three quality levels: good, medium, and poor.

For each task, we evaluate all algorithms using three different training seeds. Each trained model is tested over 32 independent episodes.

B. Baselines

We compare COMAT with four recent offline MARL algorithms: OMIGA [23], OMAR [22], Multi-agent version CQL [29] (MA-CQL), and MADT [7]. MA-CQL is a variant of CQL [29], modified to use a central critic for continuous action tasks and a QMIX-style mixer [16] for discrete action tasks. The performance scores of MA-CQL on SMAC are taken from the results reported in OG-MARL [14]. MADT originally includes both offline and online training phases, but we only conduct offline training as our focus is solely on offline MARL evaluation.

C. Results

Table I and Table III present the means and standard deviations of the average returns for multi-agent MuJoCo and SMAC tasks, computed across the three different training seeds. The results show that COMAT demonstrates superior performance on average compared to the baselines, both in continuous and discrete action domains. While OMIGA [23] and MADT [7] achieve competitive performance in both multi-agent MuJoCo and SMAC tasks, COMAT consistently achieves higher scores in the experiments, except for the 4-Ant experiment with the medium dataset. In contrast, OMAR [22] and MA-CQL [29] exhibit unstable learning dynamics in MuJoCo tasks and with few exceptions, perform significantly worse than COMAT and other baselines. Even in cases where certain baselines achieve higher scores, we observe that COMAT maintains stable learning and delivers competitive performance without exhibiting fluctuations in scores.

In both multi-agent MuJoCo and SMAC tasks, coordinating multiple agents to plan effectively requires each agent not only to perform well individually, but also to consider and synchronize with the actions of others. These results show that COMAT successfully coordinated the agents in planning their actions while considering the adjacent agents,

TABLE I: **Results in multi-agent MuJoCo tasks.**

Task	Dataset	COMAT (ours)	OMIGA [23]	OMAR [22]	MA-CQL [29]	MADT [7]
2-HalfCheetah	Expert	**10393.25 ± 173.76**	10242.95 ± 142.7	555.86 ± 299.06	-109.566 ± 58.35	4724.88 ± 4351.85
	Medium	5247.44 ± 63.38	5187.22 ± 13.86	**5446.85 ± 40.82**	24.87 ± 152.69	4944.06 ± 142.43
	Medium-replay	**4766.11 ± 189.84**	4765.38 ± 141.68	597.77 ± 1458.00	-139.62 ± 43.47	527.13 ± 418.84
	Medium-expert	**9101.05 ± 1170.76**	7549.05 ± 1463.71	910.05 ± 1815.79	-57.03 ± 24.78	4801.77 ± 118.32
4-Ant	Expert	**4326.17 ± 100.29**	3751.08 ± 869.90	424.01 ± 186.60	834.77 ± 94.69	3594.36 ± 2500.17
	Medium	3283.83 ± 70.88	3433.22 ± 253.17	919.541 ± 509.51	815.54 ± 58.35	**4029.81 ± 9.97**
	Medium-replay	**2738.40 ± 395.04**	1611.66 ± 641.26	311.18 ± 328.38	2074.60 ± 1332.10	1480.04 ± 676.56
	Medium-expert	**4177.87 ± 61.10**	3811.56 ± 544.61	858.17 ± 69.03	1285.01 ± 2246.30	2899.45 ± 2052.59

TABLE II: **Ablation studies of COMAT.**

Task	Dataset	COMAT (ours)	COMAT-MLP	COMAT-BC	MAT (w/o context)
2-HalfCheetah	Expert	**10393.25 ± 173.76**	9750.45 ± 1276.19	10140.69 ± 351.83	6368.79 ± 2832.56
	Medium	**5247.44 ± 63.38**	5124.39 ± 226.49	4841.58 ± 69.53	4935.91 ± 217.33
	Medium-replay	**4766.11 ± 189.84**	4671.31 ± 70.72	3923.49 ± 74.32	4378.57 ± 118.55
	Medium-expert	**9101.05 ± 1170.76**	8371.44 ± 1278.26	6281.35 ± 481.40	7810.10 ± 3055.27
4-Ant	Expert	4326.17 ± 100.29	**4526.24 ± 172.56**	3897.00 ± 325.30	4278.08 ± 312.14
	Medium	**3283.83 ± 70.88**	3251.97 ± 108.25	2370.79 ± 242.04	2999.98 ± 113.42
	Medium-replay	**2738.40 ± 395.04**	2572.12 ± 502.99	2057.96 ± 99.19	2699.97 ± 12.12
	Medium-expert	**4177.87 ± 61.10**	3953.98 ± 190.91	2924.54 ± 274.27	2423.89 ± 1484.68

TABLE III: **Results in SMAC tasks.**

$5m\_vs\_6m$

	Good	Medium	Poor
OMIGA [23]	9.46 ± 0.90	9.96 ± 0.86	7.65 ± 0.45
OMAR [22]	15.86 ± 0.74	15.01 ± 0.80	7.89 ± 0.19
MA-CQL [29]	13.80 ± 3.90	16.90 ± 1.20	**10.40 ± 1.00**
MADT [7]	15.93 ± 1.82	16.20 ± 0.83	9.43 ± 0.25
COMAT (ours)	**16.25 ± 1.39**	**17.14 ± 0.30**	8.53 ± 0.84

$3s5z\_vs\_3s6z$

	Good	Medium	Poor
OMIGA [23]	16.56 ± 0.98	11.39 ± 0.32	1.30 ± 0.02
OMAR [22]	12.33 ± 3.52	12.69 ± 0.50	8.11 ± 0.14
MA-CQL [29]	7.30 ± 1.90	8.10 ± 3.10	2.90 ± 0.90
MADT [7]	13.89 ± 0.66	12.66 ± 0.56	8.32 ± 0.26
COMAT (ours)	**17.82 ± 0.41**	**13.23 ± 0.13**	**8.73 ± 0.45**

by effectively encoding the necessary information from the *contexts* and aggregating them into its sequence model.

D. Ablation studies

We conduct ablation studies on multiple variants of COMAT to analyze the impact of different components on performance. We introduce three variants of COMAT, namely, COMAT-MLP, COMAT-BC, and MAT. COMAT-MLP replaces the proposed cross-attention-based approach in the context encoder with a simple two-layer MLP network. In this version, the trajectory embeddings from the history are independently encoded through the MLP, and their mean is used to compute the context embedding. COMAT-BC, on the other hand, determines the next action by taking the argmax of its probability, eliminating the process for future trajectory sampling or beam search. This approach is akin to conventional behavior cloning (BC). In addition, we evaluate a variant of COMAT with the same architecture but without utilizing the *contexts* (MAT). MAT does not include the context encoder and the context aggregator. Trajectory embeddings are directly mapped to the logits of the next tokens using MLP decoders.

Table II shows the results of ablation studies for COMAT on the multi-agent MuJoCo tasks. The results indicate that COMAT consistently outperforms MAT, highlighting the advantages of incorporating *contexts* into the sequence model for multi-agent trajectory prediction. While MAT also demonstrates competitive performance, it occasionally exhibits high variance, suggesting instability in its planning. Moreover, we find that although the MLP-based approach used in the context encoder of COMAT-MLP outperforms MAT, which does not use *context*, it still underperforms on average compared to our proposed cross-attention-based approach. We interpret this as the cross-attention mechanism effectively assigning weights to and extracting crucial information from the context embeddings. Finally, the weaker performance of COMAT-BC compared to COMAT suggests that the combination of generating diverse and valid trajectories with beam search plays a crucial role in discovering better actions. This further emphasizes the importance of accurately estimating rewards and reward-to-go values for improved decision-making.

V. CONCLUSIONS

In this work, we propose COMAT, a context-aware multi-agent trajectory transformer for offline MARL that integrates *context* information from adjacent agents into the trajectory modeling process. Our model leverages a transformer-based architecture with key modules for encoding and aggregating *context* information from adjacent agents, enabling more accurate trajectory prediction and improved decision-making. Through experiments on multi-agent MuJoCo and SMAC tasks, COMAT demonstrated superior performance compared to baseline methods. Ablation studies further validated the crucial role of each COMAT component in improving its performance. These results highlight the effectiveness of integrating *context* into multi-agent trajectory modeling, offering a promising approach for future advancements in offline MARL tasks.

REFERENCES

[1] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," *in Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2021.

[2] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *in Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2021.

[3] A. O'Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee *et al.*, "Open x-embodiment: Robotic learning datasets and RT-X models : Open x-embodiment collaboration," in *Proc. of the International Conference on Robotics and Automation (ICRA)*, May. 2024.

[4] K.-H. Lee, O. Nachum, M. S. Yang, L. Lee, D. Freeman, S. Guadarrama, I. Fischer, W. Xu, E. Jang, H. Michalewski *et al.*, "Multi-game decision transformers," *in Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2022.

[5] Y.-H. Wu, X. Wang, and M. Hamaya, "Elastic decision transformer," *in Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2023.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *in Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2017.

[7] L. Meng, M. Wen, C. Le, X. Li, D. Xing, W. Zhang, Y. Wen, H. Zhang, J. Wang, Y. Yang *et al.*, "Offline pre-trained multi-agent decision transformer," *Machine Intelligence Research*, vol. 20, no. 2, pp. 233–248, 2023.

[8] W.-C. Tseng, T.-H. J. Wang, Y.-C. Lin, and P. Isola, "Offline multi-agent reinforcement learning with knowledge distillation," *in Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2022.

[9] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[10] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: A survey," *Artificial Intelligence Review*, vol. 55, no. 2, pp. 895–943, 2022.

[11] B. Peng, T. Rashid, C. Schroeder de Witt, P.-A. Kamienny, P. Torr, W. Böhmer, and S. Whiteson, "Facmac: Factored multi-agent centralised policy gradients," *in Proc. of Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2021.

[12] M. Samvelyan, T. Rashid, C. S. de Witt, G. Farquhar, N. Nardelli, T. G. J. Rudner, C. Hung, P. H. S. Torr, J. N. Foerster, and S. Whiteson, "The starcraft multi-agent challenge," in *Proc. of the International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, May. 2019.

[13] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," *arxiv.org/abs/2004.07219*, Apr. 2020.

[14] C. Formanek, A. Jeewa, J. Shock, and A. Pretorius, "Off-the-grid marl: Datasets and baselines for offline multi-agent reinforcement learning," in *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Jun. 2023.

[15] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward," in *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May. 2017.

[16] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proc. of the International Conference on Machine Learning (ICLR)*, Jul. 2018.

[17] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *in Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2017.

[18] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative multi-agent games," *in Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2022.

[19] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi, "Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning," in *Proc. of the International conference on machine learning (ICLR)*, Jul. 2019.

[20] J. Wang, Z. Ren, T. Liu, Y. Yu, and C. Zhang, "QPLEX: Duplex Dueling Multi-Agent Q-Learning," in *Proc. of the International Conference on Learning Representations (ICLR)*, May 2021.

[21] T. Rashid, G. Farquhar, B. Peng, and S. Whiteson, "Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning," *in Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2020.

[22] L. Pan, L. Huang, T. Ma, and H. Xu, "Plan better amid conservatism: Offline multi-agent reinforcement learning with actor rectification," in *Proc. of the International Conference on Machine Learning (ICLR)*, Jul. 2022.

[23] X. Wang, H. Xu, Y. Zheng, and X. Zhan, "Offline multi-agent reinforcement llearning with implicit global-to-local value regularization," *in Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2023.

[24] Y. Yang, X. Ma, C. Li, Z. Zheng, Q. Zhang, G. Huang, J. Yang, and Q. Zhao, "Believe what you see: Implicit constraint approach for offline multi-agent reinforcement learning," *in Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2021.

[25] X. Wang and X. Zhan, "Offline multi-agent reinforcement learning with coupled value factorization," in *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Jun. 2023.

[26] J. Shao, Y. Qu, C. Chen, H. Zhang, and X. Ji, "Counterfactual conservative q learning for offline multi-agent reinforcement learning," *in Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2023.

[27] M. Wen, J. Kuba, R. Lin, W. Zhang, Y. Wen, J. Wang, and Y. Yang, "Multi-agent reinforcement learning is a sequence modeling problem," *in Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2022.

[28] Y. Yuan, X. Weng, Y. Ou, and K. M. Kitani, "Agentformer: Agent-aware transformers for socio-temporal multi-agent forecasting," in *Proc. of the International Conference on Computer Vision (CVPR)*, Jun. 2021.

[29] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *in Proc. of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec. 2020.

[30] J. Jiang and Z. Lu, "Online tuning for offline decentralized multi-agent reinforcement learning," in *Proc. of the AAAI Conference on Artificial Intelligence*, Feb. 2023.

[31] P. Barde, J. Foerster, D. Nowrouzezahrai, and A. Zhang, "A model-based solution to the offline multi-agent reinforcement learning coordination problem," in *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May. 2024.

[32] J. Jiang and Z. Lu, "Offline decentralized multi-agent reinforcement learning." in *Proc. of the European Conference on Artificial Intelligence (ECAI)*, Apr. 2023.