Université de Montréal

Morphology-Agnostic Reinforcement Learning with BERT

par
Parnika Parnika

Département d’informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l’obtention du grade de Maître ès sciences (M.Sc.)
en computer science

March, 2025

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé:

Morphology-Agnostic Reinforcement Learning with BERT

présenté par:

Parnika Parnika

a été évalué par un jury composé des personnes suivantes:

_ _,	président-rapporteur
Glen Berseth,	directeur de recherche
_ _,	membre du jury
_ _,	examinateur externe

Mémoire accepté le: ....

RÉSUMÉ

L'apprentissage par renforcement implique généralement l'apprentissage d'une politique pour une morphologie d'agent spécifique afin de résoudre une tâche. Cela nécessite un réapprentissage complet pour chaque nouvelle morphologie afin de résoudre la même tâche, ce qui ne permet pas de tirer parti des expériences d'apprentissage précédentes. Entraîner une politique unique sur plusieurs morphologies, performante lorsqu'elle est transférée à des morphologies non détectées, constitue un défi permanent pour la recherche. La difficulté est d'autant plus grande lorsque l'objectif est d'obtenir une politique performante sans ajustement supplémentaire, c'est-à-dire sans transfert de politique sur la morphologie non détectée. Des travaux antérieurs ont permis d'avancer significativement dans cette direction. Certains travaux considèrent une représentation graphique de la morphologie et l'apprennent avec l'objectif de politique d'apprentissage par renforcement. D'autres travaux ne considèrent pas la morphologie comme un graphe et apprennent sa représentation directement avec l'apprentissage par renforcement. Ces travaux ont obtenu de bons résultats sur les tâches de référence standard pour le contrôle indépendant de la morphologie, et nous souhaitons améliorer les capacités de généralisation dans ce travail. Nous exploitons la description textuelle d'une morphologie enregistrée dans son URDF et utilisons BERT pour obtenir des représentations pré-entraînées représentant cette morphologie. Ces représentations sont ensuite traitées par une politique de transformation entraînée avec l'algorithme RL TD3. Compte tenu des avancées significatives dans les capacités de généralisation des modèles fondamentaux, nous pensons que BERT peut fournir des représentations pré-entraînées de haute qualité de morphologies invisibles, permettant une meilleure généralisation. Nous évaluons notre méthode sur 8 tâches de référence, sur 3 ensembles d'entraînement et de test de morphologies différents, avec un ordre de complexité croissant. Les résultats montrent que notre méthode généralise bien ces tâches, comparable à la méthode de pointe Any-Morph, sans toutefois la surpasser.

mots clés: RL, URDF, morphology, morphology-agnostic, BERT

ABSTRACT

RL typically involves training a policy for a specific agent morphology to solve a task. This requires retraining from scratch for every new morphology to solve the same task. It's an ongoing research challenge to train a single policy across multiple morphologies that performs well when transferred to unseen ones. This is further challenging when the objective is to zero-shot policy transfer on the unseen morphology, that is, obtaining a policy that performs well without further fine-tuning. Prior work has taken valuable steps towards this goal. Some of the works consider a graphical representation of the morphology and learn it with the RL policy objective. Another work doesn't assume morphology as a graph and learns its representation directly with RL. These works performed well on the standard benchmark tasks for morphology-agnostic control, and we aim to advance the generalization capabilities in this work. We leverage the text description of a morphology enlisted in its URDF and leverage BERT to obtain pre-trained embeddings to represent the morphology. These are then processed by a transformer policy trained with RL algorithm, TD3. Considering the significant advances in the generalization capabilities of foundation models, our intuition is that BERT can provide high-quality pre-trained representations of unseen morphologies, enabling better generalization on them. We evaluate our method on the 8 benchmark tasks on 3 different training and test sets of morphologies with increasing order of complexity. The results show that our method generalizes well on these tasks comparable to the prior state-of-the-art, AnyMorph, however, it is unable to outperform it.

Keywords: RL, URDF, morphology, morphology-agnostic, BERT

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

LIST OF ABBREVIATIONS

RL	Reinforcement Learning
MTRL	Multi-Task Reinforcement Learning
BERT	Bidirectional Encoder Representations from Transformers
MDP	Markov Decision Process
GNN	Graph Neural Network
URDF	Unified Robot Description Format

(dedicace) Dedicated to my loved ones!

ACKNOWLEDGMENTS

I am extremely grateful to Prof. Glen Berseth for giving me the opportunity to obtain valuable research experience on an important research problem in robotics domain. I am also very grateful to his PhD student Adriana Knatchbull-Hugessen whose insights were valuable towards progressing on the project. Research is hard because coming up with the right set of experiments among multiple others and asking the right questions to make any progress is not a trivial task. The insights from my project supervisor and collaborator were extremely important for both my learning and progress on this project.

CHAPTER 1

INTRODUCTION

Deep reinforcement learning (RL) has enabled breakthroughs in complex tasks like mastering the game of Go, predicting 3D protein structures, and advancing robot learning. RL typically involves training a policy neural network from scratch for a specific agent morphology to maximize cumulative returns on a specific task. However, transferring a pre-trained policy to a different agent morphology is one of the ongoing research challenges in robotics domain. This is also referred to as **"morphology-agnostic RL"**. Using previous learning experience to train a different agent could improve learning efficiency and performance, and cut computational costs and time compared to training from scratch. **The objective of this work is to evaluate the utility of BERT for policy generalization across unseen agent morphologies during training.**

An agent morphology can be intuitively defined as a graph where vertices are the joints of limbs and edges are connections between these joints depending on whether two limbs are connected or not. The URDF files describe the geometry of the agent's morphology in a human readable text format. We extract this text information and leverage the vast learning experience on text of the foundation model BERT to encode this information. Our policy, similar to anymorph, is a transformer architecture that accepts a state vector from the environment and outputs the action vector. While training, we send these text encodings concatenated with the current state vector as input to the transformer architecture. In anymorph, authors propose to represent sensors and actuators as a sequence of tokens and learn their representation with the RL policy from scratch. We instead extract the names of these sensors and actuators and use BERT to obtain pre-trained encodings of these names. Our insight is that text is naturally used to describe any agent's morphology in its URDF. Leveraging the advanced generalization capability of BERT on text, we can obtain good quality learned representations for the names of sensors and actuators for any morphology unseen during training, enabling our model to generalize well on those morphologies. We evaluate the generalization performance of our method on multiple held-out morphologies and their mixtures.

Advances in morphology-agnostic RL can be significantly useful in the robotics domain. In recent years, the integration of foundation models with robotics has been evolving and these large models are being used within robotics for perception, decision making and control. These models are trained on large and diverse data and already enabled advanced generalization and learning capabilities in natural language processing and computer vision domains[7]. They are well-known for their zero-shot generalization capability, that is they can perform a task without training on data for that task. Hence, they have been leveraged in a plug and play manner or fine-tuned for robotics tasks. In our approach, we do not need to fine-tune the model; instead we plug and use the output of the BERT model to achieve our objective. The details of our approach are elaborated in the Method section. Our proposed method is able to learn with these pre-trained representations on mixtures of morphologies in the training set and can generalize well on unseen morphologies comparable to anymorph. However, our approach has not been able to outperform AnyMorph but it entails useful pointers towards future work. The details of experiments are demonstrated in the Experiments and Results section.

CHAPTER 2

PRELIMINARIES

2.1 Reinforcement Learning

In Reinforcement Learning, we work with Markov Decision Process(MDP). An MDP is a tuple $\langle \mathrm{S}, \mathrm{A}, \mathrm{R}, \mathrm{T}, \rho_0 \rangle$ where S is the set of all states, A is set of all actions, R is the reward function depending on state, action and the next state, $\mathrm{R}(s,a,s')$ where s,s' $\in$ S, T is the transition probability distribution over s'$\in$ S when an action a is taken on state s and is given by $\mathrm{T}(s'|s,a)$. The ρ_0 is the initial state distribution.

A policy is a probability distribution over actions given a state given by $\pi(a|s)$. The objective of an agent in RL is to learn a policy that maximizes the expectation over discounted cumulative returns.$\mathrm{J} = \sum_{t=0}^{\infty} \gamma^t r_t$, where $\gamma \in (0,1)$ is a discount factor, t is the discrete environment step and r_t is the reward at step t.

In the Multi-task Reinforcement Learning setting, the agent's goal is to maximize the average returns across N tasks. In this work, we use tasks and environments synonymously. The MDPs of 2 tasks are incompatible if the dimensionality of their states or actions is not the same. Let's denote MDP_1 has S_1 and MDP_2 has S_2 as the set of states where $s_1 \in \mathrm{R}^y \quad \forall s_1 \in S_1$ and $s_2 \in \mathrm{R}^{\hat{y}} \quad \forall s_2 \in S_2$. The MDP_1 and MDP_2 are incompatible as $dim(s_1) = y \neq dim(s_2) = \hat{y}$. In this work, our objective is to learn a single global policy across N incompatible MDPs. Due to this, we can't use MLPs and need a model architecture suitable for varying input sequence lengths. For this reason, we use Transformers. [9]

2.2 Transformers

Transformers[2] is a model architecture that avoids using recurrence unlike recurrent architectures used for NLP such as RNNs and LSTM. It relies on the attention mechanism to incorporate the global dependencies between the input and output sequences and allows parallelization making it more beneficial than the recurrent models. The attention mechanism is a building block in this model architecture where each element in the input sequence is linearly projected as **Query**,**Key** and **Value**.Each value **v** and key **k** have dimension as d_h. For each element in the sequence, a weighted sum over the values in the sequence is computed. For computing the attention weights, a dot product is computed between a "Query" **q** with all the "Keys" **k** in the sequence.

A $= softmax(\frac{qk^T}{\sqrt{d_h}})v$.

The multi-head attention is the extension of the attention mechanism where instead h different linear projections of the q,k and v are learned. All the head outputs are concatenated and again linearly projected. This allows the model to attend to the input sequence in h different ways and can be represented as $MHA(q,k,v) = concat(head_1,...head_h)W^o$ where $head_i = A(qW_i^q, kW_i^k, vW_i^v)$. If the input sequence $z \in \mathrm{R}^{n \times d_{model}}$ then $W_i^q \in \mathrm{R}^{d_h \times d_{model}}$, $W_i^k \in \mathrm{R}^{d_h \times d_{model}}$ and $W_i^v \in \mathrm{R}^{d_h \times d_{model}}$ and $W^o \in \mathrm{R}^{hd_h \times d_{model}}$. [2]

2.3 BERT

BERT, short for Bidirectional Encoder Representations from Transformers, is a variant of transformer architecture popularly used for downstream NLP tasks. In this work, we use this model to obtain contextualized word embeddings. The building block of this architecture is the stack of Transformer Encoder Layers and it can look at words bidirectionally in an input sequence, allowing to incorporate bidirectional context when encoding. However, this leads to the issue where the words can look at themselves. To avoid this, 15% of the input words are masked and the model is trained on the task of predicting the masked words. It is trained on large datasets of wikipedia (2.5B words) and Book Corpus(800M words) on the masked language modeling task.

The input to the model is the sequence of word tokens appended with a special token at the beginning of the sequence, termed 'CLS'. For the BERT-base which contains a stack of 12 encoder layers, each position in the sequence outputs a vector of dimension 768. In our work, we use the output of the first position which is a 'CLS' token embedding. This embedding represents the entire context of every input sequence. It is also used for sentence classification tasks. [6] [1]

CHAPTER 3

LITERATURE REVIEW

The morphology of an agent can be seen as a labelled graph where nodes are labelled with observable feature information from the sensors of the corresponding limb e.g. rotational velocities, translational velocities, limb type, etc. and an edge represents connection between two nodes if their corresponding limbs are connected.[9] The URDF is an xml file and commonly used to represent a morphology. In this work, we focus on mujoco environments namely, half cheetah, hopper, walker and humanoid where the mujoco engine loads the URDF of these morphologies to create corresponding reinforcement learning environment. Multi-task reinforcement learning(MTRL) can leverage the common information across tasks(morphologies in our case)[9]. It's objective is to learn a single global policy across environments of multiple morphologies for improved sample efficiency, learning performance and generalization. The work in this report is an instance of multi-task RL. We deal with incompatible environments where the state and action spaces are incompatible across tasks. This is because each agent morphology consists of varying numbers of limbs and the number of sensors and actuators per limb might also vary leading to different state and action spaces. Hence, MTRL can't be done with functional approximators. In our work, for an agent morphology each limb has more than 1 sensor and one actuator. However, for more complicated agent morphologies there can be more than 1 actuator per limb leading to higher precision and control.

3.1 Shared Modular Policies

Graph Neural Networks(GNNs) can process graphs of arbitrary size, thereby allowing MTRL on incompatible environments. This work leverages GNNs to encode morphological information. They take inspiration from the nature of modularity and reuse in sensorimotor systems[10]. Instead of creating a centralized neural network model that outputs control values for all the agent's actuators, they introduce "Shared Modular Policies", where a single reusable module is instantiated per actuator and gets observation only from the actuator's local sensors. For coordination between the modules, learned message vectors are passed to the neighbouring modules. For an agent, each module's policy outputs the torques for each limb individually. The module is said to be shared because the parameters are shared across all limbs for all agents. Each limb's policy module is conditioned on the local state of the limb as well as the message vector. This message vector is a 32-dimensional learned vector generated by the neighbouring limb's policy alongside the control value of the neighbouring limb's actuator. The message vectors from all neighbouring limbs are aggregated before being passed to the shared policy module. They use TD3 RL algorithm to train the policy to optimize the joint reward function across all limbs for the morphology. The authors propose three message passing approaches, i.e., top-down, bottom-up and both-way. The both-way message passing enables back and forth communication between the limb's policy modules and both learn as well as generalize the best among all message-passing approaches as demonstrated in their experiments.[10]

3.2 Amorpheus

SMPs use GNNs to incorporate the structural information of a morphology. This work[9] points out that we don't need this information encoded by a graph for improving learning performance. It is motivated by the fact that any benefit that GNNs provide is out-weighted by the difficulty it creates through message passing. This is especially important for sparsely connected graphs where information loss can occur when the information is communicated across multiple hops due to finite size of MLPs[8]. Instead, the authors use transformer encoders in this work which obviates the need to do message propagation far away into the graph.[9] Even though they don't leverage graph neural networks for training the policy, they still consider each limb as a node of a morphological graph and encode each node independently through a linear encoder and then send the encoded information to the Transformer which is decoded by a MLP layer to output action per limb. The critic is also the same architecture as the policy. They also use the TD3 RL algorithm for training their policy. In their work, they demonstrate the effectiveness of this approach over SMPs.

3.3 AnyMorph

Previous works use graphs to represent an agent's morphology. This work[3] points out the limitations of representing morphology as a graph. Such morphologies require us to know the sensors and actuators for each limb of an agent; however the authors point out that the number of limbs is independent of the dimensionality of state and action space. The authors continue to categorize the limitations into two parts. The first limitation is referred to as the "graph structure assumption" where the graph definition of a morphology can be used to denote agents with rigid limbs only and not for further complicated physical structures of agents. The second limitation is referred to as "alignment assumption". This assumes the sensors and actuators can be aligned/categorized according to limb but the authors argue that some RL agents might not have been well-defined limbs. Due to these limitations, the authors proposed an alternative definition to represent an agent morphology. Instead of defining morphology in terms of sensors and actuators per limb and predicting control values per limb, they represent morphology directly in terms of all the sensors and actuators that comprise the agent bypassing any dependence on limbs. The authors propose encoding these sensors and actuators as a sequence of tokens and learn their corresponding embeddings jointly with the policy from the RL objective. The policy is a variant of Transformer[2](does not use causal attention masking for the decoder) and returns action values for all the agent's actuators. The TD3 algorithm is used to train the policy with the objective to maximize the returns across all morphologies in the training set. The policy model is trained across multiple agent morphologies for better generalization on morphologies in the test set. Through experiments on 8 benchmark RL tasks, the authors show their method outperforms both SMP and Amorpheus on both learning and generalization performance.

AnyMorph relies on learning the tokens for in-distribution morphologies. This can't transfer well to a completely out of distribution morphology because of lack of learned embeddings for sensors and actuators per limb for that unseen morphology. In this work, we take a step towards better policy transfer performance to a completely out of distribution morphology. During generalization evaluation, instead of randomly initializing the sensor and actuator embeddings for the unseen morphology we use BERT to encode the sensor and actuator names and obtain pre-trained embeddings instead.

CHAPTER 4

METHOD

We call our method **MA-BERT**, referring to **Morphology-Agnostic with BERT**. For all limbs given a morphology, instead of learning the embeddings of their sensors and actuators from scratch like AnyMorph, we leverage BERT for obtaining their pre-trained representations. We use the transformer architecture for our policy as it can operate on varying states and action spaces, enabling us to train a single global policy on incompatible environments. We train the transformer policy on multiple morphologies. The mujoco environments for these morphologies are, namely, Cheetahs, Walkers, Humanoids and Hoppers. We also train our method on the mixture of these morphologies.

4.1 Encoding limb sensors

In our method, we leverage the mujoco environment of a morphology to get the names of all its sensors and actuators of all limbs. The sensor names are e.g., position, translational velocity, rotational velocity and angle among others. We use BERT to encode these names and obtain corresponding pre-trained text encodings. In figure 4.1, we show a morphology having 'M' number of limbs while position, angular velocity etc are the sensors per limb and there are 'n' number of sensors per limb. Consider a limb having the name foot and one of the limb sensors named as position. The text "foot position" is sent as input to the BERT model to obtain it's pre-trained embedding. Similarly, the texts of all limb sensors are sent to the BERT model to obtain their respective embeddings. We extract the 'CLS' token embedding from BERT. The names of limbs for all morphologies in our experiments and their sensors and actuators are detailed in the Appendix section.III.I Since we have M and n as the number of limbs and sensors per limb respectively, the total number of observation token embeddings will therefore be Mxn or simply 'N'. Since we use the BERT base model, the dimension of the output embedding vector is 768. Therefore, our pre-trained observation embedding matrix is of

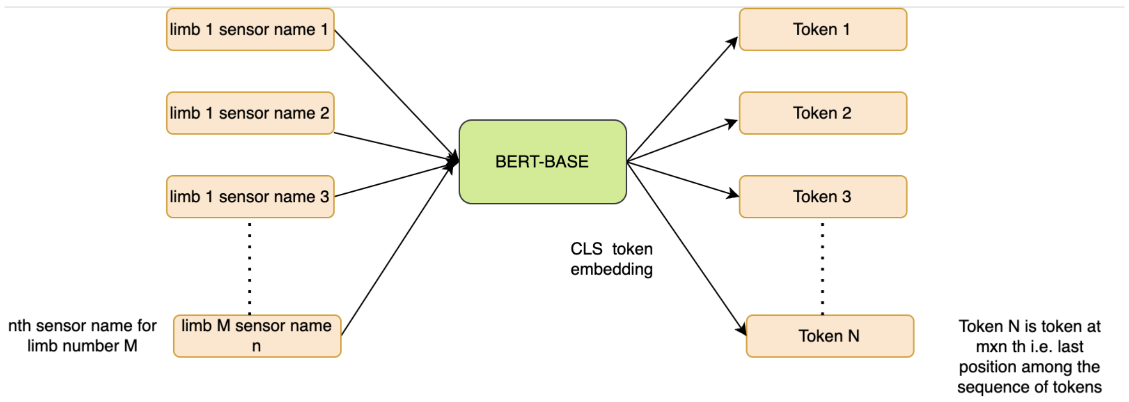

Figure 4.1 – Visualization of the process to obtain pre-trained observation embeddings where each token refer to a single sensor or actuator

size Nx768. Similarly, we derive the action embedding matrix for the actuator per limb. In our work, all the morphologies have one actuator per limb. Therefore, our pre-trained action token embedding matrix is of size Mx768. The transformer maps the input sequence of 'N' elements to the output sequence of 'M' elements. The major difference between our approach and anymorph is in the way we embed observation and actions.

4.2 Actor

Let the observation embeddings matrix be given by obs_N^{emb} and the observation vector at time step t be given by s_t. Similar to anymorph, we map each value in the observation vector into a Z- dimensional vector with the frequency encoding given by equation 4.1. Each value of the observation at any time step t is passed through a series of sinusoidal embedding functions with increasing frequencies $k1, k2, ...k_{M/2}$ from 1/10 to 1000. This is found to work well in previous work[3] and can also be seen as the positional encoding function from this work [2].

$$X = [obs_N^{emb}; cos(k_1 o_t); sin(k_1 o_t); cos(k_2 o_t); sin(k_2 o_t); ...] \tag{4.1}$$

In figure 4.2, we show that the current observation at timestep t is concatenated with the observation token embeddings obs_N^{emb}. The concatenation operator is denoted

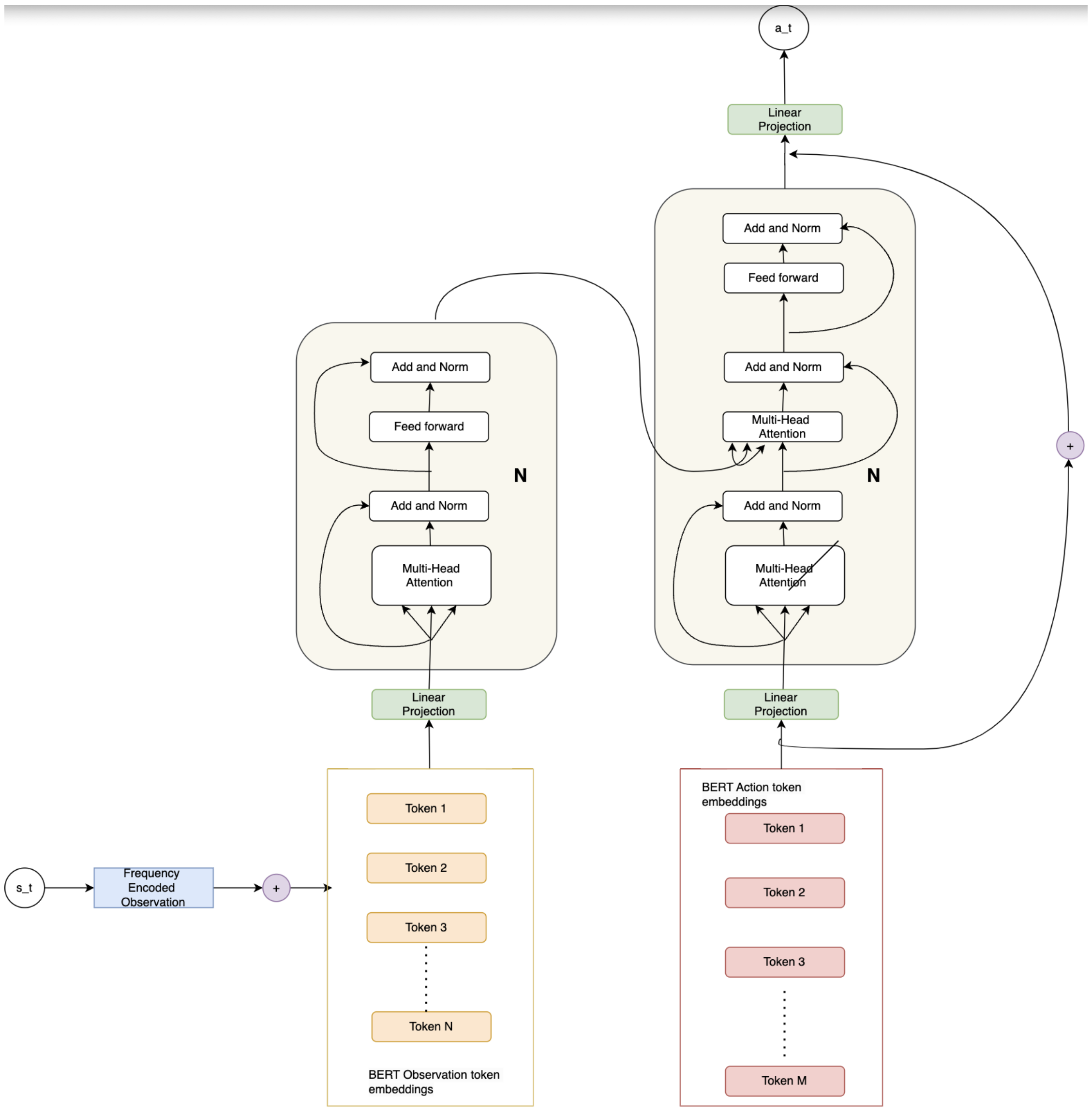

Figure 4.2 – Policy architecture; The observation at current time step s_t is encoded and concatenated with BERT Observation Token embeddings. This is input to the Transformer model with BERT Action Token embeddings. The model outputs M number of actions.

by purple circle in the figure. The observation sequence of tokens and action token embeddings are linearly projected to match the dimension of the Transformer model hidden state, d_{model}[2]. The input to the Transformer encoder is the linearly projected observation tokens while input to the Transformer decoder is the linearly projected action token embeddings and the output of the transformer encoder. The weights of these linear projections W_{proj}^{obs} and W_{proj}^{act} are given by equation 4.2.

$$W_{proj}^{obs} \in \mathbb{R}^{(768+Z)\times d_{model}}, W_{proj}^{act} \in \mathbb{R}^{768\times d_{model}} \tag{4.2}$$

,where 768 is the BERT 'CLS' token embedding dimension and Z is the output dimension of the frequency encoding function.

The decoder outputs M vectors and we learn another linear transformation W_{proj}^{out} to map these vectors to scalars 4.3, thereby returning M number of pre-tanh actions. This is equivalent to the number of limbs of the morphology. For the decoder, we don't need to use auto-regressive masking for our task.

$$output_t = transformer(XW_{proj}^{obs}, act_M^{emb} W_{proj}^{act}) W_{proj}^{out} \tag{4.3}$$

where act_M^{emb} are the action token embeddings where M is the number of tokens equivalent to the number of limbs in our case. Finally, the action at time step t is given by: $a_t = tanh(output_t)$

4.3 Critic

To optimize the transformer policy, we need the feedback from the critic network. The critic gives its feedback through Q values which represent the long term discounted cumulative rewards for taking an action given a state of an RL agent. We keep the critic architecture similar to anymorph. It is a Transformer encoder which takes as input the state and the action taken by the policy at that state. It returns the Q values for the state-action pair. The dimension of the Q values vector is equal to the dimension of the action vector.

4.4 Training process

Following prior work [10] [3], we train the transformer policy with the TD3 RL algorithm and use adam optimizer. Initially we used 0.126 as the exploration hyperparameter which we changed to 0.2 for reasons detailed in the Experiments section. The full list of hyperparameters is in the Appendix section.II.II We train the actor on every morphology in the training set for 3 million timesteps and then evaluate the learning and generalization performance of morphologies in the test set. All morphologies have a different number and type of limbs. We alternate between collecting data for morphologies in the training set and updating parameters of the transformer model on that data iteratively across all morphologies. Each morphology has a separate replay buffer for data collection. Our training methodology is aligned to previous works[10] [3].

CHAPTER 5

EXPERIMENTS AND RESULTS

To evaluate the effectiveness of pre-trained BERT representations for morphology-agnostic RL, we use the benchmark introduced in [10]. The benchmark consists of 8 tasks with the objective to maximize the cumulative discounted returns across N agents having different morphologies. These agents are from OpenAI Gym RL environments, specifically mujoco environments and are considered more complex to be solved by a policy.[5]. The tasks are on 4 types of agents namely: HalfCheetah-v2, Walker2d-v2, Hopper-v2, and Humanoid-v2. We used the same set of mujoco environment versions as used by prior work[3]. For the Cheetahs morphology type, there are 15 types of morphologies. There is 1 full cheetah morphology i.e. the morphology with all limbs while the remaining 14 morphologies are created by removing some of the limbs. Similarly, Walkers, Humanoids and Hoppers have 6, 8 and 3 numbers of morphologies in total respectively. Learning a single global policy is challenging because not only it should learn to control a morphology as sophisticated as humanoids but also it should be able to control multiple morphologies with varying types of limbs. This is further challenging when we mix different types of agents and learn a global policy to control them all. Therefore, learning to control Cheetah-Walker-Humanoids-Hopper i.e. totalling to the 32 number of morphologies with a single policy is the most challenging. We compare our method MA-BERT with AnyMorph, the current state-of-the-art on these benchmark tasks. There are a total of 4 experiments on a single type of morphology namely Cheetahs, Walkers, Humanoids and Hoppers and remaining 4 experiments on the mixture of morphologies namely, Cheetah-Walker-humanoids, Walker-Humanoids, Walker-Humanoids-Hopper and Cheetah-Walker-Humanoids-Hopper. Furthermore, there are training and test sets of morphologies for each of these experiments where we train our method and anymorph on the morphologies present in the training set while we evaluate the zero-shot generalization performance on the separate held-out morphologies in the test set.

5.1 Train-Test Splits

In our experiments, we use three different types of training and test sets.I.II.III.III One used in prior work[10][3], while we create the other two. The former one includes 1 full morphology during training for every morphology type. This means that the policy has seen all the limbs during training. For the second train-test split we remove the full morphology from the training set. Then pick a specific limb and remove morphologies containing that limb from the training set and transfer those containing it into the test set. For the third train-test split, we pick more limbs and move morphologies containing those limbs from the training set to the test set. The intuition behind this is that the importance of our method will be more profound when we zero-shot transfer our policy to unseen limbs in the training set. This is because we can derive quality pre-trained embeddings for those unseen limbs through BERT while AnyMorph does not have any learned embedding associated with that unseen limb. Detailed information about these two train-test splits is given in the Appendix section.

5.2 Learning Performance on Train-Test Split 1

In figure 5.1, we observe the learning performance of our method and anymorph however we consider two variants of each method; one including limb type vector and another not including this vector in the state representation. AnyMorph, originally, includes this information in its state representation. In fig 4.1 above, we observe what constitutes the state of the RL agent. In our approach, each limb sensor is represented by its token embedding which is a pre-trained BERT embedding with input being the name of the limb sensor. For example, the input foot position where limb name is foot and sensor name is position. For both the “MA-BERT” and “MA-BERT-no-ltv” variants of our method, we use the same mujoco RL environments for these morphologies as used in prior work[10][3] except for one change in the state representation. For “MA-BERT”, we include a 4 dimensional vector representing the limb type while for “MA-BERT-no-ltv” we don’t include this information. We also include one experiment where we remove this information from the state for AnyMorph as well which we denote as

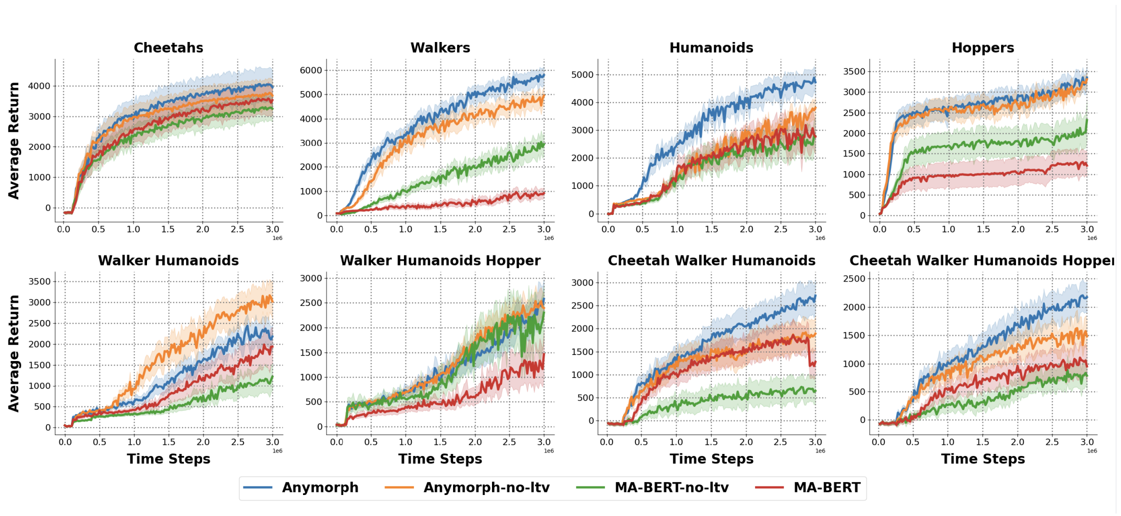

Figure 5.1 – Learning Performance of anymorph and our method including limb type vector and excluding it from the state, on 8 tasks across 3 random seeds. The y-axis is the Average Return per episode per training morphology and seed while the x-axis is the total steps across all training morphologies, i.e., 3-million timesteps. Dark coloured lines refer to the average and the shading around each line is a confidence interval of 95%.

"AnyMorph-no-ltv".

In figure 5.1, in most of the environments, we observe that "AnyMorph" and "MA-BERT" are performing better than "AnyMorph-no-ltv" and "MA-BERT-no-ltv" respectively. **This shows that providing more information to the agent about the state of it's environment can lead to better learning enabling higher cumulative returns.** However, overall we observe that either of the variants of our method are not learning as well as anymorph on this train-test split.

5.3 Generalization Performance on Train-Test Split 1

In figure 5.2, we observe the generalization performance of our method and anymorph on 3 seeds and 7 tasks as we don't have a test set for the Hoppers morphology. We observe that generalization performance of both the variants of anymorph is better than ours on this split used in prior work. However, Our method's performance is better when the limb type vector is included in the state representation than when it is not.

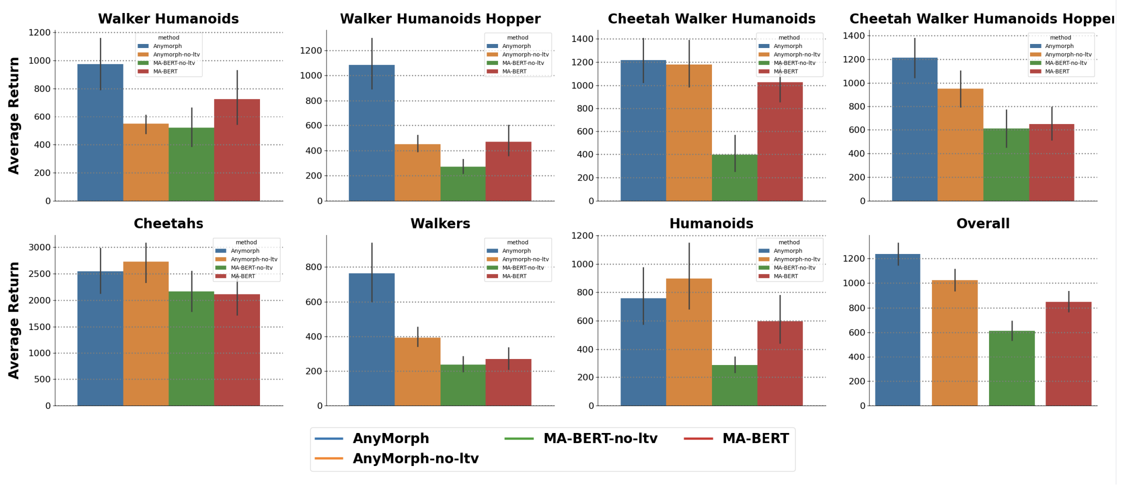

Figure 5.2 – Generalization Performance of anymorph and our method with limb type vector and without it in the state representation, on 7 tasks. The evaluation is done on morphologies in the test set. The y-axis is the Average Return per 10 episodes per test morphology and seed. Dark coloured bars refer to the average return evaluated at 3 million timesteps and the error bars show a confidence interval of 95%.

Similar observation is seen for anymorph. Therefore, for the next set of experiments we include this information in the state representation and compare our method MA-BERT with AnyMorph. One issue our method has faced during training leading to low learning and generalization performance is the following. We usually train on at least 3 random seeds due to the stochasticity inherent in RL. While anymorph does not face any issue during training and successfully learns across all seeds our method could not learn on some specific seeds. The reason behind this might be that our RL agent is facing a hard time to get out of bad states on those seeds. To alleviate this, **we increase the exploration hyperparameter to from 0.126 to 0.2** for TD3 so that the agent explores better and prevents itself from encountering bad states.

5.4 Learning Performance on New Train-Test Splits

For these train-test splits, each morphology in the training set except one does not contain all the limbs of that morphology. We don't do this for hopper morphology as it has only 3 numbers of different sub-morphologies. Furthermore, for the second training set and for every type of morphology except hoppers, we pick a specific limb and remove all morphologies containing that limb into the test set. For the third training set, for all morphologies except hoppers and humanoids, we pick more than one limb and remove all morphologies containing those limbs into the test set. For Humanoids morphology, only one limb could be left unseen in the training set.

While training on both second and third train-test splits, we increase the exploration hyperparameter to 0.2. For training on the third set, we added one extra hidden layer before sending the observation token embeddings to the transformer to evaluate if learning performance increases. We observe in figures 5.3 5.4 that the learning performance of our method on both the training sets across all tasks is either similar to anymorph's performance or lower than it.

5.5 Generalization Performance on New Train-Test Splits

It is expected to be harder for our method and anymorph to generalize on held-out morphologies from the second set because of one limb unseen in the training set per morphology. However, it is the hardest to generalize on the test morphologies from the third set with multiple limbs unseen in the training set per morphology. We expect our method to generalize well on both these sets because we are able to obtain quality embeddings for these unseen limbs leveraging BERT. While anymorph has no learned embeddings for these unseen limbs hence we initialize these embeddings with zeros. We make the following two observations from figures 5.5 5.6.

1. In figure 5.5, we observe that our intuition is correct for most of the morphologies and overall our method generalizes somewhat better than anymorph. Our method performs much better than anymorph for Humanoids morphology. Our method's performance should be much better than anymorph on the third train-test set but this is not the case as observed in figure 5.6. The reason behind this might be, **To generalize zero-shot to multiple unseen limbs, we need not only good quality embeddings for those unseen limbs but also might need to fine-tune the policy on them.**

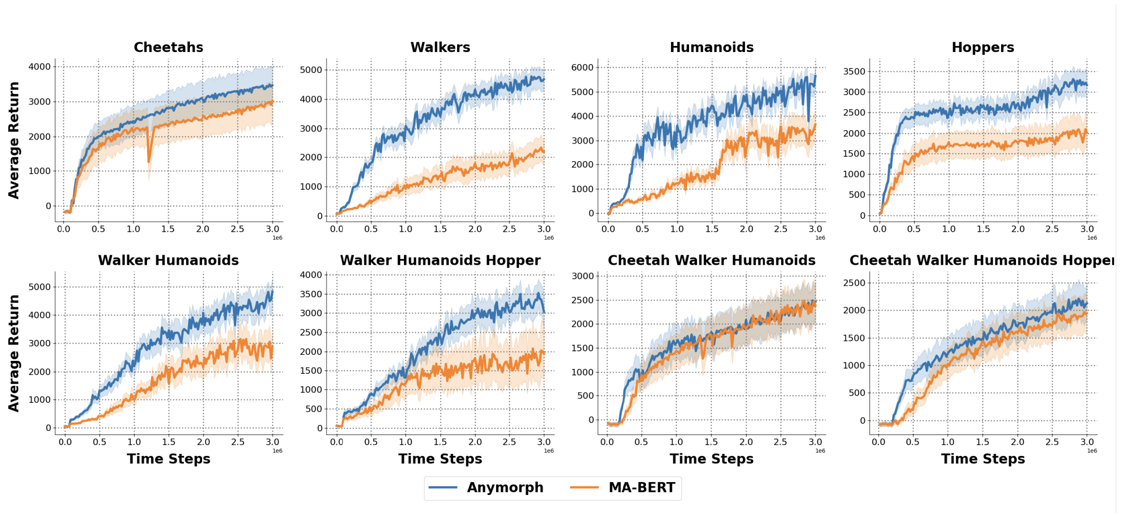

Figure 5.3 – Learning Performance of anymorph and our method on 8 tasks across 3 random seeds. Second train-test split is used with one limb unseen in the training set. The y-axis is the Average Return per episode per training morphology and seed while the x-axis is the total steps across all training morphologies, i.e., 3-million timesteps.

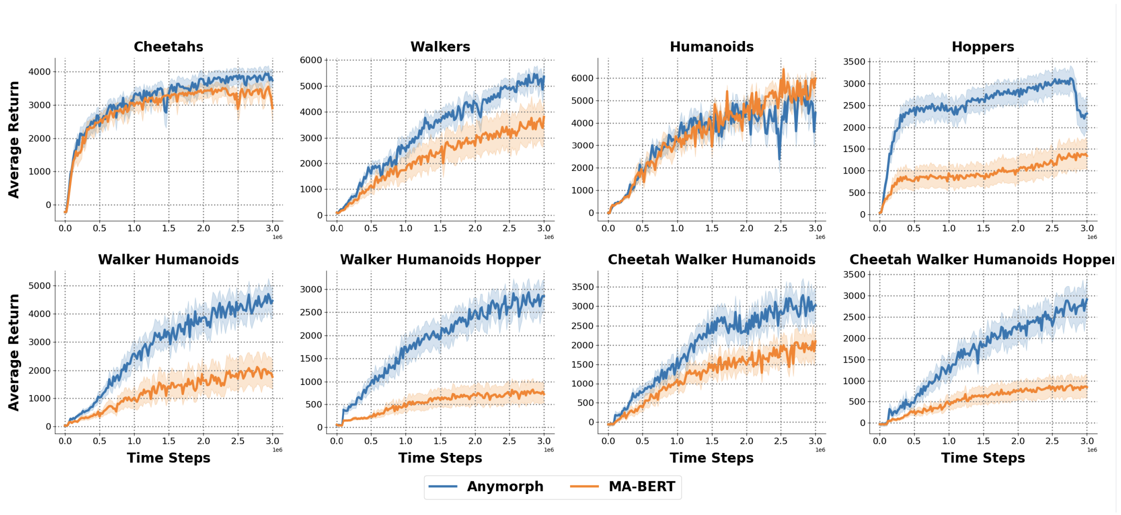

Figure 5.4 – Learning Performance of anymorph and our method on 8 tasks across 3 random seeds. Third train-test split is used with more limbs unseen in the training set. The y-axis is the Average Return per episode per training morphology and seed while the x-axis is the total steps across all training morphologies, i.e., 3-million timesteps.

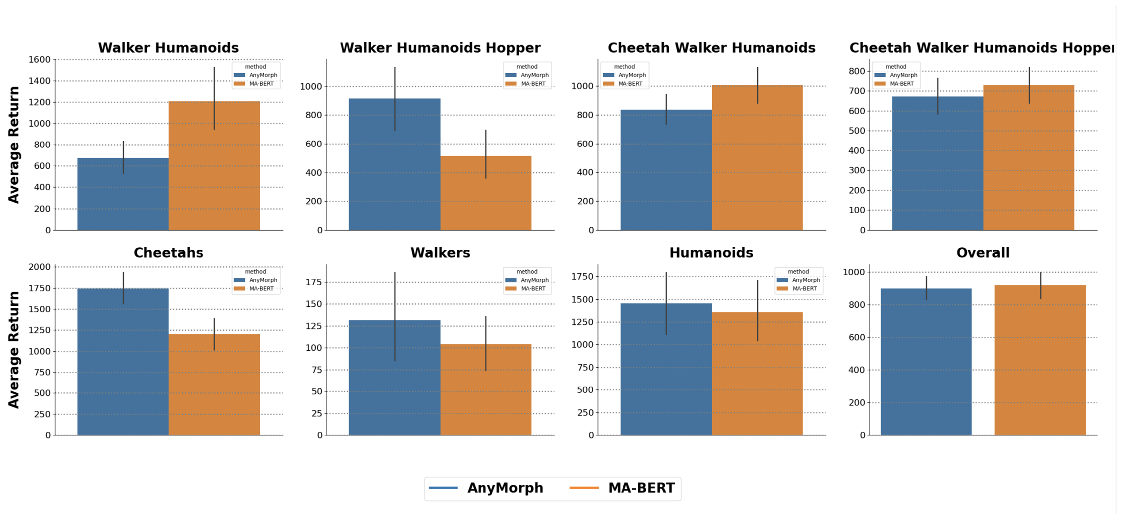

Figure 5.5 – Generalization Performance of anymorph and our method on 7 tasks. The evaluation is done on morphologies in the test set 2 with one unseen limb during training. The y-axis is the Average Return per 10 episodes per test morphology and seed. Dark coloured bars refer to the average return evaluated at 3 million timesteps and the error bars show a 95% confidence interval.

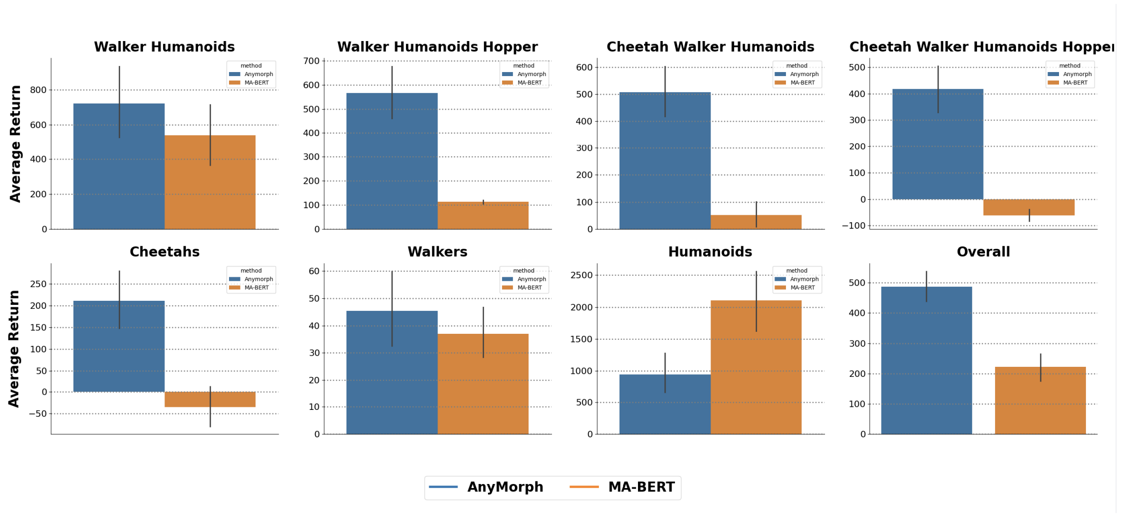

Figure 5.6 – Generalization Performance of anymorph and our method on 7 tasks. The evaluation is done on morphologies in the test set 3 with more unseen limbs during training. The y-axis is the Average Return per 10 episodes per test morphology and seed. Dark coloured bars refer to the average return evaluated at 3 million timesteps and the error bars show a 95% confidence interval.

2. Another observation is that the generalization performance of both our method and anymorph reduces when there are more unseen limbs in the training set. This is expected because it's harder to generalize in this case. Finally, our method is susceptible to seeds. Though increasing the exploration hyperparameter to 0.2 helped increase the learning and generalization performance on most tasks, however our method is still susceptible and performs much better on some seeds than others while this variance across seeds is much lower in anymorph.

It is to note that in our approach, the observation embeddings are derived from the limb and sensor names. These differ across different morphology types because for the same sensor, the limb names are different across morphologies. This means the embeddings won't be shared during training across different morphology types. However, this is not the case with anymorph where token embeddings are both shared and trained across same type or different type of morphologies during training. This might also be the reason why anymorph's generalization performance is better on the third training set.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this work, we investigated the application of BERT for learning transferrable policies to morphologies unseen during training. Our investigation was done on three different training and test set of morphologies with increasing order of complexity. For the training and test sets where all the limbs of a morphology were not seen during training, our approach enabled us to obtain pre-trained representations for the sensors and actuators of the missing limbs unlike AnyMorph which does not learn any representation for unseen limbs. Our approach leverages the text description of a morphology in its URDF to obtain the sensor and actuator names of all limbs of a morphology and encoded them with BERT for learning a Transformer-based policy. From the results on the second train-test set with one missing limb during training, we observe our method MA-BERT's generalization performance outperforms AnyMorph on most of the tasks. However, for the third train-test set with more missing limbs during training, our method generalizes less than AnyMorph. Our method also generalizes lesser than AnyMorph on the original train-test split as well with no missing limb during training but even more so on the third set.

The reason behind this might be two-fold. First, in Anymorph, we have same embedding representation for shared limbs across morphologies. That is limbs that have similar limb types across morphologies. However, our pretrained embeddings depend on the limb names which are different for every morphology. No embeddings are shared across similar limb types in our case. Secondly, Anymorph's learned embeddings might be better than BERT pretrained embeddings as they are learned with the RL policy objective. In our approach we don't fine-tune BERT embeddings for the limbs sensors and actuators. Thirdly, our method was unable to learn well on specific seeds. This suggests future work in the following directions:

1. It might help to obtain same BERT embeddings for shared limbs across morphologies. That is, giving the same name to similar limb types e.g. foot, shoulder, thigh etc.

irrespectively of the morphology. This will lead to shared embedding if the limb type is similar across morphologies, enabling better inference of unseen agent's morphology leading to better generalization.

2. We can also fine-tune BERT's embeddings with the RL policy objective. Then during generalization evaluation, we load these fine-tuned embeddings for limbs common in both training and test sets across morphologies and obtain pre-trained BERT embeddings for limbs unseen during training.

3. Our method faced the issue of encountering bad states and was unable to explore to rewarding states. Increasing the exploration hyperparameter helped but not much. However, increasing the exploration hyperparameter and resetting parameters for one or more of the transformer policy's last layer can help mitigate this issue considering the primacy bias in RL[4]. It might be a possibility that our policy is overfitting to early encountered bad states. Resetting some of model parameters may act as a regularizer and can therefore lead to better learning performance.

CHAPTER 7

KNOWLEDGE ACQUIRED

During this research internship, I encountered the following challenges: coming up with the right input and architecture choice for our task, challenges with learning performance due to stochasticity in RL and when our approach finally started learning, there was another challenge to improve our model's learning and generalization performance over AnyMorph's.

My experience working with collaborators led to insights on how to approach a problem with different angles if the solution doesn't work out. This is further elaborated below. Finally, I also gained experience on Singularity containers, writing slurm job scripts for running on the cluster and wandb for experiment logging. **Following elaborates my journey through the hurdles, their solution and my learning on the way**.

7.1 Input and Architecture Choices

1. Policy and Critic Architecture: **MLP**, Input to BERT: Combination of floating point values and text as **"slider position is 4.9314 hinge position is 0.2099 slider velocity is -0.3086 . . . "**. Slider and hinge are the limb names for the Inverted Pendulum RL environment while position and velocity are the senor names. Output of MLP: action values Initially, for the observation input to the model. We first extracted the sensor names and created a new text with the names alongside their floating point values received from the RL environment. We started testing our approach with a simpler RL environment i.e. Inverted Pendulum. Therefore the new text created was like,"slider position is 4.9314 hinge position is 0.2099 slider velocity is -0.3086". This text input was sent to BERT for obtaining the embedding. The 768-dimensional embedding was sent to the MLP for generating an output. We input the concatenated state-action values to the critic MLP as is usually done. The RL algorithm was **SAC**. This **did not work**.

2. Policy and Critic Architecture: **Transformer Decoder**, Input to BERT: Combi-

nation of floating point values and text as **“slider position is 4.9314 hinge position is 0.2099 slider velocity is -0.3086 . . . .”**. Output: action values Here we use the same input as above but instead of using MLP for the policy, we use Transformer Decoder. We obtain the pre-trained observation embeddings from BERT and train action embeddings from scratch with the RL policy objective. This is input to the transformer decoder. The RL algorithm was **SAC**. This also **did not lead to any learning on the Inverted Pendulum Environment**.

3. Policy and Critic Architecture: **MLP**, Input to BERT: Only floating point values as **“ 4.9314 0.2099 -0.3086 . . . .”**. Output: action values The RL algorithm was **SAC**. This led to some learning improvement on the Inverted Pendulum Environment but not considerable.

4. Our approach shown in figures 4.1 and 4.2. Policy Architecture: **Transformer**, Critic Architecture: **Transformer Encoder**, example BERT input:**[“ffoot position”,.... “bthigh velocity”, ...]**, Input to sinusoidal function:**[4.9314,.... 0.2099,... -0.3086,....]**, Output: action values. The RL algorithm used here is **TD3**. This led to **considerably well learning on the HalfCheetah environment** which is more complex than Inverted Pendulum. We then tested learning the global policy on 3 cheetah morphologies which also worked. However, increasing the cheetah morphologies to 12 did not lead to learning(average returns < 1000).

Overall, here I learned that BERT is not good for floating point values representation. It does not tokenize floating point values well and thereby unable to obtain good quality pre-trained representations. This situation is worsened if you give the input of combined text and floating point values. But BERT is good for text representations so dividing the problem into BERT for text representations while sinusoidal functions to convert a floating point-value into a d-dimensional vector turns out to be the right input design choice in our case. This is analogous to the divide and conquer strategy to solve a problem. We then combine them via concat or add operators. Furthermore, using a transformer is indeed the right choice for our approach needs to handle varying input sequence length.

7.2 Hyperparameter Tuning

Tuning the transformer hyperparameters led to learning on all 12 cheetah sub-morphologies. The average returns were greater than 1000. However, our performance on Cheetahs was still lower than AnyMorph.

7.3 Increasing the number of sensors and actuators in State representation

For the state representation of the RL environment for any agent morphology, we were using only 2 sensors which tracked the ‘position’ and ‘velocity’ of all the limbs of the agent. Increasing this to incorporate the 3-d position, 3-d rotational velocity, 3d-translational velocity et. for all limbs of the agent **improved our method’s performance comparable to anymorph on all 12 cheetah morphologies.** This helped for all the other agent morphologies as well i.e. Walkers, Humanoids, Hoppers. This allowed us to move to the next step and test on a mixture of varying types of morphologies which is Cheetah-Humanoids-hopper, Walker-Humanoids etc. Also, enabling focusing towards the generalization performance of our method as well. **Our method’s generalization performance was lower than anymorph.**

7.4 Ablation experiments

To improve the generalization performance of our method, the experiments performed included the following

1. Include limb type vector or not in the state representation. The limb type vector denotes the type of limb for an agent morphology. For the Cheetah agent, the type of limbs are “shin”, “thigh”, “foot” and “torso”. So far, in all our experiments, this was not included in the state representation. We found that **including the limb type vector gives overall better learning and generalization performance across all morphologies.** This again shows that **more information about the agent’s state leads to better performance.**

2. Choice between using CLS(classification token) or mean pooled BERT embed-

ding. We give limb and sensor name as input to the BERT and receive token embeddings as the output. So far, we have been averaging the token embeddings to give input to the transformer. This experiment evaluates whether to use sentence embeddings referred to as 'CLS' token embedding from the BERT model output or average across token embeddings(mean pooling). We found that the **'CLS' token embedding works better than mean pooling on most of the tasks.**

This emphasized the importance to perform ablation for improving the model performance. However, our model still didn't perform better than anymorph.

7.5 Importance of evaluating on multiple seeds

Initially when evaluating the model's learning and generalization performance with AnyMorph on 1 seed, the model performance was comparable. However when evaluated with 3 seeds, our method's performance was much lower than AnyMorph on most of the tasks. This is because **our method was not learning on specific seeds on those tasks. Increasing the hyperparameter to 0.2 improved our method's performance.** This helped on most of the tasks except the Walker-Humanoids-Hopper. Also, our method was still not performing better than AnyMorph.

7.6 Other learning

Working with collaborators gave me insight on approaching the problem from multiple angles. One of the examples is that when on the original task our method did not work, we created new training and test splits where there were one or more missing limbs from the training set. The intuition behind this was that given our approach we have access to pre-trained BERT embeddings for the unseen limbs while AnyMorph does not have any learned embedding associated with the unseen limbs. And this should allow our method to perform better. Our method gets somewhat better performance than AnyMorph when we have only 1 missing limb from the training set on the 3 seeds. However, this does not scale with more missing limbs from the training set. Through this journey, I believe I got to face numerous challenges as outlined above, gaining hands-on experience to solve these and interesting knowledge along the way.

BIBLIOGRAPHY

[1] ALAMMAR, J. The illustrated bert, elmo, and co.

[2] ASHISH VASWANI, NOAM SHAZEER, N. P. J. U. L. J. A. N. G. K. I. P. Attention is all you need. In *NIPS* (2017).

[3] BRANDON TRABUCCO, MARIANO PHIELIPP, G. B. Anymorph: Learning transferable polices by inferring agent morphology. In *ICML* (2022).

[4] EVGENII NIKISHIN, MAX SCHWARZER, P. D. P.-L. B. A. C. The primacy bias in deep reinforcement learning. In *ICML* (2022).

[5] *Gym Mujcoco Documentation*.

[6] JACOB DEVLIN, MING-WEI CHANG, K. L.-K. T. Bert: Pre-training of deep bidirectional transformers for language understanding. In *ICLR* (2021).

[7] ROYA FIROOZI, JOHNATHAN TUCKER, S. T. A. M. J. S. W. L. Y. Z. S. S. A. K. K. H. B. I. D. D. J. W. C. L. M. S. Foundation models in robotics: Applications, challenges, and the future.

[8] URI ALON, E. Y. On the bottleneck of graph neural networks and its practical implications. In *ICLR* (2021).

[9] VITALY KURIN, MAXIMILIAN IGL, T. R. W. B. S. W. My body is a cage: The role of morphology in graph-based incompatible control. In *ICLR* (2021).

[10] WENLONG HUANG, IGOR MORDATCH, D. P. One policy to control them all: Shared modular policies for agent-agnostic control. In *ICML* (2020).

Appendix I

Train-Test Splits

We train our method, MA-BERT, and AnyMorph on three train sets of morphologies and evaluate its generalization performance on test set of morphologies also referred to as held-out morphologies. In this section, we list the morphologies in training and test sets across the three sets. We observe that for each of the four types of morphologies in the benchmark tasks, there is a full morphology containing all the limbs for that morphology. Then there are sub-morphologies formed by removing one or more limbs from the full morphology. In the first train-test set in table I.I, the full morphology is included during training for all morphologies so no limb is unseen during training. The generalization evaluation is done on the unseen sub-morphologies. In the other two sets, we create the train-test sets such that there is one or more limbs unseen per morphology, respectively, during training. These are given in tables I.II and I.III respectively. We also move the full morphologies per agent type to the test set. This further complicates the generalization task.

For the second train-test set in I.II , for cheetahs, we move all the morphologies containing "foot" limb to the test set. Similarly, for walkers we move morphologies containing "left3_joint" and for humanoids, we move morphologies containing "left_shoulder1" limb to the test set.

The third train-test set of morphologies is given by the table I.III. This set contains relatively much lesser morphologies in the training set than the test set compared to the above two.

Task	**Training Morphologies**	**Testing Morphologies**
Cheetahs	cheetah_2_back cheetah_2_front cheetah_3_back cheetah_3_front cheetah_4_allback cheetah_4_allfront cheetah_4_back cheetah_4_front cheetah_5_balanced cheetah_5_front cheetah_6_back cheetah_7_full	cheetah_3_balanced cheetah_5_back cheetah_6_front
Walkers	walker_2_main walker_4_main walker_5_main walker_7_main	walker_3_main walker_6_main
Humanoids	humanoid_2d_7_left_arm humanoid_2d_7_lower_arms humanoid_2d_7_right_arm humanoid_2d_7_right_leg humanoid_2d_8_left_knee humanoid_2d_9_full	humanoid_2d_7_left_leg humanoid_2d_8_right_knee
Hoppers	hopper_3 hopper_4 hopper_5	

Table I.I – Train-Test Split 1

Task	**Training Morphologies**	**Testing Morphologies**
Cheetahs	cheetah_2_back cheetah_2_front cheetah_3_back cheetah_3_front cheetah_4_allback cheetah_3_balanced cheetah_4_back cheetah_4_front cheetah_5_back cheetah_5_balanced cheetah_6_back	cheetah_7_full cheetah_5_front cheetah_6_front cheetah_4_allfront
Walkers	walker_2_main walker_3_main walker_4_main walker_5_main	walker_6_main walker_7_main
Humanoids	humanoid_2d_7_left_arm humanoid_2d_7_lower_arms	humanoid_2d_7_left_leg humanoid_2d_8_right_knee humanoid_2d_7_right_arm humanoid_2d_7_right_leg humanoid_2d_8_left_knee humanoid_2d_9_full
Hoppers	hopper_3 hopper_4 hopper_5	

Table I.II – Train-Test Split 2

Task	**Training Morphologies**	**Testing Morphologies**
Cheetahs	cheetah_2_front cheetah_3_front cheetah_4_allfront	cheetah_7_full cheetah_5_front cheetah_6_front cheetah_3_back cheetah_2_back cheetah_4_allback cheetah_3_balanced cheetah_4_back cheetah_4_front cheetah_5_back cheetah_5_balanced cheetah_6_back
Walkers	walker_2_main walker_3_main walker_4_main	walker_6_main walker_7_main walker_5_main
Humanoids	humanoid_2d_7_left_arm humanoid_2d_7_lower_arms	humanoid_2d_7_left_leg humanoid_2d_8_right_knee humanoid_2d_7_right_arm humanoid_2d_7_right_leg humanoid_2d_8_left_knee humanoid_2d_9_full
Hoppers	hopper_3 hopper_4 hopper_5	

Table I.III – Train-Test Split 3

Appendix II

Hyperparameters

We list the hyperparameters used for training the transformer policy and for TD3 RL algorithm across all 3 sets. We change the exploration hyperparameter σ for the last two sets from 0.126 to 0.2.

Hyperparameter Names	**Value**
Z (Sinusoidal Embedding size)	96
Transformer Hidden Size	128
Transformer Feedforward Size	256
Attention Heads	2
Transformer Encoder Layers	3
Transformer Decoder Layers	3
Transformer Activation	relu
Dropout Rate	0.0

Table II.I – Hyperparameters for the Transformer Architecture

Hyperparameter Names	**Value**	**Modified Value set 2**	**Modified Value set 3**
Number of Random Seeds	3		
Batch Size	100		
Max Episode Length	1000		
Max Replay Size Total	10000000		
Max Environment Steps	3000000		
Policy Update Interval	2		
Initial Exploration Steps	10000		
Policy Noise	0.2		
Policy Noise Clip	0.5		
τ	0.046		
σ	0.126	0.2	0.2
Discount Factor	0.99		
Gradient Clipping	0.1		
Learning Rate	0.00005		

Table II.II – Hyperparameters for TD3

Appendix III

Encoded Names

In this section, we list down the names of limbs and their sensors and actuators for all morphologies. For the benchmark used, **we have the same sensors for all morphologies**. These names are encoded by BERT to obtain the token embeddings for each limb's sensor. We have not listed the dimensionality of each sensor in this table, e.g., the position, velocities are 3-d. Considering cheetahs morphology, limb and sensor names as ffoot and position respectively , we name along the three dimensions as as ffoot_position_x, ffoot_position_y, ffoot_position_z. **The names of all actuators and limbs are same as we have one actuator per limb in all morphologies**.

Sensor Names	**Cheetahs**	**Walkers**	**Humanoids**	**Hoppers**
Position	FFoot	Left1_joint	Right_hip_y	Thigh_joint
Translational Velocity	FShin	Left2_joint	Right_knee	Leg_joint
Rotational Velocity	FThigh	Left3_joint	Left_hip_y	Lower_leg_joint
Exponential Map	BFoot	Right1_joint	Left_knee	Foot_joint
Type	BShin	Right2_joint	Right_shoulder1	
Angle	BThigh	Right3_joint	Right_elbow	
Joint Range			Left_shoulder1	
			Left_elbow	

Table III.I – Sensor and actuator names across all morphologies