Think Before You Act: Decision Transformers with Working Memory

Jikun Kang [1 2]   Romain Laroche [*]   Xingdi Yuan [* 3]   Adam Trischler [*]   Xue Liu [* 1 2]   Jie Fu [2]

Abstract

Decision Transformer-based decision-making agents have shown the ability to generalize across multiple tasks. However, their performance relies on massive data and computation. We argue that this inefficiency stems from the forgetting phenomenon, in which a model memorizes its behaviors in parameters throughout training. As a result, training on a new task may deteriorate the model's performance on previous tasks. In contrast to LLMs' implicit memory mechanism, the human brain utilizes distributed memory storage, which helps manage and organize multiple skills efficiently, mitigating the forgetting phenomenon. Inspired by this, we propose a working memory module to store, blend, and retrieve information for different downstream tasks. Evaluation results show that the proposed method improves training efficiency and generalization in Atari games and Meta-World object manipulation tasks. Moreover, we demonstrate that memory fine-tuning further enhances the adaptability of the proposed architecture. [1]

1. Introduction

Recently, with the tremendous success of decoder-only Transformer models (Brown et al., 2020; OpenAI, 2023; Dosovitskiy et al., 2021; Touvron et al., 2023), an increasing number of researchers have focused on decoder-only Transformer-based decision-making agents. As shown with GPT-3 (Brown et al., 2020) and follow-up work (Kaplan et al., 2020; Clark et al., 2022), the generalization of these LLMs depends significantly on the model size, *i.e.* the num-
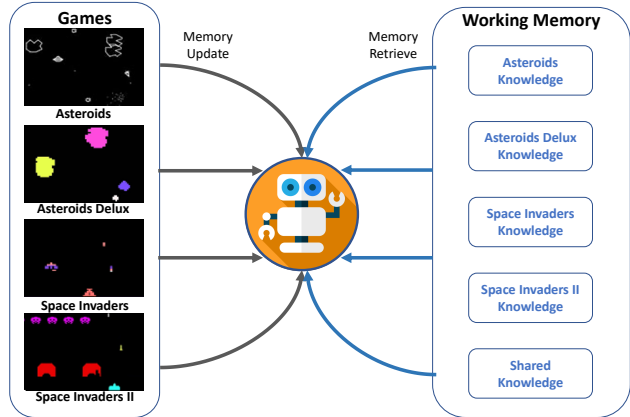

Figure 1: Illustrating how a robot can use its memory to guide its playing strategy.

ber of parameters. This is partly because neural network parameters act as implicit memory (Neyshabur et al., 2019), enabling models to "memorize" a huge amount of training data by fitting these parameters. However, relying purely on scale has practical and ethical limits: there are economic and ecological costs, and it reduces accessibility To address some limits of the implicit, parameter-based memory of large models, we take inspiration from the concept of "working memory" (Baddeley, 2003; Cowan, 2008) to *explicitly* store and recall past experiences for use in future decision-making. The concept, "working memory", originates from cognitive psychology and neuroscience (Baddeley, 2003; Goldman-Rakic, 1995), where it refers to the system responsible for the temporary storage and manipulation of information during cognitive tasks.

Our motivation comes from how humans think before they act: they can reason on past experiences to generate appropriate behavior in new situations. We want to equip our robots with similar abilities. Imagine training a robot to play four different Atari games: Asteroids, Asteroids Deluxe, Space Invaders, and Space Invaders II (Figure 1). Asteroids Deluxe is a sequel to Asteroids that introduces new boss fights and enemies, similarly, Space Invaders II is a sequel to Space Invaders. For a robot to play these four games, it must actively store what it has learned in memory and choose the appropriate strategy for each game. Throughout training, the robot's memory module continuously processes

---

*Equal advising [1]Department of Computer Science, McGill University, Montréal, Canada [2]Mila - Québec AI Institute, Montréal, Canada [3]Microsoft Research, Montréal, Canada. Correspondence to: Jikun Kang <jikun.kang@mail.mcgill.ca>, Jie Fu <jie.fu@polymtl.ca>.

[1]We open source the code at https://github.com/luciferkonn/DT_Mem.

and updates relevant game information, allowing it to make informed decisions and adapt its strategies.

Followed by this intuition, we introduce **D**ecision **T**ransformers with **M**emory (DT-Mem): which represents a working memory as a matrix, the functioning of the memory entails two primary steps, namely **memory update** and **memory retrieval**. DT-Mem builds on earlier work on memory-augmented neural networks (Santoro et al., 2016)—including neural Turing machines (Graves et al., 2014) and memory networks (Sukhbaatar et al., 2015)—in several ways, as we detail in Section 2.

We use content-based addressing (Eslami et al., 2016) to locate the memory position to update information into or retrieve information from it. The memory update involves modifying or replacing existing information. This enables the system to keep track of changes, maintain task-relevant information, and facilitate decision-making. More specifically, we first map both the input sequence and memory into three entities: *query*, *key*, and *value*. Next, we use an attention-based mechanism to calculate the correlations between the input and memory, and subsequently use the attended weight of the input sequence to update the memory. Memory retrieval refers to the process of accessing and recovering stored information. It involves bringing relevant information back to condition decision-making. We read from the updated memory at the content-based address to achieve that.

Humans can make analogies by mapping experience between tasks, this enables us to leverage experience when encountering new tasks. Therefore, we also equip our memory module with an adaptable mapping capability. Specifically, for adapting the memory module to a new task, we employ the Low-Rank Adaptation (LoRA) method as described in (Hu et al., 2022) as our fine-tuning strategy. The main idea behind LoRA is to train a low-rank projection matrix on a small amount of labeled data from a new task. This matrix maps the parameters of a pre-trained model to a new task. In this work, we fine-tune only the memory module because we rely on the generalization capacity of a pre-trained Decision Transformer (DT). In prior works, Transformers are often pre-trained on large-scale datasets (Lee et al., 2022; Xu et al., 2023), such pre-training enables them to capture broad knowledge that is transferable across tasks. In contrast, the memory module we propose is designed to store knowledge explicitly that can be modified and utilized to new tasks.

DT-Mem differs from external memory and information retrieval-based methods in several ways. First, external memory methods often require a large dataset that serves as a look-up table; whereas in our system, the working memory can be maintained at a handy size. Second, external memory methods require an extra step of representation learning to convert entries in the look-up table into vector space; in contrast, information in our system naturally operates in a latent space. Finally, external/retrieval-based memory methods often resort to a k-nearest neighbor search during retrieval, mainly because it is computationally impractical for an attention-based mechanism to operate on large sets (the look-up table); our method, on the other side, could naturally leverage attention-based module during both memory update and retrieval.

To validate our approach, we evaluate DT-Mem in two environments and compare against a set of strong baselines: (a) Atari games: we compare against Multi-game Decision Transformer (MDT, Lee et al., 2022) and Recurrent Memory Decision Transformer (RMDT, Bessonov et al., 2023); (b) Meta-World environments: we compare against Prompt Decision Transformer (PDT, Xu et al., 2022) and Hyper-Decision Transformer (HDT, Xu et al., 2023). Our results show that DT-Mem improves generalization and adaptability with fewer model parameters and less training time.

2. Related work

Transformer-based Reinforcement Learning methods Transformer (Vaswani et al., 2017) is a powerful architecture designed for sequence modeling. Owing to the capabilities that emerge as model and data size scale up, the Transformer has become a foundational model in several domains, including natural language processing (Brown et al., 2020; OpenAI, 2023; Touvron et al., 2023) and computer vision (Dosovitskiy et al., 2021). However, applying Transformer in reinforcement learning settings, such that it generalizes to multiple tasks, remains an open problem.

Recently, Chen et al. (2021) and Janner et al. (2021) treat the RL problem as a sequence modeling problem and proposed a Transformer-based architecture to solve it with offline RL. These findings inspired researchers to develop more advanced Transformer-based RL methods. Subsequent efforts mainly focus on two aspects: generalization and adaptability. To improve model online adaptability, Zheng et al. (2022) propose the Online Decision Transformer (Online DT), which utilizes the maximum-entropy idea to encourage pre-trained policies to explore during a phase of online adaptation. To improve offline adaptation, Xu et al. (2023) propose a Hyper-network-based module that helps DT adapt to unseen tasks efficiently. To facilitate task adaptation, Xu et al. (2022) introduce the prompt-based DT, which selects short trajectories to use in a task prompt in analogy with in-context learning for large language models. Furthermore, Lee et al. (2022) propose a multi-game DT (MDT), which use the expert action inference to produce actions of highly-rewarding behavior consistently. MDT demonstrates that DT can generalize to various Atari games with human-level performance.

We argue that the generalization of the above-mentioned works relies on the size of the models and does not learn the data efficiently. To address this issue, we introduce a memory module that can store, mix, and retrieve training information for better model and training efficiency.

Working memory In the context of machine learning, there is a long history of neural network-based models that incorporate memory mechanisms (Das et al., 1992; Schmidhuber, 1992; Hochreiter & Schmidhuber, 1997; Santoro et al., 2016; Ba et al., 2016; Munkhdalai & Yu, 2017; Munkhdalai et al., 2018; Csordás & Schmidhuber, 2019; Ramsauer et al., 2020; Wu et al., 2022a). Generally, this research aims to enhance the capacity of neural networks to store and manipulate information over extended periods of time, leading to improved performance on a range of tasks. It often takes inspiration from human cognitive function. Most salient to our work, Graves et al. (2014) merge concepts from Turing machines and deep learning in "Neural Turing Machines" (NTMs), neural networks that include a content-addressable matrix memory space for storing and updating information throughout time. They show NTMs to be effective for various algorithmic tasks. Concurrently, Sukhbaatar et al. (2015) introduce "memory networks," which use a content-addressable matrix memory to store and retrieve information from previous computational steps, facilitating complex reasoning and inference tasks.

Infinity-former excels in handling unbounded contexts with precision and flexibility, ideal for extensive and complex datasets (Martins et al., 2022). LONGMEM decoupled architecture and token-to-chunk retrieval make it adept at managing large contexts and overcoming memory staleness (Wang et al., 2024). kNN-augmented Transformer offers flexibility in context length and rapid adaptation to new data, enhancing the model's real-time applicability (Wu et al., 2022b).

More recently, Bessonov et al. (2023) introduces a recurrent memory mechanism to address reinforcement learning challenges, which preserves a hidden state throughout the decision-making process. However, this method overlooks the storage and retrieval of task-related information, thereby falling short of fostering model generalization and task adaptation. Munkhdalai et al. (2019) propose a rapidly adaptable neural memory system, with which they instantiate as a feedforward neural network trained by meta-learning (Chen et al., 2022). They evaluate the memory's effectiveness in a simple RL setting, maze exploration, and on various NLP tasks. Alternatively, Goyal et al. (2022) builds on the "global workspace" theory from cognitive science, which posits that different input entities share information through a common communication channel. The proposed shared global workspace method employs the attention mechanism to encourage the most useful information to be shared among

neural modules. It is closely related to working memory and inspires us to explore how an explicit working memory can improve the generalization of Transformer-based models. An upshot of our work is that it may be valuable to revisit earlier memory-augmentation methods in light of more powerful foundation models.

Comparisons with classic memory-based methods There are several classic memory-based methods; however, none of them can be directly applied to DT in the same manner as RMDT. Consequently, we have not included them in our evaluations. To distinguish our approach, we compare these methods in terms of both memory writing and memory reading. **Memory Writing:** ∞-former (Martins et al., 2022) represents memory as a continuous signal using radial basis functions (RBFs). When new information is encountered, it is integrated into this continuous representation. This process involves evaluating the continuous signal at specific locations and then concatenating these evaluations with new vectors coming from the short-term memory. DT-Mem involves sophisticated mechanisms using attention to manage the significance of new and existing data. This process would be based on calculating correlations between the input and memory and updating the memory with the attended weight of the input sequence. LONGMEM (Wang et al., 2024) caches paired attention keys and values from the previous context into a non-differentiable memory bank using a frozen backbone Large Language Model (LLM) as the memory encoder. KNN-transformer (Fan et al., 2021) uses (key, value) pairs from the local context are appended to the end of an external memory. **Memory Reading:** The reading process of ∞-former utilizes a continuous-space attention framework. DT-Mem uses content-based addressing for memory retrieval. This process would involve using attention mechanisms to read from the updated memory, focusing on the content relevant to the current task or context. LONGMEM uses a decoupled memory module, specifically a residual side-network (SideNet), for memory retrieval and reading. The SideNet retrieves cached key-value pairs of previous contexts from memory using the attention query of the current input. KNN-transformer features a kNN-augmented attention layer that combines standard dense self-attention with approximate k-nearest-neighbor search into the external memory. The kNN lookup retrieves the top-k (key, value) pairs for each query from the input subsequence, constructing an attention matrix that represents these memories differently for each query.

3. Preliminaries

3.1. Offline Reinforcement Learning

A trajectory consists of a series of states, actions, and rewards, expressed as $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \cdots, s_T, a_T, r_T)$. In the context

of offline RL, data acquisition does not come from active interaction with the environment. Instead, we rely solely on a predefined and limited dataset containing various trajectories generated by different policies. This scenario presents greater challenges as it restricts the agent's ability to actively explore the environment and gather new information, which is a crucial aspect of traditional RL approaches.

Formally, in the context of model evaluation, we can define a set of training tasks and testing tasks as T^{train} and T^{test}, respectively. These two sets deliberately have no overlapping tasks but may share the same or similar observation and action spaces. To be more specific, for each training task $\mathcal{T}^i \in T^{\text{train}}$, we have access to a large training dataset, which contains trajectories $\tau^{0:H} = (s_0, a_0, r_0, \cdots, s_H, a_H, r_H)$, where H is the episode length. However, we assume access to only a small amount of data for the testing tasks.

Our goal is to evaluate the proposed model in two dimensions. First, we want to assess the model's **generalization**, which refers to its ability to solve the testing tasks within a finite time with no additional fine-tuning. Second, we want to test the model's **adaptability**, which refers to its ability to improve its performance on the testing tasks through fine-tuning on limited data after pre-training on separate tasks.

3.2. Low-rank Adaptation

Low-rank adaptation (LoRA, Hu et al., 2022) is a transfer learning technique used to adapt a pre-trained model to a new task with limited labeled data. The main idea behind LoRA is to utilize a small amount of labeled data from a new task to learn a low-rank projection matrix. This matrix maps the parameters of a pre-trained model to the new task. Specifically, for a pre-trained weight matrix $W \in \mathbb{R}^{d \times k}$, we assume a low-rank decomposition for the weight update: $W + \Delta W = W + BA$, where $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$. Once the projection matrix is learned, it can transform the pre-trained model's parameters to a new subspace that is more suitable and which alters its forward pass output. In other words, $Wx + \Delta W x = Wx + BAx$.

4. Methodology

4.1. Overview

In Figure 2, we depict the architecture of DT-Mem, which consists of four components: the attention module, the encoder module associated with positional encoding, the action decoder module, and the memory module. The primary role of the attention module is to capture dependencies and relationships between states, actions, and returns in a sequence. The encoder with positional encoder module embeds the inputs and persists the temporal correlations be-

tween states and actions. The input of the encoder module is a fixed-length sequence of trajectories, denoted as $\tau^{t+1:t+K}$. The output is a sequence of embeddings, where each entry can be attended state embeddings, action embeddings, or return-to-go embeddings. The action decoder is a multi-layer perceptron (MLP) that responds to decode the latent parameters of actions. We introduce a memory module for storing and manipulating intermediate information on top of the standard Decision Transformer architecture. This memory design is inspired by the Neural Turing Machine (Graves et al., 2014), where the memory is utilized to infer multiple algorithms. The details of memory module design and functionality are introduced in Section 4.2.

4.2. Memory Module

The design for the memory module is inspired by the way humans think before they act. Its functioning consists of three parts: identifying salient information output from the Transformer module; determining where to store new information and how to integrate it with existing memory slots; and considering how to use these memory slots for future decision-making. We break down these questions and design the following steps to address them.

Step 0: Memory Module Initialization. The memory is initialized as a random matrix M, where each row $m_i \in \mathbb{R}^d$, with $i \in [0, N]$, represents a memory slot.

Step 1: Input Sequence Organizing. As illustrated in Section 3.1, an input sequence comprises multiple steps of the tuple $< \hat{r}_t, s_t, a_t >$. Instead of directly feeding this sequence into the Transformer module, we treat each tuple as an entity and embed them into the same latent space. Specifically, we define embedding functions $g_s(s) = e_s$, $g_a(a) = e_a$, and $g_r(\hat{r}) = e_{\hat{r}}$, where e_s, e_a, and $e_{\hat{r}} \in \mathbb{R}^d$ with d representing the dimension in the latent space. The final input sequence emerges from the concatenation of embeddings $E = [\cdots, [e_{s_t}; e_{a_t}; e_{\hat{r}_t}], \cdots]$.

Recall that we represent our memory as a matrix with fixed dimensions (i.e., number of slots $\times$ hidden dimensions). It is crucial to synchronize the input dimensions for efficient storage. Notably, in our design, we maintain the relationships among trajectories as posited in the Decision Transformer paper, although this is not a requisite. For instance, in another prior work, Trajectory Transformer (Janner et al., 2021), states, rewards, and others are grouped individually. We demonstrate in Appendix B.6 that these varied designs exhibit no significant difference.

Step 2: Content-based Address. We use an attention-based mechanism to locate a proper memory slot for new input by identifying correlated information. This approach is based on the idea that humans tend to store and group similar or related information together (e.g., in Documentation Sci-
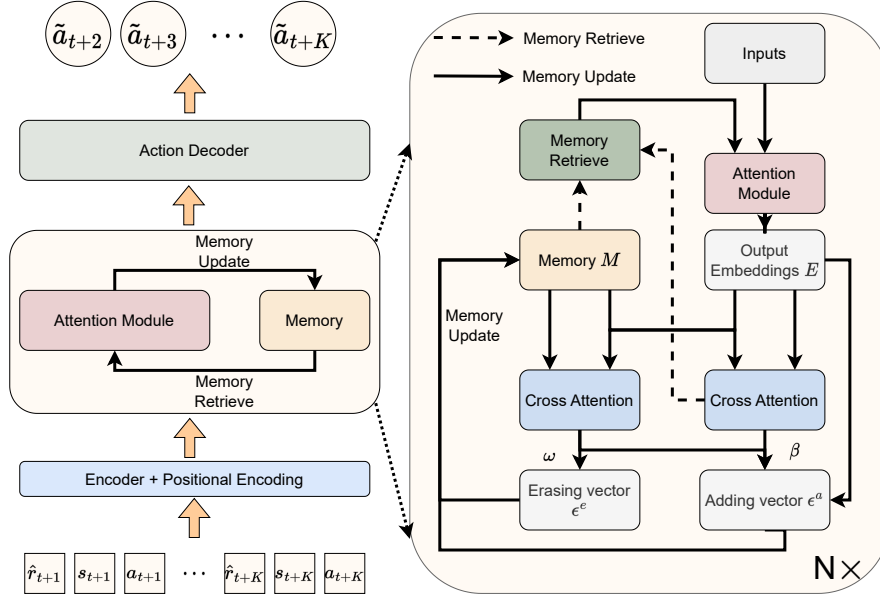
Figure 2: An overview of the proposed DT-Mem architecture. The input of the encoder is a fixed-length sequence of trajectories. The encoder with positional encoder module embeds the inputs and persists the temporal correlations between states and actions. The primary role of the attention module is to capture dependencies and relationships between states, actions, and returns in a sequence. Note that there are multiple attention modules stack together. Our design deconstructs this module and manages the memory flows between the attention module within each block. The output from attention blocks flows to the action decoder, which decodes back to the real actions.

ence and Archival Science (Dooley, 2007)). To locate the memory position, we compute the position address w as follows: $w = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)$. Here, $Q = MW^q$ and $K = EW^k$, where W^q and W^k are parameters in the MLP. The objective is to map the memory and input information into the query and key matrix, and then use the dot product to determine the similarities between these two matrices. The softmax function guarantees that the sum of all addresses equals one.

Step 3: Memory update. To store incoming information and blend it with existing memory, we calculate two vectors: an erasing vector, ϵ^e, and an adding vector, ϵ^a. The erasing vector erases the current memory, while the adding vector controls information flow to the memory. To achieve this goal, we again utilize the attention mechanism. First, we map memory and input information to query, key, and value vectors, denoted as $\hat{Q} = M\hat{W}^q$, $\hat{K} = E\hat{W}^k$, and $\hat{V} = E\hat{W}^v$, respectively, where $\hat{W}^q$, $\hat{W}^k$, and $\hat{W}^v$ are parameters. Next, we calculate the writing strength, $\beta = \text{softmax}\left(\frac{\hat{Q}\hat{K}^T}{\sqrt{d}}\right)$. The erasing vector is used to selectively erase information from the memory matrix and is computed as a function of the content-based addressing vector and the write strength. The erasing vector is calculated as $\epsilon^e = w \odot (1 - \beta)$, where $\odot$ indicates element-wise multiplication. The complement of the write strength is

1 minus the write strength, so this will result in a vector where the elements corresponding to the selected memory locations are set to 0, and the elements corresponding to the unselected memory locations are unchanged.

The adding vector is used to selectively add information to the memory matrix and is computed as a function of the write strength and the input vector. Specifically, the adding vector is calculated as $\epsilon^a = (w \odot \beta)\hat{W}^v x$.

Finally, the memory is updated as $M_t = M_{t-1} \odot (1 - \epsilon^e) + \epsilon^a$. The new information will be stored if the selected memory slot is empty or erased. Otherwise, the new information will be blended with the existing memory contents.

Step 4: Memory retrieval. We retrieve information from the updated memory slots to utilize memory for decision-making. Reading from the memory matrix is done by computing a read position vector. This vector can be computed using the above content-based addressing mechanism that compares the query vector with the contents of the memory matrix. Note that in other retrieval-based methods (Humphreys et al., 2022; Borgeaud et al., 2022), the nearest neighbor is the common way to retrieve related information. However, in our case, the working memory is considerably smaller than typical external memory, which makes attention-based retrieval feasible. Since the query information is the same as the input information, we use the same

content address to retrieve the memory: $\boldsymbol{E}_{\text{out}} = \boldsymbol{w} \odot \boldsymbol{M}_t$.

4.3. Pre-training

We use a set of training tasks T^{train}, where each task $\mathcal{T}_i \in T^{\text{train}}$ has an associated offline dataset $\mathcal{D}_i$ consisting of hundreds of trajectories τ generated by a behavior policy. The behavior policy can either be a pre-trained policy (such as DQN), a rule-based policy, or even human players — depending on what is available. Each trajectory $\tau = (s_0, a_0, r_0, \cdots, s_H, a_H, r_H)$, where $s_i \in \mathcal{S}, a_i \in \mathcal{A}, r_i \in \mathcal{R}$, and H is the episode length.

To serve as an input to the DT-Mem, we first segment the trajectory τ into several pieces, each with length K. We denote $\tau_{t+1:t+K} = (s_{t+1}, a_{t+1}, r_{t+1}, \cdots, s_{t+K}, a_{t+K}, r_{t+K})$ as one of the input sequence. However, we modify these trajectories instead of inputting them directly. Specifically, we follow the return-to-go Decision Transformer idea (Chen et al., 2021) and calculate the return to go, $\hat{r}_t = \sum_{t+1}^{t+K} r_t$, which is performed at every timestep. This is effective because $\hat{r}_t$ acts as a subgoal, encouraging the Transformer module to generate actions that can reduce the negative of this value as close to zero as possible. Therefore, we feed the modified trajectories $\hat{\tau}_{t+1:t+K} = (\hat{r}_{t+1}, s_{t+1}, a_{t+1}, \cdots, \hat{r}_{t+K}, s_{t+K}, a_{t+K})$ as input to the Transformer module. The output of the Transformer module is a sequence embedding $\boldsymbol{e}_{\text{seq}} \in \mathbb{R}^{d \times 3K}$, where d is the dimension of the embedding space.

Next, we transmit $\boldsymbol{e}_{\text{seq}}$ to the memory module to update and retrieve the memory information. Finally, we feed the retrieved information $\boldsymbol{E}_{\text{out}}$ into the MLPs to generate the corresponding actions $\hat{a}_t$. We minimize a supervised training loss with three terms: predicted actions $\tilde{a}_t$, predicted reward $\tilde{r}_t$, and predicted return-to-go $\tilde{R}_t$. The loss function is:

$$\mathcal{L} = \sum_{t+1}^{t+K} ||\tilde{a}_t - a_t||^2 + \alpha||\tilde{r}_t - \hat{r}_t||^2 + \lambda||\tilde{R}_t - r_t||^2, \ (1)$$

where α and λ are scalar hyper-parameters. Empirically, we find that the final performance is not sensitive to specific α and λ values, so we set them to 1 for simplicity.

The full pre-training process is summarized in Appendix A.2 Algorithm 1.

4.4. Fine-tuning with LoRA

Fine-tuning LLMs involves heavy computation due to the large number of parameter updates required. We argue that fine-tuning only the memory module can achieve results comparable to those of fine-tuning the entire parameter space. LLMs such as BERT (Devlin et al., 2019) or GPT (Radford et al., 2019) greatly benefit from training on large-scale datasets, which expose the model to a diverse range of linguistic patterns and semantic relationships. This exposure helps the model learn robust and generalized representations that capture different language understanding and generation aspects. After pre-training, the model can be fine-tuned on specific downstream tasks with task-specific labeled data. In our case, this task-specific knowledge is stored in the memory module. Thus, fine-tuning the memory module helps the model update its memory module to adapt to the new task.

We apply the low-rank adaptation approach (LoRA, Hu et al., 2022) to fine-tune the memory module. Specifically, we modify the forward pass by adding low-rank matrices to $\boldsymbol{W}^q$, $\boldsymbol{W}^k$, $\boldsymbol{W}^v$, $\hat{\boldsymbol{W}}^q$, and $\hat{\boldsymbol{W}}^k$. Taking $\boldsymbol{W}^q$ as an example. Assuming the original output for query information is $\boldsymbol{Q} = \boldsymbol{M}\boldsymbol{W}^q$, we adapt this query value to a new task as $\boldsymbol{Q}' = \boldsymbol{M}(\boldsymbol{W}^q + \boldsymbol{B}^q\boldsymbol{A}^q)$, where $\boldsymbol{W}^q \in \mathbb{R}^{n \times d}$, $\boldsymbol{B} \in \mathbb{R}^{n \times m}$, and $\boldsymbol{A} \in \mathbb{R}^{m \times d}$, and m is the size of the memory module. Since the rank $m \ll \min(n, d)$, fine-tuning the parameters $\boldsymbol{B}^q$ and $\boldsymbol{A}^q$ reduces the number of trainable parameters for downstream tasks. We perform supervised training by computing the loss between the model's output and the labels in the fine-tuning dataset. Only $\boldsymbol{B}^q$ and $\boldsymbol{A}^q$ are updated during this process. The detailed fine-tuning procedure can be found in Appendix A.2 Algorithm 2.

5. Evaluation

We design our experiments to answer the following questions:

- **Q1:** Does DT-Mem improve model generalization?
- **Q2:** Does DT-Mem improve pre-training results and training efficiency?
- **Q3:** Does DT-Mem scales with model size?
- **Q4:** Does fine-tuning only the memory module improve model adaptability?

We use generalization to refer to performance on tasks the model has never been trained on (zero-shot), and adaptability to refer to performance after fine-tuning.

5.1. Environments and Models Setup

Atari Games. To ensure a fair comparison with the Multi-Game Decision Transformer, we use the same Atari dataset, which comprises multiple training runs of DQN trajectories. Due to limited compute resources and to prevent cherry-picking, we select 17 games from a total of 41 based on their alphabetical order, as introduced in (Lee et al., 2022). For each game, the data contains 50 policy checkpoints, each containing 500k environment steps. For the fine-tuning dataset, we randomly selected 10% of the data from the unseen dataset, yielding 50k environment steps. Following the settings from Lee et al. (2022), we choose four games (Alien, Ms. Pac-Man, Space Invaders, and Star Gunner)

Model	Layers	Hidden size (d)	Heads	Params	Memory Size	Memory Module Params
HDT	4	512	8	13M	N.A.	N.A.
MDT-200M	10	1280	20	200M	N.A.	N.A.
DT-Mem	4	512	8	13M	559K	7M

Table 1: Implementation details of model sizes

	Alien	MsPacman	SpaceInvaders	StarGunner
MDT	3.8% ($\pm$0.4%)	13.2% ($\pm$1.3%)	8.6% ($\pm$1.6%)	2.3% ($\pm$0.1%)
RMDT	22.3% ($\pm$10.7%)	22.9% ($\pm$8.9%)	17.6% ($\pm$9.2%)	27.7% ($\pm$11.5%)
DT-Mem	51.0% ($\pm$32.2%)	69.3% ($\pm$19.3%)	53.6% ($\pm$29.0%)	62.2% ($\pm$19.1%)

Table 2: Evaluation results on 4 held-out games after pre-training on other Atari Games. Each value represents the DQN-normalized score, computed with a 95% confidence interval.

to be used only for fine-tuning. Moreover, Brandfonbrener et al. (2022) suggests that return-conditioned supervised learning (RCSL) algorithms require strong dataset coverage to select a near-optimal policy. Therefore, our dataset contains both expert and non-expert behaviors.

Meta-World. To make a fair comparison with Hyper-DT and Prompt-DT, we evaluate the proposed method on the Meta-World environment (Yu et al., 2019). We evaluate using the Meta-World ML45 benchmark, which includes 45 training tasks and 5 testing tasks. Following the approach taken in (Xu et al., 2023), for each training task, we generate an offline dataset containing 1000 episodes for each game, using a rule-based script policy. For fine-tuning data, we randomly pick 10k episodes from the testing dataset, as compared to 20k-80k episodes used in Hyper-DT.

Implementation details. We report results for DT-Mem 20M (20 million parameters), which consists of 13M Transformer parameters and 7M memory module parameters. For all games, we use eight V100 GPUs for model training and one V100 GPU for fine-tuning. We train on both Atari games and Meta-World for 10M steps. For fine-tuning on unseen scenarios, we train for 100k steps.

Table 1 summarizes the different model configurations used for evaluation. In this section, we describe these model configurations in detail. While Table 1 provides a summary, we will provide additional information here. DT-Mem, PDT, and HDT all share the same Transformer architectures. However, for task-adaptation, HDT utilizes a pre-trained 2.3M hyper-network, while DT-Mem introduces 147K LoRA parameters. To compare with MDT, we use the same parameter size as reported in (Lee et al., 2022).

5.2. Baseline Methods

We compare DT-Mem's performance against the following baselines. **MDT**: Multi-game Decision Transformer (Lee et al., 2022), which trains a large Transformer-based model on multi-game domains. For a fair comparison, we train an MDT with 20M parameters, which is approximately the same size of DT-Mem. **RMDT**: Recurrent Memory Decision Transformer (Bessonov et al., 2023), which utilizes a recurrent memory mechanism for solving reinforcement learning problems. This is the most related memory-based DT that is close to our work. **HDT**: Hyper-Decision Transformer (Xu et al., 2023), which utilizes a hyper-network module to help DT adapt rapidly to unseen tasks. Since we do not have access to the implementation at the time of writing, for the sake of correctness, we compare our model with HDT on Meta-World only. The results reported in our evaluation section come from the HDT paper. **PDT**: The Prompt Decision Transformer (Xu et al., 2022) generates actions by considering both recent context and pre-collected demonstrations from the target task.

5.3. DT-Mem improves model generalization.

We evaluate four held-out games fine-tuning results as listed in Table 2. Each evaluation signifies an average derived from 16 runs, each under differing random seeds. The derived results show that the memory-incorporated methods, RMDT and DT-Mem, enhance model generalization when compared to their ablation method MDT. A noteworthy observation is that DT-Mem demonstrates superior generalization performance than RMDT in all four games. This suggests that RMDT does not adequately address the storage and retrieval of information related to tasks, consequently not effectively supporting model generalization and task adaptation development. This contrasts DT-Mem, which is designed to accomplish these objectives.

5.4. DT-Mem enables more computationally efficient training and scale with model parameters.

To demonstrate training efficiency, we illustrate the model training time in Table 3 and the training curve in Appendix B.2 Figure 5. During training, we find that DT-Mem reduces
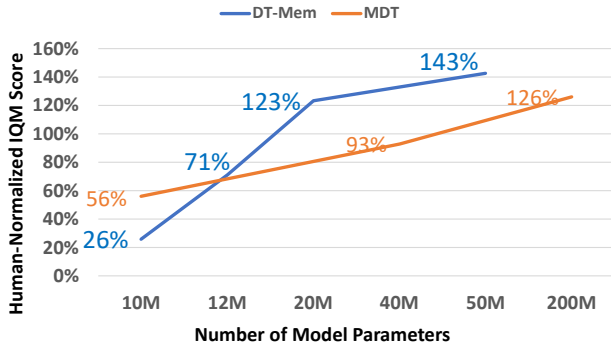
Figure 3: Scaling of IQM scores

Model	Training time (hours)
DT-Mem-20M	**50**
MDT-13M	200
MDT-40M	400
MDT-200M	1600

Table 3: Model training time

the training time by approximately 4 times, 8 times, and 32 times compared to MDT-13M, MDT-40M, and MDT-200M, respectively. It is reasonable to report the prediction loss on the training dataset for the training curve since we use a supervised loss. As defined in Section 4.3, the prediction accuracy consists of three parts: action prediction accuracy, reward prediction accuracy, and return prediction accuracy.

Figure 3 showcases the scaling laws of the proposed DT-Mem model. We measure performance using the human-normalized IQM score. It's crucial to note that for all instances of DT-Mem, we maintained a consistent number of memory slots. The result shows that the performance of DT-Mem scales with the number of parameters. Notably, the generalization of DT-Mem with 20M parameters is approximately on par with MDT's 200M parameter version. Furthermore, the 50M DT-Mem surpasses MDT by a margin of 16.7%.

5.5. Fine-tuning only the memory module improves model adaptability.

Another question we care about is how the pre-trained DT-Mem performs on unseen tasks. We randomly selected nine unseen Atari games and evaluated their performance through relative improvement scores, as shown in Figure 4 (top). DT-Mem consistently outperforms RMDT and MDT in most of the games listed, with the exception of Seaquest, where MDT excels. MDT exhibits the least superior performance across most games, with its performance particularly lagging in KungFuMaster, Robotank, and Phoenix. RMDT holds an intermediate performance level between DT-Mem

and MDT across most games. The consistent superior performance of DT-Mem across most games suggests that this method might have a more adaptable approach. The singular superior performance of MDT in Seaquest prompts a further investigation into the unique attributes of this game that may favor the MDT method.

To further understand the adaptability of the proposed method, we compare DT-Mem with HDT and PDT in Meta-World environments. The quantitative fine-tuning results are shown in Table 4. Overall, DT-Mem achieves the best performance in the comparison. As we can see, compared to HDT, DT-Mem increases training, testing (no-FT), and testing (FT) scores by an average of 3%, 8%, and 3%, respectively. Moreover, the HDT adaptation module (hyper-network module), while small (69K) relative to the full model (13M), relies on the pre-trained hyper-network, which contains 2.3M parameters. We argue that the hyper-net is more burdensome than our design: it uses more than 10x the number of adaptation parameters (147K) used by DT-Mem and requires an extra compute phase to pre-train the hyper-network module, which again could cause extra issues in optimization.

5.6. DT-Mem improves training performance.

In this section, we evaluate whether adding the memory module helps improve the pre-training performance. We report relative improvement for clear comparison: rel-imp(%) = $\frac{(\text{model score} - \text{best score in data})}{\text{best score in data}} \times 100$ to measure the model performance. For better visualization, we take the logarithm of the rel-imp(%). As shown in Figure 4 (bottom), the proposed DT-Mem out performs MDT in 13 out of 17 games. DT-Mem outperforms RMDT in 15 out of 17 games. These results demonstrate that the memory module improves the policy training performance.

6. Conclusion

LLM-based RL algorithms have shown generalization across multiple tasks and games. We argue that this ability comes from implicit memory that fits a large number of parameters to the training data, which is inefficient in terms of model size. In contrast, we propose a new approach inspired by the concept of "working memory" called **D**ecision **T**ransformers with **M**emory (DT-Mem), which stores training experience explicitly in a content-addressable matrix module for later retrieval and use. The evaluation demonstrates that DT-Mem achieves better generalization on Atari games with only 10% of the model parameters compared to the state-of-the-art method. We also show that DT-Mem outperforms other memory-based DT methods regarding generalization and adaptability. Furthermore, we demonstrate that fine-tuning DT-Mem with a small amount of data can produce state-of-the-art results on both Atari games and the Meta-World environment, when compared to MDT,
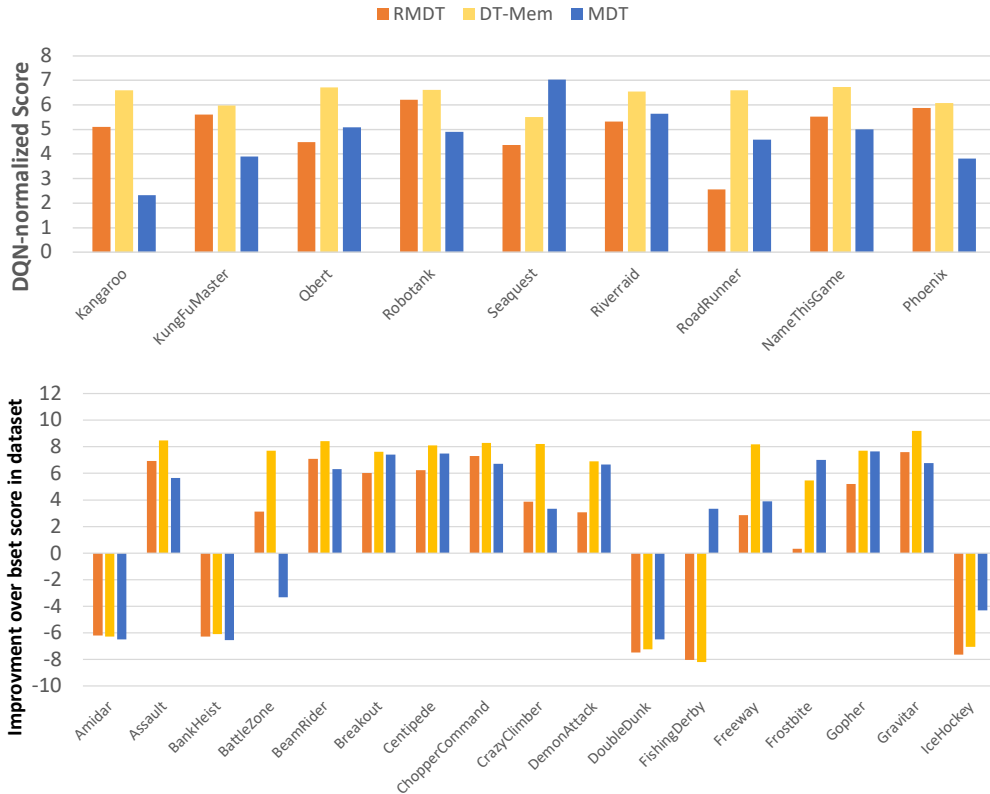
Figure 4: **Top**: Fine-tuning performance on 10% of dataset in unseen Atari games. For better visualization, the y-axis is the logarithm of DQN-normalized score. **Bottom**: The performance improvement for the training dataset.

	Model Sizes		Meta-World ML45 Performances		
	Adaptation	Percentage	Train	Test (no-FT)	Test (FT)
HDT	69K	0.5%	0.89 ± 0.00	0.12 ± 0.01	0.92 ± 0.10
PDT	6K	0.05%	0.88 ± 0.00	0.06 ± 0.05	0.09 ± 0.01
DT-Mem	147K	0.7%	**0.92 ± 0.00**	**0.20 ± 0.01**	**0.95 ± 0.10**

Table 4: Evaluation results on Meta-World ML45 benchmarks

RMDT, PDT, and HDT.

Impact Statement

We do not foresee any significant societal impact resulting from our proposed method. The current algorithm is not yet designed to interact with humans or any realistic environment. It is primarily developed within a controlled and theoretical framework, focusing on improving computational techniques rather than practical applications at this stage. If one chooses to extend our methods to more interactive or real-world scenarios, caution should be exercised to ensure that any safety and ethical concerns are appropriately addressed. As our work is categorized in the offline-RL domain, it is feasible to supplement its training with a dataset that aligns with human intents and values.

This alignment could help in creating systems that are more beneficial and less harmful when eventually deployed in practical applications. However, one must be wary that the way our architecture generalizes across tasks is still not well understood, and as a consequence, we cannot guarantee the generalization of its desirable features: performance, robustness, fairness, etc. Moreover, we contribute to increasing large models accessibility and reducing their ecological impact. Efficient algorithms can lower the computational resources required, leading to less energy consumption and a smaller carbon footprint. In summary, while our current work is confined to theoretical advancements, the potential extension to practical applications must be approached with diligence. Ensuring safety, ethical integrity, and sustainability will be paramount as these methods evolve from offline settings to more dynamic and human-centric environments.

References

Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., and Ionescu, C. Using fast weights to attend to the recent past. *Advances in neural information processing systems*, 29, 2016.

Baddeley, A. Working memory: looking back and looking forward. *Nature reviews neuroscience*, 4(10):829–839, 2003.

Bessonov, A., Staroverov, A., Zhang, H., Kovalev, A. K., Yudin, D., and Panov, A. I. Recurrent memory decision transformer. *arXiv preprint arXiv:2306.09459*, 2023.

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., van den Driessche, G., Lespiau, J., Damoc, B., Clark, A., de Las Casas, D., Guy, A., Menick, J., Ring, R., Hennigan, T., Huang, S., Maggiore, L., Jones, C., Cassirer, A., Brock, A., Paganini, M., Irving, G., Vinyals, O., Osindero, S., Simonyan, K., Rae, J. W., Elsen, E., and Sifre, L. Improving language models by retrieving from trillions of tokens. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2206–2240. PMLR, 2022.

Brandfonbrener, D., Bietti, A., Buckman, J., Laroche, R., and Bruna, J. When does return-conditioned supervised learning work for offline reinforcement learning? In *NeurIPS*, 2022.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

Chen, C., Chen, X., Ma, C., Liu, Z., and Liu, X. Gradient-based bi-level optimization for deep learning: A survey. *CoRR*, abs/2207.11719, 2022.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. In *NeurIPS*, pp. 15084–15097, 2021.

Clark, A., de Las Casas, D., Guy, A., Mensch, A., Paganini, M., Hoffmann, J., Damoc, B., Hechtman, B. A., Cai, T., Borgeaud, S., van den Driessche, G., Rutherford, E., Hennigan, T., Johnson, M. J., Cassirer, A., Jones, C., Buchatskaya, E., Budden, D., Sifre, L., Osindero, S., Vinyals, O., Ranzato, M., Rae, J. W., Elsen, E., Kavukcuoglu, K., and Simonyan, K. Unified scaling laws for routed language models. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pp. 4057–4086. PMLR, 2022.

Cowan, N. What are the differences between long-term, short-term, and working memory? *Progress in brain research*, 169:323–338, 2008.

Csordás, R. and Schmidhuber, J. Improving differentiable neural computers through memory masking, deallocation, and link distribution sharpness control. *arXiv preprint arXiv:1904.10278*, 2019.

Das, S., Giles, C. L., and Sun, G.-Z. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Proceedings of The Fourteenth Annual Conference of Cognitive Science Society. Indiana University*, volume 14, 1992.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, pp. 4171–4186. Association for Computational Linguistics, 2019.

Dooley, J. The archival advantage: Integrating archival expertise into management of born-digital library materials. *Archival Science Special Issue on Archiving Research Data*, 7(1), March 2007.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*. OpenReview.net, 2021.

Eslami, S. M. A., Heess, N., Weber, T., Tassa, Y., Szepesvari, D., Kavukcuoglu, K., and Hinton, G. E. Attend, infer, repeat: Fast scene understanding with generative models. In *NIPS*, pp. 3225–3233, 2016.

Fan, A., Gardent, C., Braud, C., and Bordes, A. Augmenting transformers with knn-based composite memory for dialog. *Transactions of the Association for Computational Linguistics*, 9:82–99, 2021.

Goldman-Rakic, P. S. Cellular basis of working memory. *Neuron*, 14(3):477–485, 1995.

Goyal, A., Didolkar, A. R., Lamb, A., Badola, K., Ke, N. R., Rahaman, N., Binas, J., Blundell, C., Mozer, M. C., and Bengio, Y. Coordination among neural modules through a shared global workspace. In *ICLR*. OpenReview.net, 2022.

Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. In *ICLR*. OpenReview.net, 2022.

Humphreys, P. C., Guez, A., Tieleman, O., Sifre, L., Weber, T., and Lillicrap, T. P. Large-scale retrieval for reinforcement learning. In *NeurIPS*, 2022.

Janner, M., Li, Q., and Levine, S. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2021.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Lee, K., Nachum, O., Yang, M., Lee, L., Freeman, D., Guadarrama, S., Fischer, I., Xu, W., Jang, E., Michalewski, H., and Mordatch, I. Multi-game decision transformers. In *NeurIPS*, 2022.

Martins, P. H., Marinho, Z., and Martins, A. F. T. ∞-former: Infinite memory transformer. In *ACL (1)*, pp. 5468–5485. Association for Computational Linguistics, 2022.

Munkhdalai, T. and Yu, H. Meta networks. In *International conference on machine learning*, pp. 2554–2563. PMLR, 2017.

Munkhdalai, T., Yuan, X., Mehri, S., and Trischler, A. Rapid adaptation with conditionally shifted neurons. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3664–3673. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/munkhdalai18a.html.

Munkhdalai, T., Sordoni, A., Wang, T., and Trischler, A. Metalearned neural memory. *Advances in Neural Information Processing Systems*, 32, 2019.

Neyshabur, B., Li, Z., Bhojanapalli, S., LeCun, Y., and Srebro, N. The role of over-parametrization in generalization of neural networks. In *ICLR (Poster)*. OpenReview.net, 2019.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Adler, T., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G. K., et al. Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*, 2020.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pp. 1842–1850. PMLR, 2016.

Schmidhuber, J. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Sukhbaatar, S., Weston, J., Fergus, R., et al. End-to-end memory networks. *Advances in neural information processing systems*, 28, 2015.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NIPS*, pp. 5998–6008, 2017.

von Oswald, J., Henning, C., Sacramento, J., and Grewe, B. F. Continual learning with hypernetworks. In *ICLR*. OpenReview.net, 2020.

Wang, W., Dong, L., Cheng, H., Liu, X., Yan, X., Gao, J., and Wei, F. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 36, 2024.

Wu, C.-Y., Li, Y., Mangalam, K., Fan, H., Xiong, B., Malik, J., and Feichtenhofer, C. Memvit: Memory-augmented multiscale vision transformer for efficient long-term video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13587–13597, 2022a.

Wu, Y., Rabe, M. N., Hutchins, D., and Szegedy, C. Memorizing transformers. *arXiv preprint arXiv:2203.08913*, 2022b.

Xu, M., Shen, Y., Zhang, S., Lu, Y., Zhao, D., Tenenbaum, J. B., and Gan, C. Prompting decision transformer for few-shot policy generalization. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pp. 24631–24645. PMLR, 2022.

Xu, M., Lu, Y., Shen, Y., Zhang, S., Zhao, D., and Gan, C. Hyper-decision transformer for efficient online policy adaptation. *CoRR*, abs/2304.08487, 2023.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019. URL https://arxiv.org/abs/1910.10897.

Zheng, Q., Zhang, A., and Grover, A. Online decision transformer. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pp. 27042–27059. PMLR, 2022.

A. Implementation Details

A.1. Hyper-parameters

This section will delve into the specifics of the model parameters. Understanding these parameters is key to understanding the model's workings. It is worth noting that the source code for this model is publicly available at https://anonymous.4open.science/r/DT-Mem-Submission277/README.md. This allows for a deeper understanding of the model's inner workings and may facilitate replicating its results.

Hyperparameters	Value
K (length of context)	28
dropout rate	0.1
maximum epochs	1000
steps for each epoch	1000
optimizer learning rate	1e-4
weight decay	1e-4
gradient norm clip	1.
data points for each dataset	500,000
batch size	64
memory slots	1290
activation	GELU
optimizer	AdamW
scheduler	LambdaLR

Table 5: Hyperparameters for DT-Mem training

A.2. Training and fine-tuning algorithm

In this section, we present the pre-training DT-Mem in Appendix A.2 and fine-tuning DT-Mem with LoRA in Appendix 5.5.

We pre-train DT-Mem on multiple offline datasets. Each gradient update of the DT-Mem model considers information from each training task.

Algorithm 1 Pre-train DT-Mem

1: **for** T episodes **do**
2:     **for** Task $\mathcal{T}_i \in T^{\text{train}}$ **do**
3:         Sample trajectories $\tau = (s_0, a_0, r_0, \cdots, s_H, a_H, r_H)$ from the dataset $\mathcal{D}_i$.
4:         Split trajectories into different segments with length K and calculate return-to-go in the input sequence.
5:         Given $\hat{\tau}_{t+1:t+K}$, compute the sequence embedding e_{seq}.
6:         Update the memory module and retrieve the relative information as $\boldsymbol{E}_{\text{out}}$
7:         Given $\boldsymbol{E}_{\text{out}}$, predict actions $\tilde{a}_t$, reward $\tilde{r}_t$, and return-to-go $\tilde{R}_t$.
8:         Compute the loss according to Eqn. 1.
9:         Update all module parameters.
10:     **end for**
11: **end for**

We fine-tune the memory module to adapt to each downstream task. To achieve this, we fix the pre-trained DT-Mem model parameters and add additional LoRA parameters for the memory module feed-forward neural networks. The fine-tuning dataset is used to update these LoRA parameters only.

Algorithm 2 Fine-tuning DT-Mem

Require: Fine-tuning dataset $\mathcal{T}^i \in T^{\text{test}}$ dataset $\mathcal{D}^i$ for $\mathcal{T}^i$. Initialize LoRA parameters $\hat{B}^q, \hat{B}^k, \hat{B}^v, \hat{A}^q, \hat{A}^k, \hat{A}^v, B^q, A^q, B^k, A^k$.

1: **for** T steps **do**
2:     Split trajectories into different segments with length K and calculate return-to-go in the input sequence.
3:     Given $\hat{\tau}_{t+1:t+K}$, compute the sequence embedding e_{seq}.
4:     Update memory module using $\hat{Q} = M(\hat{W}^q + \hat{B}^q\hat{A}^q)$, $\hat{K} = M(\hat{W}^k + \hat{B}^k\hat{A}^k)$, $\hat{V} = M(\hat{W}^v + \hat{B}^v\hat{A}^v)$, $Q = M(W^q + B^qA^q)$, $K = M(W^k + B^kA^k)$
5:     Retrieve the relative information as E_{out}
6:     Given E_{out}, predict actions $\tilde{a}_t$, reward $\tilde{r}_t$, and return-to-go $\tilde{R}_t$.
7:     Compute the loss according to Eqn. 1.
8:     Update LoRA parameters only.
9: **end for**

Algorithm 3 Memory Operations

1: **Step 0: Memory Module Initialization**
2: Initialize memory as a random matrix M where each row $m_i \in \mathbb{R}^d$ and $i \in [0, N]$.
3:
4: **Step 1: Input Sequence Organizing**
5: Restructure input sequence into format $< \hat{r}_t, s_t, a_t >$.
6: Define embedding functions $g_s(s) = e_s, g_a(a) = e_a, g_r(\hat{r}) = e_{\hat{r}}$.
7: Concatenate embeddings to form input sequence $E = [\cdots; e_{s_t}, e_{a_t}, e_{\hat{r}_t}; \cdots]$.
8:
9: **Step 2: Content-based Address**
10: Use attention to locate the memory slot for new input.
11: Calculate position address $w = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)$.
12: Define $Q = MW^q$ and $K = EW^k$.
13:
14: **for** N Times memory operations do **do**
15:     **Step 3: Memory Update**
16:     Calculate erasing vector ϵ^e and adding vector ϵ^a.
17:     Define $\hat{Q} = M\hat{W}^q, \hat{K} = E\hat{W}^k, \hat{V} = E\hat{W}^v$.
18:     Compute writing strength $\beta = \text{softmax}\left(\frac{\hat{Q}\hat{K}^T}{\sqrt{d}}\right)$.
19:     Calculate $\epsilon^e = w \odot (1 - \beta)$.
20:     Calculate $\epsilon^a = (w \odot \beta)\hat{W}^v x$.
21:     Update memory $M_n = M_{n-1} \odot (1 - \epsilon^e) + \epsilon^a$.
22:
23:     **Step 4: Memory Retrieve**
24:     Retrieve information from memory for decision-making.
25:     Compute read position vector using content-based address.
26:     Retrieve memory $E_{\text{out}} = w \odot M_n$.
27:     $E = E_{\text{out}}$
28: **end for**
29: output E for action decoder.

B. Additional Experiments

B.1. Evaluation Parameters

To evaluate the performance of our model on Atari games, we randomly selected 16 different random seeds for evaluation. We chose the random seed by multiplying the number of runs by 100. For example, the random seed for run 6 is $6 \times 100 = 600$.

B.2. Training Efficiencies

We illustrate the model training curve in Figure 5 to demonstrate training efficiency. It is reasonable to report the prediction loss on the training dataset for the training curve since we use a supervised loss. Here, the prediction accuracy consists of three parts: action prediction accuracy, reward prediction accuracy, and return prediction accuracy. The y-axis shows the average value of these three predictions, and the x-axis is the relative walltime based on the same computing resources.
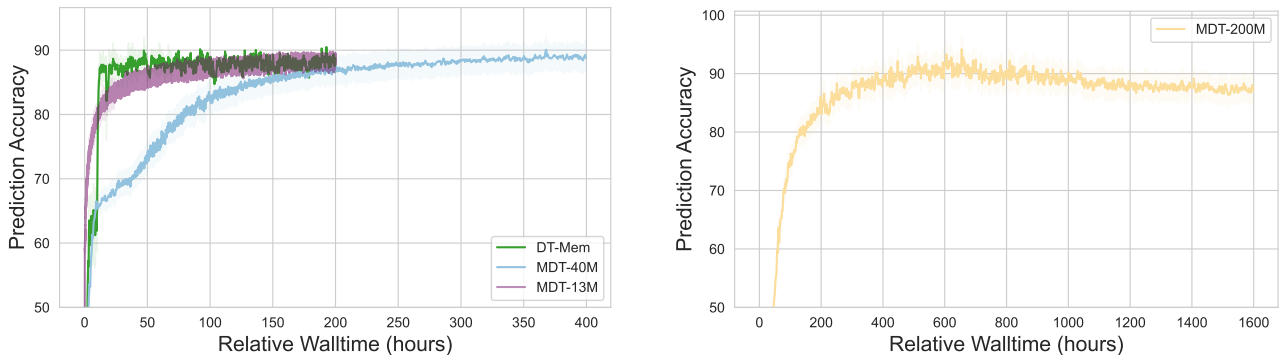

Figure 5: This graph shows the prediction accuracy during training. Each curve represents three runs with different random seeds. For better visualization, MDT-200M is displayed in a separate figure.

B.3. The analysis of memory size

In this section, we investigate the impact of the memory module size on the performance of DT-Mem. We employ the Bayes optimization strategy to tune the parameters. It's worth noting that the memory size is calculated by multiplying the number of memory slots by the size of each slot, which is fixed at 512 dimensions for the sake of evaluation simplicity. To expedite the hyper-parameter tuning process, we present the evaluation results based on 100k training steps of the StarGunner game. We assess various configurations of memory slots and calculate their corresponding average rewards over 16 runs. Figure 6 reveals several key findings: (1)Increasing the size of memory slots leads to a higher reward accumulation. Notably, there is a significant performance boost when the number exceeds 1200. (2)In summary, when the number of memory slots exceeds 1800, the system's performance decreases. This decline occurs because there is a trade-off between the number of memory slots and the training steps. With a larger number of memory slots, it becomes necessary to allocate more training time.

B.4. Ablation study of LoRA adaptor

| | Meta-World ML45 Performances | | | Data size | Model | |
	Train	Test (no-FT)	Test (FT)		Adap.	Per.
DT-Mem (hyper-net)	0.92 ± 0.01	$\mathbf{0.23 \pm 0.10}$	0.81 ± 0.15	30	5.7M	43.8%
DT-Mem	$\mathbf{0.92 \pm 0.00}$	0.20 ± 0.01	$\mathbf{0.95 \pm 0.10}$	10	147K	0.7%

Table 6: Ablation study results on Meta-World ML45 benchmarks. DT-Mem (hyper-net) denotes the variation of DT-Mem, which substitutes the LoRA adaptation module with hyper-networks. Adap. stands for adaptation parameters, and Per. stands for the percentage of the original model.

In this section, we conduct an ablation study of the LoRA-based memory adaptor. We substitute the LoRA adaptor with hyper-networks. Specifically, the parameters of the memory module are generated from hyper-networks. This approach
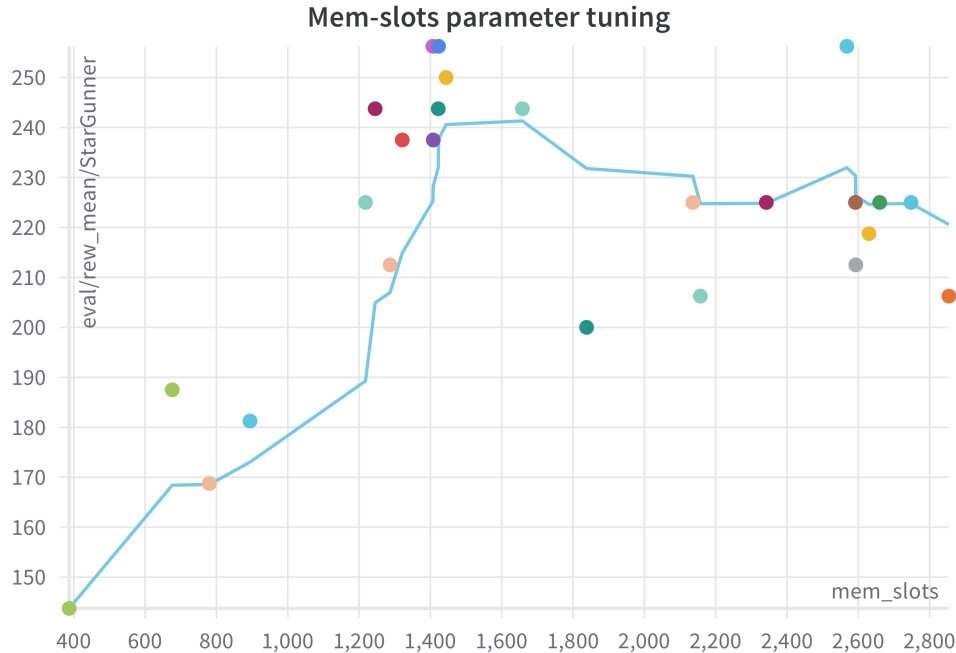
Figure 6: The parameter tuning results for the number of memory slots. The blue curve shows the like from left to right over the x-axis and plots the running average y value.

is based on (von Oswald et al., 2020), where hyper-networks take task-related information as input and generate the corresponding networks for the downstream MLP. We use the same approach and generate parameters that are conditioned on two types of inputs: the task embedding from the task encoder and the sequence embeddings from the Transformer module.

To generate task embeddings, we adopt the same idea from PDT (Xu et al., 2022), demonstrating that a small part of trajectories can represent task-related information. We further extend this idea to extract the task information fully. To achieve this goal, we use a Neural Network as a task encoder. Specifically, this task encoder is implemented as a Transformer encoder-like structure (Vaswani et al., 2017). We first formulate the first i steps of collected trajectories $\tau_{0:i} = (s_0, a_0, r_0, \cdots, s_i, a_i, r_i)$ as a task specific information. The task trajectory $\tau_{0:i}$ is treated as a sequence of inputs to the task encoder. The output of the task encoder is a task embedding $e_{task} \in \mathbb{R}^d$, where d is the dimension of the embedding.

Then, we concatenate the task embedding and sequence embedding $e = [e_{task}; e_{\text{seq}}]$ and input them to the hyper-networks. Specifically, we define the hyper-network as a function of $f_\omega(\cdot)$ parameterized by ω. The output $\Theta = f_\omega(e)$ is a set of parameters for the memory module.

According to the evaluation results in Table 6, including a hyper-network in the DT-Mem model improves generalization without the need for fine-tuning. However, it is worth noting that the hyper-network variant of DT-Mem (hyper-net) exhibits higher variance compared to DT-Mem. The primary reason for this higher variance is the uncertainty arising from the task information. Different task-related sequences are collected in each run, resulting in varying generated parameters for the memory module. Regarding the task fine-tuning results, we observe that the LoRA module outperforms other methods. This finding indicates that fine-tuning with LoRA enhances the model's adaptability. We hypothesize that the size of the hyper-networks model plays a role in these results. Fine-tuning a large model size (5.7M) with a small step-size (100k steps in our case) becomes challenging. In an effort to improve hyper-networks fine-tuning performance, we increased the fine-tuning dataset from 10k episodes to 30k episodes. These findings suggest that LoRA-based fine-tuning demonstrates better data efficiency.

The motivations for using LoRA to fine-tune the model can be concluded in the following two reasons:

Hu et al. (2022) suggests that the LoRA method, in contrast to other adapters, maintains model quality without introducing

inference latency or shortening input sequence length. Furthermore, it facilitates rapid task-switching in service deployments by sharing most model parameters. Parameter-efficient fine-tuning (PEFT) refines a limited number of model parameters, preserving most of the pre-trained LLM parameters, which reduces computational and storage demands (Hu et al., 2022). This approach also addresses catastrophic forgetting [4] and has outperformed standard fine-tuning in low-data and out-of-domain situations [5]. Besides, the results of full parameter fine-tuning vs. PEFT are shown in Table 7:

Game	PEFT	FFT-Single	FFT-All
Alien	127.4%	116.8%	113.9%
MsPacman	130.8%	122.8	77.1%
SpaceInvaders	100.8%	86.8%	73.4%
StarGunner	158.3%	55.7%	40.6%

Table 7: Performance comparison of PEFT across various games

where PEFT stands for LoRA fine-tuning for all games together, FFT-single means full-parameter fine-tuning on a single game only, FFT-All stands for full-tine-tuning on all games together. Results are DQN-normalized score.

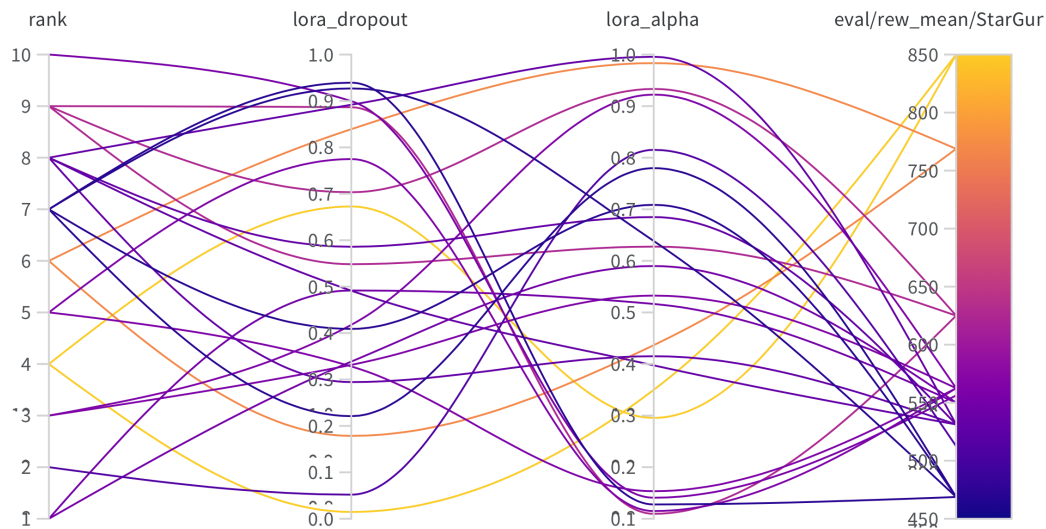## B.5. LoRA hyper-parameters tuning

Figure 7: LoRA hyper-parameters tuning results.

In this section, we explore the impact of LoRA hyper-parameters on the final fine-tuning results. LoRA employs three hyper-parameters: rank, lora_dropout, and lora_alpha. The rank parameter, denoted as m, determines the low-rank of adaptation matrices $\boldsymbol{B} \in \mathbb{R}^{n \times m}$ and $\boldsymbol{A} \in \mathbb{R}^{m \times d}$, as described in Section 4.4. The lora_dropout refers to the dropout rate applied to the LoRA neural networks, while lora_alpha controls the scaling factor of the LoRA outputs. Figure 7 presents the fine-tuning results, with the last column (**eval/rew_mean/StarGur**) specifically showcasing the fine-tuning results for the StarGunner game. To obtain the optimal set of parameters, we employ the Bayesian optimization method for parameter tuning, which suggests various parameter combinations that maximize the fine-tuning results.

We further analyze these parameters and present the findings in Table 8. To gain insights, we utilize two widely used metrics in the MLOps platform Weights&Biases[2].

[2]For better understanding, please refer to `https://docs.wandb.ai/guides/app/features/panels/parameter-importance?_gl=1*4s7cuj*_ga*MTQxNjYxODU0OC4xNjgzNjY4Nzg3*_ga_JH1SJHJQXJ*MTY4NDc5NDkzNS40MS4xLjE2ODQ3OTQ5NDIuNTMuMC4w`

Parameter	Importance score	Correlation score
rank	**0.486**	-0.132
lora_dropout	0.285	-0.561
lora_alpha	0.229	0.550

Table 8: Analysis of LoRA hyper-parameters

Regarding the **importance score**, we train a random forest model with the hyper-parameters as inputs and the metric as the target output. We report the feature importance values derived from the random forest. This hyper-parameter importance panel disentangles complex interactions among highly correlated hyper-parameters. It facilitates fine-tuning of hyper-parameter searches by highlighting the hyper-parameters that significantly impact the prediction of model performance.

The **correlation score** represents the linear correlation between each hyper-parameter and the chosen metric (in this case, val_loss). A high correlation indicates that when the hyper-parameter has a higher value, the metric also tends to have higher values, and vice versa. Correlation is a useful metric, but it does not capture second-order interactions between inputs and can be challenging to compare when inputs have widely different ranges.

As shown in Table 8, rank emerges as the most important hyper-parameter that requires careful tuning. The correlation score of rank is -0.132, indicating that a smaller rank number leads to better fine-tuning results. Based on our findings, a rank value of 4 yields the best outcome. Lora_dropout and lora_alpha exhibit similar importance scores, suggesting that these two parameters can be treated equally. The correlation score reveals that a smaller lora_dropout value and a larger lora_alpha value result in improved performance.

B.6. Ablation studies on different input sequence organizing choices

We examine two distinct approaches to input organization. The first approach is adopted from the trajectory Transformer as outlined in (Janner et al., 2021), which organizes the inputs as $(s_1, \ldots, s_t, a_1, \ldots, a_t, r_1, \ldots, r_t)$, grouping states, actions, and rewards accordingly. The second approach is derived from the decision Transformer as described in (Chen et al., 2021), and is the method utilized in this study.

Game	Choice one	Choice two (Ours)
Alien	211.9	239.6
MsPacman	637.1	713.4
SpaceInvaders	165.7	171.2
StarGunner	620.7	709.3

Table 9: Ablation studies on different choices of organizing. Each value represents raw scores in Atari games.

From the table above, we observe minor differences between the two sets of inputs. However, the variance in outcomes between the two methodologies is not significant. Therefore, we empirically adopt the second approach for our design in this paper.

B.7. Ablation studies with DT

As shown in Table 10, the leftmost column represents the size of the dataset used for training. As seen in the table above, the generalized agent DT-Mem outperforms when compared to training on the DT-20M 50k datasets. Fine-tuning DT-Mem on 50k datasets yields better results than training DT-20M on 200k datasets.

B.8. Full Fine-tuning vs. LoRA

Full Fine-tuning (FFT) vs. LoRA: To assess whether the use of LoRA adversely affects performance, we conducted experiments contrasting Full Fine-Tuning (FFT) of memory parameters with LoRA. In this context, FFT-single refers to fine-tuning all parameters exclusively on a single game, whereas FFT-All represents fine-tuning simultaneously on the entire set of games. The results are DQN-normalized score. Based on above results, we conclude the following observations:

	DT-Mem (Ave)	DT-Mem FT (Ave)	DT-20M (Ave)
10k	-	-	10.1%
20k	-	-	9.8%
30k	-	-	15.3%
40k	-	-	22.6%
50k	51.0%	127.4%	41.8%
100k	-	-	83.1%
200k	-	-	120.3%
500k	-	-	170.7%

Table 10: Comparison with DT in different fine-tuning datasets

Game	PEFT	FFT-Single	FFT-All
Alien	127.4%	116.8%	113.9%
MsPacman	130.8%	122.8	77.1%
SpaceInvaders	100.8%	86.8%	73.4%
StarGunner	158.3%	55.7%	40.6%

- LoRA appears to be the most consistently effective strategy across the games provided. - While **FFT-Single** occasionally outperforms PEFT (like in Alien), **FFT-All** consistently trails behind the other two.

The reason full fine-tuning is not comparable to PEFT comes from the following parts: 1. Fine-tuning dataset size. Note that we only use 50k data in LoRA, and full fine-tuning compares to 500k used in MDT paper 2. The advantages of LoRA include its ability to tackle catastrophic forgetting, as well as its superior performance compared to standard fine-tuning in scenarios with limited data and those that are outside of the usual domain.

B.9. Analyze of input misleading

we conducted an experiment to assess the robustness of the proposed method against input distortion. This involved adding Gaussian noise to the input frames of Atari games. Specifically, we set the mean to 0 and experimented with various standard deviation values. The results are detailed in the table below:

	Alien	MsPacman	SpaceInvaders	StarGunner
MDT	3.8%	13.2%	8.6%	2.3%
DT-Mem	51.0%	69.3%	53.6%	62.2%
DT-Mem (std=0.5)	55.3%	67.6%	53.0%	57.8%
DT-Mem (std=1)	35.6%	56.1%	40.0%	34.6%
DT-Mem (std=2)	25.9%	35.6%	30.5%	21.1%

From the results above, we conclude that the proposed DT-Mem demonstrates greater robustness to noisy inputs compared to the MDT method. This is evident as the DT-Mem consistently outperforms MDT under various levels of Gaussian noise. Notably, the performance with a standard deviation of 0.5 shows minimal difference compared to the no-noise scenario, illustrating DT-Mem's effectiveness in mitigating the impact of varying input distortions.

C. Memory Module Visualization

Figure 8 illustrates the visualization of the memory module. Since memory operations are trained in conjunction with the Transformer module, we randomly select a later training episode to mitigate uncertainties regarding operational parameters. Due to time constraints, we trained on only two games simultaneously. In the revised version of the paper, we intend to provide visualizations for all games. For clearer visualization, we opted for a memory module of a smaller size, containing 128 memory slots.

Let's first discuss how memory modules update within the same game. As observed in the figure, for the Amidar game, the actively updated memory slots concentrate around rows 18, 84, and 117. This pattern is consistent across episodes, albeit with reduced activity. Such a trend indicates that during each training iteration, the Transformer agent tends to overwrite the

same memory slot contents. We note a similar observation in the Assault game. Furthermore, we observe that the memory module's activity diminishes in later episodes. For instance, in the Assault game, the active memory slot in row 12 during episode 200k becomes less active by episode 201k. We hypothesize that as training progresses, the accumulated knowledge becomes sufficiently robust for retrieval, reducing the need for updates.

Moving on, when comparing the activity of memory slots across different games, there are intriguing overlaps. For instance, comparing Amidar 200k and Assault 200k reveals that memory slots around row 120 are active in both games. We surmise that this region retains cross-task knowledge shared between games. Additionally, the varying attention across other memory slots demonstrates how these slots assist the agent in decision-making across diverse games.
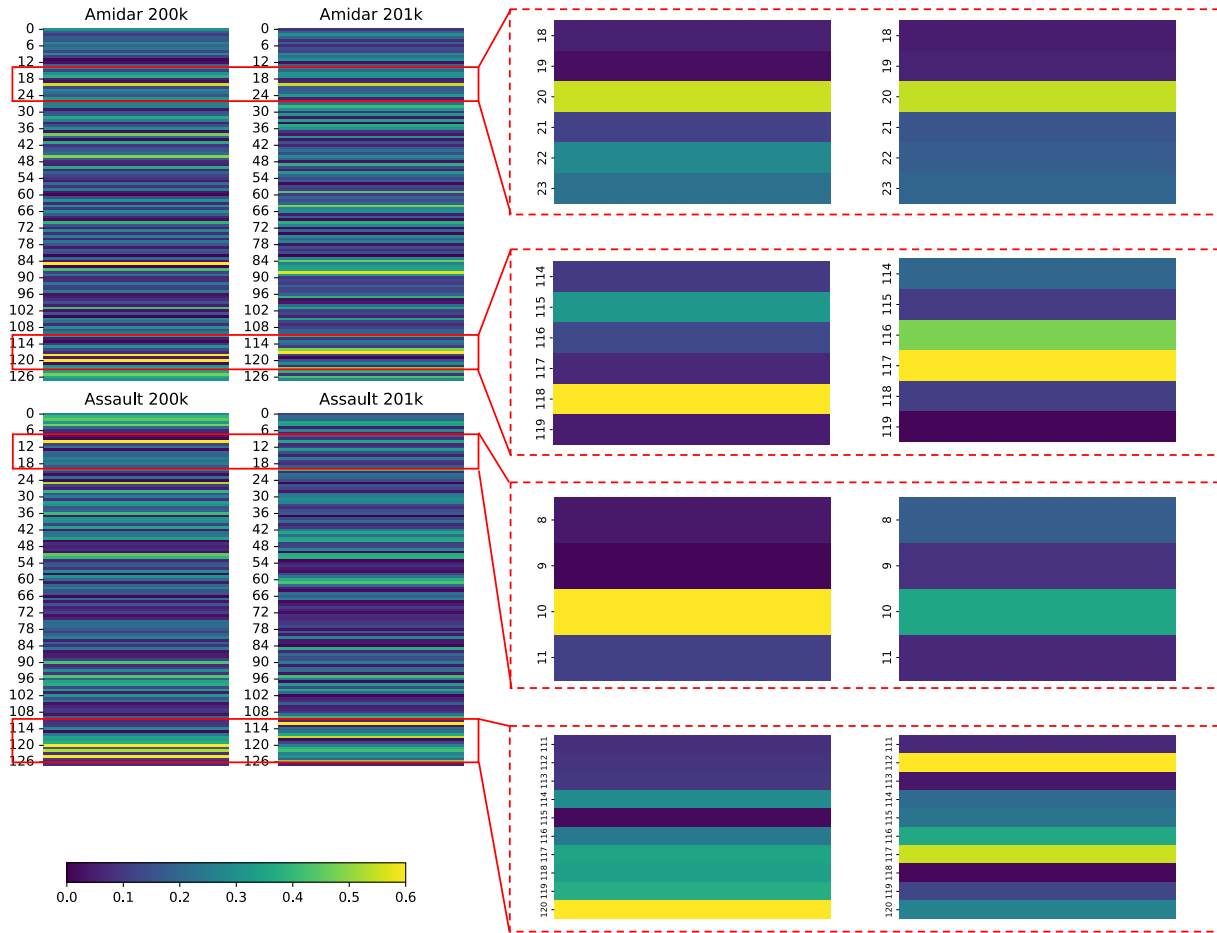

Figure 8: This visualization represents the memory module. In the figure, each row is derived from the mean of a vector that signifies a memory slot. Each depiction calculates the variation between two write operations in a single episode for each memory slot. Lighter shades indicate memory slots that have been actively updated post-write operations. The encircled areas highlight the comparison of active memory slots across different episodes.

D. Limitations

The first limitation of our work is the sample efficiency of memory fine-tuning. The 10% fine-tuning dataset is still sizeable, and we plan to explore more sample-efficient methods in the future. We could, for instance, consider a setting with more tasks, each one with less data, so that the inter-task generalization would be even more crucial to its performance. This work does not propose a control strategy for collecting data on a new task. For future work, we plan to investigate online data collection methods, including designing and learning exploration strategies for efficient fine-tuning on new tasks. Finally, the approach has been intuitively motivated, and it would be valuable to have a theoretical grounding that would show the

structural limits of large models and how equipping them with a memory component overcomes them.

E. Comparison of DT-Mem and Neural Episodic Control (NEC) in Writing and Reading Memory

Memory Mechanism

- **NEC:** Utilizes a Differentiable Neural Dictionary (DND) for storing experiences with separate memories for each action, containing state representations (keys) and value function estimates (values).
- **DT-Mem:** Integrates a working memory module within a Transformer framework, focusing on storing, blending, and retrieving information for improving training efficiency and generalization.

Writing to Memory

- **NEC:** Continuously adds new experiences and rapidly updates value function estimates in memory.
- **DT-Mem:** Modifies or replaces existing information in the memory matrix using an attention mechanism to calculate correlations and update memory with the attended weight of the input sequence.

Reading from Memory

- **NEC:** Implements context-based lookups in the DND to retrieve values, outputting a weighted sum based on the similarity between the current state and stored keys.
- **DT-Mem:** Employs content-based addressing for memory retrieval, using attention mechanisms to read from the updated memory and inform decision-making.

Distinctive Features and Advantages

- **NEC:** Designed for rapid assimilation and action upon new experiences with specialized and swift updates for each action.
- **DT-Mem:** Aims to enhance generalization across tasks and reduce catastrophic forgetting by integrating memory with the Transformer's sequential data handling capabilities.