석 사 학 위 논 문
Master's Thesis

보상반환값과 행동만이 주어진 상황에서의 순차적 의사결정

Sequential Decision Making with only Return and Action

2023

성 해 빈 (成 海 彬 Seong, Haebin)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

석 사 학 위 논 문

보상반환값과 행동만이 주어진 상황에서의 순차적 의사결정

2023

성 해 빈

한 국 과 학 기 술 원

AI대학원

보상반환값과 행동만이 주어진 상황에서의 순차적 의사결정

성 해 빈

위 논문은 한국과학기술원 석사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2023년 6월 9일

심사위원장	황 성 주	[illegible](인)
심 사 위 원	양 은 호	[illegible](인)
심 사 위 원	신 진 우	[illegible](인)

Sequential Decision Making with only Return and Action

Haebin Seong

Advisor: Sung Ju Hwang

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Master of Science in Artificial Intelligence

Seoul, Korea
June 9, 2023

Approved by

[illegible]

Sung Ju Hwang
Professor of Graduate School of AI

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

MAI

성해빈. 보상반환값과 행동만이 주어진 상황에서의 순차적 의사결정. AI 대학원 . 2023년. 17 쪽. 지도교수: 황성주. (영문 논문)
Haebin Seong. Sequential Decision Making with only Return and Action. Graduate School of AI . 2023. 17 pages. Advisor: Sung Ju Hwang. (Text in English)

초 록
최근의 트랜스포머 구조의 성공은 순차적 모델링에서 우수한 성능을 보여주었으며, 이로 인해 의사결정 트랜스포머와 같이 순차적 의사결정 및 강화학습을 포함한 다양한 분야에서 트랜스포머를 적용하는 접근 방법들이 등장하였다. 마르코프 의사결정 과정이라는 순차적 의사결정 및 강화학습의 표준적인 문제 설정이 상태, 행동 및 보상의 전이 순서에 대한 정보를 필요로 하지만, 이러한 정보들이 현실세계 문제들을 푸는 데 있어 항상 접근 가능하지는 않다. 본 논문에서는 실제 상황에서의 적용을 보다 용이하게 하기 위해 마르코프 의사결정 과정의 조건을 대폭 완화한 의사결정에 대한 새로운 문제 설정을 제안한다. 의사결정 트랜스포머의 접근 방법을 확장하여 우리가 제안한 새로운 문제 설정에서 트랜스포머의 시퀀스 모델링 능력을 활용하는 의사결정 방법을 제안한다. 또한, 불확실성 모델링과 시퀀스 생성을 이용한 목표 지향적인 액티브 러닝을 가능하게 하는 프레임워크도 제안한다.

핵 심 낱 말 순차적 의사결정, 강화 학습, 의사결정 트랜스포머, 트랜스포머 구조, 지피티 구조, 자기주도학습, 불확실성 모델링, 액티브 러닝, 실험계획법

Abstract
As recent success of transformer architectures have shown superior performance in sequence modeling, several approaches have been proposed to apply transformers in various fields, including sequential decision-making and reinforcement learning, such as the prior work on Decision Transformers. However, Markov Decision Processes (MDPs), the standard problem setting in sequential decision making and reinforcement learning, require information on the transition sequence of state, action, and reward. This information is not always available in real-world problems. In this paper, we propose a new problem setting for decision making, which is a relaxation of the MDP that requires fewer conditions, thus making it easier to apply in many real-world situations, such as robotic control or experimental design. By extending the approach used in Decision Transformers, we suggest a decision making method that leverages the sequence modeling power of transformers in this new problem setting. Additionally, we propose an active learning framework that could enable goal-oriented active learning in this new problem setting, using uncertainty modeling and sequence generation.

Keywords Sequential Decision Making, Reinforcement Learning, Decision Transformer, Transformer Architecture, GPT Architecture, Self-Supervised Learning, Uncertainty Modeling, Active Learning, Experimental Design

Contents

Chapter 1. Introduction

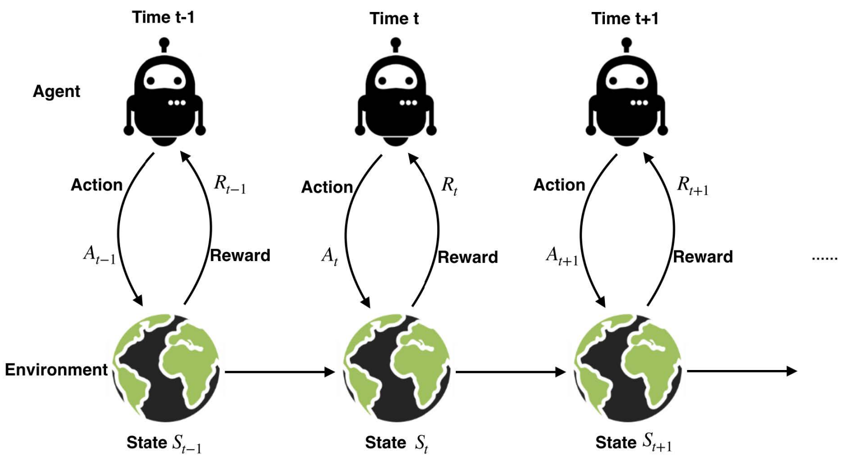


Figure 1.1: Illustration of Markov Decision Process.

The advance of deep learning and reinforcement learning (RL) made it possible to solve and even outperform human experts in many real-world sequential decision making problems [16, 20, 22]. Also, recent work has shown that the transformer architecture [21] is highly effective in sequential modeling tasks through capturing high-dimensional semantic concepts at scale, enabling zero-shot language modeling [17, 2] and out-of-distribution image recognition [6] and generation [18]. This success of transformers extends their application to various fields, such as sequential decision making and RL, as done in the prior work on Decision Transformers [3] or Trajectory Transformers [10]. These approaches have shown that conventional RL algorithms can be replaced with generative sequence modeling with transformers, which is a more flexible and powerful approach.

However, the condition of Markov Decision Processes (MDP), which is the standard problem setting in sequential decision making and reinforcement learning, is somewhat restrictive. MDP requires information on the transition sequence of state, action, and reward. This information is not always available in real-world problems. For example, in robotic control or experiment design, it is notoriously hard to manually formulate a task-specific reward [1], and state observations also need actual sensors which usually come with expensive costs. Due to the difficulty of reward formulations, there are even approaches that try to learn rewards from human preferences, which have recently shown great potential on language modeling in ChatGPT [4, 15]. In these cases, we may not be able to apply the standard RL algorithms, and relaxing the condition of MDP can be useful to express these problems.

In this paper, we propose a new problem setting for sequential decision making, which is a relaxation of the MDP that requires fewer conditions, thus making it easier to be applied in many real-world situations. Also by extending the approach used in Decision Transformers, we introduce a decision making method that leverages the sequence modeling power of transformers in this new problem setting. Additionally, we propose an active learning framework that enables goal-oriented active learning in this new problem setting, using uncertainty modeling and sequence generation. Finally, we conducted experiments simulating our proposed methods on a synthetic environment. Before elaborating the proposals, we briefly review the prior works related to it.

Chapter 2. Related Works

2.1 Sequential Decision Making & Markov Decision Process

Sequential decision making refers to the task of making a series of decisions over time, especially when the earlier decision affects the outcome of subsequent decisions. Sequential decision making is very common in real-world situations, such as robotics, finance, autonomous driving, game playing, or even planning a trip.

Markov decision process (MDP) is a popular framework used to model sequential decision making problems. In MDP, a decision making process is described by a sequence of states, actions, and rewards. At each time step, an action is applied to the environment. According to the action, the environment's current state transitions to the next state, and a reward is generated by the environment to express the correctness of the selected action for the current state. Another condition of MDP is that the next state and reward only depend on the current state and action, not on the previous states and actions. This condition is called the Markov property, and it is a very strong assumption that is not always true in real-world situations, although it is a very useful assumption to make the problem easier to solve.

2.2 Reinforcement Learning & Offline Reinforcement Learning

Reinforcement learning (RL) is a popular solution of MDP that uses a trial-and-error approach to learn the optimal policy. The agent tries to collect samples by interacting with the environment, regarding all the choices and experiences that the agent has collected so far. The agent's goal is to find the optimal policy that maximizes the expected cumulative reward (also known as *return*) over time. With the combination of conventional theoretical concepts of reinforcement learning and function estimation of deep learning, deep RL has become a popular and powerful method in solving MDPs and making decisions in real-world problems, often outperforming humans [16, 20, 22].

Offline Reinforcement Learning [14] represents a more challenging variant of RL that relies on a static dataset for learning, rather than interacting and collecting data directly from the environment. This approach could be beneficial in real-world scenarios where engaging with the environment is cost-prohibitive, such as in the domains of robotics or healthcare. Given that the context of offline RL can be handled in a manner similar to supervised learning, rather than standard online RL, sequence modeling methods within RL are often more oriented towards offline RL instead of online RL. [3, 10]

2.3 Transformers

The transformer architecture [21] is a sequence modeling architecture that uses self-attention to capture long-range dependencies. It is currently the most successful architecture in many fields of machine learning due to its powerful sequence semantic capturing ability and scalability, leading to successful applications in natural language processing [17, 2], computer vision [6], time series prediction [23], or even protein folding applications such as AlphaFold [11].

The Generative Pre-trained Transformer (GPT) architecture [17] is a special version of transformers, using only transformer decoders. The GPT architecture has shown great success in autoregressive generative sequence modeling, such as OpenAI's GPT-3 [2]. The GPT architecture is also used in our work, as it is a powerful sequence modeling architecture also proven to be capable of sequential decision making and reinforcement learning in the prior work of Decision Transformers [3] and Trajectory Transformers [10].

2.4 Uncertainty Modeling

Uncertainty modeling [8] is a process of quantifying the uncertainty of lack of knowledge of the model's prediction, due to noisy, limited, ambiguous data. For real-world applications, uncertainty modeling can aid decision making in high risk fields such as medical image analysis for autonomous vehicle control. Uncertainty modeling can also help in machine learning situations where we want to make a decision based on what we don't know, such as reinforcement learning and active learning. Monte Carlo Dropout [7] and Neural Processes [12] are two methods of uncertainty modeling that are covered in this paper.

2.5 Active Learning

Active learning [19] is a machine learning approach that involves an iterative process of selecting the most informative samples from a large unlabeled dataset and requesting labels for those samples. The goal of active learning is to train a model with high performance while minimizing the amount of labeled data required for training. Uncertainty modeling can be used as a measure of informativeness, and is often used in active learning when selecting samples.

2.6 Self-Supervised Learning

Self-supervised learning [9] is a machine learning paradigm where a model learns from unlabeled data without explicit human supervision. As its name suggests, self-supervised learning creates an auxiliary task and generates labels from the unlabeled data, and learns it in a supervised manner. One of the most successful self-supervised learning methods is generative modeling, where the model learns to generate new samples that are similar to the original data. Generative modeling is used in training GPT language models [17, 2] and also Decision Transformers [3] that use GPT models. Another popular self-supervised learning method is masking and reconstructing the data. This method is used in BERT [5] and other similar models.

Chapter 3. Approach

In this section, we provide details of our problem definition, Return and Action Only Process (RAOP). Then, we suggest a solution method for RAOP with self-supervised reward predictor and goal-oriented autoregressive sequence modeling based on Transformers. Finally, we propose a goal-oriented active learning framework for RAOP via uncertainty modeling, inspired by the goal-oriented sequence generation capability of GPT.

3.1 Problem Definition

3.1.1 Markov Decision Process

To understand our problem definition, we first take a look into Markov Decision Process (MDP) which is a standard framework for sequential decision making, since it is comparable to our setting. MDP is defined as a tuple of $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability. MDPs also should satisfy the Markov Property, which is a condition requiring the next state and reward to only depend on the current state and action, and not on the previous state and actions. Markov property can be mathematically expressed as $\mathcal{P}(s_t|s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2}, \cdots, s_0, a_0) = \mathcal{P}(s_t|s_{t-1}, a_{t-1})$. The goal of the agent is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected sum of rewards $\mathbb{E}_{\tau\sim\pi}\left[\sum_{t=1}^{T} \mathcal{R}(s_t, a_t)\right]$, where $\tau = [(s_1, a_1), \cdots, (s_T, a_T)]$ is the trajectory of state-action pairs induced by the policy π. The sum of rewards is also called as the name of *return*, which is $\mathfrak{R}(s_{1:T}, a_{1:T}) = \sum_{t=1}^{T} \mathcal{R}(s_t, a_t)$. *Return-to-go* refers to the sum of rewards starting from the current time step t', which is $\mathfrak{R}(s_{t':T}, a_{t':T}) = \sum_{t=t'}^{T} \mathcal{R}(s_t, a_t)$. The prior work of Decision Transformers [3] and Trajectory Transformers [10] showed that autoregressive generation of sequences including return-to-go can simulate reinforcement learning.

3.1.2 Return and Action Only Process (RAOP)

We propose a new problem setting which is a simplification and relaxation of Markov Decision Process that could enable much broader application of real-world problems, which we call **Return and Action Only Process** (RAOP). RAOP is defined as a tuple of $(\mathcal{A}, \mathfrak{R})$, where $\mathcal{A}$ is the action space and $\mathfrak{R} : \mathcal{A}^T \to \mathbb{R}$ is the return function, and T is the trajectory length. The goal of the agent is to find a policy $\pi : \mathcal{A} \to \mathcal{A}$ that maximizes the expected return $\mathbb{E}_{\tau^a\sim\pi}[\mathfrak{R}(a_{1:T})]$, where $\tau^a = [a_1, a_2, \cdots, a_T]$ is the trajectory without states (only actions). Note that the state space $\mathcal{S}$ and the transition probability $\mathcal{P}$ are removed from the definition of MDP, and the reward is no longer available for each time step, other than the single return value $\mathfrak{R}(a^i_{1:T})$ that is generated only when every sequence of actions in the trajectory are executed. Due to easier application of sequence modeling, we consider an offline setting of RAOP in this paper, so we do not have direct access to the RAOP environment and only have to learn from the given the action trajectory dataset $\mathcal{D} = \{(a^i_{1:T}, \mathfrak{R}(a^i_{1:T}))\}_{i=1}^{N}$, where $a^i_{1:T}$ is the action sequence and $\mathfrak{R}(a^i_{1:T})$ is the return of the action sequence. Also we would like to emphasize that RAOP does not require anything like the Markov Property, which is rather a convenient but often unrealistic constraint in MDP. Thus RAOP can model tasks better than MDP that have dependencies on past states and actions, which is quite a common situation in the real-world but is tricky to express in MDP.

3.2 Self-Supervised Reward Predictor for RAOP

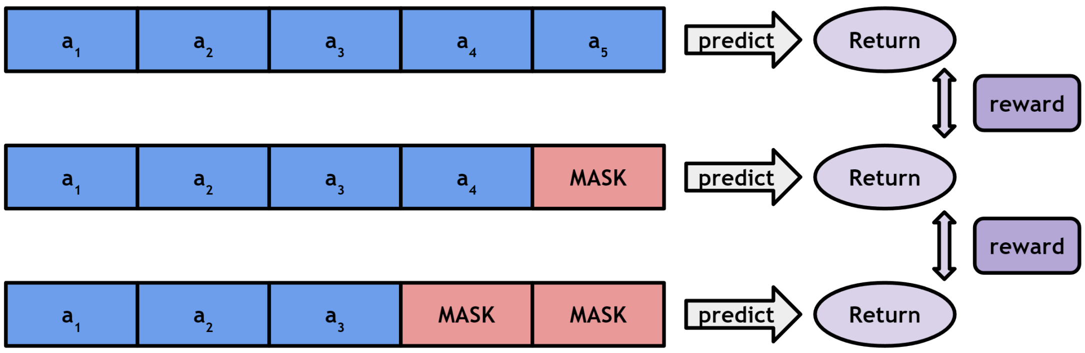


Figure 3.1: Self-Supervised Reward Predictor for RAOP.

In this section, we assume that the return of RAOP is latently a sum of rewards for each action in the trajectory sequence, which is a common assumption in RL. Based on that assumption, we propose to train a reward predictor in a self-supervised manner. For better understanding, before going into details about the self-supervised reward predictor, we could consider a simple and naïve return predictor for RAOP trained for the following loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{a^i_{1:T}, \mathfrak{R}(a^i_{1:T}) \in \mathcal{D}} \left[(f_\theta(a^i_{1:T}) - \mathfrak{R}(a^i_{1:T}))^2 \right]. \tag{3.1}$$

where $f_\theta(\cdot)$ is a neural network parameterized by θ, which takes the action sequence $a^i_{1:T}$ as an input and outputs the expected return of the input action sequence. The neural network is trained to minimize the expected Mean Square Error (MSE) between the predicted return and the ground-truth return. However, we found that this naïve application is too trivial to learn meaningful representations for predicting the rewards. To overcome this limitation, we propose to mask some parts of the action sequence data and make the neural network predict the return of the masked action sequence. Intuitively, we can estimate return-to-go values from return values with self-supervised learning, thus making room for simulating RL with return-to-go values as shown in [3, 10]. Specifically, our masked self-supervised learning for the reward prediction for RAOP is as follows:

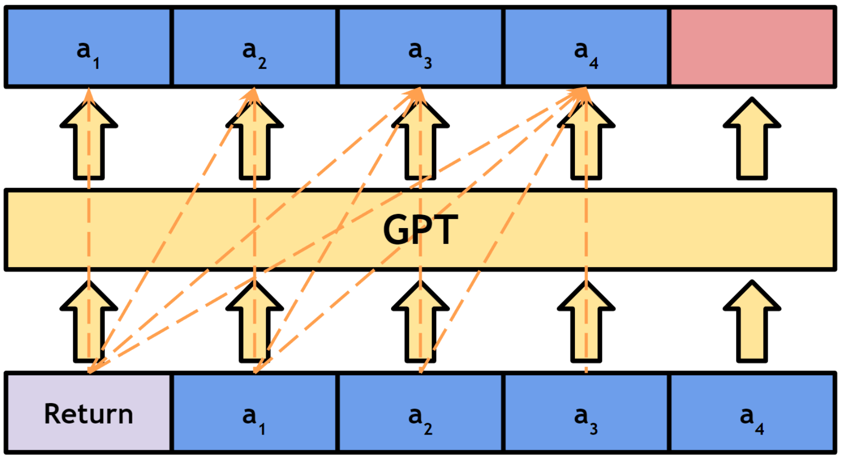
$$\begin{aligned}\texttt{mask}(a^i_{1:T}) = \{\ &([a_1, \ldots, a_{T-1}, \text{MASK}], \mathfrak{R}(a^i_{1:T})),\ ([a_1, \ldots, a_{T-2}, \text{MASK}, \text{MASK}], \mathfrak{R}(a^i_{1:T})),\ \ldots, \\ &([a_1, a_2, \text{MASK}, \ldots, \text{MASK}], \mathfrak{R}(a^i_{1:T})),\ ([a_1, \text{MASK}, \ldots, \text{MASK}], \mathfrak{R}(a^i_{1:T}))\ \}.\end{aligned} \tag{3.2}$$

$$\mathcal{D}_{\text{MASK}} = \mathcal{D} \cup \left(\bigcup_{i=1}^{N} \texttt{mask}(a^i_{1:T}) \right), \text{ where } |\mathcal{D}_{\text{MASK}}| = N + N \times (T-1) = N \times T, \tag{3.3}$$

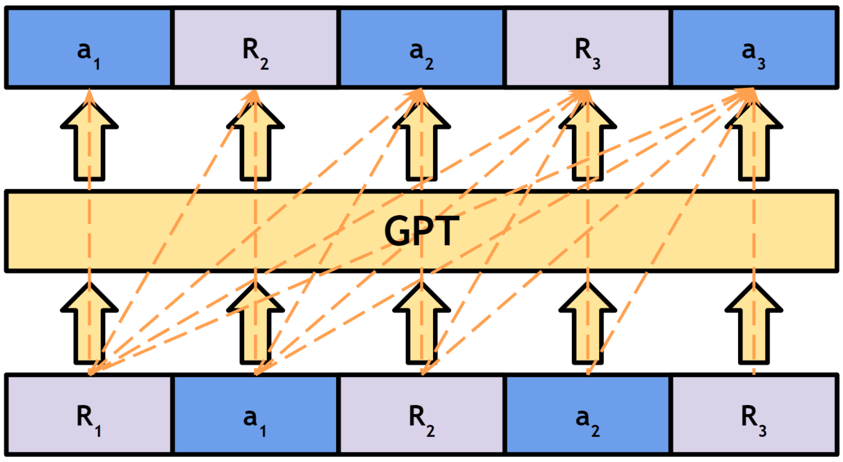
$$\mathcal{L}_{\text{MASK}}(\theta) = \mathbb{E}_{a^i_{1:T}, \mathfrak{R}(a^i_{1:T}) \in \mathcal{D}_{\text{MASK}}} \left[(f_\theta(a^i_{1:T}) - \mathfrak{R}(a^i_{1:T}))^2 \right], \tag{3.4}$$

where $\texttt{mask}(\cdot)$ is a function that masks the input action sequence from the last action a_T to the second action a_2 with the MASK token, $\mathcal{D}_{\text{MASK}}$ is the augmentation of the original trajectory dataset $\mathcal{D}$ by the masked trajectory dataset $\bigcup_{i=1}^{N} \texttt{mask}(a^i_{1:T})$, and $\mathcal{L}_{\text{MASK}}$ is the proposed masked loss of self-supervised reward predictor for RAOP. Figure 3.1 depicts the overall process for training and utilizing the self-supervised reward predictor.

3.3 Solving RAOP with Goal-Oriented Autoregressive Sequence Modeling via Transformers



(a) **GPT-Raa :** $g_\phi([\mathfrak{R}(a_{1:T}), a_1, a_2, \cdots, a_T])$



(b) **GPT-RaR :** $g_\phi([\mathfrak{R}(a_{1:T}), a_1, \hat{\mathfrak{R}}(a_{2:T}), a_2, \cdots, \hat{\mathfrak{R}}(a_T), a_T])$

Figure 3.2: Autoregressive Goal-Oriented Sequence Generator. There are two settings: **a) GPT-Raa:** predict the next action given the previous actions and the return, **b) GPT-RaR:** predict the next action given the previous actions and the return-to-go series estimated by the self-supervised reward predictor. The orange dotted arrows denote masked self-attention that enables the model to generate the next action without seeing the future action, which is commonly used in GPT autoregressive sequence generation.

In this section, we solve the RAOP problem with goal-oriented sequence modeling via transformers, especially with autoregressive sequence generation using the GPT architecture. There are two formulations that we consider: **a) GPT-Raa:** predict the next action given the previous actions and the return, **b) GPT-RaR:** predict the next action given the previous actions and the return-to-go series estimated by the self-supervised reward predictor.

For simplicity, we first consider **GPT-Raa**, which does not utilize the self-supervised reward predictor we trained in 3.2. For **GPT-Raa**, we train the model g_ϕ as follows:

$$\mathcal{L}_{\mathrm{Raa}}(\phi) = \mathbb{E}_{(a^i_{1:T}, \mathfrak{R}(a^i_{1:T})) \in \mathcal{D}} \left[(g_\phi([\mathfrak{R}(a_{1:T}), a_1, a_2, \cdots, a_T]) - a^i_{1:T})^2 \right] \tag{3.5}$$

where $g_\phi(\cdot)$ is a GPT transformer network parameterized by ϕ, which takes the sequence of return and action as input and outputs action sequence prediction. **GPT-Raa** just uses the given dataset as it is, and performs sequence modeling with given return and action sequence.

Next, we consider **GPT-RaR**, which utilizes the self-supervised reward predictor f_θ we trained in 3.2. For **GPT-RaR**, we train the model g_ϕ as follows:

$$\hat{\mathfrak{R}}(a_{t:T}) = \mathfrak{R}(a_{1:T}) - f_\theta([a_1, a_2, \cdots, a_{t-1}, \mathrm{MASK}, \mathrm{MASK}, \cdots, \mathrm{MASK}]) \tag{3.6}$$

$$\mathcal{L}_{\mathrm{RaR}}(\phi) = \mathbb{E}_{(a^i_{1:T}, \mathfrak{R}(a^i_{1:T})) \in \mathcal{D}} \left[(g_\phi([\mathfrak{R}(a_{1:T}), a_1, \hat{\mathfrak{R}}(a_{2:T}), a_2, \cdots, \hat{\mathfrak{R}}(a_{T:T}), a_T]) - a^i_{1:T})^2 \right] \tag{3.7}$$

where equation 3.6 refers to the return-to-go estimation using the self-supervised reward predictor f_θ defined in 3.4. $\mathfrak{R}(a_{1:T})$ denotes the real return given by the dataset, and $\hat{\mathfrak{R}}(a_{t:T})$ denotes the return-to-go estimated by the predictor f_θ. For **GPT-RaR**, we include the estimated return-to-go $\hat{\mathfrak{R}}(a_{t:T})$ into the sequence of inputs into the GPT model, similar to the prior work of [3].

The two formulations are graphically illustrated in Figure 3.2. In both formulations, the model targets to minimize the difference between input action and generated action, following the prior work of [3]. Cross-entropy loss is used for discrete action space and mean-squared error loss is used for continuous action space. Note that the masked self-attention architecture of GPT *enables the model to generate the next action without seeing the future actions*, which is a crucial property for sequential decision making and why it is acceptable to give inputs of $g_\phi(a^i_{1:T}, \mathfrak{R}(a^i_{1:T}))$ for estimating $a^i_{1:T}$.

3.4 Goal-Oriented Active Learning for RAOP via Uncertainty Modeling

Algorithm 1 Goal-Oriented Active Learning in RAOP

```
Input: Train dataset $\mathcal{D}_{train}$, Goal $\mathcal{G}$, Learning rate $\lambda_f, \lambda_g$, Loss function $\mathcal{L}_f, \mathcal{L}_g$, Number of acquisition $m, n$
Output: Return predictor $f_\theta$, Sequence generator $g_\phi$

1:  $reached \leftarrow False$
2:  for each active learning step do
3:      Randomly initialize $\theta$, $\phi$
4:      for each gradient step do
5:          Draw a mini-batch $\mathcal{B}$ from $\mathcal{D}_{train}$
6:          $\theta \leftarrow \theta - \lambda_f \nabla_\theta \mathcal{L}_f(\theta, \mathcal{B})$
7:          $\phi \leftarrow \phi - \lambda_g \nabla_\phi \mathcal{L}_g(\phi, \mathcal{B})$
8:      end for
9:      Draw $m$ samples as $\mathcal{M}$ from distribution of sequence generator conditioned by goal
            $\triangleright \mathcal{M} \sim g_\phi(\mathcal{G})$
10:     Compute uncertainty of each sample in $\mathcal{M}$ using the return predictor
            $\triangleright u(\mathcal{M}) = \mathrm{Var}[f_\theta(\mathcal{M})]$
11:     Select $n$ samples as $\mathcal{N}$ with highest uncertainty from $\mathcal{M}$
            $\triangleright \mathcal{N} \leftarrow \arg\max_{\mathcal{M}}[u(\mathcal{M})]$
12:     Label $\mathcal{N}$ and append them to $\mathcal{D}_{train}$
13:     if sample with label $\mathcal{G}$ exists in $\mathcal{D}_{train}$ then
14:         $reached \leftarrow True$
15:     end if
16: end for
17: return $\theta, \phi, reached$
```

The standard framework of active learning [19] is to select the most informative data points from the unlabeled dataset, add them to the labeled dataset, and then retrain the model with the new labeled dataset. However, the selection of data points from the unlabeled dataset usually involves exhaustive search of the whole unlabeled dataset, which is not efficient in real-world problems especially when we don't even have an unlabeled dataset and we have to select samples from the search space. Also, usually the samples are selected based on the uncertainty of the model, which is not always the best choice when we want to select samples that could help find a specific goal.

We thus suggest a goal-oriented active learning framework for RAOP in algorithm 1 that can generate samples from the search space without having to exhaustively search the dataset, and select the most informative samples that could help find a specific goal. For this active learning framework, we add uncertainty modeling (such as Monte Carlo Dropout [7] or Neural Processes [12]) on the return predictor and sequence generator, and utilize the uncertainty to sample from the search space and decide most informative samples. The return predictor f_θ and sequence generator g_ϕ are described in 3.2 and 3.3.

Chapter 4. Experiments

In this section, we now experimentally verify the proposed method using a synthetic environment which is similar to a "*quizshow*" setup.

4.1 Experimental Setup

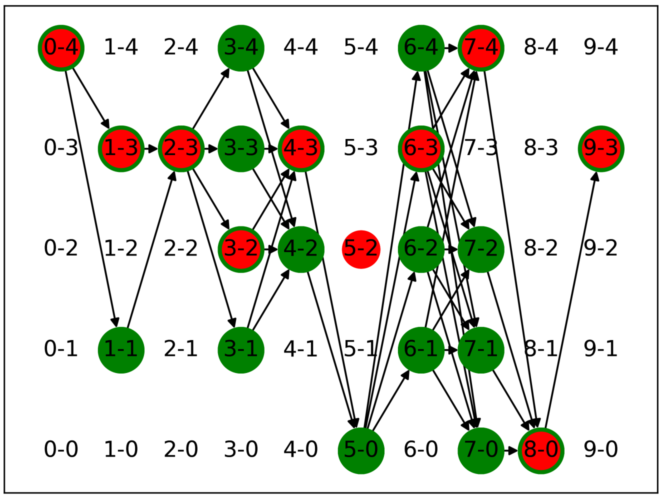


Figure 4.1: Example setting of the *"quizshow"* synthetic environment. In this illustrated example, the number of questions is $T = 10$ and the number of possible answers is $N = 5$. For each grid number, the first number represents the question number and the second number represents the answer number. So 4-1 stands for answer number 1 in question number 4. The green circles represent the correct answer and the red circles represent the input answer sequence $a^i_{1:T}$ that the agent executed. By observing the green circles, we can see that in this example, the first question has only 1 correct answer (4) and the second question has 2 correct answers (1,3). Since the input answer sequence is correct until the 5th question and wrong in the 6th question, even though the agent got correct answers for 7th to 10th questions, the agent cannot earn any points after the 6th question. So the return score $\mathfrak{R}(a^i_{1:T})$ of this input answer sequence is 5.

We first elaborate the synthetic environment used for our experiments. In this environment, our objective is to obtain consecutive answers correctly in a series of questions. $a^i_{1:T}$ is the sequence of answers, and $\mathfrak{R}(a^i_{1:T})$ is the score of the sequence of answers. Starting from the first question, the agent has to get the correct answer consecutively for each question, earning one point for each correct question. If the agent gets any question wrong, then it cannot earn any points after the wrong answer, even if the subsequent answers are correct. This attribute is contrary to the Markov property, where previous actions should not affect current decisions. When the number of questions is T, and the number of possible answers in each question is N, each question can have 1 to $N-1$ correct answers. Figure 4.1 shows an example instantiation of the quizshow environment. By default, we use $T=10$ and $N=5$ for this experiment. We generate 1000 samples so that the score distribution of the samples is evenly balanced from 0 to K. We split the dataset into 8000 and 2000 samples each for the training set and test set.

4.2 Return Predictor

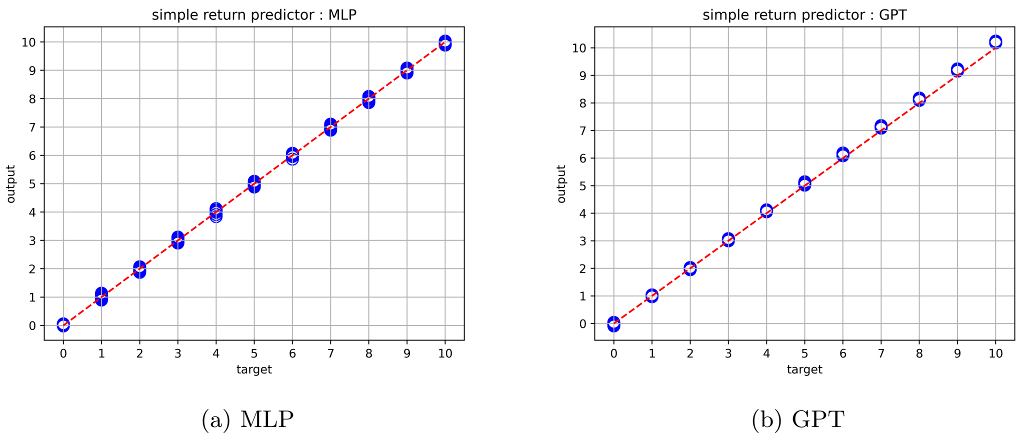


Figure 4.2: Test set scatter plot of simple return predictor f_θ. Scatter dots are drawn as blue emtpy circles. The red dotted line is drawn by the formula of $y=x$. The circle dots overlapping clearly in GPT shows that the prediction of GPT is more consistent than MLP.

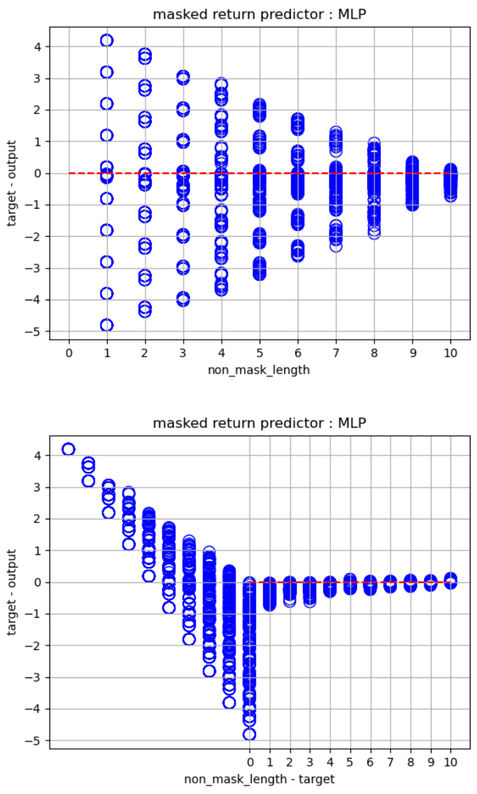
	MLP	GPT
MSE	0.0029	0.0141

Table 4.1: MSE of simple return predictor f_θ. Lower is better.

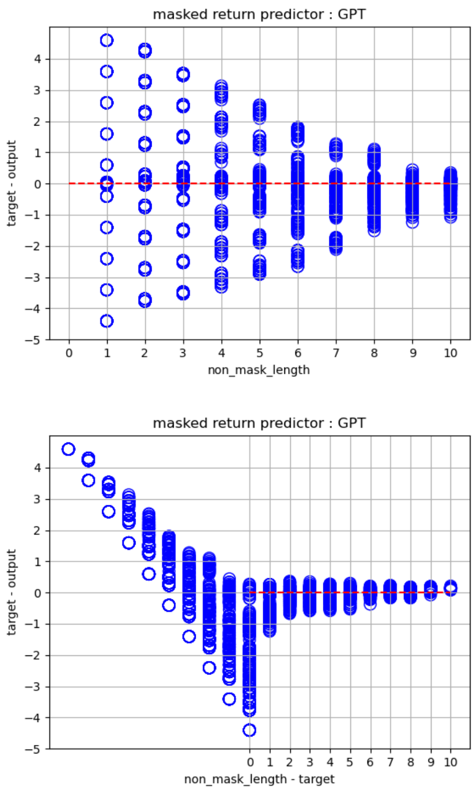
To briefly assess the difficulty of the task, we first train a simple return predictor f_θ as described in 3.1. We compare Multi-Layer Perceptron(MLP) and GPT architectures for the predictor. Note that there is no autoregressive sequence generation when we use GPT as the predictor, and the output sequence of GPT is connected to a linear head to infer the return score. Each model is trained for 100 epochs, and the batch size is 512. Figure 4.2 and Table 4.1 illustrates the scatter plot and MSE loss for test set evaluation. The results show that the MLP architecture shows better performance than GPT, but the prediction is slightly more inconsistent than GPT.

4.3 Self-Supervised Reward Predictor

Now we train a self-supervised reward predictor f_θ as described in 3.4. For each sample of $\mathcal{D}_{train}$, we generate and augment $T-1$ samples by masking the action with mask length 1 to $T-1$. We use the same MLP and GPT architectures for the predictor. Note that there is no autoregressive sequence generation when we use GPT as the predictor. Each model is trained for 30 epochs, and the batch size is 512. Figure 4.3 illustrates the scatterplot for test set evaluation. We can see not much difference in using MLP and GPT as the self-supervised reward predictor. The first row of Figure 4.3 shows that the prediction is more accurate as the action sequence mask is shorter. The second row of Figure 4.3 shows that the prediction is accurate when the result is actually predictable (x-axis bigger than 0).



(a) MLP



(b) GPT

Figure 4.3: Scatter plot of self-supervised reward predictor f_θ. Scatter dots are drawn as blue emtpy circles. The x-axis of the first row is *non_mask_length*, which is the length of the action sequence that is not masked. The x-axis of the second row is *non_mask_length - target*. If the *non_mask_length - target* value is is bigger than 0, the target is accurately predictable due to the environment setting. The y-axis of both rows is the difference of prediction and output.

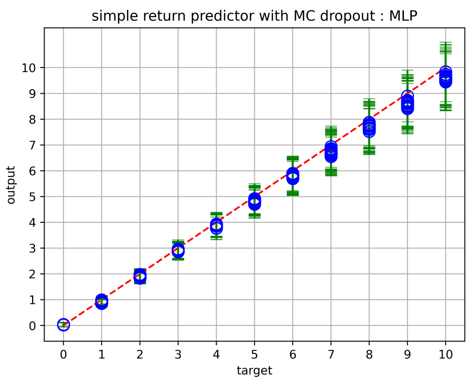
4.4 Goal-Oriented Action Sequence Generator

	MLP	**GPT-Raa**	**GPT-RaR**
CE	5.8125	5.4315	5.3590
Online Score	7/16	11/16	14/16

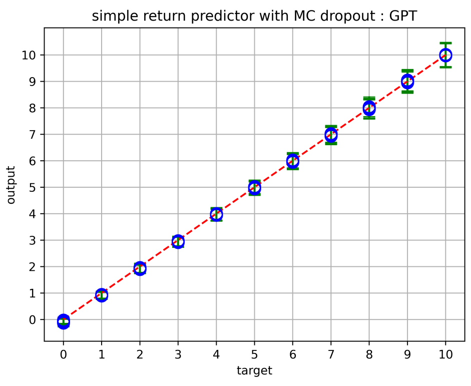
Table 4.2: CE and Online Score of action sequence generator. Lower CE and higher Online Score is better. Online score is measured on $T + 1 = 16$ samples, which represent score 0 to T.

As described in 3.3, we have 2 settings of the sequence generator: **GPT-Raa, GPT-RaR**. In **GPT-Raa**, we feed the sequence of return and actions as the input of the GPT model g_ϕ. In **GPT-RaR**, we estimate the return-to-go value with the self-supervised reward predictor f_θ trained in the previous section 4.3, and feed it as the input of the sequence generator GPT model g_ϕ. For comparison, we also train a simple classification model with MLP. We train the models for 200 epochs, and the batch size is 512. Since the environment has a discrete action space, we use cross entropy (CE) loss for training. We also score online performance, which measures how many of the sequences generated were actually sequences that have return scores of 0 to T. For better comparison, we make the task harder by using an environment setting of $T = 15$ for the sequence generator experiment. We would like to emphasize that the online score is much more meaningful than the CE loss, because the CE loss does not necessarily mean that the generated action is incorrect, rather it only means that the generated action is not identical to the action in the dataset. The online score can mitigate this limitation, since it can check whether the generated action is actually correct or not by interacting with the environment. Table 4.2 illustrates the CE loss and online score for MLP, **GPT-Raa, GPT-RaR**. The results show that the GPT model perform better than the MLP model presumably due to its sequence modeling capability. Also, we can observe that the return-to-go value generated with the self-supervised reward predictor f_θ in **GPT-RaR** actually helped training quite substantially, by boosting the online score up to 14/16.

4.5 Uncertainty Modeling



(a) MLP



(b) GPT

Figure 4.4: Test set scatter plot of simple return predictor f_θ with uncertainty modeling via MC dropout shown as errorbars. Scatter dots are drawn as blue emtpy circles. The red dotted line is drawn by the formula of $y = x$. The green bar denotes the standard variance uncertainty of the prediction.

	MLP	GPT
NLL	0.1209	-0.4083
RCE	0.8309	0.7228

Table 4.3: NLL, RCE of simple return predictor f_θ with uncertainty modeling via MC dropout. Lower is better.

To check the capability of uncertainty modeling, we model uncertainty for the simple return predictor by applying Monte Carlo dropout(MC dropout) [7]. Again, we compare MLP, GPT architectures. Each model is trained for 500 epochs, and the batch size is 512. We draw 1000 monte carlo samples for evaluation. Table 4.3 and Figure 4.4 illustrate the negative log loss (NLL), regression calibration error (RCE) [13], and scatterplot with errorbars for evaluation. Our results show that with uncertainty modeling via MC dropout, the GPT architecture shows more outstanding performance than MLP, contrary to the results with no uncertainty modeling. In both MLP and GPT, the uncertainty gets higher when the score is higher. This is likely due to the nature of the environment, since the higher the score, the more the possible answer sequences exist. We can see that GPT shows a much more accurate estimate in both mean and standard deviation than MLP.

Chapter 5. Conclusion

In this paper, we proposed a new problem setting of sequential decision making which is more relaxed than Markov Decision Processes and thus more applicable to real-world problems. Then we proposed a decision making method that leverages the sequence modeling power of transformers in this new problem setting. The experiment results showed some potential of our approach with the powerful sequence modeling capability of GPT. We also proposed an active learning framework on this new problem setting, but experiments on the active learning framework still remain to be conducted. We have checked that the uncertainty modeling performance of MC dropout is not that accurate in our setting. Better uncertainty modeling would be essential for the active learning framework to work well.

For future work, we plan to enhance uncertainty modeling by possibly better uncertainty modeling methods such as Neural Processes, and integrate it into the active learning framework to conduct experiments on active learning. We also plan to conduct experiments on real-world problems, such as the experimental design task of semiconductor manufacturing processes, which represents a task that is impossible or too expensive to acquire state or reward transition information with current technology, thus impossible to model with Markov Decision Processes (MDP), but possible to model with Return and Action Only Processes (RAOP).

Bibliography

[1] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *CoRR*, abs/1806.06877, 2018.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[3] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[4] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[7] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA, 20–22 Jun 2016. PMLR.

[8] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, Muhammad Shahzad, Wen Yang, Richard Bamler, and Xiao Xiang Zhu. A survey of uncertainty in deep neural networks, 2022.

[9] Jie Gui, Tuo Chen, Qiong Cao, Zhenan Sun, Hao Luo, and Dacheng Tao. A survey of self-supervised learning from multiple perspectives: Algorithms, theory, applications and future trends, 2023.

[10] Michael Janner, Qiyang Li, and Sergey Levine. Reinforcement learning as one big sequence modeling problem. In *ICML 2021 Workshop on Unsupervised Reinforcement Learning*, 2021.

[11] J. Jumper, R. Evans, A. Pritzel, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583–589, 2021.

[12] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *International Conference on Learning Representations*, 2019.

[13] Volodymyr Kuleshov, Nathan Fenner, and Stefano Ermon. Accurate uncertainties for deep learning using calibrated regression. *CoRR*, abs/1807.00263, 2018.

[14] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020.

[15] Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, Zihao Wu, Dajiang Zhu, Xiang Li, Ning Qiang, Dingang Shen, Tianming Liu, and Bao Ge. Summary of chatgpt/gpt-4 research and perspective towards the future of large language models, 2023.

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

[17] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[18] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8821–8831. PMLR, 18–24 Jul 2021.

[19] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B. Gupta, Xiaojiang Chen, and Xin Wang. A survey of deep active learning, 2021.

[20] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[22] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[23] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey, 2023.