2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Merging Decision Transformers: Weight Averaging for Forming Multi-Task Policies

Daniel Lawson and Ahmed H. Qureshi

***Abstract*— Recent work has shown the promise of creating generalist, transformer-based, models for language, vision, and sequential decision-making problems. To create such models, we generally require centralized training objectives, data, and compute. It is of interest if we can more flexibly create generalist policies by merging together multiple, task-specific, individually trained policies. In this work, we take a preliminary step in this direction through merging, or averaging, subsets of Decision Transformers in parameter space trained on different MuJoCo locomotion problems, forming multi-task models without centralized training. We also demonstrate the importance of various methodological choices when merging policies, such as utilizing common pre-trained initializations, increasing model capacity, and utilizing Fisher information for weighting parameter importance. In general, we believe research in this direction could help democratize and distribute the process that forms multi-task robotics policies. Our implementation is available at https://github.com/daniellawson9999/merging-decision-transformer.**

I. INTRODUCTION

Following advances in sequence modeling, there have been various works studying the feasibility of creating generalist transformer-based policies across many control tasks, with work such as Multi-game Decision Transformer [21] creating multi-task policies for Atari, or Gato [30] which learns a single generalist across many modalities, tasks, and robot embodiments. We consider the nature of which these multi-task robotics policies can be formed. Is it possible to more flexibly create multi-task control policies with reduced demands for centralizing all data and training simultaneously?

We believe an initial step in this direction is investigating the feasibility of combining single Decision Transformers (DT) trained on different tasks. Specifically, we consider DTs trained on MuJoCo locomotion problems. In this work, we look at combining through the perspective of weight merging [5], [24], [36]. For example, given two trained DTs on different environments, merging refers to taking all parameters or a subset of parameters, say those associated with attention, and replacing the parameters in both models with combinations of their original parameters.

In this work, we assess the feasibility of merging transformers for control tasks, and provide several techniques to achieve successful merging, forming our contributions:

1) We begin by investigating how merging individual layers and subsets of DTs affect model performance and find that we can directly merge and swap certain parameters of Decision Transformers trained on different MuJoCo environments (HalfCheetah, Walker, Hopper) with, in some cases, minimal decrease in performance. This leads us to investigate the role of attention in DTs, finding some DTs do not heavily rely on attention.
2) We propose a method for creating multi-task DTs through merging certain parameters, freezing those merged, and then independently finetuning un-merged parts with *Merge-Freeze-Finetune* (MFF). This creates a multi-task DT without centralized data or training objectives. We also demonstrate improving existing sub-optimal multi-task policies through merging without MFF of finetuned policies on higher quality data.
3) We show that common initialization can lead to better performance after merging with Decision Transformers. Specifically, we use common initializations from language modeling [31], [33]. We also investigate methods to improve merging, such as co-training and regularization to model initializations, increasing model size, and utilizing Fisher information.

The authors are with the Department of Computer Science, Purdue University, West Lafayette, IN 47907 USA (email: lawson95@purdue.edu; ahqureshi@purdue.edu)

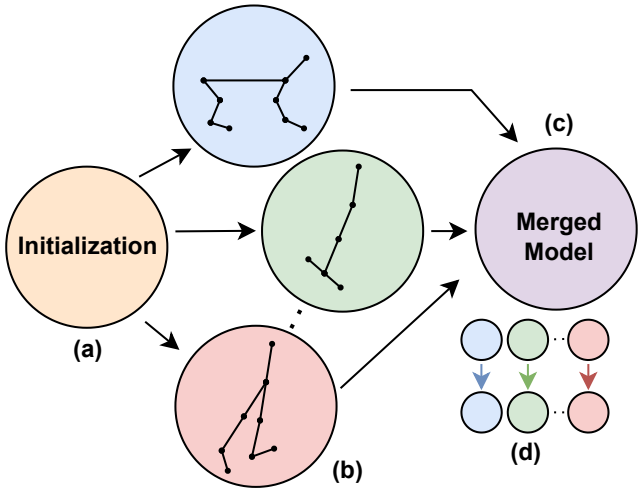


Fig. 1: **Process for merging Decision Transformers**. Starting from common, or unique initializations (a), we separately train models for different offline reinforcement problems (b). After training, task-specific models are combined by merging over a subset of parameters (c). After merging, we may improve performance by freezing merged parameters, and then separately finetuning, updating unique un-merged parameters (*Merge-Freeze-Finetune*) (d).

II. BACKGROUND

A. Transformers

Transformers [35] are a common neural network architecture for modeling sequences. We consider causal decoder-

only transformers like GPT [29]. A transformer is composed of several successive transformer blocks, which each consist of (multi-headed) self-attention layers, multi-layer perceptron (MLP) layers, as well as layer normalization [2]. Given a sequence of inputs of length n and embedding size d $X \in \mathbb{R}^{d \times n}$, a self-attention layer projects each input to queries (Q), keys (K), and values (V) with parameters $\{W_q, b_q\}, \{W_k, b_k\}, \{W_v, b_v\}$ respectively. We then perform Attention(X): $\text{softmax}(\frac{QK^T}{\sqrt{d}})V$. We apply this operation for each head in parallel, stack outputs, resulting in $Y \in \mathbb{R}^{Hd \times n}$, where H is the number of heads, which is then projected by W_o, resulting in $X' = W_o Y \in \mathbb{R}^{d \times n}$ [28]. Decision Transformers, apply layer normalization before and after attention layers, where each layer has learnable affine transform parameters γ, β. Each MLP contains one hidden layer (and one output layer), with parameters W_1, W_2. Transformers also have a residual connection after each attention and MLP layer respectively, so the actual output consists of the transformation from the layer added with its input.

B. Offline reinforcement learning with decision transformers

We consider problems that are modeled by a Markov decision process (MDP) with states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, unknown transition dynamics $p(s'|s,a)$ and reward function $r(s,a)$. Traditionally in RL, we aim to learn an optimal policy that maximizes expected (discounted) return through interaction with the environment. Instead, in offline RL, we learn without interaction using a static dataset. A dataset consists of a set of trajectories, where each trajectory has the form $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_N, a_N, r_N)$, consisting of a sequence of states, actions, and return at each timestep until timestep N. With this data, we aim to find a policy $\pi(a|\cdot)$ that maximizes expected return $\mathbb{E}[\sum_{t=0}^{N} r_t]$, where the expectation is over the distribution induced by transition dynamics and the policy π. Decision Transformer [4] casts offline RL as a sequence modeling problem using causal autoregressive transformers. Particularly, DT models the actions in the sequence $\tau = (\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, \ldots, \hat{R}_T, s_T, a_T)$, where the return-to-go (RTG) is the undiscounted sum of future reward: $\hat{R}_t = \sum_{t'=t}^{T} r_{t'}$. To make a prediction, each input is embedded using linear projections with added positional encoding and normalization, passed through the transformer, and then a final linear layer predicts the action. We refer to transformer parameters as those only associated with transformer layers and not input or output projections.

III. Methods

In this work, we consider the problem of merging or combining neural networks trained on independent robotics control tasks. Given m networks $\{\theta_1, \ldots, \theta_m\}$, we aim to obtain a merged network θ_M which combines the weights of each individual network, obtaining a new model which satisfies all individual task but with a similar number of parameters as just a single model.

To obtain multi-task models, we investigate both direct averaging and fisher merging, which do not require using the prior data, or performing additional training. We demonstrate the success of merging without additional training after the merging step for improving sub-optimal multi-task models in Section V-E.

When merging single-task models trained on separate robotic embodiments, we may encounter interference when merging. If we relax the constraint of performing additional training, and having access to prior data, we study whether we can still form a multi-task model without centralized training by separately finetuning only a few parameters for each task after merging.

A. Merging

Given two trained neural networks with the same architecture, and their sets of parameters or weights θ_A, θ_B, we can interpolate between weights, getting a new model $\theta_\lambda = (1-\lambda)\theta_A + \lambda\theta_B$. With $\lambda = .5$, we directly average weights, obtaining a merged model $\theta_{\lambda=.5} = \theta_M = .5(\theta_A + \theta_B)$. In this work, we are interested in merging subsets of transformer parameters discussed in Section II-A.

Directly interpolating between parameters of randomly initialized models may result in models with high loss [1], [9], but this can be mitigated instead merging models from common initializations. When models θ_A, θ_B are from two finetuned models from a common initialization, θ_{pre}, merging is equivalent to task arithmetic for creating multi-task models [12]. Specifically, a task vector $\tau_x = \theta_x - \theta_{\text{pre}}$ stores the difference between a finetuned and pre-trained model. We can form a multi-task model by adding the averaged task vectors to the original model:

$$\begin{aligned} \theta_{\text{pre}} + \frac{1}{2}(\tau_A + \tau_B) = \theta_{\text{pre}} + \frac{1}{2}((\theta_A - \theta_{\text{pre}}) \\ +(\theta_B - \theta_{\text{pre}})) = \frac{1}{2}(\theta_A + \theta_B) = \theta_M \end{aligned} \tag{1}$$

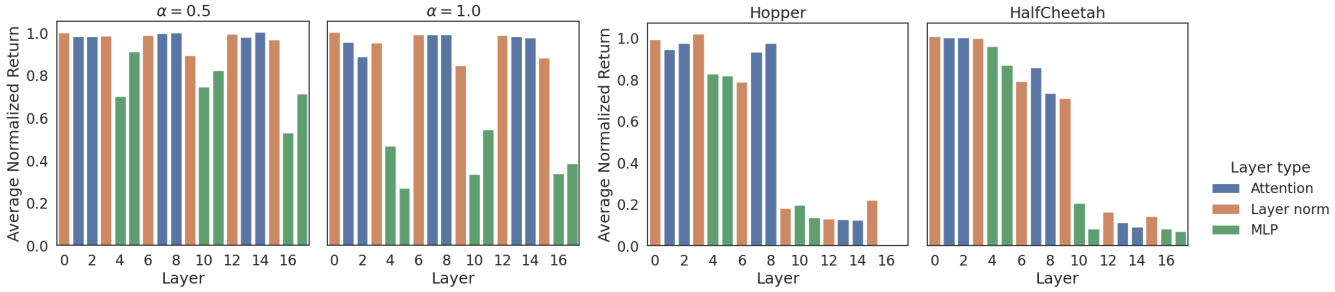
Fisher merging: Another interpretation of merging M models which start from a common initialization is that our goal is to finding the parameters θ_M which maximize the likelihoods of the posterior distribution under all models. If each posterior distribution over parameters is an isotropic Gaussian, then the merged parameters which maximize the joint posterior are:

$$\theta_M = \text{argmax}_\theta \sum_i \log p(\theta|\theta_i, I) = \frac{1}{M}\sum_i \theta_i \tag{2}$$

Instead, Fisher merging [24] considers a Gaussian approximation $p(\theta|\theta_i, F_i)$, where F_i is a diagonal precision matrix containing the Fisher information for each parameter:

$$F_i = \mathbb{E}_{x \sim D_i} \mathbb{E}_{y \sim p_{\theta_i}(y|x_i)} \nabla_{\theta_i} (\log\, p_{\theta_i}(y|x))^2 \tag{3}$$

For each parameter j of model i, this obtains a score F_i^j which represents the precision of the estimate for θ_i^j. This can be computed after training, and stored along with a model. To form a merged model, we can perform a weighted



(a) (b)

Fig. 2: (a): Normalized return after **merging a single layer** at a time, averaged between all pairs of Walker, Hopper, HalfCheetah. Return is normalized by original performance before merging. We evaluate at $\alpha = .5$. an average, and $\alpha = 1$, swapping in a layer from another DT. (b): **Incremental merging** with $\alpha = .5$ merging from HalfCheetah to Hopper (**left**) and Hopper to HalfCheetah (**right**), where the set of merged layers grows as moving further in depth.

average of model parameters in proportion to each model's Fisher information:

$$\theta_M^j = \frac{\sum_{i=1}^{M} F_i^j \theta_i^j}{\sum_{i=1}^{M} F_i^j} \tag{4}$$

As models are trained for different robotic tasks, we find that the scale of gradients vary, so we normalize each parameter's Fisher information by the sum of total information for that model before averaging.

B. Merge-Freeze-Finetune

In some instances, we expect that we can directly merge the weights of several models, but in other situations where models are significantly different, we expect significant decreases in performance after merging. In these instances, we propose *Merge-Freeze-Finetune* (MFF): Merging over a subset of two networks or more networks, freezing parameters in the subset, and then separately finetuning the transformer unmerged parameters and linear input and output projections in both original models, to separately adapt to changes in shared parameters.

C. Pre-trained Initializations

Prior work has shown the importance of common-initialization for merging models trained on different data distributions [12], [24]. In domains like language modeling, we can simply merge finetuned models from common language model initializations, but we do not have readily available equivalent initializations in robotics. However, we propose that language model initializations can be useful for merging models even in a drastically different domains such as robotics control tasks. This is motivated by work which shows that transformers pre-trained on language can learn general representations and parameters that are amenable to transfer [23]. In the context of offline reinforcement learning, [31], [33] showed that initializing Decision Transformers (DT) [4], [11] with pre-trained language models can increase convergence speed and performance on the D4RL [10] MuJoCo [34] benchmarks, showing transfer between completely different modalities.

Following this motivation, later, in Section V-D we pre-train with language, utilizing a small LM, ChibiT [31] pre-trained on Wikitext-103 [26] with the same architecture we next discuss in Section V. When training a DT with ChibiT initialization, we use the modified objective:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{MSE}} + \lambda_1 \mathcal{L}_{\cos} + \lambda_2 \mathcal{L}_{\text{LM}} + \lambda_3 \mathcal{L}_2 \tag{5}$$

Where $\mathcal{L}_{\text{MSE}}$ is the original DT objective, $\mathcal{L}_{\text{LM}}$ is the language modeling objective, $\mathcal{L}_{\cos}$ encourages cosine similarity between DT input embeddings and clustered centers of language token embeddings, and $\mathcal{L}_2 = \|\theta_{\text{LM}} - \theta\|_2$.

IV. Related work

Several papers have studied merging models with shared initialization in language and vision for the purpose of improving performance, generalization, forming pre-trained models, or for extending task capabilities [5], [6], [12], [13], [15], [22], [36], increasing robustness to distribution shift [37], and for distributed training [7], [25]. Other work has improved methods for merging weights by utilizing Fisher information for weighted averaging [18], [24], and accounting for symmetries, such as permutation [1], [16], [27] which is used to effectively merge weights found through optimizing the same loss but with different initialization. To our knowledge, we are the first work that considers merging for decision-making or robotic control settings, which presents unique challenges. Many recent works have considered creating multi-task or general models for decision-making problems [8], [14], [19], [21], [30], but rely on simultaneous and centralized training over all tasks. While we focus on experiments framed as multi-task problems, merging, could be explored for continual (reinforcement) learning [17], which has long-standing goals of creating systems that can continuously adapt to new tasks.

V. Experiments

We begin by partially merging Decision Transformers to investigate if similar parameters may be learned across environments, ignoring input and output projection layers which have unique dimensionality and function for different

environments. We use settings for MuJoCo experiments from [4], [31], as well as their implementations. Following their configuration, we use transformers with an embedding size of 128, 1 head, and 3 layers. We randomly initialize and train a model for HalfCheetah, Hopper, and Walker2D on expert D4RL [10] datasets. Given two trained models on different environments, we call the model which is having a layer altered the target model with parameters θ_t, and the model where the layer is taken from, which is trained on another environment, the source model with parameters θ_s.

We first average a single layer at a time, keeping all other layers unaltered. For one layer $\theta^i \in \theta$, we use the update $\theta_t^i = (1-\alpha)\theta_t^i + \alpha(\theta_s^i)$, where $\alpha \in \{.5, 1\}$, relating to averaging and swapping parameters. We merge between each pair of models out of the three models covering each environment. For each pair, we look at merging in each direction, swapping source and target models. This leads to six evaluations per layer, which we average and display in Figure 2a. We report normalized return over 25 episodes, where 1 corresponds to the original return before any merging ($\alpha = 0$).

We find that we can both average ($\alpha = .5$), or directly use a layer from another transformer ($\alpha = 1$), showing that Decision Transformers trained on different MuJoCo tasks may learn functionally similar parameters. We see a larger drop-off from merging parameters within the multi-layer perception (MLP) layers. However, we find that we can directly use the attention parameters from another trained model at any depth with little to no decrease in empirical return.

A. Merging subsets of multiple layers

We should draw attention to that in Figure 2a we are merge one layer at a time, and possibly, a transformer could be robust to single-layer interference. We must see how well merging performs with multiple layers at a time, as errors could compound. We look at this from two perspectives. We begin by following a similar procedure to the former per-layer merging, but instead incrementally add an additional layer, forming a larger subset as we go move further in depth, in Figure 2b. Instead of averaging across different environment pairs, we show this within a single pair to be able to clearly see the effects of adding layers. We see that as we add layers, performance decreases, reaching close to zero performance when merging the entire transformer. We also tend to see large drop-offs at MLP layers, but less so at attention layers, and varying results with layer normalization.

1) Merging all attention layers: When merging individual layers, we see the least reduction in performance with attention layers. Thus, we see what happens when merging all attention parameters (the parameters associated with the query, key, value, and output projections) at once. Attention parameters consist of $\sim 33.18\%$ of the 3 transformer layers which have a total size of 596K parameters. We display results in Figure 3. On the x-axis, we interpolate between parameters where $\alpha = 0$ corresponds to using unchanged attention parameters and $\alpha = 1$ corresponds to using all attention parameters from the other environment. It appears

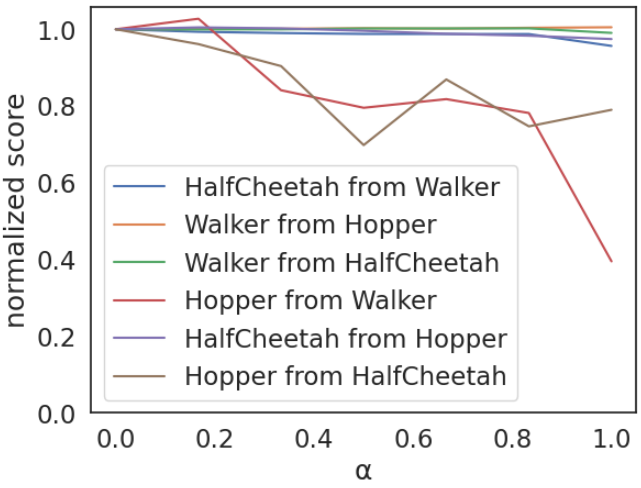


Fig. 3: **Merging all attention parameters** of transformer across pairs of MuJoCo environments. We vary α, which interpolates between original parameters ($\alpha = 0$) and to parameters from the second environment ($\alpha = 1$).

that Decision Transformers trained on different MuJoCo can learn functionally similar attention weights. We can directly swap in the weights of HalfCheetah into the Walker2D transformer, and vice versa, with no decrease in return. We do not see this hold as strongly between Hopper/Walker2d, and Hopper/HalfCheetah, but we still see good results.

B. Analysing attention merging

To explain the success of swapping attention parameters in the prior section, we perturb attention parameters after training to see how much randomly initialized DTs rely on the attention mechanism. We replace the attention weights of trained DTs using both random parameters (fixed for all experiments), identity parameters (weights set to 1, and biases equal to 0), and removing attention, having information pass through residual connections. We show the results of these variations in Table I. We see varied results depending on the environment and dataset pair. We see decreases over all datasets with Hopper. With Walker and HalfCheetah, while we see little impact on the homogeneous medium or expert datasets, we see performance decrease after perturbation for models trained on medium-expert.

C. Merge-Freeze-Finetune

In previous sections, even when merging over a compatible subset, such as attention layers, we still see some drop-off in return after merging (such as between Hopper and HalfCheetah), as shown in Figure 3. We aim to see if we can merge without losing performance and over parameters which previously saw large decreases in performance, such as MLP parameters as in Figure 2a. To overcome this, we test *Merge-Freeze-Finetune* on several subsets, reporting the performance of the merged model as a percent of the original performance. We also report the size increase of the multi-task model within transformer layers, which does not include linear input/output projections that are kept unique for each

TABLE I: We report the impact of **altering attention parameters of trained DTs**. We report D4RL normalized scores of original performance, replacing attention parameters with randomly initialized parameters, identity parameters, and removing attention, relying on residual connections.

Dataset	Env	Original	Random	Identity	Removed
Medium Expert	Cheetah	88.6	41.0	41.2	41.5
	Hopper	108.3	77.2	66.8	74.1
	Walker	109.4	84.6	82.6	86.9
Medium	Cheetah	40.4	40.3	39.7	40.8
	Hopper	76.4	52.2	52.6	52.2
	Walker	75.3	74.1	76.2	74.3
Expert	Cheetah	89.9	90.1	89.9	90.1
	Hopper	108.8	67.1	67.9	68.8
	Walker	109.4	109.7	109.8	109.8

task. For example, without any merging, we maintain original performance but need unique transformer layers for each task (200% parameters). We use subsets of all attention related parameters, MLP and attention parameters, and the entire transformer which also adds layer normalization layers. We also report attention merging without finetuning (M), as previously shown in Section V-A.1, and an alternative to equally merging, where we keep one model unmodified, but copy and freeze its parameters to the DT in the second environment, and have this model update its non-transformer parameters.

With this setup, we report results between merging Cheetah and Hopper models in Table II. When merging over attention layers, plus finetuning (MFF), we can recover the original performance. We can also retain greater compression with Attention+MLP merging, obtaining a multi-task model, with only 1.0026 times more transformer parameters, but with some reduced performance.

TABLE II: Evaluating performance after **merging between a pair of HalfCheetah and Hopper expert models** with *Merge-Freeze-Finetune* (MFF) and just merging (M), over different model subsets. Original performance (100%) in this experiment corresponds to D4RL normalized scores of 110.51 and 92.49 for Hopper and HalfCheetah.

Configuration	Hopper	Cheetah	Transformer Size
Original	100%	100%	200%
Frozen from Hopper	100%	3%	100%
Frozen from HalfCheetah	48%	100%	100%
Transformer (MFF)	61%	86%	100%
Attention+MLP (MFF)	90%	98%	100.26%
Attention (MFF)	101%	101%	166.82%
Attention (M)	79%	98%	166.82%

D. Merging with language pre-training

We may obtain better results if we merge two (or more) models which share a common initialization. Additionally, if models are encouraged to stay close to this original initialization, then we may also see better merging results.

We train using Equation 5, reporting several variants that all decay λ_1 to 0.0 after 5000 following [31], this includes using language co-training (ChibiT$_{\mathcal{CO}}$: $\lambda_2 = 1, \lambda_3 = 0$), using l_2 regularization to the initialization (ChibiT$_{\mathcal{R}}$: $\lambda_2 = 0, \lambda_3 = 1$), or no co-training or regularization (ChibiT: $\lambda_2 = 0, \lambda_3 = 0$).

We train models for HalfCheetah, Hopper, and Walker medium and medium-expert tasks and evaluate *Merge-Freeze-Finetune* (MFF), merging across all three models, instead of between pairs. We also merge over attention and MLP layers. We display results in Table III. In addition to language-initialized models, we report several baselines. This includes repeating the MFF process but with randomly initialized models as in previous sections (Random). We also train a multi-task baseline (Multi), which simultaneously trains a decision transformer on all three tasks. We also report the results of training individual DTs for each task (Single). First, we notice a performance gap between the multi-task and single-task models, as multi-task models are the same size as single-task models, but must learn all tasks. Our goal is to obtain similar performance to the multi-task model but without simultaneous, multi-task training. We find that this is successful on MuJoCo medium, with ChibiT and ChibiT$_{\mathcal{R}}$ exceeding the performance of the multi-task model, and coming close to single-task performance. Because merged models share attention and MLP parameters, we obtain a model with just 100.52% the transformer parameters of a single task model, for all three tasks, as each transformer block for each task only has unique layer-normalization layers, but shares all other parameters. On medium-expert, we see a larger gap between both our merged, and multi-task models to the single-task models. We continue by discussing two complementary methods for closing this gap on the medium-expert benchmark.

E. Fisher merging and larger pre-trained initializations

In this section, we jointly consider two sources of improvement. First, we consider repeating our procedure of ChibiT$_{\mathcal{R}}$ in Section V-D but with **larger models**, specifically GPT2 small, with 84M transformer parameters. Additionally, we consider **Fisher merging**, to replace directly averaging. We evaluate these two improvements on medium-expert, as shown in Table IV, reporting average results over two instances of MFF. We can see that combining both larger model initializations and Fisher merging results in a multi-task model that is achieved without any centralized training, sharing transformer attention and MLP parameters over all tasks, and attaining an average D4RL normalized score of 94.97, close to the score of training separate individual Decision Transformers (99.73). Additionally, while training with larger models results in similar initial performance, we see significantly smaller reduction in performance after merging.

F. Merging finetuned multi-task models

We also examine if we can have settings where we can improve models, through merging, but without additional

TABLE III: **Merging Decision Transformers trained with shared language model initialization** on D4RL medium and medium-expert datasets. We evaluate merging attention + MLP layers with *Merge-Freeze-Finetune* (MFF) over all three environments at once. We report mean D4RL normalized scores and standard error over two instances of MFF, and two multi-task models. Single-task results are reported from DT [4].

Dataset	Environment	ChibiT	ChibiT$_{\mathcal{CO}}$	ChibiT$_{\mathcal{R}}$	Random	Multi	Single
Medium	HalfCheetah	39.9 ± 0.3	39.9 ± 0.3	41.2 ± 0.2	33.8 ± 0.5	38.7 ± 0.4	42.6
	Hopper	59.4 ± 0.9	55.4 ± 0.4	56.8 ± 0.4	58.0 ± 0.8	61.1 ± 4.1	67.6
	Walker	81.3 ± 0.3	77.6 ± 0.9	79.2 ± 0.6	82.2 ± 0.0	75.4 ± 1.0	74.0
	Average	60.2	57.6	59.1	58.0	58.4	61.4
Medium Expert	HalfCheetah	47.4 ± 0.5	40.1 ± 1.0	39.3 ± 0.1	35.4 ± 1.0	36.4 ± 0.2	82.2
	Hopper	49.4 ± 3.1	41.0 ± 1.2	71.5 ± 3.3	47.7 ± 1.8	91.5 ± 1.7	109.1
	Walker	101.9 ± 0	94.9 ± 2.2	98.6 ± 5.5	83.6 ± 2.2	106.9 ± 1.0	107.9
	Average	66.2	58.6	69.8	55.6	78.2	99.7

TABLE IV: **Merging with increased model size and Fisher information** with *Merge-Freeze-Finetune* (MFF) over all three models at once on D4RL medium-expert, extending Table III. We list the score of the model after performing MFF, followed by the initial score after training for a specific task, before merging (MFF / Initial). We evaluate merging with both ChibiT$_{\mathcal{R}}$ and GPT2$_{\mathcal{R}}$ initializations.

Dataset	Environment	ChibiT$_{\mathcal{R}}$		GPT2$_{\mathcal{R}}$		Single
		Direct	Fisher	Direct	Fisher	
Medium Expert	HalfCheetah	39.3 / 89.5	42.6 / 89.5	84 / 89.9	87.2 / 89.9	82.2
	Hopper	71.5 / 106.9	77.2 / 106.9	84.1 / 106.6	88.3 / 106.6	109.1
	Walker	98.6 / 109.3	103.3 / 109.3	109 / 107.8	109.4 / 107.8	107.9
	Average	69.8 / 101.9	74.4 / 101.9	92.4 / 101.4	95.0 / 101.4	99.7

TABLE V: **Merging with finetuned multi-task models**. We display performance of initial multi-task model, and then after finetuning on expert data for each task separately. Next, we show merging the resulting models. We repeat initial training twice, and merging and evaluating performance twice, reporting the mean and standard error over four evaluations.

Model(s)	Hopper	Cheetah	Walker
Multi-Task Medium	72 ± 1.9	39.9 ± 0.1	74.9 ± 0.9
Finetuned Experts	108.3 ± 1.4	70.5 ± 0.3	108.8 ± 0.2
Direct Merge	60.2 ± 3.2	43.7 ± 0.9	70.4 ± 7.8
Fisher Merge	$\mathbf{84.3 \pm 2.8}$	$\mathbf{45.5 \pm 0.6}$	$\mathbf{99 \pm 1.2}$

finetuning after merging. We consider multi-task models, and as report a gap between small multi-task and single-task models in Section V-D, we also use a larger model with embedding dimension of 512, and 4 heads, and 3 layers for this experiment. We begin by training a multi-task model on the medium datasets. After training this model, we independently finetune the model on expert data for each task with initialization regularization as in Section V-D, resulting in three separate finetuned models. We then merge all transformer parameters of the resulting finetuned models, to obtain one multi-task model. We evaluate both regular merging, and Fisher merging in Table V. When directly averaging the weights of finetuned expert models, we see performance decrease on Walker and Hopper, and slightly improve from the medium model with HalfCheetah. However, by performing Fisher merging, we improve considerably on Hopper and Walker, with still a slight improvement on HalfCheetah. While a gap still exists between our merged multi-task model, and the maximum performance achieved by each separate finetuned model, it is promising to find that we can have settings where merging improves performance on all tasks, even without MFF, as used in previous sections.

VI. Discussion & Conclusion

In this work, we studied merging Decision Transformers trained on MuJoCo locomotion problems. We found that it is possible to merge randomly initialized models, leading us to analyze the role of attention in DTs, use merging to obtain or improve multi-task models, and investigate using additional finetuning, common initializations, and utilizing Fisher information.

While we can come close to performance of individually trained policies, further work could explore merging with reduced penalty. Additionally, we believe follow-ups of our work could see if our findings more generally hold beyond merging policies locomotion tasks. This includes other multi-task settings [21] with DTs, testing other kinds of pre-training [3], [20] besides language, or merging other kinds of models in the RL settings which incorporate pre-training and finetuning [19], [32], [38]. In general, we view a broader impact of merging as providing a possible direction for more accessibly creating multi-task DTs or generalist models.

REFERENCES

[1] Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries, 2022.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[3] Rogerio Bonatti, Sai Vemprala, Shuang Ma, Felipe Frujeri, Shuhang Chen, and Ashish Kapoor. Pact: Perception-action causal transformer for autoregressive robotics pre-training, 2022.

[4] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling, 2021.

[5] Leshem Choshen, Elad Venezian, Noam Slonim, and Yoav Katz. Fusing finetuned models for better pretraining, 2022.

[6] Alexandra Chronopoulou, Matthew E. Peters, Alexander Fraser, and Jesse Dodge. Adaptersoup: Weight averaging to improve generalization of pretrained language models, 2023.

[7] Shachar Don-Yehiya, Elad Venezian, Colin Raffel, Noam Slonim, Yoav Katz, and Leshem Choshen. Cold fusion: Collaborative descent for distributed multitask finetuning, 2022.

[8] Yilun Du, Mengjiao Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Joshua B. Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning universal policies via text-guided video generation, 2023.

[9] Rahim Entezari, Hanie Sedghi, Olga Saukh, and Behnam Neyshabur. The role of permutation invariance in linear mode connectivity of neural networks, 2021.

[10] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

[11] Hiroki Furuta, Yutaka Matsuo, and Shixiang Shane Gu. Generalized decision transformer for offline hindsight information matching, 2021.

[12] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic, 2022.

[13] Gabriel Ilharco, Mitchell Wortsman, Samir Yitzhak Gadre, Shuran Song, Hannaneh Hajishirzi, Simon Kornblith, Ali Farhadi, and Ludwig Schmidt. Patching open-vocabulary models by interpolating weights, 2022.

[14] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima-manipulation, 2022.

[15] Xisen Jin, Xiang Ren, Daniel Preotiuc-Pietro, and Pengxiang Cheng. Dataless knowledge fusion by merging weights of language models, 2022.

[16] Keller Jordan, Hanie Sedghi, Olga Saukh, Rahim Entezari, and Behnam Neyshabur. Repair: Renormalizing permuted activations for interpolation repair. *arXiv preprint arXiv:2211.08403*, 2022.

[17] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives, 2020.

[18] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[19] Aviral Kumar, Rishabh Agarwal, Xinyang Geng, George Tucker, and Sergey Levine. Offline q-learning on diverse multi-task data both scales and generalizes, 2022.

[20] Aviral Kumar, Anikait Singh, Frederik Ebert, Yanlai Yang, Chelsea Finn, and Sergey Levine. Pre-training for robots: Offline rl enables learning new tasks from a handful of trials, 2022.

[21] Kuang-Huei Lee, Ofir Nachum, Mengjiao Yang, Lisa Lee, Daniel Freeman, Winnie Xu, Sergio Guadarrama, Ian Fischer, Eric Jang, Henryk Michalewski, and Igor Mordatch. Multi-game decision transformers, 2022.

[22] Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A. Smith, and Luke Zettlemoyer. Branch-train-merge: Embarrassingly parallel training of expert language models, 2022.

[23] Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Pretrained transformers as universal computation engines, 2021.

[24] Michael Matena and Colin Raffel. Merging models with fisher-weighted averaging, 2021.

[25] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. 2016.

[26] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

[27] Fidel A. Guerrero Peña, Heitor Rapela Medeiros, Thomas Dubail, Masih Aminbeidokhti, Eric Granger, and Marco Pedersoli. Re-basin via implicit sinkhorn differentiation, 2022.

[28] Mary Phuong and Marcus Hutter. Formal algorithms for transformers. 2022.

[29] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[30] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. A generalist agent. 2022.

[31] Machel Reid, Yutaro Yamada, and Shixiang Shane Gu. Can wikipedia help offline reinforcement learning?, 2022.

[32] Adrien Ali Taiga, Rishabh Agarwal, Jesse Farebrother, Aaron Courville, and Marc G Bellemare. Investigating multi-task pretraining and generalization in reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023.

[33] Shiro Takagi. On the effect of pre-training for transformer in different modality on offline reinforcement learning. 2022.

[34] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[36] Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time, 2022.

[37] Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo-Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, and Ludwig Schmidt. Robust fine-tuning of zero-shot models, 2021.

[38] Yifan Xu, Nicklas Hansen, Zirui Wang, Yung-Chieh Chan, Hao Su, and Zhuowen Tu. On the feasibility of cross-task transfer with model-based reinforcement learning, 2022.