Offline Actor-Critic Reinforcement Learning Scales to Large Models

Jost Tobias Springenberg [*1] **Abbas Abdolmaleki** [*1] **Jingwei Zhang** [*1] **Oliver Groth** [*1] **Michael Bloesch** [*1] **Thomas Lampe** [*1] **Philemon Brakel** [*1] **Sarah Bechtle** [*1] **Steven Kapturowski** [*1] **Roland Hafner** [*1] **Nicolas Heess** [1] **Martin Riedmiller** [1]

Abstract

We show that offline actor-critic reinforcement learning can scale to large models – such as transformers – and follows similar scaling laws as supervised learning. We find that offline actor-critic algorithms can outperform strong, supervised, behavioral cloning baselines for multi-task training on a large dataset; containing both sub-optimal and expert behavior on 132 continuous control tasks. We introduce a Perceiver-based actor-critic model and elucidate the key features needed to make offline RL work with self- and cross-attention modules. Overall, we find that: i) simple offline actor critic algorithms are a natural choice for gradually moving away from the currently predominant paradigm of behavioral cloning, and ii) via offline RL it is possible to learn multi-task policies that master many domains simultaneously, including real robotics tasks, from sub-optimal demonstrations or self-generated data.

1. Introduction

In recent years, scaling both model and dataset sizes has led to multiple breakthroughs in machine learning. In particular, generative pre-training of large (vision-)language models on web-scale data is now the standard way to solve many language and vision tasks (OpenAI, 2023; Alayrac et al., 2022) and generative models of images and music have, in the last years, reached unprecedented quality (Rombach et al., 2021; Kang et al., 2023).

Recent work on scaling up policy learning for control has shown that, when similar model architectures are used (e.g. transformers), supervised behaviour cloning (BC) from large datasets can lead to surprisingly capable multi-task policies (Reed et al., 2022; Bousmalis et al., 2023; Brohan et al., 2022; Octo Model Team et al., 2023). Although impressive, these examples come with the drawback that high-quality ('expert' demonstration) data is needed for training. While such high quality data is readily available for language and vision domains via the internet, in robotics (and other real world domains) expert data is scarce and expensive to obtain – and in many cases not available in the first place. It is thus desirable to use different training methods, such as reinforcement learning (RL), that can utilize sub-optimal data or data generated without a human in the loop, i.e. generated by an agent, – which can be more readily available – while retaining scaling benefits.

However, training large behaviour models via offline RL methods[1] is a largely unexplored area of research. While first explorations of applying pure Q-learning on larger multi-task datasets exist (Kumar et al., 2022a; Chebotar et al., 2023) they either consider non-transformer models of moderate size (Kumar et al., 2022a) or adapt relatively small models and incur significant computational overhead during training (Chebotar et al., 2023). What is missing is a clear recipe detailing how to scale offline RL to large transformers accompanied by an efficient model.

In this work we provide such a recipe and introduce the Perceiver-Actor-Critic (PAC) approach outlined in Figure 1. We show that *a specific class of offline RL algorithms (offline actor-critic methods) can indeed scale to large models and datasets without incurring a large additional computational cost*. In addition we etablish, for the first time, that offline RL *follows similar scaling laws to those observed in the supervised learning regime* (Henighan et al., 2020; Kaplan et al., 2020). We further establish that this class of methods is ideally suited for slowly moving away from supervised BC towards RL during training, allowing us to run large and expensive experiments without fear of instability and to adapt our method depending on the quality of the data.

We introduce a simple offline actor-critic algorithm that optimises a KL-regularized RL objective and can be seen as a simplified variant of MPO/DIME (Abdolmaleki et al., 2018;

[*]Equal contribution [1]Google Deepmind. Correspondence to: Jost Tobias Springenberg <springenberg@google.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[1]This is in contrast to online RL of transformer models which is often applied when large language models are fine-tuned with RLHF, but is prohibitively expensive in real-world settings.

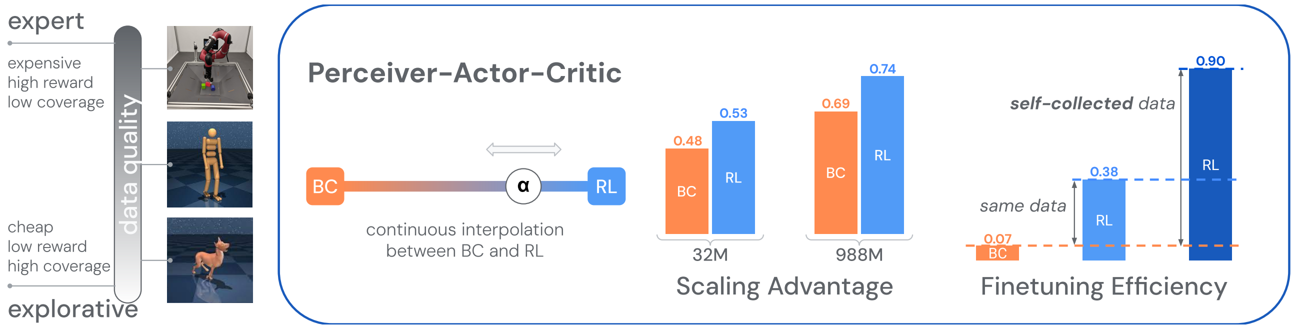


Figure 1. PAC is a scalable neural architecture for continuous control able to smoothly interpolate between BC and offline RL. The system design enables training on heterogenous, multi-modal data of varying quality. We demonstrate that our system achieves higher performance than BC across a series of model scales. The method also enables a seamless transition into offline and online RL finetuning for fast adaptation and mastery of control tasks.

2022). We find that regularizing the policy towards the data distribution (via BC) is sufficient to stabilize offline RL for large models and also allows convenient interpolation between BC and RL. We additionally introduce architectural advances which enable training with RL at scale. E.g. incorporating the action into the Q-function via cross-attention (allowing fast estimation of Q-values for multiple actions) and incorporating a large number of inputs via Perceiver-style cross-attention to learned latent variables; enabling training with many inputs of different modalities (text, proprioception, vision) while enabling inference of a large 1 B parameter model at 20 Hz on a local machine.

PAC outperforms BC on a number of benchmarks in continuous control, including outperforming Gato (Reed et al., 2022) on Control Suite (Tunyasuvunakool et al., 2020) tasks and recovers expert performance from heterogeneous data in a real robot benchmark. This establishes that RL should be considered a viable alternative to BC for large policies. Videos of our agent can be found at `https://sites.google.com/view/perceiver-actor-critic`.

2. Background and Related Work

Supervised Generalist Agents Several recent works have trained large transformer-based (Vaswani et al., 2017) generalist agents via BC by building on previous works in which control tasks were transformed into sequence prediction problems (Chen et al., 2021; Janner et al., 2021). Gato (Reed et al., 2022) for example, was trained on tasks ranging from Atari games to robotics manipulation. Subsequently, large generalist robotics agents (Brohan et al., 2022; Bousmalis et al., 2023; Zitkovich et al., 2023) have been trained on large datasets with multiple tasks, object sets and embodiments, and have been shown to generalize to new tasks and domains after fine-tuning (Bousmalis et al., 2023; Open X-Embodiment Collaboration, 2023). Perceiver-based networks with cross-attention (Jaegle et al., 2021) have also been applied to robotics to minimize computational demands when handling voxel observations (Shridhar et al., 2023; Ze et al., 2023). Finally, Octo Model Team et al. (2023) used multi-headed attention to predict outputs in a similar way to the cross-attention in our system.

Offline RL Offline RL methods (Levine et al., 2020; Lange et al., 2012) learn from fixed datasets without online exploration. Unlike supervised algorithms, they can learn from suboptimal trajectories and thus more data. However, they are at risk of issues like overoptimism for unseen state-action pairs. This is often addressed by regularizing the policy to stay close to the data (Peng et al., 2019; Wang et al., 2020; Fujimoto et al., 2019; Wu et al., 2019). Like prior work (Abdolmaleki et al., 2022; Fujimoto & Gu, 2021), we combine a BC regularization term with an off-policy RL method. Other offline RL methods penalize the value function (Kumar et al., 2020) or prevent value propagation (Kostrikov et al., 2021) for unseen state-action pairs. While most offline RL works use relatively small benchmarks, recent ones have tackled challenging multi-task problems (Kumar et al., 2022a) and pre-trained robotics generalists that can be fine-tuned to new tasks (Kumar et al., 2022b). However, to our knowledge, only the recent Q-Transformer (Chebotar et al., 2023) provides an example of a transformer trained with offline RL on larger datasets, albeit with a relatively small model. Our actor-critic-based approach is more naturally suited for extending BC-based methods and less computationally demanding. This allows us to explore much larger models and perform a scaling law analysis.

Scaling Law Analysis Our scaling law analysis mirrors analyses of large language models for which several studies have shown smooth power-law relations between model size and performance (Kaplan et al., 2020; Hoffmann et al., 2022; Henighan et al., 2020). Some recent works have also investigated scaling behavior of neural networks for online RL (Neumann & Gros, 2022; Hilton et al., 2023) albeit with relatively small (<40M parameter) models. Lee et al. (2022) analyzed how performance scaled with the number of parameters of Decision Transformer (Chen et al., 2021) style networks and includes plots for a CQL (Kumar et al., 2020) offline RL baseline for models up to 200M parameters finding no favourable scaling. In contrast, we find scaling to work for actor-critic methods and provide a thorough scaling law analysis. Concurrent work also shows promising scaling behavior of model-based RL methods (Hansen et al., 2023) for models up to 300M parameters.

3. Scalable Offline Actor-Critic Learning

We scale up offline actor-critic methods to large models. To achieve this, we adapt methods from the offline RL literature and present our proposed algorithm in Section 3.2. We adapt a Perceiver-IO (Jaegle et al., 2022) architecture to the actor-critic setting and present our model in Section 3.3.

3.1. Background and Notation

We consider learning in a multi-task Markov decision process (MDP), where at each time step t the agent selects an action $a_t \in \mathcal{A}$ for its current state $s_t \in \mathcal{S}$, receives a reward $r_{t+1} = R(s_t, a_t, \tau) \in \mathcal{R}$ specific to the task $\tau \in \mathcal{T}$ and transits to the next state $s_{t+1} \sim p(\cdot|s_t, a_t)$. We use the term state and multimodal observations interchangably, although the true environment state is often not fully observable.

An RL algorithm seeks to find a policy $\pi(a_t|s_t,\tau)$ that maximizes the per-task discounted cumulative return $\mathbb{E}_{p_\pi}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t,\tau)\right]$ under the trajectory distribution p_π induced by the policy π. The Q-function, the V-function and the advantage function are defined as: $Q^\pi(s_t,a_t,\tau) = \mathbb{E}_{p_\pi, s_k=s_t, a_k=a_t}\left[\sum_{k=t}^{\infty}\gamma^{k-t}R(s_k,a_k,\tau)\right]$, $V^\pi(s_t,\tau) = \mathbb{E}_{a_t\sim\pi(\cdot|s_t,\tau)}\left[Q^\pi(s_t,a_t,\tau)\right]$, $A^\pi(s_t,a_t,\tau) = Q^\pi(s_t,a_t,\tau) - V^\pi(s_t,\tau)$. We assume the availability of an offline dataset $\mathcal{D} = \{(s_t, a_t, s_{t+1}, \tau)\}$, generated by following a behavior policy $b(a_t|s_t,\tau)$, and access to either the reward function R or reward annotations.

We also make use of behaviour cloning (BC) terms for training which can be formalized as minimizing $\mathbb{E}_{(s_t,\tau)\in\mathcal{D}}\,\mathrm{D}_{\mathrm{KL}}[b,\pi|s_t,\tau] = -\mathbb{E}_{\mathcal{D}}\log\pi(a_t|s_t,\tau) + K_{BC}$ between the behavior policy b that generated the dataset and the learned policy π (K_{BC} is a constant offset).

3.2. Offline KL-Regularized Actor-Critic

We target a KL-regularized RL objective, where the goal is to find a policy π_{imp} that improves over a reference policy $\tilde{\pi}$ via $\pi_{\text{imp}} = \arg\max_\pi J(\pi)$ where $J(\pi)$ is given as:

$$J(\pi) = \mathop{\mathbb{E}}_{(s_t,\tau)\in\mathcal{D}}\left[\mathop{\mathbb{E}}_{a_t\sim\pi}\left[Q^\pi(s_t,a_t,\tau)\right] - \eta\,\mathrm{D}_{\mathrm{KL}}\left[\pi,\tilde{\pi}|s_t,\tau\right]\right] \tag{1}$$

where η is a hyperparameter determining the strength of the regularization towards the reference policy $\tilde{\pi}$. The solution to this maximization problem is given as (see Appendix A.1 for derivation):

$$\begin{aligned}\pi_{\text{imp}}(a_t|s_t,\tau) &\propto \exp(Q^{\pi_{\text{imp}}}(s_t,a_t,\tau)/\eta)\tilde{\pi}(a_t|s_t,\tau),\\ &\propto \exp(A^{\pi_{\text{imp}}}(s_t,a_t,\tau)/\eta)\tilde{\pi}(a_t|s_t,\tau).\end{aligned} \tag{2}$$

This observation allows us to transform the RL problem of finding an optimal policy into a weighted supervised learning problem (cf. Abdolmaleki et al. (2018)). Assuming access to an estimate of $Q^{\pi_{\text{imp}}}$ or $A^{\pi_{\text{imp}}}$, we can fit a parametric policy π_θ by minimizing its divergence $D_{KL}[\pi_{\text{imp}}, \pi_\theta|s_t,\tau]$ to π_{imp} using a sample based estimate. *Turning the policy optimisation problem into an instance of supervised learning has the major benefit that it is easy to trade-off the policy optimisation objective with a behavior cloning term, since all loss terms will be (weighted) negative log likelihoods.*

Different choices for estimating $Q^{\pi_{\text{imp}}}$ or $A^{\pi_{\text{imp}}}$ as well as the reference policy $\tilde{\pi}$ lead to different algorithmic variants. We will concentrate on a Q-function based variant in the main paper but describe a state-value function (V-function) based variant in the appendix; which has similar scaling benefits.

We train the policy π_θ together with an estimate $Q_\theta \approx Q^{\pi_\theta} \approx Q^{\pi_{\text{imp}}}$ of the state-action value function. To balance losses, we employ tools from the distributional reinforcement learning literature (Bellemare et al., 2017) which transform the problem of learning Q_θ into minimizing the negative log likelihood of a discretized Q-function distribution $p_\theta(q|s_t,a_t,\tau)$. Using the distributional TD operator (Bellemare et al., 2017) we can compute a sample-based target Q-distribution $\Gamma_{\theta'}(q|s_t,a_t,\tau)$ (see Appendix A.2) where θ' are the parameters of a target network which is periodically updated to a time-lagged version of θ. The same target parameters also give rise to a target policy $\pi_{\theta'}$ which we use as the reference policy in Equation (2), i.e. $\tilde{\pi} = \pi_{\theta'}$. Combining the policy loss, a BC loss, and the KL-based Q-value loss yields a total loss containing three KL terms:

$$\begin{aligned}L^Q(\theta) = \mathop{\mathbb{E}}_{\mathcal{D}}\Big[&(1-\alpha)\,\mathrm{D}_{\mathrm{KL}}[\pi_{\text{imp}}, \pi_\theta|s_t,\tau,\tilde{\pi}=\pi_{\theta'}]\\ &+\alpha\,\mathrm{D}_{\mathrm{KL}}[b,\pi_\theta|s_t,\tau]\\ &+\beta\,\mathrm{D}_{\mathrm{KL}}[\Gamma_{\theta'}(q|s_t,a_t,\tau), p_\theta(q|s_t,a_t,\tau)]\Big]\\ = -\mathop{\mathbb{E}}_{\mathcal{D}}\Big[&(1-\alpha)\mathop{\mathbb{E}}_{a'\sim\pi_{\theta'}}\left[w(a',s_t,\tau)\log\pi_\theta(a'|s_t,\tau)\right]\\ &+\alpha\log\pi_\theta(a_t|s_t,\tau)\\ &+\beta\mathop{\mathbb{E}}_{q\sim\Gamma_{\theta'}}\log p_\theta(q|s_t,a_t,\tau)\Big] + K_H,\end{aligned} \tag{3}$$

where $w(a',s_t,\tau) = \frac{\exp(Q_\theta(s_t,a',\tau)/\eta)}{\mathbb{E}_{a'\sim\pi_{\theta'}}[\exp(Q_{\theta'}(s_t,a',\tau)/\eta)]}$ and K_H is a constant entropy related offset independent of θ. The expectation over the data is estimated by sampling $(s_t,a_t,s_{t+1},\tau)\in\mathcal{D}$, the expectation over action samples

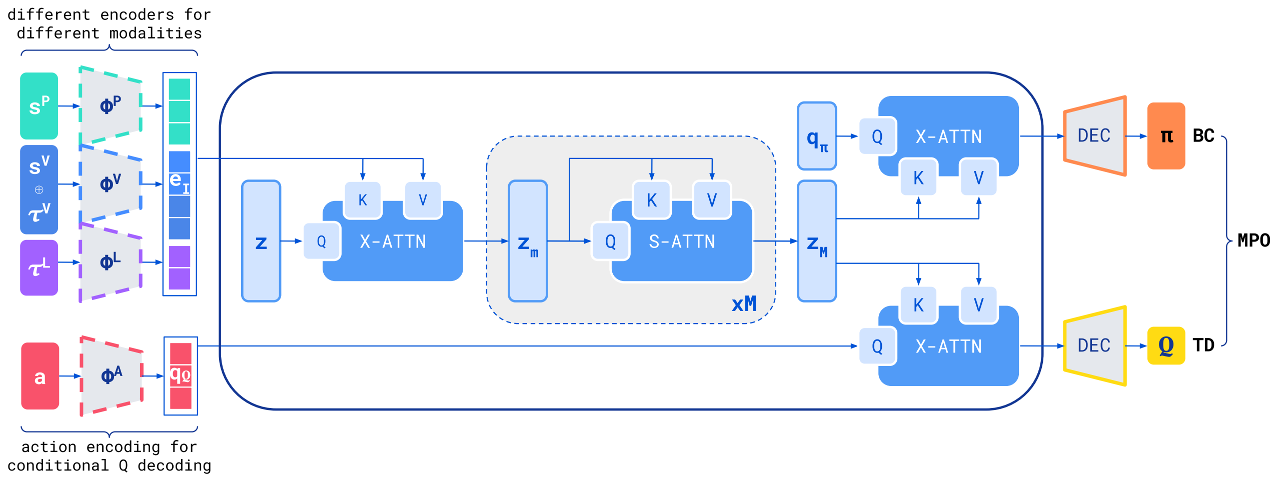


Figure 2. High-level PAC model architecture. Modality-specific encoders transform proprioceptive (P), visual (V), and language (L) inputs into embedding vectors e_I, which are cross-attended by learnable latent queries z_0. This is followed by a series of self-attention blocks to yield the latent encoding z_M, which is then queried via additional cross-attention modules to decode the desired outputs. The policy decoder employs a learnable query q_π to cross-attend z_M and outputs the logits of action distributions. The Q-value decoder employs a query q_Q based on the encoded actions to cross-attend z_M and outputs the action-specific logits of the distributional Q-function.

from $\pi_{\theta'}$ is estimated based on $N = 10$ samples and the expectation $\mathbb{E}_{q\sim\Gamma_{\theta'}}$ can be evaluated analytically. Finally α and β are multipliers trading off different loss components (which are relatively easy to set due to all losses corresponding to weighted categorical log likelihoods). We refer to Appendix A.2 for a step-by-step derivation.

Notably, aside from the KL towards the improved policy π_{imp}, Equation (3) also includes a KL towards the behaviour policy b. This additional regularization is necessary to prevent π_θ from converging to action samples that have high Q-values but are far away from those observed in the data (and are thus at the risk of being overestimated); a common issue in offline RL with Q-functions (Levine et al., 2020). The additional BC term prevents this, following prior examples for using a BC loss as a simple regularisation technique in offline RL (Abdolmaleki et al., 2022; Fujimoto & Gu, 2021). We find that this is the only term needed to stabilize learning. In addition, it gives us a natural way for moving away from learning via pure behavioral cloning ($\alpha = 1$) towards pure policy optimisation against the learned Q-function ($\alpha = 0$). This also allows us to perform expensive training runs of large models with confidence since we can set α to a larger value such that the policy stays close to BC, guaranteeing stable training, and can reduce it later during fine-tuning.

3.3. Scalable Architecture for Actor-Critic Learning

With the proposed offline actor-critic algorithm, we now describe how π_θ and Q_θ are instantiated with scalable network architectures. In particular, we aim for an architecture that is flexible enough to incorporate different modalities of state observations and task descriptions as well as various action specifications, while also being computationally efficient for consuming high-dimensional inputs during learning and at inference time (to enable 20 Hz control of real robots). In this section, we describe how we adopt a Perceiver-IO architecture (Jaegle et al., 2021) to achieve the above. The model is depicted in Figure 2.

Observation Encoding Given multimodal inputs, in particular proprioceptive and visual observations $s_t = (s_t^P, s_t^V)$ along with visual and language task descriptions $\tau = \tau^V, \tau^L)$, our model first deploys one encoder (ϕ) per modality to encode the inputs into embedding vectors: $e_I = \phi^P(s_t^P) \oplus \phi^V(s_t^V) \oplus \phi^V(\tau^V) \oplus \phi^L(\tau^L) \in \mathbb{R}^{N\times D_I}$, with N and D_I denoting the number and dimensionality of the embedding vectors. Details of each modality encoder are provided in Appendix B.2. For the proprioception encoder ϕ^P we propose a novel *multi-scale* normalizer to account for arbitrary input scales and provide further details and ablations on this encoder choice in Appendices B.1 and D.2.1. We highlight that our model uses task descriptions of different modalities (text and vision) and we analyse this multimodal task conditioning in Appendix D.2.4.

Transformer on Latent Space At this point, the commonly adopted approach would be to feed the embedding sequence $e_I \in \mathbb{R}^{N\times D_I}$ directly into a transformer consisting of multiple stacked self-attention blocks. However, for the domains we consider, the input sequence length amounts to thousands of tokens for a single time step. As the computational complexity and memory usage of self-attention scales quadratically with the sequence length, this common treatment potentially hinders the learned controller from being applicable to real robotic systems that impose real-time constraints. To address this, we adopt the methodology from the perceiver model (Jaegle et al., 2021). Specifically, a cross-attention block is placed at the front-end of the network in which the input sequence of embeddings e_I are queried by N_Z trainable latent vectors each of size D_Z: $z \in \mathbb{R}^{N_Z \times D_Z}$, which outputs latent embeddings z_0. This is followed by M self-attention operations on the latents which finally yield $z_M \in \mathbb{R}^{N_Z \times D_Z}$. Since the number of latent vectors is typically much smaller than the input sequence length ($N_Z \ll N$) and the self-attention operation is shifted from the input embeddings to the latent vectors, this effectively reduces the computation and memory usage to $O({N_Z}^2)$. We provide more details on the perceiver backbone in Appendix B.3.

Policy and Value Decoding To implement an actor-critic algorithm, the model needs to output both a Q-value estimate and an action prediction. While the action prediction $\hat{a}_t$ can be directly modeled as a function of the inputs (s_t, τ) which are encoded into e_I and thus z_M, the value estimate $Q_\theta(s_t, a_t, \tau)$ also depends on the action a_t which is not encoded in z_M. To obtain the two types of outputs we cross-attend the latent embeddings z_M with dedicated queries. While the queries for the policy are learned vectors, the Q-value queries are computed by encoding the action $a_t \in \mathbb{R}^{N^A}$ via our multi-scale normalizer. This has the advantage that the model is less prone to ignoring the action compared to when the action would be presented as an input (a common problem when learning Q-values). It also allows efficient evaluation of the Q-function for multiple action samples via caching of the action-independent latent z_M. We provide more details in Appendix B.4 and ablate the importance of the cross-attention for Q-value prediction in Appendix D.2.2.

4. Experiments

We present three sets of experiments investigating different aspects of PAC. Section 4.1 analyzes whether PAC follows scaling laws similar to established supervised learning settings. Section 4.2 compares PAC's performance after large-scale training with the RL objective to different BC baselines across over 100 continuous control tasks. Finally, Section 4.3 studies how PAC can be finetuned by leveraging its Q-function to hone in on a real robot task and further improve its performance using self-generated data.

We use a large dataset throughout all experiments which combines tasks from three different sources: Gato data (Reed et al., 2022) consist of records of an RL agent solving 32 simulation tasks in Control Suite (Tunyasuvunakool et al., 2020). RoboCat data (Bousmalis et al., 2023) operates on the RGB Stacking benchmark (Lee et al., 2021) using RL in simulation to build pyramid and tower structures using a 7-DoF Panda robot. It also contains an Insertion task featuring teleoperated simulation data of the same robot inserting differently sized gears onto pegs. Lastly, CHEF (Lampe et al., 2023) data contains simulated and real-world records of a 5-DoF Sawyer robot stacking two objects in the RGB Stacking benchmark using an RL algorithm. For all episodes in our dataset, a short language instruction describing the task is added to each frame, e.g. `humamoid.run` or `panda.sim.pyramid`, which serves as a unique goal instruction to differentiate between the different tasks. For all RoboCat tasks an additional goal image is provided as the goal instruciton. We again emphasize that our model can handle both language and visual goal descriptions (where present) and refer to Appendix D.2.4 for details about the goal conditioning. In total, our data mix consists of 3.64M episodes across 102 simulated and 30 real continuous control tasks which equates to approximately 2.45T tokens for model training (cf. Appendices C.3 and C.4).

4.1. Scaling Analysis for Offline RL Objectives

A central part of our investigation is to understand the interaction between offline actor-critic algorithms and scalable neural network architectures that use (self-)attention. When trained with supervised objectives, such as next-token prediction, architectures of this type usually follow *scaling laws* (Kaplan et al., 2020), i.e. for all performance-optimal models the number of tokens consumed and the number of model parameters used follow power-laws in the number of FLOPs spent. However, it has so far been unclear whether these scaling laws also extend to RL. To investigate this relationship, we adopt the methodology from Hoffmann et al. (2022) (also known as 'Chinchilla scaling laws') and apply it to PAC. We define five different model scales (XXS, XS, S, M and L) ranging from 32M to 988M parameters to study the scaling behavior of PAC and report the full model architecture hyper-parameters in Appendix C.1.

To conduct our analysis, we train PAC across the different scales with two different values of α for the BC/RL trade-off. Setting $\alpha = 1.0$ results in a BC objective for the policy and constitutes our baseline `BC+Q`[2] while `PAC` performs offline RL with $\alpha = 0.75$. With a batch size of 512 trajectories of length five, one epoch of our data mix takes approximately 2.7M steps. Therefore we train each model for 3M updates to stay in a single-epoch regime.

Following Kaplan et al. (2020); Hoffmann et al. (2022), the power laws between compute operations C, number of

[2]Using a Q-value loss term with $\beta > 0$ never decreased the performance in our BC experiments; we keep it for comparability.

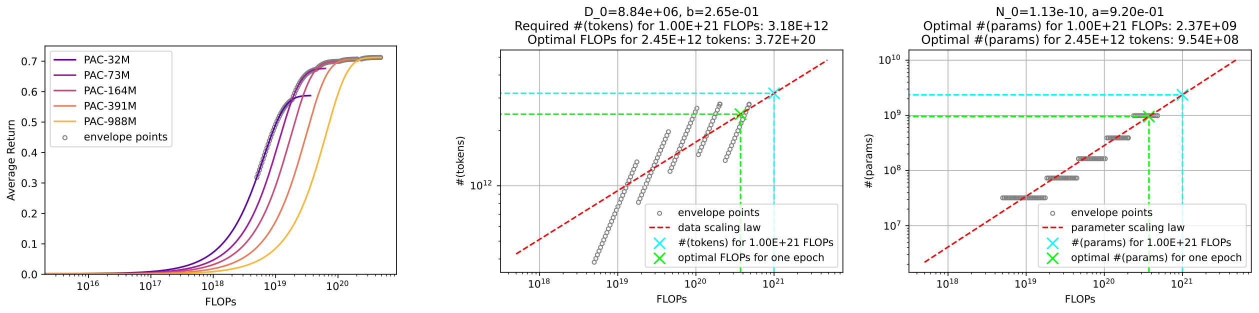


Figure 3. Scaling laws based on the return profile envelope for PAC. We select 100 logarithmically spaced points between 5E+18 and 5E+20 FLOPs on the envelope of the return profiles (left) and use them to fit the scaling laws (middle, right). For both the token and parameter scaling plots, we indicate the scaling trend with a dashed red line. The dashed green line represents the optimal number of parameters and compute budget needed to fit the data in one epoch of training. The dashed teal line represents the optimal data and parameter trade-off for a FLOP budget of 1E+21.

tokens D and number of parameters N for performance-optimal models of the family are:

$$N(C) = N_0 * C^a, \quad D(C) = D_0 * C^b. \tag{4}$$

Normally, the coefficients a and b are fitted using compute-optimal model checkpoints along the *loss envelope* of the different training runs for different compute budgets. However, we observe that the training loss is not a reliable indicator for model performance in our setting (cf. Appendix E.3). We therefore use an approximation of the average episode return as a means to select the best performing model for each compute budget from the respective model family. To extrapolate from average returns we fit a logistic function to regress the training steps against average return across all tasks, normalized in $[0, 1]$ (cf. Appendix E.1) to obtain a *return profile* for each model. We plot the return profiles for the PAC family against FLOPs in the left column of Figure 3 and use them to select 100 points on the profiles' envelopes to fit the scaling laws of Equation (4). Scaling plots for all model families are presented in Figure 12 in the Appendix.

The scaling laws are different for the BC and offline RL settings. When we constrain the data budget to a single epoch, i.e. 2.45T tokens, the fits suggest to train a 1.33B parameter model in the BC+Q case whereas in the case of PAC a smaller model of only 954M parameters is suggested. This is consistent with our observation that the L-size of PAC with 988M parameters performs best which is close to the predicted optimality point while the BC+Q model likely would benefit from being scaled up further. Data-wise, BC+Q and PAC scale nearly the same ($b(\text{PAC}) \approx b(\text{BC+Q}) \approx 0.266$). However, the RL objective seems to benefit more from additional parameters as the compute budget increases compared to BC ($a(\text{PAC}) = 0.920 > a(\text{BC+Q}) = 0.828$) suggesting that the capacity needed for the Q-function is larger (though as we will see the Q-function can learn from lower quality data).

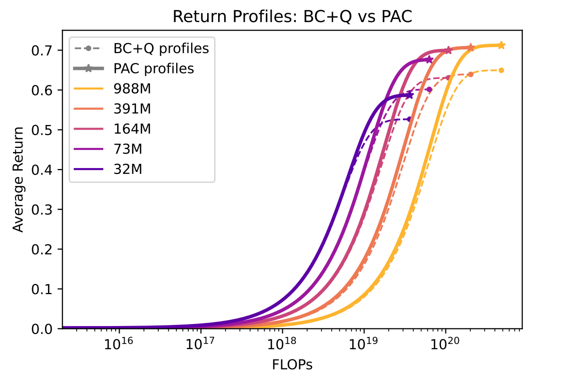
Another way to compare the scaling behaviors between the BC and offline RL objectives is through the lens of the Iso-Return contours (analogous to the Iso-Loss landscape of Hoffmann et al. (2022)) as presented in Figure 4. The Iso-Return landscape projects the return vs. FLOPs from the return profiles into the parameters vs. FLOPs space and indicates the average return via the color gradient. We draw Isolines between the data points observed during model evaluation to survey the return landscape of the two model families. The Isolines of PAC are drawn solid, the ones of BC+Q are dashed. We also plot the parameter scaling laws for both model families in red over the landscape to indicate the direction of 'optimal scaling' for both families, also called the 'efficient frontier' (Hoffmann et al., 2022). In this direction, the return landscape for both is likely to incline. Comparing the Isolines between both families, we observe that the RL objective shifts all return plateaus to the top left compared to the BC baseline. This suggests that the RL objective can use additional parameters and training FLOPs increasingly more effectively compared to a pure BC objective and *scales better* with increased compute.

4.2. Large-scale Offline Actor-Critic Learning

The scaling analysis above suggests that PAC's offline RL outperforms a BC objective when scaled up. We now investigate whether this still holds when comparing against two strong BC baselines: Gato (Reed et al., 2022) and RoboCat (Bousmalis et al., 2023). The pre-training phase of such large models typically only uses a BC objective to ensure 'safe' optimization and reduce the risk of divergence for these costly training runs. However, if an offline RL objective could be used safely, this would allow using sub-optimal data from the start and further enhance subsequent self-improvement (since a Q-function is available).

Table 1. Policy success rates across $\#(\mathcal{T})$ tasks in each domain for 100 evaluations per task. The average success rate in the training data is reported as p_D. For Gato:Control, the percentage of achieved expert average reward and the standard-error-based 95% CIs are reported. For all other domains, the average success rates and their corresponding Wilson score intervals for $\alpha_W = 0.05$ are reported. Best results (within CI of the best mean) in each row are bold. [† cited from Reed et al. (2022); ⋆ cited from Bousmalis et al. (2023)]

Domain	$\#(\mathcal{T})$	p_D	**BC (Gato† / RC⋆)**	**FilteredBC**	**BC+Q**	**PAC**	**α-PAC**
Gato:Control	32	N/A	63.6†	75.8 [62.5, 78.6]	84.6 [79.6, 89.7]	87.7 [83.8, 91.6]	**92.1 [88.4, 95.9]**
RC:Tower	7	75	61.0⋆ [57.3, 64.5]	64.0 [60.4, 67.5]	**71.3 [67.8, 74.5]**	**69.3 [65.8, 72.6]**	**69.6 [65.9, 72.7]**
RC:Pyramid	30	75	**64.5⋆ [62.8, 66.2]**	**64.0 [62.3, 65.7]**	62.4 [60.7, 64.1]	**63.5 [61.7, 65.1]**	**64.9 [63.1, 66.6]**
RC:Insertion	3	97	71.3⋆ [66.0, 76.2]	81.0 [75.8, 84.7]	79.7 [74.8, 83.8]	80.3 [75.5, 84.4]	**89.3 [85.0, 92.1]**
CHEF:sim	1	28	N/A	17.0 [10.9, 25.5]	11.0 [6.3, 18.6]	**55.0 [45.2, 64.4]**	**52.0 [42.3, 61.5]**



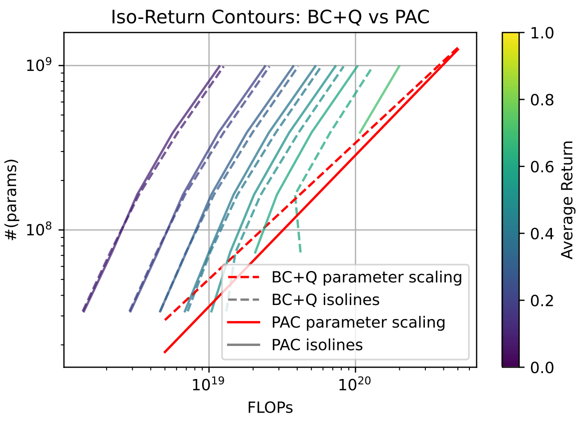


Figure 4. Iso-Return comparison of BC+Q vs PAC. The return profile (top) contrasts the expected average return between the BC baseline and the RL objective across all model scales. The Iso-Return contours (bottom) depict how the reward landscape over the parameter-FLOPs landscape shifts between using the BC objective (dashed contours) and the RL objectives (solid contours). The parameter scaling laws indicating the 'efficient frontier' for both model families are also plotted in red as a reference for the likely scaling progression of the return landscape.

For our comparison, we consider the following PAC-based models: PAC ($\alpha = 0.75$) our main actor-critic model; BC+Q ($\alpha = 1, \beta > 0$) as a baseline which also learns a Q-function, but never leverages it for policy optimization (we found this to always be at least as good as pure BC in preliminary experiments); and FilteredBC ($\alpha = 1, \beta = 0$) which does not learn a Q-function and is only trained on successful episodes of our data mix to mimic a 'pure' BC setting. We also add α-PAC as our best actor-critic model which uses a different value for the BC/RL trade-off α for each dataset to obtain the best performance and demonstrate that our method can be optimally tuned to deal with data of widely varying quality in the same training mixture. More detailed ablations on the choice of α and β are presented in Appendix D.1. For a fair comparison to the 1.2B parameter versions of Gato and RoboCat, we use PAC in its L-size with about 1B parameters and train for 3M updates. All details of the pre-training data mix and optimizer hyperparameters are reported in Appendix C.5. Each PAC model is evaluated across all task families in simulation and the results are reported in Table 1. Where available, we cite the baseline results for Gato and RoboCat (RC) directly from their respective papers. In general, the Q-function-based PAC outperforms BC across tasks, confirming our hypothesis that offline RL is a viable alternative for training large models and we note that a V-function based variant also achieves similar results (see Appendix F.1).

In more detail: On the Control Suite tasks PAC outperforms all baseline tasks reaching 87.7% of expert performance and α-PAC even boosts it further to 92.1%.[3] It is also worth noting that our BC baselines already outperform the Gato results, potentially due to PAC's improved architecture. On the RoboCat tasks, PAC performs commensurately with all BC baselines and outperforms prior work especially on the more difficult Tower task achieving $\approx$ 70% success rate, but the difference is less pronounced since the respective datasets come from near expert policies (> 75% success). The biggest difference is observed on the insertion task where FilteredBC and BC+Q already improve $\approx$ 10% over the RoboCat baseline and α-PAC yields another significant improvement to 89.3%. Finally, for the stacking task from CHEF which has the poorest data quality – collected form a sub-optimal policy that only achieved 28% success

[3]For compatibility we also use the expert performance definition of Reed et al. (2022).

Table 2. Success rates with Wilson score intervals for $\alpha_W = 0.05$ for CHEF:real tasks (400 trials per task) for different objectives, as well as for an RL finetuning run with self-generated data (RLFT).

Domain	$\#(\mathcal{T})$	**BC+Q**	α-**PAC**	α-**PAC (RLFT)**
CHEF: real	5	7.1 [6.1, 8.2]	69.8 [67.8, 71.8]	**93.2** **[92.0, 94.2]**

– we can observe that PAC learns policies with good success rates while all BC baseline are barely able to match the average performance of the data collecting policy. This highlights that our method fulfills one of the main promises of offline RL: it can learn successful policies even from severely sub-optimal data. We provide an additional comparison to a Q-transformer (Chebotar et al., 2023) baseline (which is non-trivial to scale and does not outperform BC) in the Appendix D.2.3. Additional ablations regarding the architectural choices are also presented in the appendix.

4.3. RL Fine-tuning and Self-improvement

We now demonstrate how PAC's built-in critic can be leveraged to transition into different finetuning scenarios and use this to 'master' a target task (i.e. success rate > 90%). For this we replicate the 5-DoF object stacking scenario of (Lee et al., 2021) on a Rethink Sawyer arm in the real world.

Initially, we deploy different PAC models which have been pre-trained for 3M steps on the full data mix from Section 4.2. The best of these models (α-PAC) achieves a success rate of 69.8% which far exceeds what is learnable from this data with BC (see Table 2). Additionally we verify that we can change the value of α during training, by first training with $\alpha = 1$ for 3M steps (cf. BC+Q inTable 2) followed by 3M steps with $\alpha = 0$, which achieves 61.9% $[60.0, 63.8]$, in line with the α-PAC result. That demonstrates that we can safely transition from BC to RL at any point during the training process.

Next, we follow the iterative improvement protocol of Lampe et al. (2023) and collect the evaluation trials in an additional dataset. Afterwards, we add this data to the data mix (retaining all initial data used in previous sections) and train the model for another $\approx 250k$ steps. We repeat this process multiple times, each time adding more data. This cycle of feeding back self-generated data to the offline RL optimization provides a significant performance boost, increasing the success rate in each round, until eventually reaching a near-mastery level of 93.2%. Average scores for each round, and the number of episodes collected for self-improvement, are summarized in Table 3. More detailed scores across the sub-tasks can be found in Appendix F.2.

Finally, we repeat this self-improvement experiment for all Control Suite tasks, adding 10,000 episodes per task and performing RL finetuning for $500k$ steps starting from the

Table 3. Success rates with Wilson score intervals for $\alpha_W = 0.05$ for CHEF:real tasks (400 trials per task) across self-improvement iterations, as well as number of additional episodes collected for each iteration. Rates are reported for the most challenging object flipping task ('set 2'), and the average across all test object sets ($\#(\mathcal{T}) = 5$).

Iteration	**Episodes**	**Flipping**	**CHEF:real**
α-PAC	330k	53.5 [48.6, 58.3]	69.8 [67.8, 71.8]
RLFT #1	+ 110k	66.8 [62.0, 71.2]	84.7 [83.1, 86.2]
RLFT #2	+ 75k	76.2 [71.8, 80.2]	89.8 [88.4, 91.1]
RLFT #3	+ 11k	91.5 [88.4, 93.9]	93.2 [92.0, 94.2]

checkpoint after three rounds of RLFT in Table 3. This results in an increase to 94.3% $[91.3, 97.3]$, up from 92.1% achieved by α-PAC.

The fine-tuning experiments highlight that PAC both outperforms BC on this challenging, low-quality data and can hill-climb its performance towards mastery using self-generated data – a feat that is only possible with an RL style self-improvement loop. Interestingly, even after mastering the CHEF:real domain, α-PAC's performance on the other domains does not decline as a side-effect (cf. Table 20 in the Appendix). It is also worth noting that the L-sized version of PAC runs at 20 Hz on a local Nvidia RTX 3090 GPU during this real-robot experiment.

5. Discussion

In this work, we demonstrated that offline actor-critic methods can scale to large models of up to 1B parameters and learn a wide variety of 132 control and robotics tasks. On these tasks, our RL-trained models outperform strong BC baselines, especially in the presence of sub-optimal training data. Our finetuning experiments also showed that RL can be effectively applied after pre-training without any model changes, which enabled the mastery of a real robot task improving from a 70% to a 90% success rate using RL and autonomously collected data. The scaling analysis provides insights into the optimal model sizes and training durations for our datasets and indicates that the performance of offline RL scales better with compute than pure BC. Finally, our system allows for a gradual and stable transition between BC and RL learning, and can process data of various modalities simultaneously, while remaining efficient enough to allow our biggest model to control a real robot at 20 Hz.

However, our work also has some limitations: First, offline RL requires reward annotations, which can be costly. Progress in the development of universal reward functions (Du et al., 2023) or unsupervised reward labeling (Chebotar et al., 2021) could therefore greatly broaden the applicability of our method. Second, given the wide variety of domains considered, we saw no strong indications of transfer across tasks. However, we would expect generalization to improve with the use of datasets which have more overlap between tasks and domains as in Zitkovich et al. (2023).

Overall, we believe our work could pave the way for training large models via offline actor-critic methods on ever larger datasets of robot data. Additionally, an exciting opportunity lies in further scaling offline actor-critic learning to models of multiple billion parameters, and combining our systems with pre-trained VLMs, or even exploring offline actor-critic RL as an alternative method for generative pre-training in language models.

Acknowledgements

We would like to thank Konstantinos Bousmalis, Francesco Nori and the members of the DeepMind Control team for feedback on early versions of this manuscript. We would like to thank the greater robotics community at Google DeepMind team for their feedback and contributions with respect to software and robot hardware support. Finally, we thank the anonymous reviewers for their valuable feedback.

Impact Statement

This work presents new methods for training generalist agents for control applications including robotic manipulation. The general impact on society from generalist robotics agents at this point is not well understood, and we encourage further work into their risks and benefits. We emphasize that both our model and the actor-critic methods introduced for training at scale are for research use only and are not currently deployed in any production scenario to any users, and we thus expect no direct impact resulting from this work.

In a broader sense, Perceiver-Actor-Critic shares the majority of safety concerns discussed in Gato (Reed et al., 2022) and RoboCat (Bousmalis et al., 2023). In particular, our self-improvement loop has the same safety concerns attached to the BC-style self improvement in Bousmalis et al. (2023). It is worth emphasising that our improvement step is carried out offline from human defined reward functions, and no learning happens while interacting with any real world system. Additionally, in some sense the fact that we use rewards to 'shape' the behaviour of the learned policies makes work on safety via value alignment to human preferences (Russell, 2019; Christiano et al., 2017) more directly applicable although much work still remains to be done on this front.

References

Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., and Riedmiller, M. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations*, 2018.

Abdolmaleki, A., Huang, S., Vezzani, G., Shahriari, B., Springenberg, J. T., Mishra, S., Tirumala, D., Byravan, A., Bousmalis, K., György, A., et al. On multi-objective policy optimization as a tool for reinforcement learning: Case studies in offline rl and finetuning, 2022. URL `https://openreview.net/forum?id=bilHNPhT6-`.

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736, 2022.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., and Zaremba, W. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pp. 449–458. PMLR, 2017.

Bousmalis, K., Vezzani, G., Rao, D., Devin, C., Lee, A. X., Bauza, M., Davchev, T., Zhou, Y., Gupta, A., Raju, A., et al. Robocat: A self-improving foundation agent for robotic manipulation. *arXiv preprint arXiv:2306.11706*, 2023.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Dabis, J., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jackson, T., Jesmonth, S., Joshi, N., Julian, R., Kalashnikov, D., Kuang, Y., Leal, I., Lee, K.-H., Levine, S., Lu, Y., Malla, U., Manjunath, D., Mordatch, I., Nachum, O., Parada, C., Peralta, J., Perez, E., Pertsch, K., Quiambao, J., Rao, K., Ryoo, M., Salazar, G., Sanketi, P., Sayed, K., Singh, J., Sontakke, S., Stone, A., Tan, C., Tran, H., Vanhoucke, V., Vega, S., Vuong, Q., Xia, F., Xiao, T., Xu, P., Xu, S., Yu, T., and Zitkovich, B. Rt-1: Robotics transformer for real-world control at scale. In *arXiv preprint arXiv:2212.06817*, 2022.

Chebotar, Y., Hausman, K., Lu, Y., Xiao, T., Kalashnikov, D., Varley, J., Irpan, A., Eysenbach, B., Julian, R., Finn, C., et al. Actionable models: Unsupervised offline reinforcement learning of robotic skills. *arXiv preprint arXiv:2104.07749*, 2021.

Chebotar, Y., Vuong, Q., Hausman, K., Xia, F., Lu, Y., Irpan, A., Kumar, A., Yu, T., Herzog, A., Pertsch, K., et al. Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions. In *7th Annual Conference on Robot Learning*, 2023.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf.

Du, Y., Konyushkova, K., Denil, M., Raju, A., Landon, J., Hill, F., de Freitas, N., and Cabi, S. Vision-language models as success detectors. *arXiv preprint arXiv:2303.07280*, 2023.

Fujimoto, S. and Gu, S. S. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

Fujimoto, S., Meger, D., and Precup, D. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pp. 2052–2062. PMLR, 2019.

Fujimoto, S., Meger, D., Precup, D., Nachum, O., and Gu, S. S. Why should i trust you, bellman? the bellman error is a poor replacement for value error. *arXiv preprint arXiv:2201.12417*, 2022.

Hansen, N., Su, H., and Wang, X. Td-mpc2: Scalable, robust world models for continuous control. *arXiv preprint arXiv:2310.16828*, 2023.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Henighan, T., Kaplan, J., Katz, M., Chen, M., Hesse, C., Jackson, J., Jun, H., Brown, T. B., Dhariwal, P., Gray, S., et al. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*, 2020.

Hilton, J., Tang, J., and Schulman, J. Scaling laws for single-agent reinforcement learning. *arXiv preprint arXiv:2301.13442*, 2023.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Jaegle, A., Gimeno, F., Brock, A., Vinyals, O., Zisserman, A., and Carreira, J. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pp. 4651–4664. PMLR, 2021.

Jaegle, A., Borgeaud, S., Alayrac, J.-B., Doersch, C., Ionescu, C., Ding, D., Koppula, S., Zoran, D., Brock, A., Shelhamer, E., Henaff, O. J., Botvinick, M., Zisserman, A., Vinyals, O., and Carreira, J. Perceiver IO: A general architecture for structured inputs & outputs. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=fILj7WpI-g.

Janner, M., Li, Q., and Levine, S. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.

Kang, M., Zhu, J.-Y., Zhang, R., Park, J., Shechtman, E., Paris, S., and Park, T. Scaling up gans for text-to-image synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

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

Kostrikov, I., Nair, A., and Levine, S. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

Kudo, T. and Richardson, J. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Kumar, A., Agarwal, R., Geng, X., Tucker, G., and Levine, S. Offline q-learning on diverse multi-task data both scales and generalizes. *arXiv preprint arXiv:2211.15144*, 2022a.

Kumar, A., Singh, A., Ebert, F., Nakamoto, M., Yang, Y., Finn, C., and Levine, S. Pre-training for robots: Offline rl enables learning new tasks from a handful of trials. *arXiv preprint arXiv:2210.05178*, 2022b.

Lampe, T., Abdolmaleki, A., Bechtle, S., Huang, S. H., Springenberg, J. T., Bloesch, M., Groth, O., Hafner, R., Hertweck, T., Neunert, M., Wulfmeier, M., Zhang, J., Nori, F., Heess, N., and Riedmiller, M. Mastering stacking of diverse shapes with large-scale iterative reinforcement learning on real robots, 2023.

Lange, S., Gabel, T., and Riedmiller, M. Batch reinforcement learning. In *Reinforcement learning: State-of-the-art*, pp. 45–73. Springer, 2012.

Lee, A. X., Devin, C., Zhou, Y., Lampe, T., Bousmalis, K., Springenberg, J. T., Byravan, A., Abdolmaleki, A., Gileadi, N., Khosid, D., Fantacci, C., Chen, J. E., Raju, A., Jeong, R., Neunert, M., Laurens, A., Saliceti, S., Casarini, F., Riedmiller, M., Hadsell, R., and Nori, F. Beyond pick-and-place: Tackling robotic stacking of diverse shapes. *arXiv preprint arXiv:2110.06192*, 2021.

Lee, K.-H., Nachum, O., Yang, M. S., Lee, L., Freeman, D., Guadarrama, S., Fischer, I., Xu, W., Jang, E., Michalewski, H., et al. Multi-game decision transformers. *Advances in Neural Information Processing Systems*, 35: 27921–27936, 2022.

Levine, S., Kumar, A., Tucker, G., and Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Neumann, O. and Gros, C. Scaling laws for a multi-agent reinforcement learning model. *arXiv preprint arXiv:2210.00849*, 2022.

Octo Model Team, Ghosh, D., Walke, H., Pertsch, K., Black, K., Mees, O., Dasari, S., Hejna, J., Xu, C., Luo, J., Kreiman, T., Tan, Y., Sadigh, D., Finn, C., and Levine, S. Octo: An open-source generalist robot policy. https://octo-models.github.io, 2023.

Open X-Embodiment Collaboration. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

OpenAI. Gpt-4 technical report, 2023.

Peng, X. B., Kumar, A., Zhang, G., and Levine, S. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-maron, G., Giménez, M., Sulsky, Y., Kay, J., Springenberg, J. T., et al. A generalist agent. *Transactions on Machine Learning Research*, 2022.

Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Wiele, T., Mnih, V., Heess, N., and Springenberg, J. T. Learning by playing solving sparse reward tasks from scratch. In *International conference on machine learning*, pp. 4344–4353. PMLR, 2018.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models, 2021.

Russell, S. *Human compatible: Artificial intelligence and the problem of control*. Penguin, 2019.

Seyde, T., Werner, P., Schwarting, W., Gilitschenski, I., Riedmiller, M., Rus, D., and Wulfmeier, M. Solving continuous control via q-learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=U5XOGxAgccS.

Shridhar, M., Manuelli, L., and Fox, D. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Conference on Robot Learning*, pp. 785–799. PMLR, 2023.

Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., Heess, N., and Tassa, Y. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. ISSN 2665-9638. doi: https://doi.org/10.1016/j.simpa.2020.100022. URL https://www.sciencedirect.com/science/article/pii/S2665963820300099.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, Z., Novikov, A., Zolna, K., Merel, J. S., Springenberg, J. T., Reed, S. E., Shahriari, B., Siegel, N., Gulcehre, C., Heess, N., et al. Critic regularized regression. *Advances in Neural Information Processing Systems*, 33: 7768–7778, 2020.

Wu, Y., Tucker, G., and Nachum, O. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

Ze, Y., Yan, G., Wu, Y.-H., Macaluso, A., Ge, Y., Ye, J., Hansen, N., Li, L. E., and Wang, X. Gnfactor: Multi-task real robot learning with generalizable neural feature fields. *arXiv preprint arXiv:2308.16891*, 2023.

Zitkovich, B., Yu, T., Xu, S., Xu, P., Xiao, T., Xia, F., Wu, J., Wohlhart, P., Welker, S., Wahid, A., et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *7th Annual Conference on Robot Learning*, 2023.

Appendix

This appendix presents further details and additional experiments of the proposed Perceiver-Actor-Critic (PAC) model and is arranged as follows: Methodological details are discussed in Sections A and B, with Section A focusing on algorithmic details to complement Section 3.2, and Section B discussing architecture details accompanying Section 3.3. Then further details on the experimental setups are given in Appendix C, following which are algorithm-wise sensitivity analysis and architecture-wise ablation studies in Section D. Finally, additional experiments are presented in Appendix F.

A. Method Details - Algorithm

A.1. Necessary Optimality Conditions for Policy

We give a short derivation following previous work by Abdolmaleki et al. (2018). The solution to Equation (1) has to be optimal for every state s and task τ and we will thus drop these dependencies for simplicity but add a Lagrangian because the policy is a probability distribution:

$$\begin{aligned}\pi_{\text{imp}} &= \arg\max_{\pi} J(\pi) \\ &= \arg\max_{\pi} \mathop{\mathbb{E}}_{\pi} Q(a) - \eta\, \mathrm{D}_{\mathrm{KL}}[\pi, \tilde{\pi}] + \lambda(1 - \mathop{\mathbb{E}}_{\pi} 1).\end{aligned} \tag{5}$$

We can now compute the partial derivative w.r.t. π and set it to zero:

$$\partial J / \partial \pi = Q(a) - \eta(\log(\pi(a)/\tilde{\pi}(a)) + 1) - \lambda = 0 \quad \forall a \tag{6}$$

After exponentiation this can be re-arranged into:

$$\exp(Q(a)/\eta)\exp(1 + \lambda/\eta) = \pi(a)/\tilde{\pi}(a) \tag{7}$$

Since λ is a constant and takes on the role of a normalizing constant we can write:

$$\pi(a) \sim \exp(Q(a)/\eta)\tilde{\pi}(a) \tag{8}$$

And Equation (2) can be retrieved by re-introducing the dependencies on states s and task τ.

A.2. PAC+Q Details

Our default implementation of PAC (called PAC+Q here for clarity) trains the policy π_θ together with an estimate $Q_\theta \approx Q^{\pi_\theta} \approx Q^{\pi_{\text{imp}}}$ of the state-action value function. To enable us to balance losses easily we transform the problem of learning Q_θ into minimizing some negative log likelihood of a categorical distribution. This is possible by employing tools from the distributional reinforcement learning literature: Instead of parameterizing Q_θ directly, we learn a discretized representation: a categorical distribution $p_\theta(q|s_t, a_t, \tau)$ over binned values $q \in \{q_{\min}, q_{\min} + \epsilon, q_{\min} + 2\epsilon, \dots, q_{\max}\}$ with bin size ϵ, giving rise to the Q-value $Q_\theta(s_t, a_t, \tau) = \mathbb{E}_{p_\theta}\, q$.

We use the definition of the distributional TD operator[4] following Bellemare et al. (2017) to compute a target Q-distribution as:

$$\begin{aligned}&\Gamma_\theta(q \mid s_t, a_t, \tau) = \\ &\mathop{\mathbb{E}}_{s_{t+1}} \mathop{\mathbb{E}}_{\substack{a' \sim \\ \pi_\theta(\cdot|s_{t+1},\tau)}} \mathop{\mathbb{E}}_{\substack{q' \sim \\ p_\theta(\cdot|s_{t+1},a',\tau)}} \left[\mathbf{1}_{[q_t - \epsilon/2, q_t + \epsilon/2]}(r_t + \gamma q')\right],\end{aligned} \tag{9}$$

where the indicator function $\mathbf{1}$ is used to map the probability mass of the transformed Q-value to the nearest target bin. We can now compute the KL-divergence of our estimate against this target

$$\mathrm{D}_{\mathrm{KL}}[\Gamma_{\theta'}(q|s_t, a_t, \tau), p_\theta(q|s_t, a_t, \tau)], \tag{10}$$

based on transitions $(s_t, a_t, s_{t+1}, \tau) \in \mathcal{D}$ and where θ' refers to the parameters of a target network which we periodically update to a time-lagged version of model parameters θ.

The above target network also gives rise to a target policy $\pi_{\theta'}$ and this is what we use as reference policy for the Q-value based maximisation of Equation (2), i.e. $\tilde{\pi} = \pi_{\theta'}$. Combining the policy loss, a BC loss, and the Q-Value loss into a total loss yields three KL terms:

$$\begin{aligned}L^Q(\theta) = &\mathop{\mathbb{E}}_{\mathcal{D}}\Big[(1 - \alpha)\, \mathrm{D}_{\mathrm{KL}}[\pi_{\text{imp}}, \pi_\theta | s_t, \tau, \tilde{\pi} = \pi_{\theta'}] \\ &+ \alpha\, \mathrm{D}_{\mathrm{KL}}[b, \pi_\theta | s_t, \tau] \\ &+ \beta\, \mathrm{D}_{\mathrm{KL}}[\Gamma_{\theta'}(q|s_t, a_t, \tau), p_\theta(q|s_t, a_t, \tau)]\Big] \\ = -&\mathop{\mathbb{E}}_{\mathcal{D}}\Big[(1 - \alpha) \mathop{\mathbb{E}}_{a' \sim \pi_{\theta'}}\left[e^{Q_{\theta'}(s_t, a', \tau)/\eta - K_Q} \log \pi_\theta(a'|s_t, \tau)\right] \\ &+ \alpha \log \pi_\theta(a_t|s_t, \tau) \\ &+ \beta \mathop{\mathbb{E}}_{q \sim \Gamma_{\theta'}} \log p_\theta(q|s_t, a_t, \tau)\Big] + K_H,\end{aligned} \tag{11}$$

where $K_Q = \log \mathbb{E}_{a' \sim \pi_{\theta'}}[\exp(Q_{\theta'}(s_t, a', \tau)/\eta)]$ is a normalizing constant, K_H a constant entropy related offset independent of θ, the expectation over the data is estimated by sampling $(s_t, a_t, s_{t+1}, \tau) \in \mathcal{D}$, the expectation over action samples from $\pi_{\theta'}$ is estimated based on $N = 10$ samples and the expectation $\mathbb{E}_{q \sim \Gamma_{\theta'}}$ can be evaluated analytically. Finally α and β are multipliers trading off different loss components (which are relatively easy to set due to all losses corresponding to categorical log likelihoods).

A.3. PAC+V Details

Our alternative implementation of PAC uses a state-value function and is called PAC+V here for clarity. It directly uses the data generating behavior policy as the reference, i.e.

[4] Analogous to the standard temporal difference operator using the relation $Q^\pi(s_t, a_t) = r_t + \gamma E_\pi[Q^\pi(s_{t+1}, a_{t+1})]$.

$\tilde{\pi} = b$, thus simplifying the losses. Instead of estimating Q^π we can thus rely on an empirical estimate of the advantage $A^\pi(s_t, a_t, \tau) \approx A_\theta(s_t, a_t, \tau) = r_t + \gamma V_\theta(s_{t+1}, \tau) - V_\theta(s_t, \tau)$ using samples $(s_t, a_t, s_{t+1}, \tau) \in \mathcal{D}$ from which we learn a V-function estimate $V_\theta \approx V^\pi$ only As an aside: this also eliminates the need for a target network.

Analogously to the Q-function we use a categorical distribution $p_\theta(v|s_t, \tau)$ over binned values $v \in \{v_{\text{min}}, v_{\text{min}} + \epsilon, v_{\text{min}} + 2\epsilon, \dots, v_{\text{max}}\}$ (note that this is now not conditioned on actions a) yielding the value estimate $V_\theta(s_t, \tau) = \mathbb{E}_{p_\theta} v$. We can again compute target Q-values using a distributional TD operator:

$$\Gamma_\theta(q|s_t, a_t, \tau) = \mathop{\mathbb{E}}_{s_{t+1}} \mathop{\mathbb{E}}_{v' \sim p_\theta(\cdot|s_{t+1}, \tau)} \left[\mathbf{1}_{[q - \epsilon/2, q + \epsilon/2]}(r_t + \gamma v') \right]. \tag{12}$$

In order to retrieve target values we can take the expectation over our policy. However, given that we only have access to transitions from a behavior policy we require an importance weighting based correction:

$$\Gamma_\theta(v|s_t, \tau) = \mathop{\mathbb{E}}_{a_t \sim b(\cdot|s_t, \tau)} \frac{\pi_\theta(a_t|s_t, \tau)}{b(a_t|s_t, \tau)} \Gamma_\theta(q|s_t, a_t, \tau). \tag{13}$$

Using the second definition from Equation (2) and choosing the reference to be the behavior policy b as mentioned above we can then define the total loss for the V-function based actor-critic via the two KL terms:

$$\begin{aligned} L^V(\theta) &= \mathop{\mathbb{E}}_{\mathcal{D}} \Big[\mathrm{D}_{\mathrm{KL}}[\pi_{\text{imp}}, \pi_\theta | s_t, \tau, \tilde{\pi} = b] \\ &\qquad + \beta \, \mathrm{D}_{\mathrm{KL}}[\Gamma_{\theta'}(v|s_t, \tau), p_\theta(v|s_t, \tau)] \Big] \\ &= -\mathop{\mathbb{E}}_{\mathcal{D}} \Big[\exp\left((r_t + \gamma V_{\theta'}(s_{t+1}, \tau) - V_{\theta'}(s_t, \tau)) / \eta \right) \log \pi_\theta(a_t|s_t, \tau) \\ &\quad + \beta \frac{\pi_\theta(a_t|s_t, \tau)}{b(a_t|s_t, \tau)} \mathop{\mathbb{E}}_{v \sim \Gamma_{\theta'}(\cdot|s_t, a_t, \tau)} \log p_\theta(v|s_t, \tau) \Big] + K_H, \end{aligned} \tag{14}$$

where K_H is again a constant entropy related offset and where we dropped the normalization constant for the exponentiated advantage as is common (see e.g. Peng et al. (2019)). Even though we do not use target networks here, we keep the notation θ' to indicate that we do not take gradient w.r.t. the corresponding term. Given that the behavior policy is used as reference for the policy improvement we do not need to include an additional BC term here, using only observed actions is enough to ensure stable learning without overestimation issues – and this improvement step reverts to BC for $\eta \to \infty$. The only additional complication in this equation is that we need to estimate $b(a_t|s_t, \tau)$ to compute importance weights. Two simple strategies are possible for this: we could either assume the behaviour policy executed actions from a fixed set, i.e. $b(a_t|s_t, \tau) = \text{constant}$ in which case it can be dropped, or we can learn an estimate of b via maximum likelihood (BC).

B. Method Details - Architecture

In the following we discuss architectural details of the proposed approach. The complete procedure of how the input observations are processed and how the policy logits and Q-value logits are calculated is outlined in Algorithm 1, with detailed shape annotations for all variables; all hyperparameters noted in the algorithm are listed in Table 5. Each sub-procedure outlined in Algorithm 1 is also visually grouped together in Figure 5.

B.1. Multi-scale Normalizer

We propose *multi-scale* normalizer that maps an input float of an arbitrary scale to an N_G-dimensional normalized representation: $\phi^{\text{multi-scale}} : \mathbb{R} \to [-1, 1]^{N_G}$. It does so by employing N_G fixed gains across a logarithmic scale (e.g. $\sigma = [10^{-4}, 10^{-3}, \dots, 10^2, 10^3]$ for $N_G = 8$) to generate $\tanh$-bounded embeddings of the form:

$$\phi^{\text{multi-scale}}(x) = [\tanh(\sigma_1 x), \dots, \tanh(\sigma_{N_G} x)]. \tag{15}$$

For each input value the multi-scale normalizer thus generates a multi-dimensional representation vector using gains on a logarithmic scale. Following this, the attention mechanism, for which the multi-scale normalizer is specifically designed to pair with, can determine the most suitable gain levels to attend to. This is in contrast to the commonly used μ-Law scaling (e.g. in Reed et al. (2022)) for encoding continuous values which is at risk of either insufficient scaling or saturation if the input data is not within a known range.

B.2. Observation Encoding

As introduced in Section 3.1, the state s and the task description τ can both be composed of different modalities: proprioceptive measurements (P), visual observations (V) and language descriptions (L), all of possibly domain-dependent variable sizes; and the actions (A) are treated as just another input modality. Those input modalities are indicated by superscripts in notations. Note that we assume, without loss of generality, that the task description τ is episode-dependent and thus is denoted without a subscript indicating time step. Also note that the task description for an offline episode can be sampled and relabeled in hindsight (Andrychowicz et al., 2017; Riedmiller et al., 2018). So, we make the dependence on τ explicit in notations. In our experiments we allow for vision- and language-based task descriptions.

While easily extendable to other input data modalities, our current format accommodates proprioceptive and visual observations (s^P, s^V) as well as visual and language task

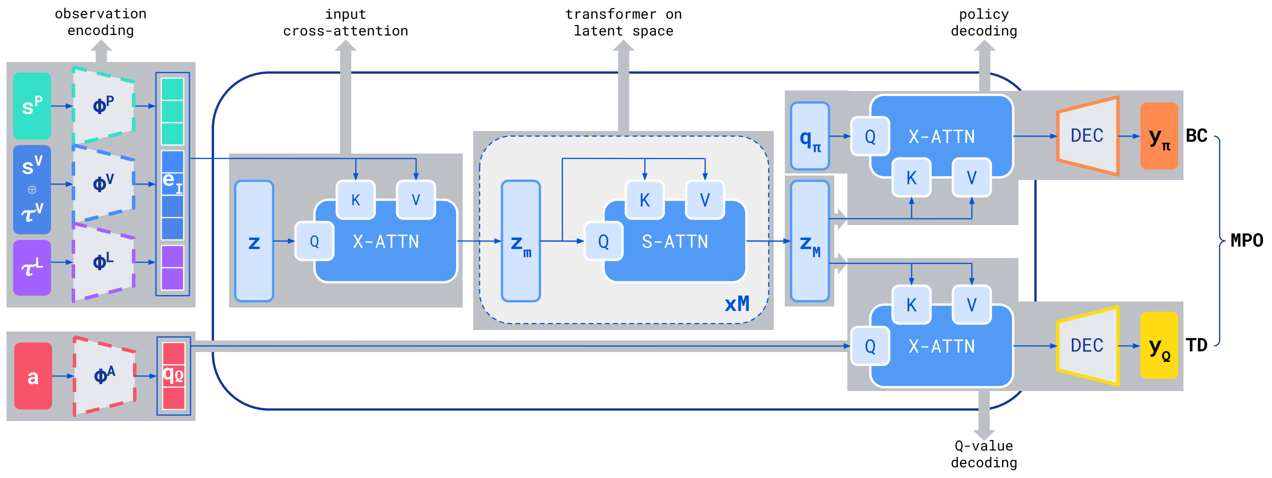


Figure 5. High-level PAC model architecture, with each sub-procedure outlined in Algorithm 1 visually grouped; note that z_M is used in both policy decoding and Q-value decoding. Details are discussed in Section B.

descriptions (τ^V, τ^L):

- Proprioception observations $s^P \in \mathbb{R}^{N^P}$ with N^P denoting the dimensions of proprioceptive measurements.
- Visual observations $s^V \in \mathbb{R}^{N^V \times H \times W \times C}$ with number of image observations N^V, height H, width W, and number of channels C for each image.
- Visual task descriptions $\tau^V \in \mathbb{R}^{N^V_\tau \times H \times W \times C}$ with N^V_τ number of images depicting the desired task (e.g., the last frame from a successful episode).
- Language task descriptions $\tau^L \in \mathbb{R}^{N^L_\tau}$ with number of text tokens N^L_τ.

All input modalities across all domains are padded in order to match this format and we keep track of the valid entries with a mask. The complete input at time step t is therefore the concatenation of $(s_t = (s^P_t, s^V_t), \tau = (\tau^V, \tau^L))$. This could also be easily extended to incorporate more than a single timestep in case of partially observable tasks.

The concatenated input is then mapped via one encoder per modality into multiple embedding vectors of dimension D_I. The details of each modality encoder are given below. The output of this mapping will be fed into the perceiver encoder.

The proprioception encoder ϕ^P is an instantiation of the multi-scale normalizer (see Appendix B.1) followed by a linear layer L^P with D_I output dimensions to map the multi-scale representation to the desired shape: $\phi^P = L^P \cdot \phi^{\text{multi-scale}}_{N_G} : \mathbb{R} \to \mathbb{R}^{N_G \times D_I}$.

The image inputs are encoded via a ResNet (He et al., 2016). We omit the last projection layer to obtain N_E spatial dimensions for our image embedding $\phi^V : \mathbb{R}^{H \times W \times C} \to \mathbb{R}^{N_E \times D_I}$, where N_E depends on the input image size and the down-sampling specifications of the ResNet. We note that just like the proprioception embedding, and in contrast to other approaches, we do not discretize or tokenize image inputs but use continuous mappings instead.

We assume language inputs to be tokenized by the SentencePiece tokenizer (Kudo & Richardson, 2018) during data loading. These are then directly embedded using a learnable look-up table $\phi^L : [1..N_T] \to \mathbb{R}^{D_I}$, with N_T the total number of different language tokens.

Applying each encoder to the corresponding input modality thus generates an encoded input, $e_I = \phi^P(s^P_t) \oplus \phi^V(s^V_t) \oplus \phi^V(\tau^V) \oplus \phi^L(\tau^L) \in \mathbb{R}^{N \times D_I}$, with $N = N^P N_G + N^V N_E + N^V_\tau N_E + N^L_\tau$.

B.3. Transformer on Latent Space

Most state-of-the-art large-scale models for control take the encoded input $e_I \in \mathbb{R}^{N \times D_I}$ and directly feed it into a transformer consisting of multiple stacked self-attention blocks. For the domains in our experiments and the data modalities that we consider, this input sequence would be of length $N = 2,634$ for a single time step (cf. Appendix C.3). As the computational complexity and memory usage of the self-attention mechanism scales quadratically with the input sequence length, this commonly adopted treatment potentially hinders the learned generalist controller from being applicable to real robotic systems that impose real-time constraints, and more generally restricts efficient learning and inference, let alone providing feasible solutions to extend

Algorithm 1 Perceiver Actor Critic Model

```
Input:
$s^P \in \mathbb{R}^{N^P}$: proprioceptive observations;
$s^V \in \mathbb{R}^{N^V \times H \times W \times C}$: visual observations;
$\tau^V \in \mathbb{R}^{N^V_\tau \times H \times W \times C}$: visual task descriptions;
$\tau^L \in \mathbb{R}^{N^L_\tau}$: language task descriptions;
$a \in \mathbb{R}^{N^A}$: actions.
Trainable parameters (randomly initialized):
$\phi^P : \mathbb{R} \to \mathbb{R}^{N_G \times D_I}$, proprioception encoder;
$\phi^V : \mathbb{R}^{H \times W \times C} \to \mathbb{R}^{N_E \times D_I}$, vision encoder;
$\phi^L : [1..N_T] \to \mathbb{R}^{D_I}$, language encoder;
$z \in \mathbb{R}^{N^Z \times D^Z}$: latents;
$q_\pi \in \mathbb{R}^{N^A \times N_O}$: policy queries;
$L_\pi : \mathbb{R}^{N_O} \to \mathbb{R}^{N_B}$: final projection for policy logits;
$\phi^A : \mathbb{R}^{N^A} \to \mathbb{R}^{1 \times D_O}$: action encoder.
$L_Q : \mathbb{R}^{N_O} \to \mathbb{R}^{N_Q}$: final projection for Q-value logits.
Output:
$y_\pi \in \mathbb{R}^{N^A \times N_B}$: policy logits;
$y_Q \in \mathbb{R}^{1 \times N_Q}$: Q-value logits.

// Observation Encoding.
    $e^P = \phi^P(s^P)$                          ▷ $e^P \in \mathbb{R}^{(N^P N_G) \times D_I}$
    $e^V = \phi^V(s^V)$                          ▷ $e^V \in \mathbb{R}^{(N^V N_E) \times D_I}$
    $e^V_\tau = \phi^V(s^V_\tau)$                ▷ $e^V_\tau \in \mathbb{R}^{(N^V_\tau N_E) \times D_I}$
    $e^L_\tau = \phi^L(\tau^L)$                  ▷ $e^L_\tau \in \mathbb{R}^{N^L_\tau \times D_I}$
    $e_I = e^P \oplus e^V \oplus e^V_\tau \oplus e^L_\tau$   ▷ $e_I \in \mathbb{R}^{N \times D_I}$
// Input cross-attention.
$z_0 = \text{X-ATTN}(z, e_I)$                    ▷ $z_0 \in \mathbb{R}^{N_Z \times D_Z}$
// Transformer on latent space.
for $m = 1$ to $M$ do
    $z_m = \text{S-ATTN}(z_{m-1})$               ▷ $z_m \in \mathbb{R}^{N_Z \times D_Z}$
end for
// Policy decoding.
$\tilde{y}_\pi = \text{X-ATTN}(q_\pi, z_M)$      ▷ $\tilde{y}_\pi \in \mathbb{R}^{N^A \times N_O}$
$y_\pi = L_\pi(\tilde{y}_\pi)$                   ▷ $y_\pi \in \mathbb{R}^{N^A \times N_B}$
// Q-value decoding.
$q_Q = \phi^A(a)$                                ▷ $q_Q \in \mathbb{R}^{1 \times D_O}$
$\tilde{y}_Q = \text{X-ATTN}(q_Q, z_M)$          ▷ $\tilde{y}_Q \in \mathbb{R}^{1 \times N_O}$
$y_Q = L_Q(\tilde{y}_Q)$                         ▷ $y_Q \in \mathbb{R}^{1 \times N_Q}$
Return $y_\pi, y_Q$
```

the temporal context window beyond a few time steps or include more high-dimensional inputs. To address this challenge, we adopt the methodology from the perceiver model (Jaegle et al., 2021) to shift the operation of stacked self-attention from the input sequence onto a few trainable latent vectors, which in turn reduces the computational complexity from a quadratic dependence on the input sequence length to a linear one.

For ease of discussion, we first give an overview of the attention operator (f_{ATTN}) and the attention block (ATTN), which is the main building block of the perceiver model. The attention operator takes as inputs three matrices: the queries $Q \in \mathbb{R}^{N_Q \times D_Q}$, the keys $K \in \mathbb{R}^{N_K \times D_Q}$ and the values $V \in \mathbb{R}^{N_K \times D_V}$, where the queries and the keys match in their vector dimensionality (D_Q) while the keys and values contain the same number of vectors (N_K). The attention operator is then defined as $f_{\text{ATTN}}(Q, K, V) = \text{softmax}\left(QK^T/\sqrt{D_Q}\right) V \in \mathbb{R}^{N_Q \times D_V}$, for which the majority of the computational complexity and memory usage scale with $O\left(N_Q \cdot N_K\right)$ (the dependency of the scaling factor on the vector dimensionality is left out to keep the discussion concise). Combining the attention operator f_{ATTN} with linear projection layers, normalization layers, post processing MLPs and skip connections yields the attention block $\text{ATTN}(x_Q, x_K, x_V)$ (we refer to (Vaswani et al., 2017; Jaegle et al., 2021) for more details). Note that this most general form admits possibly three different sources of inputs to be linearly projected to act as Q, K and V respectively: $x_Q \in \mathbb{R}^{N_Q \times N_1}$, $x_K \in \mathbb{R}^{N_K \times N_2}$, $x_V \in \mathbb{R}^{N_K \times N_3}$ (there is no constraint of N_1 matching N_2 here as the three input sources will each go through a linear projection before being fed into f_{ATTN}).

Getting back at the encoded input $e_I \in \mathbb{R}^{N \times D_I}$, a standard transformer would instantiate the attention block in the form of self-attention (using the same data source for the queries as for the keys-and-values): $\text{S-ATTN}(e_I) = \text{ATTN}(x_Q = e_I, x_K = e_I, x_V = e_I)$. With e_I containing N vectors of dimentionality D_I, the computation required by the self-attention blocks will scale quadratically with the input sequence length: $O\left(N^2\right)$, which imposes challenges for meeting real-time control constraints or enabling efficient learning and inference.

We address this issue with a perceiver-style backbone. Instead of applying self-attention directly on the input sequence, N_Z learnable vectors are created with each being of size D_Z. We refer to those trainable vectors as the latents $z \in \mathbb{R}^{N_Z \times D_Z}$. The number of latent vectors is typically much smaller than the length of the input sequence: $N_Z \ll N$ (e.g., $N_Z = 32$ in our experiments). With this, we instantiate the attention block in another form, namely a cross-attention between the latents and the normal inputs: $\text{X-ATTN}(z, e_I) = \text{ATTN}(x_Q = z, x_K = e_I, x_V = e_I)$, where the latents are used as the data source for the queries and the input sequence for the keys-and-values. This cross-attention block thus scales linearly with the input sequence length: $O(N_Z \cdot N)$, and is positioned at the front-end of our PAC model (cf. Figure 2), incorporating information from the inputs into the latent vectors. Following this, a standard transformer stack composed of M self-attention blocks is applied on the latent space: $z_{m+1} = \text{S-ATTN}(z_m)$, which finally outputs an encoding $z_M \in \mathbb{R}^{N_Z \times D_Z}$. We note that the quadratic computational scale of those self-attention blocks is decoupled from the input sequence length to be $O(N_Z{}^2)$. This effectively shifts conducting the self-attention operation from the inputs to the latents, in our case reducing the computation and memory usage by a factor of $({}^{2,634}/_{32})^2 \approx 6,775$.

B.4. Policy and Value Decoding

To obtain the two types of outputs, we create dedicated query vectors for each output which are then used to query the latent encodings z_M; querying information from z_M is again implemented via cross-attention following the decoding architecture of Perceiver-IO (Jaegle et al., 2022)) and deriving keys and values from the latents z_M.

Specifically, to acquire an action prediction of shape $N^A \times N_B$, with N^A denoting the cardinality of the action space $|\mathcal{A}|$ and N_B the number of bins each element gets discretized into, we create N^A learnable policy queries $q_\pi \in \mathbb{R}^{N^A \times N_O}$ to cross-attend to z_M, X-ATTN(q_π, z_M), and get a $N^A \times N_O$ shaped output, which are then linearly projected to $N^A \times N_B$ shaped policy logits.

Whereas for the Q-value estimate, since the information about the action are not contained in the encoded latents z_M but required for getting the estimate, the Q-value queries q_Q should not be simply created as randomly initialized trainable vectors as for q_π. Instead, they are computed by encoding the action $a_t \in \mathbb{R}^{N^A}$ via an action encoder ϕ^A composed of an multi-scale normalizer ($\mathbb{R}^{N^A} \to [-1,1]^{N^A \times N_G}$, cf. Equation (15)) followed by two linear projections ($L^{A_1} : [-1,1]^{N^A \times N_G} \to \mathbb{R}^{(N^A \times N_G) \times D_O}$, $L^{A_2} : \mathbb{R}^{(N^A \times N_G) \times D_O} \to \mathbb{R}^{1 \times D_O}$): $\phi^A = L^{A_2} \cdot L^{A_1} \cdot \phi^{\text{multi-scale}}_{N_G} : \mathbb{R}^{N^A} \to \mathbb{R}^{1 \times D_O}$. The generated value query $q_Q \in \mathbb{R}^{1 \times D_O}$ contains only one vector of size D_O, since for each action the Q-value estimate only needs to output one quantity. Note that in constrast to how the observation encoders (ϕ^P, ϕ^V, ϕ^L) are introduced in Section B.2 as mappings from each individual element to the corresponding encoding, here the action encoder ϕ^A is denoted as the mapping from the full action dimension N^A to its corresponding encoding. q_Q is then used to query the latents via cross-attention, X-ATTN(q_Q, z_M), the output of which $(1 \times N_O)$ is then mapped to $1 \times N_Q$ to generate the N_Q logits. Incorporating the action information by encoding it into a query at the decoding stage instead of concatenating it along with the other observations at the input has the advantage that this way the action is less likely to be ignored by the model (a common problem encountered when learning Q-values). It also allows efficient evaluation of the Q-function for multiple action samples: the latent representation z_M is not dependent on the action and therefore needs to be computed only once to be queried by multiple action samples.

C. Experimental Details

C.1. Architecture Hyperparameters for Scaling

We vary three parameters of the architecture as detailed in Table 4: The size of the latent vectors D_Z, the number of self-attention blocks `M` and the widening factor `W` of the attention blocks which define the ratio between the residual MLPs' hidden size to input size. With all the other fixed hyperparameters reported in Appendix C.2, the resulting model sizes range from 32M parameters (comparable to RT-1 (Brohan et al., 2022) and Q-Transformer (Chebotar et al., 2023)) to 988M parameters (close to the largest versions of Gato (Reed et al., 2022) and RoboCat (Bousmalis et al., 2023).

Table 4. Hyperparameters of PAC's different model scales.

Scale	#(params)	D_Z	M	W
XXS	32M	768	4	1
XS	73M	1024	8	1
S	164M	1280	10	2
M	391M	1536	12	4
L	988M	2048	18	4

C.2. Fixed Architecture Hyperparameters

Table 5 provides an overview of all employed model parameters which are kept fixed across model scales. N^P, N^V, N^V_τ, N^L_τ, N^A all refer to input dimensions and are chosen to accommodate all tasks the data is originating from. N_E and N_T also relate to input dimensions, but depend on the pre-processing that is applied to the data (e.g. the SentencePiece tokenizer for text input, or a ResNet and for image input). N_G is the number of scales (gains) for our proposed multi-scale normalizer. Finally, N_B and N_Q refer to the number of discrete value bins used for the discretization of the action and Q-value respectively, N_Z refers to the number of latent vectors.

Table 5. Constant hyperparameters of PAC across all model scales.

Hyperparameter	**Value**
N^P: proprioception dimensions	223
N^V: image observations	5
N^V_τ: goal images	3
N^L_τ: text tokens for task descriptions	50
N^A: action dimensions	38
N_G: multi-scale normalizer scales	8
N_E: visual tokens per image	100
N_T: language token vocabulary size	32,000
N: total number of input embedding vectors	2634
N_Z: number of Perceiver latents	32
N_B: action bins	101
N_Q: Q-value bins	101
D_I: token embedding size for each modality	256

Table 6. The data mixture used for the scaling experiments. For each domain we report the modalities (P = proprioception, V = vision, L = language, A = actions), the number of tasks, the number of episodes recorded, the effective number of trajectories (each consisting of five timesteps), the number of tokens contributed, the percentage of successful episodes and the weight λ of the domain during the data sampling process. [* = success rate only available for five stacking tasks; ** = success rate average only computed across 82 tasks with available success rates]

Domain	**Mod**	$\#(\mathcal{T})$	**#(Ep)**	**#(Traj)**	**#(Tok)**	**success**	λ
Gato: Control	P, L, A	32	468k	187M	430B	47%	8
CHEF: sim	P, V, L, A	30	755k	60M	430B	28%*	2
CHEF: real	P, V, L, A	30	2M	169M	1.12T	12%*	2
RC: Tower	P, V, L, A	7	121k	19M	122B	72%	1
RC: Pyramid	P, V, L, A	30	194k	31M	195B	52%	1
RC: Insertion	P, V, L, A	3	100k	40M	154B	97%	2
	$\sum =$	132	3.638M	506M	2.45T	avg = 42%**	

C.3. Data Modalities and Tokenization

Images At each timestep, PAC processes up $N^V + N^V_\tau$ input images. For domains which provide fewer observations or goal images, these inputs are zero-padded and masked out for the subsequent processing. Across all our experiments, we use an image resolution of 80 × 80 pixels. Each image is encoded using a simple ResNet (He et al., 2016) with three blocks using (32, 64, 64) channels, 3 × 3 pooling kernels and 2 × 2 striding. After the convolutions, each image has been downsampled to 10 × 10 'image tokens' which are projected into an embedding space of D_I dimensions. Under this embedding scheme, each image is counted as 100 tokens in our experiments. Our default implementation uses eight image observations at each timestep resulting in the processing of 800 image tokens.

Proprioception and Actions Our *multi-scale normalizer* (cf. Appendix B.1) represents each floating point number of the proprioceptive reading as N_G tokens. Each of these tokens is then projected into an embedding space of D_I dimensions. Our default implementation uses 223 proprioception observations at each timestep resulting in the processing of 1,784 proprioception tokens. In domains with fewer proprioception observations, the remaining inputs are zero-padded and masked out for the subsequent processing.

Language In order to differentiate between different tasks in the same domain, e.g. to tell the model whether to execute a `run`, `stand` or `walk` policy in Control Suite's `humanoid` domain, we feed a simplified task instruction as a language task description (τ^L) at each timestep. For each dataset, the task instruction is constructed from the dataset name (cf. Table 9) and looks like `humanoid.walk` or `sim.insert.large_gear`. We use the SentencePiece tokenizer (Kudo & Richardson, 2018) to tokenize the task instruction where each token represents an integer index into a vocabulary of N_T tokens. Each language token is subsequently projected into an embedding space of D_I dimensions. In our default implementation, we use at most 50 language tokens for the task instruction and zero-pad and mask language tokens in cases of shorter instructions.

In summary, our default implementation processes a total amount of 2,634 input tokens per timestep broken down into: 800 image tokens, 1,784 proprioception tokens and 50 text tokens.

C.4. Data Mixtures

During our scaling experiments (cf. Section 4.1), we use a data mixture consisting of 51 datasets and depict a selection of them in Figure 6. We partition the datasets into six groups as outlined in Table 6. When sampling trajectories during training, we first sample uniformly across all groups with a weight of λ and then sample uniformly within the group to draw a trajectory sample from a concrete dataset. The effective sampling ratio for each dataset is shown as P_{eff} in Table 9. To maintain the sampling ratios over the entire training process, we simply loop all underlying datasets deterministically such that none of them is ever exhausted.

During our large-scale pre-training experiments (cf. Section 4.2), we augment the data mixture already used in the scaling experiments (see the previous section), but add the full amount of RoboCat data for the Tower and Pyramid domains from Bousmalis et al. (2023) to enable a fair comparison with prior work. The adjusted overall data mixture is shown in Table 10 and the changes to the individual datasets are reported in Table 11.

C.5. Optimization Hyperparameters

For all large-scale experiments (cf. Sections 4.1 and 4.2) we use optimizer hyperparameters as reported in Table 12. Importantly, we use the AdamW optimizer (Loshchilov & Hutter, 2017) with a learning rate schedule which starts at `lr_init`, ramps up linearly for `lr_warmup_steps` to `lr_peak` and is then cosine-annealed to `lr_end` over `lr_decay_steps` which amounts to approximately one epoch in our data mix. In line with the protocol of Hoff-

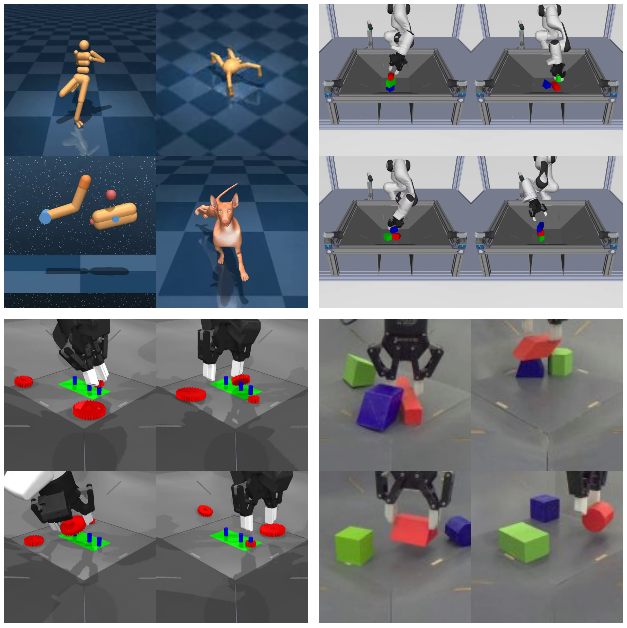


Figure 6. A selection of the domains and tasks in our data mix. **Top left:** Control Suite features 32 different continuous control tasks across 15 different embodiments with a great variance in proprioception and action spaces. **Top right:** Stacking RGB objects into different configurations (pyramides and towers) with a simulated Panda arm. **Bottom left:** Inserting gears onto pegs in simulation. **Bottom right:** Performing the RGB stacking task on a real Sawyer robot.

mann et al. (2022), we decay the learning rate by one order of magnitude over one epoch. We set `lr_decay_steps` according to the respective epoch lengths for each of our experiments, i.e. to 2.7e6 in the scaling experiments and to 4.7e6 in the pre-training experiments.

For all `PAC` models, we keep the TD loss scale β constant at 38 while varying the BC vs RL trade-off α between 1.0 for our `BC+Q` and `FilteredBC` baselines and 0.75 for the `PAC` model series. α-`PAC` sets $\beta = 19$ and α of 0.75 for the Control Suite and RoboCat data (i.e. leaning more towards BC given the high average success rates in the data) and $\beta = 1,900, \alpha = 0.0$ for the CHEF datasets (i.e. relying fully on the RL term in the presence of highly sub-optimal data). All `PAC+V` models use $\alpha = 0.0$, $\beta = 38$ and a temperature τ of 1e-4.

D. Sensitivity and Ablation Experiments

D.1. Sensitivity to Hyperparameters

Here we report additional experiments that demonstrate the influence of some of the hyperparameters and that informed the settings that we used in other experiments.

D.1.1. BC Loss Scale α

In order to perform offline reinforcement learning (RL), we simply interpolate between the behavioral cloning (BC) loss and the RL loss via a parameter α. We found that, depending on the nature of the data, different parameters work best for different datasets. Empirically, we observed that the more 'expert data' there is (i.e. a higher base success rate of the tasks in the dataset), the higher the contribution of the BC loss should be. Conversely, less expert data can benefit more from RL. As seen in Table 7, for the control suite, averaged over 32 tasks, setting α at around 0.8 works best. This is because most of the data has been collected by expert agents and therefore small divergence from BC distribution can gain improvement over the BC. This value is different for, for example, CHEF data, where $\alpha = 0.0$ (i.e pure RL) works best mainly because the data expert level is low and this dataset contains more low quality data.

Table 7. Control Suite performance of an XS-sized model using different BC loss scales α. The percentage of achieved expert average reward across 100 trials per task and the standard-error-based 95% CIs are reported.

Domain	$\#(\mathcal{T})$	α**=0.0**	α**=0.4**	α**=0.8**	α**=1.0**
Gato: Control	32	36.8	74.3	85.1	82.4
		[33.9, 39.6]	[70.2, 78.3]	[80.4, 89.8]	[76.5, 88.2]

D.1.2. Policy Loss Scale β

For efficient learning we share the network parameters between the policy and critic networks. This, however, requires some adjustments to balance these two losses for optimal performance. Therefore, we introduce a hyperparameter, β, that trades off the policy loss and the critic loss. To illustrate the sensitivity of this parameter, we use the CHEF tasks and perform pure reinforcement learning by setting BC loss scale $\alpha = 0.0$ while sweeping over different values of β. As can be seen in Table 8, for this task, a lower contribution of the policy loss leads to better results. In general, we have found that setting the β value to 0.005 achieves good performance across various tasks.

Table 8. Simulated RGB stacking performance of an XS-sized model using different policy loss scales β. The average success rates across 100 trials per task and their corresponding Wilson score intervals for $\alpha_W = 0.05$ are reported.

Domain	$\#(\mathcal{T})$	β**=1.0**	β**=0.1**	β**=0.01**	β**=0.005**
CHEF: sim	1	22.0	29.0	76.0	83.0
		[15.0, 31.7]	[21.0, 38.5]	[66.8, 83.3]	[74.5, 89.1]

D.2. Architecture Ablations

To understand what architectural decisions contribute to the performance we conduct a set of ablation studies. In particular we look into the multi-scale normalizer for encoding continuous inputs, the output action-cross-attention to generate Q-values, and the task conditioning. For compute efficiency reasons we perform these experiments on a subset of the data and with smaller model sizes.

D.2.1. Input Encoding

Our multi-scale normalizer is used for embedding both the proprioception observations and actions. We compare the multi-scale normalizer against a vanilla linear encoder and a token-based encoder similar to the one used by Reed et al. (2022). The linear encoder directly embeds the inputs to a single embedding using a linear layer followed by a normalization layer. The token-based encoder uses a lookup-table-based encoder analogous to the language encoder. Across all tokenization variants, the embedded proprioception tokens are separately cross-attended while the embedded action tokens are projected to a single embedding vector.

Table 13 reports separate results on the control suite tasks (using our XS-sized model) and RoboCat pyramid and tower tasks (using our S-sized model), where for better comparability with RoboCat (Bousmalis et al., 2023) we set the number of action bins to 1024. For Control Suite the linear embedder shows the lowest performance, likely caused by not being able to deal with the range of proprioception measurements. Based on the μ-law discretization, encoder is able to address the high range of measurements. However for the RoboCat tasks the μ-law encoder performs the worst, possibly due to discarding neighborship relations in the data. It also comes with a larger memory footprint due to the lookup table embedding. The multi-scale normalizer addresses all these issues and performs best.

D.2.2. Action Cross-attention

In addition to using a perceiver backbone for our model, a key design choice was also to use the action as output query rather than having it as an input. This has the advantage that we can use a common encoder for the policy and the Q-value estimator and enables quick evaluation of actions because the latent can be cashed. Furthermore, this shortens the path from actions to Q-values and the action is thus less likely to be ignored. In the last column of Table 13 we ablate this architectural choice against using the action as input modality for control suite tasks, where we made use of attention masking in order to hide the action input from the policy output. We observe a decrease in performance, likely due to the aforementioned reasons.

D.2.3. Comparison to Q-transformer and Return Conditioned Offline RL

We present additional results for a Q-transformer baseline when training a XS (73M parameters) model on the 32 domains from the control suite in table Table 14. As can be observed from the table, in aggregate the Q-transformer baseline is not better than BC. However, when separating results out by domain we can observe that Q-transformer matches the performance of PAC for lower dimensional domains (e.g. cartpole, finger turn) but fails in some of the higher dimensional domains such as dog walk and humanoid walk. In addition we ran a baseline that uses return conditioning to train on all data but then uses the same threshold as used for FilteredBC at test time (again training an XS model). This serves as a sensible reference roughly corresponding to a Decision Transformer implementation. We see that this recovers performance roughly equivalent to BC again but does not result in further improvement. Finally, we also ablated the choice of the distributional Q-function (no dist. Q) in the table, verifying that the performance benefit of PAC over other methods is not just due to the distributional Q-function (though it clearly benefits from it).

Details on Q-transformer implementation A naive application of Q-transformer (Chebotar et al., 2023) to our setting is not feasible due to the costly iterative (arg-)max computation involved; which would be prohitively expensive in our setting where we consider up to 38 action dimensions. We thus implement a version with independent Q-functions for each action dimension. In particular, we parameterize Q values per action dimension as $Q_\theta(a^i, s_t, \tau)$ and aggregate Q-values accross dimensions for bootstrapping in the TD-update according to:

$$V_\theta(s_t) = \frac{1}{|\mathcal{A}|} \sum_{i=o}^{|\mathcal{A}|} \max_{a^i} Q_\theta(a^i, s_t, \tau), \tag{16}$$

using this definition we can implement the loss

$$\begin{aligned}\mathcal{L}^{\mathrm{QT}}(\theta) = &\mathop{\mathbb{E}}_{(s',a,r,s')\in\mathcal{D}}\Big[\sum_{i=0}^{|\mathcal{A}|}(r + \gamma V_\theta(s') - Q_\theta(a^i, s, \tau))^2\Big] \\ &+\alpha \mathop{\mathbb{E}}_{(s',a,r,s')\in\mathcal{D}}\Big[\sum_{a'\neq a^i}(Q_\theta(a', s, \tau) - 0)^2\Big],\end{aligned} \tag{17}$$

where the first term corresponds to the TD-loss and the second term implements a conservatism bias.

This is essentially a cooperative multi-agent RL formulation as in Seyde et al. (2023). We also experimented with learning an additional V-function instead of aggregating across dimensions (and then regressing Q-values by using this V-function for bootstrapping which however gave statistically equivalent results. We swept over the hyperparameter alpha (in $[0.1, 1., 10.]$) as well as learning rates in order to provide a fair comparison.

Table 9. Data mixture of our scaling experiments. For each dataset, we report the number of trajectories and tokens contributed (#(Traj) and #(Tok)), the number of proprioception, vision and action dimensions in the raw data (N^P, N^V, N^A) and the probability P_{eff} with which we sample from each dataset during training.

Dataset Group	#(Traj)	#(Tok)	N^P	N^V	N^A	P_{eff}
Control Suite						
acrobot.swingup	10,843,200	5.75E+09	6	0	1	1/64
ball_in_cup.catch	5,292,800	3.44E+09	8	0	2	1/64
cartpole.balance	10,473,600	5.13E+09	5	0	1	1/64
cartpole.swingup	10,704,800	5.25E+09	5	0	1	1/64
cartpole.three_poles	10,012,800	7.31E+09	11	0	1	1/64
cartpole.two_poles	4,875,200	2.97E+09	8	0	1	1/64
cheetah.run	7,672,800	8.12E+09	17	0	6	1/64
dog.run	4,238,800	4.43E+10	223	0	38	1/64
dog.stand	4,162,400	4.45E+10	223	0	38	1/64
dog.trot	4,163,200	4.45E+10	223	0	38	1/64
dog.walk	5,861,600	6.27E+10	223	0	38	1/64
finger.spin	4,257,200	2.94E+09	9	0	2	1/64
finger.turn_easy	9,893,200	8.01E+09	12	0	2	1/64
finger.turn_hard	6,794,000	5.50E+09	12	0	2	1/64
fish.swim	6,172,800	7.96E+09	21	0	5	1/64
fish.upright	6,682,400	8.62E+09	21	0	5	1/64
hopper.hop	7,054,000	7.12E+09	15	0	4	1/64
hopper.stand	3,620,400	3.66E+09	15	0	4	1/64
humanoid.run	4,632,000	1.75E+10	67	0	21	1/64
humanoid.stand	5,181,200	1.95E+10	67	0	21	1/64
humanoid.walk	7,566,800	2.85E+10	67	0	21	1/64
pendulum.swingup	7,992,400	3.28E+09	3	0	1	1/64
point_mass.easy	3,980,400	1.95E+09	4	0	2	1/64
quadruped.escape	3,102,000	1.48E+10	101	0	12	1/64
quadruped.run	1,730,400	6.66E+09	78	0	12	1/64
quadruped.walk	4,457,600	1.72E+10	78	0	12	1/64
reacher.easy	5,501,200	3.14E+09	6	0	2	1/64
swimmer.swimmer15	5,483,600	1.78E+10	61	0	14	1/64
swimmer.swimmer6	98,000	1.42E+08	25	0	5	1/64
walker.run	1,988,000	2.88E+09	24	0	6	1/64
walker.stand	7,401,200	1.07E+10	24	0	6	1/64
walker.walk	5,334,400	7.73E+09	24	0	6	1/64
CHEF						
sim.lift	10,072,080	7.16E+10	129	19,200	5	1/48
sim.open	10,072,080	7.16E+10	129	19,200	5	1/48
sim.place	10,072,080	7.16E+10	129	19,200	5	1/48
sim.reach_grasp	10,072,080	7.16E+10	129	19,200	5	1/48
sim.stack	10,072,080	7.16E+10	129	19,200	5	1/48
sim.stack_leave	10,072,080	7.16E+10	129	19,200	5	1/48
real.lift	28,224,012	1.87E+11	129	12,800	5	1/48
real.open	28,224,012	1.87E+11	129	12,800	5	1/48
real.place	28,224,012	1.87E+11	129	12,800	5	1/48
real.reach_grasp	28,224,012	1.87E+11	129	12,800	5	1/48
real.stack	28,224,012	1.87E+11	129	12,800	5	1/48
real.stack_leave	28,224,012	1.87E+11	129	12,800	5	1/48
RoboCat						
panda.sim.triple_stack.success	13,992,960	8.80E+10	44	51,200	7	1/32
panda.sim.triple_stack.failure	5,439,066	3.42E+10	44	51,200	7	1/32
panda.sim.pyramid.success	15,998,400	1.01E+11	44	51,200	7	1/32
panda.sim.pyramid.failure	15,024,082	9.45E+10	44	51,200	7	1/32
sim.insert_large_gear	13,372,152	5.18E+10	21	32,000	7	1/24
sim.insert_medium_gear	14,182,310	5.49E+10	21	32,000	7	1/24
sim.insert_small_gear	12,260,934	4.74E+10	21	32,000	7	1/24

Table 10. The data mixture used for the pre-training experiments. For each domain we report the modalities (P = proprioception, V = vision, L = language, A = actions), the number of tasks, the number of episodes recorded, the effective number of trajectories (each consisting of five timesteps), the number of tokens contributed, the percentage of successful episodes and the weight λ of the domain during the data sampling process. [* = success rate only available for five stacking tasks; ** = success rate average only computed across 82 tasks with available success rates]

Domain	**Mod**	$\#(\mathcal{T})$	**#(Ep)**	**#(Traj)**	**#(Tok)**	**success**	λ
Gato: Control	P, L, A	32	468k	187M	430B	47%	4
CHEF: sim	V, P, L, A	30	755k	60M	430B	28%*	1
CHEF: real	V, P, L, A	30	2M	169M	1.12T	12%*	1
RC: Tower	V, P, L, A	7	100k	16M	100B	75%	2
RC: Pyramid	V, P, L, A	30	601k	96M	604B	75%	5
RC: Insertion	V, P, L, A	3	100k	40M	154B	97%	1
	$\sum =$	132	4.024M	567M	2.84T	avg = 58%**	

Table 11. Data mixture of our scaling experiments. For each dataset, we report the number of trajectories and tokens contributed (#(Traj) and #(Tok)), the number of proprioception, vision and action dimensions in the raw data (N^P, N^V, N^A) and the probability P_{eff} with which we sample from each dataset during training. Group entries abbreviated by ... are identical with the ones reported in Table 9.

Dataset Group	#(Traj)	#(Tok)	N^P	N^V	N^A	P_{eff}
`Control Suite`						
...						4/14
`CHEF`						
...						2/14
`RoboCat`						
`panda.sim.triple_stack.success.eval_set_2`	1,244,000	7.82E+9	44	51,200	7	1/42
`panda.sim.triple_stack.failure.eval_set_2`	681,818	4.29E+9	44	51,200	7	1/42
`panda.sim.triple_stack.success.eval_set_4`	7,363,360	4.63E+10	44	51,200	7	1/42
`panda.sim.triple_stack.failure.eval_set_4`	1,843,906	1.16E+10	44	51,200	7	1/42
`panda.sim.triple_stack.success.eval_set_5`	3,439,040	2.16E+10	44	51,200	7	1/42
`panda.sim.triple_stack.failure.eval_set_5`	1,366,208	8.59E+10	44	51,200	7	1/42
`panda.sim.pyramid.success.eval_set_1`	15,024,000	9.45E+10	44	51,200	7	5/140
`panda.sim.pyramid.failure.eval_set_1`	4,367,962	2.75E+10	44	51,200	7	5/140
`panda.sim.pyramid.success.eval_set_2`	13,409,760	8.43E+10	44	51,200	7	5/140
`panda.sim.pyramid.failure.eval_set_2`	5,763,134	3.63E+10	44	51,200	7	5/140
`panda.sim.pyramid.success.eval_set_3`	13,249,600	8.33E+10	44	51,200	7	5/140
`panda.sim.pyramid.failure.eval_set_3`	5,749,294	3.62E+10	44	51,200	7	5/140
`panda.sim.pyramid.success.eval_set_4`	15,385,920	9.68E+10	44	51,200	7	5/140
`panda.sim.pyramid.failure.eval_set_4`	3,767,426	2.47E+10	44	51,200	7	5/140
`panda.sim.pyramid.success.eval_set_5`	15,318,240	9.64E+10	44	51,200	7	5/140
`panda.sim.pyramid.failure.eval_set_5`	4,053,880	2.55E+10	44	51,200	7	5/140
`sim.insert_large_gear`	13,372,152	5.18E+10	21	32,000	7	1/42
`sim.insert_medium_gear`	14,182,310	5.49E+10	21	32,000	7	1/42
`sim.insert_small_gear`	12,260,934	4.74E+10	21	32,000	7	1/42

Table 12. Optimizer hyperparameters for all model scales across all experiments.

Hyperparameter	**XXS (32M)**	**XS (73M)**	**S (164M)**	**M (391M)**	**L (988M)**
`lr_init`	1e-6	1e-6	1e-7	1e-7	1e-7
`lr_peak`	1e-4	1e-4	5e-5	3e-5	3e-5
`lr_end`	1e-5	1e-5	5e-6	3e-6	3e-6
`lr_warmup_steps`	1.5e4	1.5e4	1.5e4	1.5e4	1.5e4
`adamw_beta1`	0.9	0.9	0.9	0.9	0.9
`adamw_beta2`	0.95	0.95	0.95	0.95	0.95
`adamw_weight_decay`	1e-3	1e-3	1e-3	1e-3	1e-3

Table 13. Different architectural ablations trained separately on Control Suite (XS-sized model) and RoboCat pyramid and tower (S-sized model). For Gato:Control we report the percentage of achieved expert average reward and the standard-error-based 95% CIs. For RoboCat (RC) we report the average success rates and their corresponding Wilson score intervals for $\alpha_W = 0.05$. We use 100 trials per task. Best results (within CI of the best mean) in each row are bold.

Domain	$\#(\mathcal{T})$	**Linear Encoder**	**μ-Law Encoder**	**Multi-scale Normalizer**	**w/o output action**
Gato:Control	32	79.6 [74.5, 84.6]	**82.7 [77.2, 88.1]**	**86.7 [82.4, 90.9]**	76.8 [72.6, 81.1]
RC:Tower	7	**68.1 [64.6, 71.5]**	50.7 [47.0, 54.4]	**69.1 [65.6, 72.5]**	N/A
RC:Pyramid	30	63.5 [61.7, 65.2]	36.1 [34.4, 37.8]	**65.8 [64.1, 67.5]**	N/A

Table 14. Comparison between XS sized models for different objectives on Control domains. We show both aggregate performance over tasks as well as per domain results for selected lower dimensional and higher dimensional domains. We implement a Q-transformer style update as well as a simple return conditioned baseline (RC). Neither outperforms BC. Ind. denotes that the Q-transformer is implemented with the simplification of independent Q-values per action dimension. All results are percentage of expert performance, established by an independent RL training run per domain, following (Reed et al., 2022).

Domain	$\#(\mathcal{T})$	**PAC**	**PAC** (no dist. Q)	**BC**	**Q-transformer (ind.)**	**RC**
Gato:Control	32	**86.7 [82.4, 90.9]**	83.5 [81.2, 84.2]	82.4 [81.6, 83.8]	64.8 [60.1, 65.8]	80.8 [78.3, 82.4]
Cartpole		97.0	96.8	95.3	99.5	95.6
Finger turn-hard		98.3	98.4	95.7	96.3	97.5
Dog trot		86.5	75.7	69.8	12.7	70.4
Humanoid walk		92.4	71.2	60.6	24.5	62.9

D.2.4. Task Conditioning

In this ablation study we investigate how PAC uses its two different task modalities: the goal image τ^V and the language instruction τ^L. Specifically, we probe whether a trained model can react to task specifications where one of the modalities is missing and whether it can be adapted to perform better on task conditions which differ from the training setup. To conduct this investigation, we take a PAC model of size M that has been pre-trained on all data (Control Suite, CHEF and RoboCat). Note that out of all these domains, Robocat Tower and Robocat Pyramid are the only ones that have visual task descriptions available besides language task descriptions while all other domains have only language ones and therefore their visual task descriptions are merely padded zeros. We then evaluate this model with different modalities for task specification being available: **Vision+Lang**: both modalities τ^V and τ^L are present therefore the same as the setup during pretraining; **Vision**: only visual task descriptions τ^V are present and the language ones are masked out; **Lang**: only τ^L are available; **No Goal**: both task description modalities are masked out.

The evaluation results are presented in the **Pretrained** rows of Table 15. A notable observation is that the pretrained model is still very performant when only the language task description τ^V is available, albeit this differs from the task conditioning during pretraining where the model has only seen visual and language task descriptions both being present for the Tower and Pyramid domains; while the success rate drops substaintially when only visual task descriptions τ^V are present. One hypothesis of such an imbalanced performance of the pretrained model when conditioning on τ^V or τ^L could be the following: Since all data domains the model has seen during the pretraining phase have τ^L available while Tower and Pyramid are the only ones with τ^V, the $N^L = 50$ language task description tokens are attended to for every single datapoint while the $N^V_\tau \times N_E = 300$ visual task description tokens are masked out except for when learning on the Tower and Pyramid domains of robocat. Therefore we suspect the model would learn to attend more to the language task tokens than the vision task tokens in order to be able to achieve good performance averaging over all task domains. We conduct additional experiments for this hypothesis and present the results in Table 16 which will be discussed directly following.

In order to see if above said model could be quickly adapted such that its performance is more amenable to varying modalities for task description, we conduct a finetuning experiment of the model where either task description modality will be present for a certain percentage p_τ of the time and masked out the rest of the time. In particular, for this experiment we set $p_{\tau^V} = 0.99$ and $p_{\tau^L} = 0.9$ where τ^V are masked out less often to compensate for the fact that visual task descriptions are only present for two domains while the finetuning is carried over all tasks, but with adjusted sampling weights compared to Table 6: from $\lambda = (8, 2, 2, 1, 1, 2)$ to $\lambda = (6, 2, 2, 6, 6, 2)$. We also use a fixed learning rate of 3e-6. The task conditioning evaluation results of this finetuned model (after 200K steps) are shown in the **FT** rows of Table 15. We observe that the success rate of the model increases by a large margin when only visual task descrip-

Table 15. Adapting a pre-trained PAC model (size M) to a new task-conditioning setup during finetuning phase setting $p_{\tau^V} = 0.99, p_{\tau^L} = 0.9$. The average success rates across 100 trials per task and their corresponding Wilson score intervals for $\alpha_W = 0.05$ are reported.

Model	Domain	$\#(\mathcal{T})$	Vision+Lang	Vision	Lang	No Goal
PT@3M	RC: Tower	7	59.7 [49.6, 69.3]	0.1 [0.0, 3.9]	58.1 [48.1, 67.7]	0.0 [0.0, 3.6]
	RC: Pyramid	30	53.3 [43.2, 63.1]	11.4 [10.3, 12.6]	53.1 [43.1, 62.9]	11.0 [9.9, 12.2]
FT@200K	RC: Tower	7	62.0 [51.8, 71.5]	44.7 [35.4, 54.3]	60.1 [50.1, 69.6]	8.7 [7.1, 15.2]
	RC: Pyramid	30	53.1 [43.1, 62.9]	47.7 [37.7, 57.8]	53.5 [43.5, 63.3]	8.2 [4.3, 14.9]

Table 16. Pre-training a PAC model (size S) with varying goal conditions. Evaluation after 500k training steps. The average success rates across 100 trials per task and their corresponding Wilson score intervals for $\alpha_W = 0.05$ are reported.

Model	Domain	$\#(\mathcal{T})$	Vision+Lang	Vision	Lang	No Goal
$p_{\tau^V} = 1.0$	RC: Tower	7	55.6 [43.4, 66.3]	5.57 [4.0, 7.5]	57.0 [47.0, 66.6]	5.1 [3.6, 7.1]
$p_{\tau^L} = 1.0$	RC: Pyramid	30	53.1 [43.0, 62.9]	0.933 [0.7, 5.1]	51.9 [41.9, 61.7]	0.9 [0.9, 5.0]
$p_{\tau^V} = 0.9$	RC: Tower	7	52.6 [42.6, 62.4]	54.6 [44.6, 64.3]	51.4 [41.5, 61.3]	6.3 [2.5, 12.9]
$p_{\tau^L} = 0.9$	RC: Pyramid	30	48.2 [38.3, 58.2]	46.8 [36.9, 56.8]	49.8 [39.9, 59.8]	13.1 [7.4, 21.2]

tion is present compared to the pretrained model, while the model is able to keep (or slightly improves) its performance for the conditions where it already perform well after pre-training: **Vision+Lang** and **Lang**. This is promising as it shows the flexbility of the proposed model that it is quickly adaptable to be reactive to task specification conditions not encountered during training.

With Table 15 showing viable approaches to adapt pretrained models to varying task conditionings, as just mentioned, we conduct an extra experiment to test the hypothesis that attributes the imbalanced performance of the pretrained model under τ^V only versus τ^L only to these two modalities being improportionally present in the pretraining data. The experiment that tests this hypothesis is where we train a PAC model (size S) from scratch, but on RoboCat Tower and RoboCat Pyramid only such that both visual and language task descriptions are available for every datapoint. We then evaluate this model after 500k pretraining steps and present the results in the $\mathbf{p_{\tau^V} = 1.0, p_{\tau^L} = 1.0}$ rows of Table 16. However, we find that with balanced presence of τ^V and τ^L, the model is still much more reactive to τ^L (reaching success rate of 57.0% and 51.9%) than τ^V (5.6% and 0.9%). This refutes the hypothesis and we suspect that the performance imbalance might be caused by the fact that the language task descriptions is simply more discriminative than a goal image.

Setting the modality present rate to $\mathbf{p_{\tau^V} = 0.9, p_{\tau^L} = 0.9}$, we train another model from scratch with otherwise the same setup of above and present the evaluation results in the corresponding rows in Table 16. The results show that with varying goal conditionings being present in the pre-training phase, the resulting model can perform well on all conditioning variations as expected.

E. Scaling Details

E.1. Return Profile Fitting

We fit the average return of a models as a logistic function of the number of training steps:

$$R(n) = \frac{a}{1 + \exp(-k * (n - n_0))} + b. \tag{18}$$

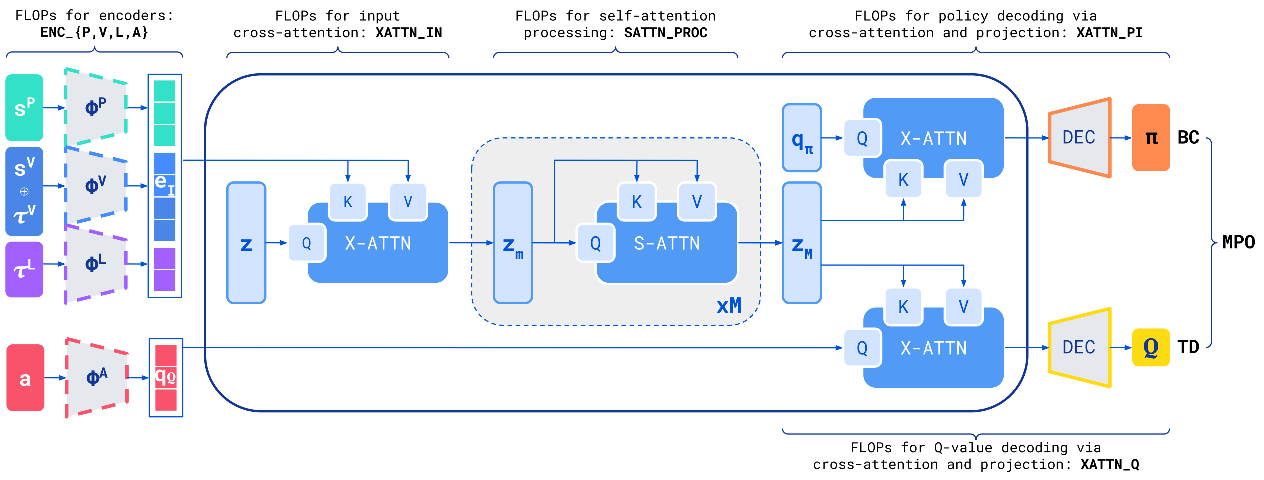
We evaluate six checkpoints of each training run evenly spaced across the 3M updates. Each evaluation runs 100 trials per task and computes the average episode return across 102 simulation tasks. The parameters a, k, n_0, b of Equation (18) are then fitted to the evaluation data points to obtain a *return profile* for each model.

E.2. FLOP Counting

Model Scale	FWD	BWD	BATCH_UPDATE
XXS (32M)	7.826E+09	1.573E+10	1.206E+13
XS (73M)	1.360E+10	2.733E+10	2.096E+13
S (164M)	2.341E+10	4.705E+10	3.607E+13
M (391M)	4.380E+10	8.804E+10	6.750E+13
L (988M)	1.040E+11	2.090E+11	1.602E+14

Table 17. FLOPs costs for forward pass, backward pass and batch update for all scales of our PAC model family. All FLOPs are computed for a batch size $B = 512$ and a target update frequency $f_{\theta'} = 100$.

When counting the FLOPs of the PAC architecture for the scaling analysis, we follow the FLOP counting scheme established by Hoffmann et al. (2022) since the bulk of our model's computation sits in cross-attention and self-attention blocks. FLOPs of cross-attention blocks are counted similarly as in self-attention blocks with the only difference of the length of the input and output sequence being different which yields seq_len_in × seq_len_out



FLOPs for encoders: ENC_{P,V,L,A}
FLOPs for input cross-attention: XATTN_IN
FLOPs for self-attention processing: SATTN_PROC
FLOPs for policy decoding via cross-attention and projection: XATTN_PI
FLOPs for Q-value decoding via cross-attention and projection: XATTN_Q
s^P
s^V
τ^V
τ^L
a
Φ^P
Φ^V
Φ^L
Φ^A
e_I
q_Q
z
z_m
z_M
Q
K
V
X-ATTN
S-ATTN
xM
q_π
DEC
π
BC
Q
TD
MPO

Figure 7. The different parts of the PAC architecture for which FLOPs are counted. The FLOPs for the different encoders are counted as $\texttt{ENC\_}\{\texttt{P,V,L,A}\}$, respectively. All FLOPs used to aggregate the encoded input tokens e_I into the Perceiver's latent z_i during the input cross-attention are counted as XATTN_IN. The FLOPs used for processing the latents via M self-attention blocks are counted as SATTN_PROC. The FLOPs used to decode the policy π and the action-value function Q from z_M via cross-attention and subsequent projections are counted as XATTN_PI and XATTN_Q respectively.

× FLOPS_ATTN instead of seq_len_in × seq_len_in × FLOPS_ATTN like in a normal self-attention block.

After tokenization of proprioception, vision, language and action (cf. Appendix C.3) we obtain T^P, T^V, T^A and T^L tokens respectively. Their encoders are simple projections and we count the FLOPs used for the embeddings as:

- $\texttt{ENC\_P} = \texttt{MAF} \times T^P \times D_I$
- $\texttt{ENC\_V} = \texttt{MAF} \times T^V \times D_I + \texttt{FLOPS\_RESNET}$
- $\texttt{ENC\_L} = \texttt{MAF} \times T^L \times N_T \times D_I$
- $\texttt{ENC\_A} = \texttt{MAF} \times T^A \times D_I$

Similar to Hoffmann et al. (2022) we use a *multiply-accumulate-factor* (MAF) of 2 to describe the multiply accumulate cost. The FLOPs needed to transform the raw input images into tokens using a ResNet are captured by FLOPS_RESNET. We count each 2D convolution operation in the ResNet as $\texttt{num\_kernels} \times (w_1 * w_2) \times (o_1, o_2) \times \texttt{MAF}$ where $w_{\{1,2\}}$ and $o_{\{1,2\}}$ are the kernel and output dimensions respectively.

The total number of FLOPs used for one forward pass of PAC are:

$$\begin{aligned}\texttt{FLOPS\_FWD} = {} & \texttt{ENC\_P} + \texttt{ENC\_V} + \texttt{ENC\_L} + \texttt{ENC\_A} \\ & + \texttt{XATTN\_IN} \\ & + M \times \texttt{SATTN\_PROC} \\ & + \texttt{XATTN\_PI} + \texttt{XATTN\_Q}\end{aligned}$$

When estimating the FLOPs for the backward pass, we slightly deviate from Kaplan et al. (2020) and also factor in the target network updates (cf Section 3.2) which occur every $f_{\theta'}$ updates (in all of our experiments we keep $f_{\theta'} = 100$). Therefore, we count the FLOPs for PAC's backward pass as:

$$\texttt{FLOPS\_BWD} = (2 + \frac{1}{f_{\theta'}}) * \texttt{FLOPS\_BWD}$$

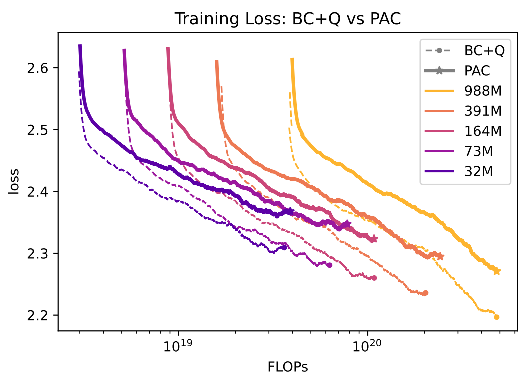
Lastly, the FLOPs for an update with batch size B are counted as:

$$\texttt{FLOPS\_UPDATE} = B * (\texttt{FLOPS\_BWD} + \texttt{FLOPS\_BWD})$$

We list the FLOPs costs for the core operations of our PAC model series in Table 17.

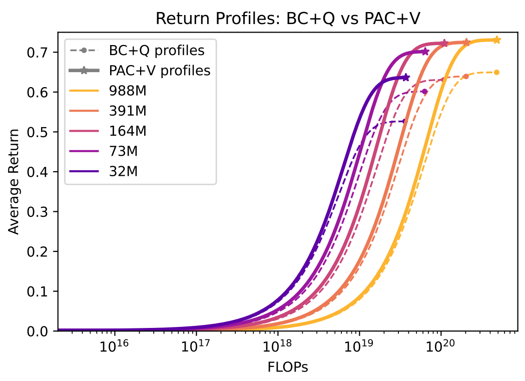
E.3. Model Loss as Performance Indicator

We plot the training loss against the FLOPs for both model sets in Figure 8 when training on the scaling data mix (cf. Table 9). The training losses suggest that the BC+Q model should outperform PAC because its final loss is about 0.1 points lower. In order to assert whether the training loss can be a reliable indicator of model performance in our experiments, we compare the final model checkpoints of BC+Q and PAC (both of size L) on 73 tasks from our training dataset in simulation and report the results in Figure 8. However, this performance-based comparison does not support the assumption that a lower training loss is indicative of higher task performance. On the contrary, PAC significantly outperforms BC on the simulated stacking task despite its



Domain	$\#(\mathcal{T})$	BC+Q	PAC
Gato: Control	32	89.2 [84.3, 94.1]	93.2 [89.6, 96.9]
RC: Tower	7	62.9 [59.1, 66.2]	63.1 [59.5, 66.6]
RC: Pyramid	30	50.5 [48.7, 52.3]	55.6 [53.8, 57.4]
RC: Insertion	3	82.0 [76.9, 85.6]	83.7 [79.1, 87.4]
CHEF: sim	1	12.0 [7.0, 19.8]	40.0 [30.9, 49.8]

Figure 8. Top: Training loss comparison of the PAC model families for $\alpha = 1.0$ (BC+Q) and $\alpha = 0.75$ (PAC). Bottom: Success rates in different simulation tasks of the final checkpoints after 3M updates for the respective L-size models. Each model was evaluated in 100 trials on each of the $\#(\mathcal{T})$ in each domain. For the Control domain, the percentage of achieved expert average reward and the standard-error-based 95% confidence intervals are reported. For all other domains, the average success rates and their corresponding Wilson score intervals for $\alpha_W = 0.05$ are reported.



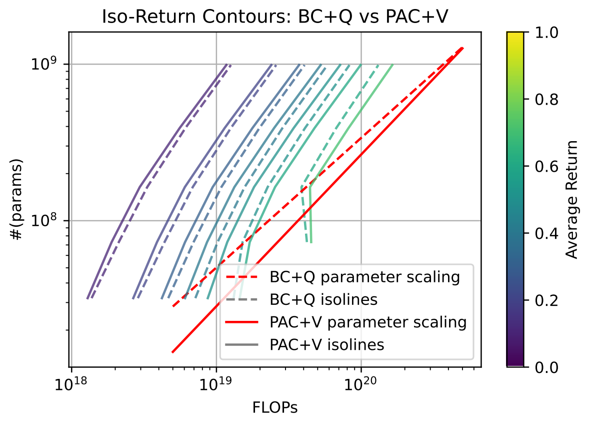


Figure 9. Iso-Return comparison of BC+Q vs PAC+V. The return profiles (top) contrast the expected average return between the BC baseline and the RL objective across all model scales. The Iso-Return contours (bottom) depict how the reward landscape over the parameter-FLOPs landscape shifts between using the BC objective (dashed contours) and the RL objectives (solid contours). For PAC+V the reward landscape is shifted towards the top left compared to the BC baseline indicating that it leads to higher average return plateaus for the same FLOP budgets.

higher training loss. Hence, we cannot use the model directly for selecting interpolants to fit the scaling laws with and need use the proxy of return profiles (cf. Appendix E.1).

Our finding is consistent with previous works (Hilton et al., 2023) as we find that loss is not necessarily a reliable indicator of model performance in RL settings. This can be due to a variety of reasons, e.g. there can be degenerate solutions when the policy overfits to the data and is unable to generalize to new states or when the Q-function collapses. Additionally, when a TD-style loss is used, losses are generally not comparable and we refer to (Fujimoto et al., 2022) for a more elaborate discussion on this issue.

E.4. PAC+V Scaling Analysis

We also conduct a scaling analysis for the V-function variant of our architecture: PAC+V (cf. Appendix A.3) following the protocol defined in Section 4.1. We plot the scaling laws for PAC+V in Figure 12 comparing it to the scaling laws for the BC+Q baseline as well. We also compare the Iso-Return contours between the two models in Figure 9.

The conclusions of the PAC+V scaling laws are consistent with the ones drawn for the Q-function variant of PAC in Section 4.1. The suggested model size for 2.45T tokens is with 852M parameters slightly smaller than PAC's with 954M parameters, but the parameters should be scaled slightly more aggressively as compute increases indicated by $a(\text{PAC+V}) > a(\text{PAC})$: $0.970 > 0.920$. However, the data scaling is in line with both PAC as well as the BC+Q baseline: $b(\text{PAC+V}) \approx 0.266$.

E.5. Scaling-based Performance Analysis of PAC on Control Suite

In this section we provide additional details about the progression of PAC 's task performance with model scale based on our experiments in Section 4.1. When expanding the task performances on Control Suite during the scaling experiments, the results fall in one of three categories: A) tasks with significant model scaling benefits; B) tasks which are

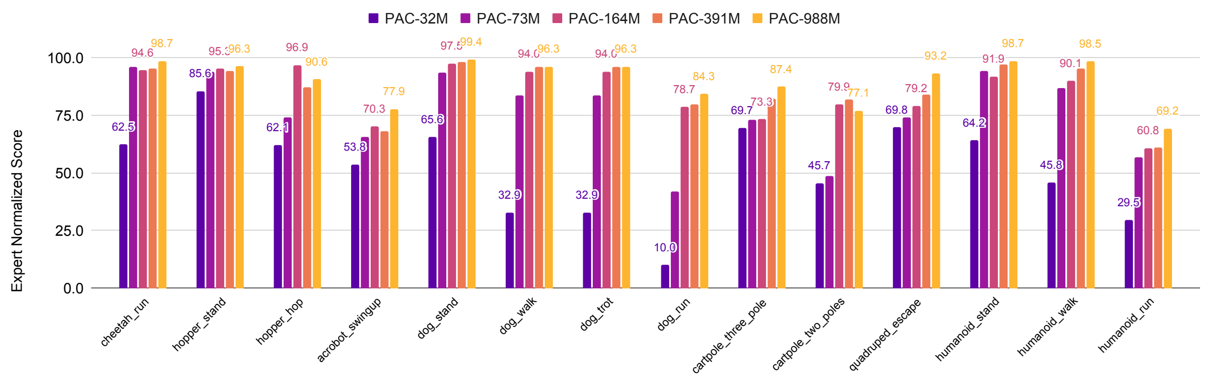


Figure 10. PAC's performance across different Control Suite tasks where significant model scaling benefits are observed. We report average task performance over 100 trials for each task and model scale.

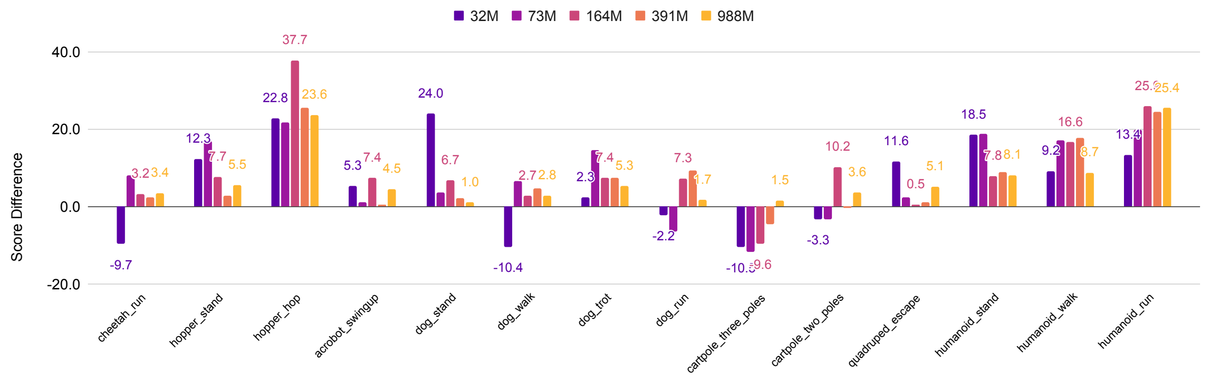


Figure 11. Average performance delta of PAC over the BC+Q baseline across all model scales reported for all Control Suite tasks which benefit significantly from model scaling.

already perfectly solved at the smallest model scale; and C) tasks which attain a score much smaller than 90 with the smallest model and do not improve with model scale.

Tasks in category B) are: ball_in_cup_catch, cartpole_{balance, swingup}, finger_spin, finger_turn_{easy, hard}, fish_{swim, upright}, pendulum_swingup, point_mass_easy, quadruped_{run, walk}, reacher_easy and walker_{run, stand, walk}. All of these tasks already achieve scores $\geq$ 95 with PAC-32M. Tasks in category C) are swimmer_{6, 15} which plateau at scores around 60 with PAC-32M and do not improve further with model scale. However, 14 out of 32 tasks fall in category A) and show significant performance improvements with model scale. We report the performance progression of PAC on these tasks in Figure 10. Among these tasks, cheetah_run, hopper_stand, dog_{stand, walk, trot} and humanoid_{stand, walk} stand out in particular as model scaling pushes them from scores as low 33 to over 95 into the 'task mastery range' without any additional data.

We also compare PAC's performance in the 14 'scaling tasks' to the BC+Q baseline across all model scales in Figure 11. We find that the improvements attributable to PAC 's offline RL objective are most pronounced in the hopper and humanoid tasks. There are tasks where the BC+Q baseline has an initial advantage, e.g. cheetah_run, dog_walk or cartpole_three_poles. However, as PAC is scaled up, it always overcomes this initial disadvantage and outperforms BC+Q consistently. This corroborates the key finding of the scaling analysis: the offline RL objective initially needs higher parameter counts compared to BC+Q (cf. Figure 3 right panel and Figure 12 top right panel), but ultimately scales better with more compute than BC+Q leading to higher performance plateaus (cf. Figure 4 bottom panel).

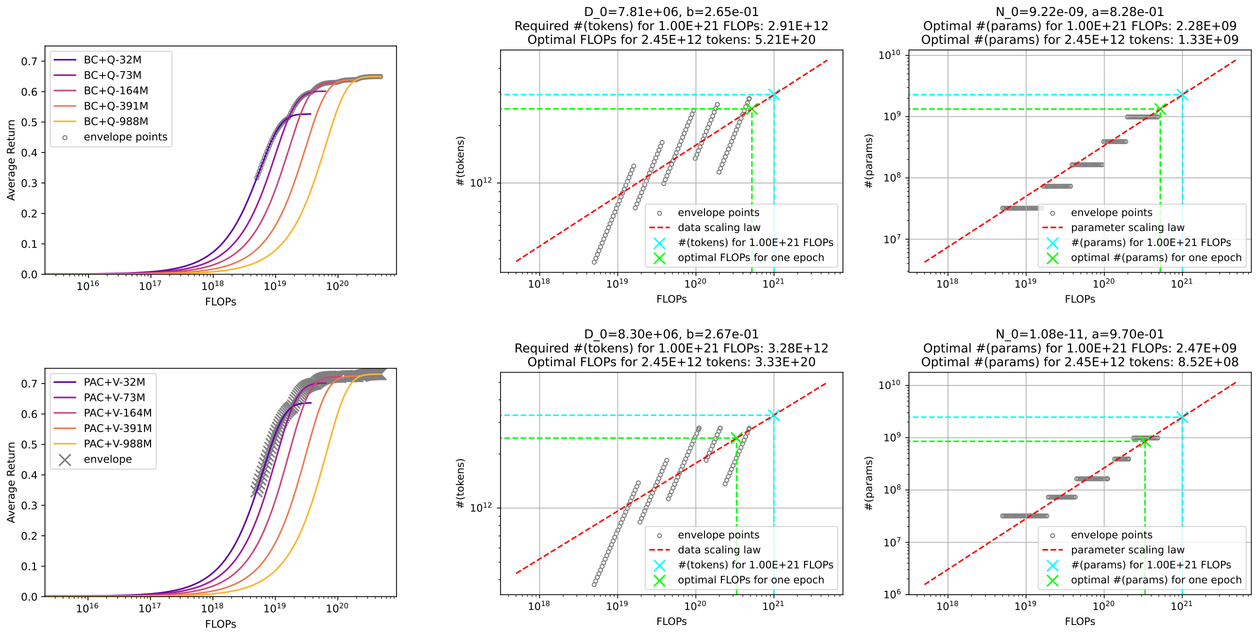


Figure 12. Scaling laws based on the return profile envelopes for `BC+Q` (top) and `PAC+V` (bottom). We select 100 logarithmically spaced points between 5E+18 and 5E+20 FLOPs on the envelope of the return profiles (left) and use them to fit the scaling laws (middle, right). For both the token and parameter scaling plots, we indicate the scaling trend with a dashed red line. The dashed green line represents the optimal number of parameters and compute budget needed to fit the data in one epoch of training. The dashed teal line represents the optimal data and parameter trade-off for a FLOP budget of 1E+21.

F. Additional Experiments

F.1. Large-scale Training of `PAC+V`

As mentioned in Section 4.2, we also conduct a large-scale training on the full pre-training corpus (cf. Table 11) following the same protocol as for the other models. We report the additional results obtained with `PAC+V` in Table 18.

The results for the V-function variant of PAC are in line with the ones obtained using the Q-function variant with two notable exceptions. First, in the case of RC:Insertion which consists exclusively of human teleoperated data, `PAC+V` provides another significant improvement over `PAC` to$\approx 89\%$ success rate. This suggests that `PAC+V` could have an edge in cases where many successful task demonstrations exist which are however beset by minor inefficiencies, e.g. motion jitter or pauses caused by human teleoperation. The results suggest that the one-step improvement over the data-generating policy afforded by `PAC+V` could be enough to prevent imitating many inefficiencies present in the training data. Second, in the case of CHEF:sim, `PAC+V` lags behind its Q-function-based counterpart `PAC`. This suggests that in cases where the data is mostly sub-optimal, a value function alone might not be sufficient to filter transitions effectively enough to improve the policy.

F.2. RL Fine-tuning and Self-improvement

In Section 4.3 we evaluated `PAC` on a robotic stacking benchmark, and performed iterative fine-tuning to improve performance in this domain. Here, we provide additional details for those results.

The RGB Stacking benchmark (Lee et al., 2021) defines five distinct sets of test objects, each highlighting a different challenge of object manipulation. For brevity, we only reported mean success rates above. In Table 19, we provide success rates for each object separately. The continuous self-improvement of `PAC` is particularly visible on ”Set 2”, which requires precise force-based control to flip objects onto their side. However, the same holds for the other object sets, which improve across fine-tuning rounds until converging at $> 90\%$ success rate.

Data collection was carried out only for the CHEF:real domain, so it is worth examining whether such focused self-improvement causes the PAC model's performance on other tasks to degrade. As Table 20 shows, performance on the simulated tasks is unaffected, even after three rounds of fine-tuning.

While we started the iterative improvement in Section 4.3 by pre-training α-`PAC`, the BC/RL trade-off parameter α also allows flexibility as to what data to start from. For

Table 18. Policy success rates across $\#(\mathcal{T})$ tasks in each domain for 100 evaluations per task. The average success rate in the training data is reported as p_D. For Gato: Control, the percentage of achieved expert average reward and the standard-error-based 95% CIs are reported (where available). For all other task families, the average success rates and their corresponding Wilson score intervals for $\alpha_W = 0.05$ are reported. Best results (within CI of the best mean) in each row are bold. [† cited from Reed et al. (2022); ⋆ cited from Bousmalis et al. (2023)]

Domain	$\#(\mathcal{T})$	p_D	**Gato† / RC⋆**	**FilteredBC**	**BC+Q**	**PAC**	**PAC+V**
Gato:Control	32	N/A	63.6†	75.8 [62.5, 78.6]	84.6 [79.6, 89.7]	**87.7 [83.8, 91.6]**	**90.2 [86.3, 94.1]**
RC:Tower	7	75	61.0⋆ [57.3, 64.5]	64.0 [60.4, 67.5]	**71.3 [67.8, 74.5]**	**69.3 [65.8, 72.6]**	67.3 [63.7, 70.7]
RC:Pyramid	30	75	**64.5⋆ [62.8, 66.2]**	**64.0 [62.3, 65.7]**	62.4 [60.7, 64.1]	**63.5 [61.7, 65.1]**	**65.3 [63.5, 66.9]**
RC:Insertion	3	97	71.3⋆ [66.0, 76.2]	81.0 [75.8, 84.7]	79.7 [74.8, 83.8]	80.3 [75.5, 84.4]	**89.0 [85.0, 92.1]**
CHEF:sim	1	28	N/A	17.0 [10.9, 25.5]	11.0 [6.3, 18.6]	**55.0 [45.2, 64.4]**	42.0 [32.8, 51.8]

Table 19. Per-group success rates and confidence intervals for real-robot stacking tasks across self-improvement iterations ($\#(\mathcal{T}) = 5$). The policies are evaluated in 400 trials per set. The average success rates and their corresponding Wilson score intervals for $\alpha_W = 0.05$ are reported.

Iteration	**Set 1**	**Set 2**	**Set 3**	**Set 4**	**Set 5**	**All Sets**
Pretraining	51.5 [46.6, 56.4]	53.5 [48.6, 58.3]	65.5 [60.7, 70.0]	84.7 [80.8, 87.9]	94.0 [91.2, 95.9]	69.8 [67.8, 71.8]
RLFT #1	83.0 [79.0, 86.4]	66.8 [62.0, 71.2]	87.5 [83.9, 90.4]	90.8 [87.6, 93.3]	95.5 [93.0, 97.1]	84.7 [83.1, 86.2]
RLFT #2	88.0 [84.4, 90.8]	76.2 [71.8, 80.1]	92.5 [89.5, 94.7]	95.8 [93.4, 97.4]	96.8 [94.6, 98.1]	89.8 [88.4, 91.1]
RLFT #3	91.8 [88.7, 94.1]	91.5 [88.4, 93.9]	90.8 [87.6, 93.3]	95.0 [92.4, 96.7]	97.0 [94.8, 98.3]	93.2 [92.0, 94.2]

Table 20. Comparison of performance across all domains after pre-training (α-PAC), after three rounds of self-improvement on the CHEF:real domain (RLFT #3).

Domain	$\#(\mathcal{T})$	α**-PAC**	**RLFT #3**
Gato: Control	32	92.1 [88.4, 95.9]	91.3 [89.6, 96.9]
RC: Tower	7	69.6 [65.9, 72.7]	70.0 [66.5, 73.3]
RC: Pyramid	30	64.9 [63.1, 66.6]	65.1 [63.3, 66.8]
RC: Insertion	3	89.3 [85.0, 92.1]	80.3 [75.5, 84.4]
CHEF: sim	1	52.0 [42.3, 61.5]	59.0 [49.2, 68.1]
CHEF: real	5	69.8 [67.8,71.8]	93.2 [92.0,94.2]

comparison, we also deploy `BC+Q` on the robot – a model which is pre-trained using only the BC part of the objective to optimize its policy ($\alpha = 1$), but which has already learned a Q-function on the pre-training data ($\beta > 0$). The initial performance of this model on the real robot is unsurprisingly low with only 3.6% success rate. When we continue the training on the *same pre-training data* (including all sim and real tasks) for another 200k updates, but lower α to 0 to fully leverage the Q-function for policy improvement, we observe a significant jump to 38.2% success rate when re-deploying on the robot. While this performance is still significantly lower than the 69.8% success of the α-`PAC` model after initial pre-training, it is high enough to feasibly be used as an alternate starting point for the self-improvement loop. This can be useful in scenarios where no non-expert data is available initially to perform α-`PAC` from the start.