Large Decision Models

Weinan Zhang
Shanghai Jiao Tong University
wnzhang@sjtu.edu.cn

Abstract

Over recent decades, sequential decision-making tasks are mostly tackled with expert systems and reinforcement learning. However, these methods are still incapable of being generalizable enough to solve new tasks at a low cost. In this article, we discuss a novel paradigm that leverages Transformer-based sequence models to tackle decision-making tasks, named large decision models. Starting from offline reinforcement learning scenarios, early attempts demonstrate that sequential modeling methods can be applied to train an effective policy given sufficient expert trajectories. When the sequence model goes large, its generalization ability over a variety of tasks and fast adaptation to new tasks has been observed, which is highly potential to enable the agent to achieve artificial general intelligence for sequential decision-making in the near future.

1 Background

Solving sequential decision-making tasks is one of the major objectives of artificial intelligence (AI) [Phillips-Wren and Jain, 2006]. Originating in the 1950s, AI techniques have been explored to solve various decision-making tasks like computer checkers, chess and other games [Schaeffer and Van den Herik, 2002]. After 70 years, AI techniques are now applied in various complex sequential decision-making tasks, including autonomous driving [Schwarting et al., 2019], multi-agent game bots [Vinyals et al., 2019], dexterous robot manipulation [Billard and Kragic, 2019], personalized recommender systems [Munemasa et al., 2018], real-time order dispatching [Li et al., 2019] etc., serving to optimize the utility or efficiency of the systems by making smart decisions.

From the perspective of technical paradigms, there are roughly three kinds of different techniques developed for sequential decision-making, namely, expert systems, reinforcement learning, and large decision models [Wen et al., 2022b].

Expert systems (ES) refer to the paradigm that human experts should first be knowledgeable about how to solve a particular sequential decision-making task, then implement the program to solve it with computers [Bohanec and Rajkovic, 1990]. For example, the design of heuristic search methods consists of the scoring function of each search node and the search algorithm [Churchill et al., 2012]; the mixed-integer programming (MIP) solution requires the human to thoroughly understand the quantitative relationship between the optimization objective and the variables as well as their constraints, then instantiate the corresponding MIP problem and use the solver to obtain the decision solution [Achterberg and Wunderling, 2013]. Despite their good explainability and controllability, the expert system methods are generally limited to the human experience and the ability to implement the experience knowledge into the solution. Further, it is common that for most sequential decision-making tasks, even the human expert cannot acquire all the valuable knowledge. Thus, expert system solutions are limited by human efforts and upper-bounded by the known human expertise.

Reinforcement learning (RL) refers to the learning-from-data methods for a sequential decision-making agent interacting with a dynamic environment [Sutton and Barto, 2018]. In each timestep of the interaction, the agent perceives the observation or state from the environment and computes the action (i.e., the decision) to deliver to the environment. Then the corresponding reward signal as feedback is sent to the agent, and the environment will transit to a new state and emit the new observation to the agent in the next timestep. The learning objective of the agent is to maximize the expected cumulative reward over the interaction episode. Powered by deep learning, deep reinforcement learning (DRL) has achieved appealing success during the last decade [Mnih et al., 2015], including the superhuman-level AI in Go and various games [Silver et al., 2016; Ye et al., 2020; Brown and Sandholm, 2019], healthcare [Yu et al., 2021], robotic control [Lee et al., 2020], real-time bidding based advertising [Jin et al., 2018] etc.

However, reinforcement learning methods generally suffer from the problems of low sample efficiency and poor generalization upon environmental change. First, as RL requires the agent to interact with the dynamic environment to obtain experience data, it is likely the sampled experience data does not help the policy improve. Thus, a variety of methods are proposed to improve the sample efficiency of RL, including exploration strategies, representation learning, environment modeling and planning, and policy transfer [Yu, 2018]. However, these methods do not fundamentally address the sample efficiency problem because of the interactive nature of RL. Second, the policy obtained from RL is highly sensitive to en-
environmental change, including the reward function and state dynamics. This is partly due to the Bellman optimality update in temporal difference learning or dynamic programming. In addition, the reward function is not necessarily well defined for each sequential decision-making task, which requires the researchers to carefully craft the reward function, but a suboptimal reward function would guide the policy to exploit its loopholes and yield an unintended policy. As a result, when facing a new sequential decision-making task, or the dynamic environment has made a remarkable change, the policy has to be (almost) trained from scratch, which substantially limits the applicability of RL in the real world.

In this article, we introduce large decision models (LDMs), a novel learning-based paradigm that is potential to address the above challenges that RL faces. We start from the origin of LDMs in offline RL, then discuss the development in online training, multi-agent settings as well as some typical applications.

## 2 Preliminaries

A general formulation of RL policy gradient is

\[
\theta \leftarrow \theta + \alpha \mathbb{E}_{(s,a) \sim \rho_m} \left[ m(s,a) \nabla_{\theta} \log \pi_\theta(a|s) \right],
\]

where \( \theta \) denotes the parameters of the learning policy \( \pi_\theta \), \( \mu \) is the behavior policy, \( \rho^\mu \) denotes the data distribution corresponding to \( \mu \)'s occupancy measure \( \rho^\mu \), \( \alpha \) is the learning rate. \( m(s,a) \) stands for an operator on the state-action pair \( (s,a) \). If \( m(s,a) \) is instantiated as \( Q(s,a) \) or \( A(s,a) \), then Eq. (1) corresponds to actor-critic methods. If \( m(s,a) \) is valued as the return right after \( (s,a) \) in a trajectory, then it corresponds to the REINFORCE method. In a multi-agent case, \( m(s,a) \) can be a credit assigner to the particular agent for its action \( a \) at the state \( s \) in a group cooperation scenario. An extreme case is when \( m(s,a) = 1 \) for all state-action pairs, then it corresponds to self-imitation of the behavior policy \( \mu \).

### Self-imitation offline or off-policy RL.

In offline or off-policy reinforcement learning, the self-imitation methods refer to reweighting the state-action pairs or resampling them from the batch or replay buffer according to value estimation, and then performing supervised learning [Peng et al., 2019; Wang et al., 2020; Chen et al., 2020]. Such a method enjoys the advantages of stable supervised learning from the reweighted (or sampled) state-action pairs and easy implementation. Generally, self-imitation would yield better efficiency if the batch experience data is more diverse. Srivastava et al. [2019] proposed a framework called upside-down RL, in which, instead of learning any action value function \( Q(s,a) \), a behavior function \( B(s,c) \) that takes the current state \( s \) and the command \( c \) (or goal) and outputs the action towards completing the command (or achieving the goal) is trained via supervised learning. For Markovian state transitions, the behavior function can be simply a multi-layer perceptron or convolutional networks. The command \( c \) can be defined as the return to achieve and left horizon.

### Sequence modeling in offline RL.

For upside-down RL [Srivastava et al., 2019], if the input is non-Markovian observations, it is straightforward to use a recurrent neural network (RNN) to implement the behavior function. Thus, the learning of the behavior function becomes a sequence modeling problem. Further, it is straightforward to replace the recurrent neural network with Transformer architectures. Compared to RNN, the Transformer architecture has the following advantages when performing decoding. First, the recurrence function is not used in Transformer, making it always work on the input data and thus suffering from lower compounding errors. Second, the positional encoding used in Transformer makes it aware of the decoding timestep, which is unavailable in RNN.

Decision Transformer (DT) [Chen et al., 2021] is a remarkable work on sequence modeling in offline RL. In DT, a casual (decoder-only) Transformer is used to implement the policy. The input data is a sequence of three tokens, i.e., return-to-go, observation, and action. An encoder is trained to project the data of each modality into the embedding space. Taking the data of previous steps, the return-to-go and the observation of the current step as input, DT predicts the action taken in the offline data. Note that the return-to-go in DT acts like the command used in upside-down RL, which is analogous to the goal in goal-conditional RL [Liu et al., 2022].

Trajectory Transformer (TT) [Janner et al., 2021] uses causal Transformer to model the whole experience sequence, predicting the observation, action, and reward of each step. As such, TT can be used as a world model and perform planning to select good action via beam search. Different from DT, TT employs a dimension-wise discretization process, which makes the vocabulary size smaller but the experience sequence much longer.

Taking back to TD learning, i.e., \( m(s,a) \) in Eq. (1) is the \( Q(s,a) \) function, the learning of \( Q \) function via standard temporal difference is like

\[
Q(s,a) \leftarrow Q(s,a)(1 - \alpha) + \alpha(r + \gamma Q(s', a'))\]

where \( a' \) is chosen by the learning policy or via a max operation. It is easy to see the message passing via TD learning is of low-efficiency, and the attribution of success is non-explicit. By contrast, for sequence modeling methods like DT, \( m(s,a) = 1 \) for all state-action pairs. The attribution of action taken can be explicitly modeled via the self-attention layers of Transformer architecture and the conditioned return-to-go signal.

## 3 Large Decision Models

In natural language processing (NLP), large language models (LLMs) refer to Transformer-based models with a vast number of parameters, e.g., BERT [Kenton and Toutanova, 2019] and GPT [Brown et al., 2020]. Besides the large parameter size, LLMs as the foundation models of NLP have the ability to perform multi-task learning [Radford et al., 2019] and few-shot adaptation [Brown et al., 2020]. Analogously, in reinforcement learning tasks, large decision models (LDMs) refer to large parameter-size models that i) are able to handle multiple decision-making tasks simultaneously and ii) can adapt to new tasks with a zero-shot or few-shot sample cost. Just like LLMs, so far the LDMs we have observed are mostly based on Transformers.

Generally, an LDM achieves multi-task handling and few-shot adaptation in the following ways, as shown in Figure 1. First, the training algorithm of an LDM is typically just su-
pervised learning, which is not directly related to the reward signal. Specifically, the trajectory experience data is selected based on the task-level success indicator or the whole return of the trajectory. Then the learning of the model is purely based on predicting the action at each step given the previous information of the trajectory, which is just like behavior cloning. Thus, LDMs naturally get rid of the problem of reward sensitivity. Second, after the trajectory data selection, the learning task itself is sequence modeling, i.e., predicting the “expert” actions in the selected trajectory data. As the network architecture of LDMs is always GPT and its variants, which thus inherit the merits of GPT as multi-task learners and few-shot learners in sequence modeling. In the remaining part of this section, we will discuss several important LDMs from the aspects of data processing, model architecture, training scheme, and generalization ability.

Gato [Reed et al., 2022] is a remarkable large decision model released by DeepMind in May 2022. Gato raises a unified data processing and model learning framework, which enables the LDM to work on 604 tasks. For the trajectory selection scheme, Gato chooses to filter out the low-return trajectories and only keep the trajectories with at least 80% of the expert return for that task, where the expert policy can be obtained via training the agent with a deep reinforcement learning method. Note that the expert policy can be regarded as overfitting the particular task and cannot transfer to other tasks. For the trajectory data processing scheme, Gato discretizes the data of different modalities into sequences of tokens, which are then mapped as sequences of embeddings. Specifically, the text and discrete data are directly embedded; the tensor data is mu-law encoded and discretized, then embedded; the images are split into patches, each of which is transformed as embeddings via ResNet [He et al., 2016]. Gato utilizes task prompts to manage the complexities of various tasks. During training, 25% of the sequences in a batch are introduced with a prompt sequence that originates from the same source agent and task episode. In the evaluation phase, a successful demonstration of the intended task is added at the beginning.

For the model training, Gato unifies the training of each task in a masked sequence modeling framework as

$$\theta \leftarrow \theta + \alpha \sum_{\tau \in \mathcal{B}} \sum_{(s, a) \in \tau} m(s, a) \log \pi_{\theta}(a|s),$$

where each trajectory $\tau = [x_1, x_2, \ldots, x_T]$ is a sequence of discretized tokens in the training batch $\mathcal{B}$, the so-called state-action pair $(s, a) = (\{x_1, \ldots, x_{t-1}\}, x_t)$ is indeed the preceding tokens and the current token at timestep $t$, $m(s, a)$ is the masking operator that specifies the token to be predicted (when $m(s, a) = 1$) or not (when $m(s, a) = 0$) depending on the task setting. In such a way, trajectory selection, data processing, and model learning from different tasks can be handled in the unified Gato framework. For policy instantiation $\pi_{\theta}$, Gato employs a decoder-only transformer model, which consists of 24 layers and 1.18 billion parameters. It uses an embedding size of 2048 and a hidden size of 8196 in the post-attention feedforward network.

Gato is trained and evaluated on 596 control tasks and 8 vision/language tasks, in 450 of which Gato surpasses the 50% expert score threshold. Such a performance demonstrates that Gato is a high-potential generalist agent across a large number of decision-making tasks. After the foundation model is pre-trained, Gato can be fine-tuned with less than 100 episodes to improve performance close to the expert’s, which demonstrates Gato’s adaptation ability to new tasks.

DB1 [Wen et al., 2022b] is a follow-up replication and validation work based on Gato. Compared to Gato, DB1 directly uses Transformer-XL architecture during training, adopts PostNorm layer normalization, and employs mixed-precision for attention computation to improve numerical stability. The number of handled tasks is 870 for DB1, where the traveling salesman problem (TSP) with 100-200 node scales, an NP-hard combinatorial optimization problem, is also included as a type of tasks. In 651 of the 870 tasks, DB1 surpasses the 50% expert score threshold.

Note that as both Gato and DB1 are partially pre-trained on natural language, they naturally work on language input. It is worth trying that task instructions can be used as the prompt prepended to the trajectory data during training, although this was not done on both LDMs. Thus, it is expected that such LDMs would be capable of understanding new task instructions for known or similar new tasks and taking actions with good generalization.

Not all LDMs are just trained with simple supervised learning methods; they can also be trained with online reinforcement learning methods. For example, AdA [Team et al., 2023] is an LDM based on Museli learning algorithm with a Transformer-XL architecture. Different from Gato and DB1, AdA is trained with the meta-RL method at scale in an open-ended task space with an auto-curriculum scheme to prioritize tasks at the frontiers of AdA’s capabilities, which gives AdA a fast adaptation ability comparable to humans. Further, LDMs are recently extended to build dynamics models to enable model predictive control [Schubert and others, 2023].

Besides single-agent tasks, preliminary attempts to enable LDMs to work on multi-agent coordination have been made. Comparing to the single-agent setting, in multi-agent systems, each agent is learning and updating its policy. Thus, from each agent’s perspective, the environment is varying across time, i.e., non-stationary, which makes multi-agent RL more challenging [Lowe et al., 2017], let alone the changes of the environment and task. Multi-agent decision Transformer (MADT) [Meng et al., 2021] is an early work that uses one causal Transformer to tackle various settings of multi-agent RL tasks based on SMAC tasks, i.e., the generalization
and adaptation across different multi-agent teams, offline pretraining and online finetuning, and multiple downstream tasks with zero-shot or few-shot learning costs. In the offline pretraining stage, MADT formulates the trajectory data of each agent as the sequence of (global state, local observation, action) and trains a parameter-sharing policy for each agent via predicting the action at each time step conditioning on previous information and a centralized critic that estimates the value of given the global state. In the online finetuning stage, each agent is initialized with the shared pretrained policy and uses the centralized critic and reward signal to calculate advantages to build a PPO method [Schulman et al., 2017] to finetune the policy.

Multi-agent Transformer (MAT) [Wen et al., 2022a] focuses on the orthogonal dimension compared to MADT, i.e., expanding the agent set as a sequence and building a sequence model to explore the interaction patterns across agents and generate the agent actions in an autoregressive manner at each timestep. With such a sequence modeling architecture, MAT works seamlessly over a varying number of agents. Unlike MADT which largely relies on pretraining over the offline dataset, MAT is trained online. Although it does not evaluate multi-task generalization, MAT demonstrates excellent zero-shot and few-shot adaptation to unseen multi-agent RL tasks on SMAC and MuJoCo tasks.

4 Applications of LDMs

With the advantages of wide generalization and low-cost adaptation to new environments or tasks, it is appealing to apply large decision models to various scenarios. Practically, there are two major implementation paradigms for LDMs depending on the application settings.

- **LDMs for general control (GC).** Based on the foundation model trained on control scenarios, e.g., robotic manipulation, locomotion, and game playing, the control policy can be finetuned with a low sample cost from the target task. The fundamental ability comes from the knowledge learned via interacting with the environment dynamics.

- **LDMs based on language backbone (LB).** Based on a pretrained large language model like ChatGPT, the agent could (i) interact with humans via language, (ii) interact in a language-based environment, e.g., dialogue system, web page navigation and operation, search engine, or (iii) interact with a general control environment if equipped with low-level perception and control APIs to interact with the environment, e.g., instruction or API based robot control. The fundamental ability comes from the knowledge learned from the large text corpus. Most LDMs based on LLMs are not trained to optimize the utility obtained by the agent, although this can be achieved by RL.

In fact, the above two paradigms are built based on the world models in dynamics and text. In general, if we can train a foundation model to understand and generate the data of a particular modality, we are close to building a world model in such a modality, based on which it is promising to build an LDM. It is also possible to build new LDMs based on large vision models in the near future. Due to the page limit, we will just discuss some applications of LDMs with the above two paradigms in the domains of robotics and game AI.

**Robotics.** Robot learning is a rising field during recent years. In general, learning from experience data makes it possible to bypass human-crafted rules or control theories based on environmental assumptions. Nonetheless, reinforcement learning methods are still difficult to enable the robot agent to generalize to different environments or tasks. As such, it is promising to leverage LDMs to overcome the above difficulties. In GC setting, for example, to train a quadrupedal robot to move over a large range of terrains and handle the sim-to-real gap, TERT [Lai et al., 2022] leverages a GPT student policy to learn from the local observations and the actions distilled from a teacher policy that accesses the true terrain information as the privileged information. To train a controller working on a wide range of embodiments of quadrupedal robots, EAT [Yu et al., 2022] is a GPT policy that takes in the vector representation of different quadrupedal robot embodiments and is trained for expert action prediction. In LB setting, the LLM can be used to understand and decompose the task instruction. With the perception APIs available, the environment state can be described via natural languages, and then based on the control APIs, the agent actions can be selected and filled with appropriate parameters via the LLM and executed in the environment. Remarkable examples include Code as Policies [Liang et al., 2022], PaLM-E [Driess et al., 2023], and ChatGPT for Robotics [Vemprala et al., 2023].

**Game AI.** The AI characters (i.e., NPC) in games play an important role to improve the user experience and even shape the game mechanism. For RTS, MOBA or MMO games, LDMs in GC settings are used to obtain superhuman combat performance. Different from robotics, game AI focuses on the interaction between agents and human users. With LDMs for GC, it is potential that the trained game AI agents are able to generalize over multiple game levels or tasks [Meng et al., 2021; Team et al., 2023]. Moreover, in role-playing and social prototyping games, the NPC agents can be implemented with LDMs based on LB. In Park et al. [2023], the NPCs are instantiated with generative agents, which enable them to move with purpose and chat with each other freely. Thus, if the game developer wants the town to host an in-game Valentine’s Day party, he does not need to craft the scripts of particular NPCs but just tell one NPC agent that she wants to host a Valentine’s Day party. Then the agent would interact with others to push for the event to happen.

5 Conclusion

In this article, we have discussed large decision models (LDMs), a new paradigm of techniques for general decision-making intelligence. LDMs refer to large parameter-size models that are able to handle multiple decision-making tasks simultaneously and can adapt to new tasks with a zero-shot or few-shot sample cost. To acquire these advantages, the LDMs we have observed are mostly based on Transformers. Research and applications centered around LDMs are on the rise. It is plausible to anticipate a surge in sequential decision-making intelligence in the imminent future.
Acknowledgments

The work is partially supported by National Key R&D Program of China (2022ZD0114804), Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102), National Natural Science Foundation of China (62076161), and Digital Brain Lab. The author thanks Ying Wen for improving this paper.

References


