A Additional Experiments

A.1 Detailed Breakdown of Downstream Tasks Results

Figure 4: [1/2] Detailed breakdown of the results obtained in the 17 fine-tuning downstream tasks for the three environments HalfCheetah, Hopper and Walker2d.

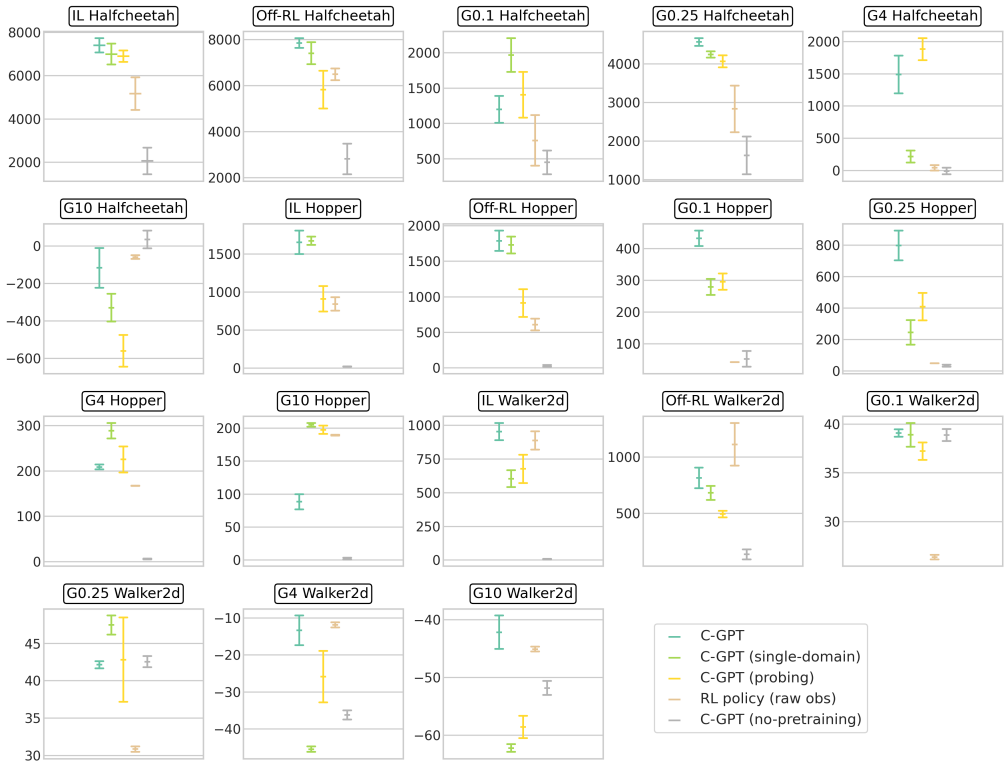

Figure 5: [2/2] Detailed breakdown of the results obtained in the 17 fine-tuning downstream tasks for the three environments HalfCheetah, Hopper and Walker2d.

B Implementation Details

In the sequence tokenization phase, we do not use return conditioning but since the representation models are pre-trained on multiple environments and tasks, we use environment conditioning, *i.e.*, during training, an environment token is appended at the beginning of the sequences in each batch, providing the model with additional contextual information. In practice, the length of the last two modalities (state and action concatenated) varies across different environments. Therefore, the maximum portion of masked tokens at the end of the sequence differs depending on the environment. For instance, in the Hopper environment with 3 actions and 11 observation tokens, the maximum portion of masked tokens is 14, while in HalfCheetah with 6 actions and 18 observation tokens, it is 24. Additionally, as we maintain a fixed-size context window of 128, the sequences' starting points will have varying truncations for different environments, ensuring a non-truncated state at the end of the window. Another design choice is the embedding aggregation, *i.e.*, how to come from a context_window x embedding_dimension tensor to a 1 x embedding_dimension tensor. We decided to use a Conv1d with a kernel size of 1.

Computational Cost. A significant advantage of the component-level sequencing approach is its reduced input dimension, allowing cheaper computational costs. By capturing the components of states and actions at different time steps, the input space expands linearly rather than quadratically mitigating the challenges associated with the curse of dimensionality. To illustrate this, consider a simple example of a 2-dimensional state space with a discretization size of 9. With a component-level granularity, the input size becomes $2 \times 9 = 18$. In contrast, a state-level granularity results in an input size of $9 \times 9 = 81$. The former exhibits linear growth within the observation space, while the latter demonstrates quadratic growth. Moreover, while it effectively multiplies the length of the input sequence by the average number of components in a state, this drawback is absorbed by the

increased context window of transformer models. Lastly, for an equal number of trajectories, the number of tokens is also trivially larger than that with a state- and action-level granularity.

C Additional Details on Masking Patterns

In this section, we provide further details on the masking patterns and schedule used in the SMART (Sun et al., 2023) and MTM (Wu et al., 2023) baselines. In C-GPT or C-BERT, we focused on reducing the technicalities to their minimum: a simple masking pattern, *i.e.*, GPT-like or BERT-like, and no masking schedule.

In SMART, the objective involves three components: Forward Dynamics Prediction, Inverse Dynamics Prediction, and Random Masked Hindsight Control. The masking schedule involves two masking sizes, k and k', which determine the number of masked actions and observations during pre-training. The masking schedule for actions (k) is designed to gradually increase the difficulty of the random masked hindsight control task. It starts with $k = 1$, ensuring the model initially predicts masked actions based on a single observed action. As training progresses, the value of k is increased in a curriculum fashion. The masking schedule for observations (k') ensures that the model learns to predict masked actions based on a revealed subsequence of observations and actions, rather than relying solely on local dynamics. Similar to the action masking schedule, k' starts at 1 and gradually increases during training. SMART's paper suggests that the masking schedule is essential for effective pre-training in control environments. By gradually increasing the masking difficulty, the model is exposed to a range of training scenarios, starting with simple local dynamics and gradually transitioning to complex long-term dependencies.

In MTM, the masking pattern is implemented by requiring at least one token in the masked sequence to be autoregressive, which means it must be predicted based solely on previous tokens, and all future tokens are masked. In addition, MTM uses a modality-specific encoder to elevate the raw trajectory inputs to a common representation space for the tokens. Finally, MTM is trained with a range (between 0.0 and 0.6) of randomly sampled masking ratios.

D Experimental Details and Hyperparameters

In this section, we provide more details about the experiments, including hyperparameter configuration and details of each environment (*e.g.*, version). For all experiments with C-GPT and the baselines, we run 64 rollouts with different random seeds and report the mean and 95% confidence interval (CI) across them.

D.1 Fair Comparison

To ensure a fair comparison between the representation models using an MLP or a transformer architecture, we made sure to have a comparable number of parameters. Both models consist of a minimum of three layers with a size of 256 for the baseline, while C-GPT uses a single layer with a size of 512 for the policy. Consequently, the maximum number of parameters comprised in the parameter-efficient fine-tuned versions (using $(IA)^3$) of PASTA methods is around 140k, which is comparable to that of the RL policies trained from raw observations.

In addition, we choose to fine-tune the MLP baselines to achieve the best performance in each environment. In contrast, we use the same set of hyperparameters for all domains involving PASTA models. This approach ensures a fair comparison between the two methods, with PASTA at a slight disadvantage. However, it also holds the promise of potentially achieving even better performance with the PASTA methods.

Finally, when comparing different pre-training methods, we save 10 checkpoints during the course of training and evaluate them all on the Imitation Learning task. For each method we select the one with highest performance and use it as initialization for all the other downstream tasks.

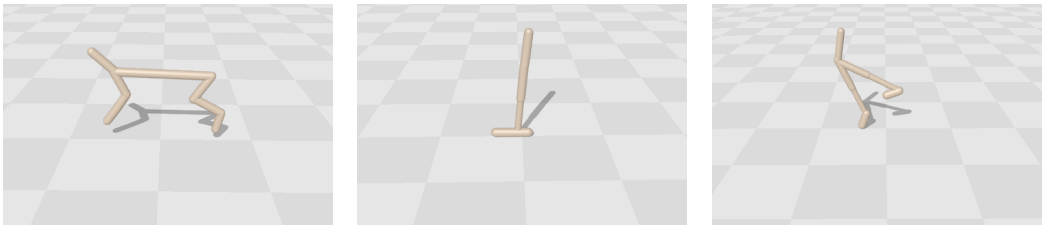

Figure 6: Continuous Control Downstream Tasks.

D.2 Environment Details

For all experiments, we use the 0.0.15 version of Brax (Freeman et al., 2021a). Each environment in Brax, illustrated in Figure 6, provides a realistic physics simulation, enabling agents to interact with objects and the environment in a physically plausible manner. The tasks studied in this paper feature (i) a HalfCheetah robot (Wawrzyński, 2009) with 9 links and 8 joints. The objective is to apply torques on the joints to make the cheetah run forward as fast as possible. The action space for the agents consists of a 6-element vector representing torques applied between the different links; (ii) a Hopper robot (Erez et al., 2011) which is a two-dimensional one-legged figure consisting of four main body parts: the torso, thigh, leg, and foot. The objective is to make hops in the forward direction by applying torques on the hinges connecting the body parts. The action space for the agent is a 3-element vector representing the torques applied to the thigh, leg, and foot joints; (iii) a Walker robot (Erez et al., 2011) which is a two-dimensional two-legged figure comprising a single torso at the top, two thighs below the torso, two legs below the thighs, and two feet attached to the legs. The objective is to coordinate the movements of both sets of feet, legs, and thighs to achieve forward motion in the right direction. The action space for the agent is a 6-element vector representing the torques applied to the thigh, leg, foot, left thigh, left leg, and left foot joints.

D.3 Dataset Details

In this section, we provide further detail on the collection of the datasets. We trained 10 SAC (Haarnoja et al., 2018) agents for a total of 5 million timesteps in each environment. From each, we select the 20% latest trajectories of size 1000, resulting in a combined total of 30 million transitions. With each environment comprising different observation and action sizes, the overall multi-domain dataset is composed of 510 million tokens. We also have one dataset for each domain.

Next, we give the hyperparameters of the SAC agents used to collect the pre-training trajectories. These are given in Table 3.

Table 3: Hyperparameters used in SAC.

Hyperparameter	**Value**
Adam stepsize	$3 \cdot 10^{-4}$
Discount (γ)	0.99
Replay buffer size	10^6
Batch size	256
Nb. hidden layers	2
Nb. hidden units per layer	256
Nonlinearity	ReLU
Target smoothing coefficient (τ)	0.005
Target update interval	1
Gradient steps per timestep	1
Training steps	20,000

We also provide a concrete example of the state and action components with their corresponding properties for the simplest robot structure, Hopper. The number of components for each property

is given in parentheses. In this case, the action space consists of torques applied to the rotors (3), while the observation space includes the following components: z-coordinate of the top (1), angle (4), velocity (2), and angular velocity (4).

D.4 Downstream Tasks Details

In this section, we provide the hyperparameters used in the training of the imitation learning algorithm Behavioural Cloning (BC) (Table 4) and the offline RL algorithm TD3-BC (Table 5).

Table 4: Hyperparameters used in the BC downstream task.

Parameter	Value
Horizon T	1000
Batch Size	1024
Non-Linearity	GELU (Hendrycks & Gimpel, 2016)
Nb. hidden layers	1
Nb. hidden units per layer	512
Adam stepsize	$3 \cdot 10^{-4}$
Training steps	8,000

Table 5: Hyperparameters used in the TD3-BC downstream task.

Parameter	Value
Horizon T	1000
Batch Size	1024
Discount γ	0.99
Non-Linearity	GELU (Hendrycks & Gimpel, 2016)
Nb. hidden layers	1
Nb. hidden units per layer	512
Adam stepsize (actor)	$1 \cdot 10^{-4}$
Adam stepsize (critic)	$3 \cdot 10^{-4}$
Target update rate	$5 \cdot 10^{-3}$
Policy noise	0.2
Policy noise clipping	(-0.5, 0.5)
Policy update frequency	2
Conservatism coefficient α	2.5
Training steps	90,000

Then, we give additional details about the sensor failures downstream tasks. In Table 6, 7 and 8 we include the correspondence between each sensor number and its associated name in all environments. In the 11 downstream tasks, we switch off each one of these sensors.

Finally, in the dynamics change downstream tasks, we vary the gravity coefficient (after training with a coefficient of 1) by multiplying it with the following constant values: 0.1, 0.25, 4, and 10.

Table 6: Sensor name / Sensor number in Halfcheetah.

Sensor name	**Sensor number**
z-coordinate of the center of mass	1
w-orientation of the front tip	2
y-orientation of the front tip	3
angle of the back thigh rotor	4
angle of the back shin rotor	5
angle of the back foot rotor	6
velocity of the tip along the y-axis	7
angular velocity of front tip	8
angular velocity of second rotor	9
x-coordinate of the front tip	10
y-coordinate of the front tip	11

Table 7: Sensor name / Sensor number in Hopper.

Sensor name	**Sensor number**
z-coordinate of the top (height of hopper)	1
angle of the top	2
angle of the thigh joint	3
angle of the leg joint	4
angle of the foot joint	5
velocity of the x-coordinate of the top	6
velocity of the z-coordinate (height) of the top	7
angular velocity of the angle of the top	8
angular velocity of the thigh hinge	9
angular velocity of the leg hinge	10
angular velocity of the foot hinge	11

Table 8: Sensor name / Sensor number in Walker2d.

Sensor name	**Sensor number**
z-coordinate of the top (height of hopper)	1
angle of the top	2
angle of the thigh joint	3
angle of the leg joint	4
angle of the foot joint	5
angle of the left thigh joint	6
angle of the left leg joint	7
angle of the left foot joint	8
velocity of the x-coordinate of the top	9
velocity of the z-coordinate (height) of the top	10
angular velocity of the angle of the top	11

D.5 HYPERPARAMETERS

In Table 9, we show the hyperparameter configuration for C-GPT across all experiments.

Table 9: Hyperparameters and configuration details for C-GPT across all experiments.

Hyperparameter	**Value**
Transformer Layers	10
Transformer Heads	8
Noising Ratio	0.15
Masking Probability	0.8
Random Token Probability	0.1
Non-Linearity	GELU
Learning Rate	$3e-4$
Num Epochs	3
Batch Size	4096
Num Quantization Tokens	1024
Embedding Dimension	256