# MIND THE BUDGET: ACCELERATING DEEP REINFORCEMENT LEARNING USING EARLY EXIT NEURAL NETWORKS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

The *"Bitter Lesson"* from Richard S. Sutton emphasizes that AI methods leveraging computation tend to outperform those relying on human insight, underscoring the value of approaches that use computational resources efficiently. In deep reinforcement learning (DRL), this highlights the importance of reducing both training and inference time. While early exit neural networks, models that adapt computation to input complexity, have proven effective in supervised learning, their use in DRL remains largely unexplored. In this paper, we propose the use of Budgeted EXit Actor (BEXA), which is a novel actor-critic architecture that integrates early exit branches into the actor network. These branches are trained via the underlying DRL method and use a constrained value-based criterion to decide when to exit, allowing the policy to dynamically adjust its computation. BEXA is general, easy to tune and compatible with any off-policy actor-critic method. We evaluate BEXA using different DRL methods such as SAC and TD3 on a suite of MuJoCo tasks. Our results demonstrate a substantial improvement in inference efficiency with minimal or no loss in performance. These findings highlight early exits as a promising direction for improving computational efficiency in DRL.

## 1 INTRODUCTION

Recent work has demonstrated favorable scaling properties of large neural networks (NNs) in deep reinforcement learning (DRL), (Farebrother et al., 2024; Nauman et al., 2024; Obando-Ceron et al., 2024). However, increasing network depth leads to higher computational costs, making DRL more expensive to train and more difficult to deploy. This is particularly problematic in areas such as robotics, where budget constraints on the inference time must be satisfied. Current methods in DRL try to speed up inference by using model compression techniques like neural network pruning and quantization, reducing the number of model computations while maintaining performance (Zhang et al., 2023). Yet, these methods can be hard to tune and might lead to potential training overhead.

Importantly, a lot of compression methods fail to leverage an inherent property of function approximation in DRL: the computational complexity required for selecting an optimal action varies with the state. For illustration, in chess, finding the best move depends on the complexity of the position. Some positions allow for quick detection of strong moves, while others require extensive computation. Thus, in DRL, where the policy is a neural network, we face a challenge: traditional neural networks are static, performing the same computations regardless of the input. This can lead to inefficiency, since for some inputs an action could be derived with significantly fewer computations.

Early exit neural networks (ENNs) are dynamic NNs that adapt their computational graph based on the input. Originating in fields with high computational demand like computer vision (CV) (Laskaridis et al., 2021) and natural language processing (NLP) (Xu & McAuley, 2023), they work by adding side branches to the network, so-called exits. A gating mechanism decides on which exit to take, adaptively controlling the network depth, enabling a trade-off between performance and efficiency, and potentially improving generalization and interpretability (Han et al., 2022). In many CV and NLP tasks, they have achieved performance comparable to that of their static counterparts while using only a fraction of floating point operations (FLOPs).

Despite their advantages, dynamic neural networks like ENNs have been hardly explored in reinforcement learning (RL). A partial explanation is that naively applying such networks to RL is not possible, as RL has some unique aspects compared to supervised learning. One of the biggest challenges is the lack of supervision, as the agent has to find the correct actions on its own. Typically, early exit branches are trained directly on fixed ground truth data, whereas in RL the behavior of the agent changes over time. In addition, the predicted actions influence the state distribution encountered by the agent. Poorly chosen actions of ENNs can therefore lead to learning instabilities.

In this work, we systematically investigate how to transfer ENNs into DRL. Based on our findings, we propose a new method called Budgeted EXit Actor (BEXA), which introduces early exit NNs with resource-constrained gating based on Q-values to speed up policy inference time during training and evaluation. Our main contributions are as follows:

1. We present BEXA, a general off-policy actor-critic method, with careful adjustments for using ENNs effectively in DRL for reducing the number of required FLOPs during training and deployment.
2. A novel, budget-aware gating mechanism that selects early exits optimally via a linear program, directly balancing expected return and computational cost.
3. We conduct extensive ablation studies to evaluate different design choices for early exit networks in actor-critic methdos.
4. We benchmark our method on well-known DRL methods such as SAC (Haarnoja et al., 2018) and TD3 (Fujimoto et al., 2018) and show the effectiveness of our method on different MuJoCo tasks.

## 2 Related Work

We divide related work into two categories: (i) early exit neural networks (ENNs), which have been primarily explored in domains outside of deep reinforcement learning (DRL), and (ii) methods for accelerating training and inference in DRL, such as model compression and software optimization. Within the second category, we also highlight the few approaches that combine both directions in a manner similar to our work.

### 2.1 Early Exit Neural Networks

Early exit neural networks (ENNs) belong to the family of dynamic neural networks (NNs). These are models that change their computational graph based on the input they receive (Han et al., 2021). Some instances adjust their depth using early exits, while others adjust their width by changing the number of neurons or channels in each layer, or by changing their parameters. We focus on early exit networks because they are conceptually straightforward and have been extensively studied (Scardapane et al., 2020b). Here, we will present only a few noteworthy works and refer the interested reader to comprehensive surveys (Laskaridis et al., 2021; Xu & McAuley, 2023; P et al., 2025).

The first works for these networks include conditional deep learning network (CDLN) (Panda et al., 2016) and BranchyNet (Teerapittayanon et al., 2016). CDLN first trains the backbone network and then adds linear early exits at multiple depths, retaining only those that improve performance. BranchyNet integrates exits into known computer vision (CV) classifier networks and uses an entropy-based criterion to terminate computation early. All exits are trained jointly with a weighted cross-entropy loss. While effective in CV, such entropy criteria are not directly applicable to DRL, where high policy entropy is beneficial for exploration.

More recent work goes beyond simple entropy-based criteria with alternative decision rules. Confidence to exit can be defined by maximum class probability (Huang et al., 2018; Wang et al., 2022) or by patience, exiting when several consecutive branches agree (Zhou et al., 2020; Zhu, 2021), a strategy common in early exit transformers. Beyond heuristics, the decision to exit can also be learned: Demir & Akbas (2024) jointly optimizes accuracy and efficiency to train exits and gates, while Vashist et al. (2022) uses DRL to learn an exit policy using a deep Q-network (DQN), though the underlying task is not a DRL one. The work presented here, can be seen as an extension to tuning the exit selection process with DRL. However, rather than learning gate decisions with reinforcement learning (RL), we formulate exit selection as a resource allocation problem: value estimates from DRL are optimized under a budget constraint via a linear program, which supervises gate policy learning.

Training strategies can also vary. The most common approach is to jointly optimize all exits under a combined loss (Berestizshevsky & Even, 2019; Scardapane et al., 2020a). However, the modular design of early exits also allows for a layer-wise training scheme (Hettinger et al., 2017), where a subset of exits is trained at a time while keeping the rest frozen.

Finally, self-distillation (Zhang et al., 2019) is a variant of knowledge distillation in which knowledge is transferred from a teacher model to one or more student models. Applied to an ENN, the final output layer can be considered the teacher, while the intermediate outputs are the students. These outputs are trained using a combination of a standard supervised loss and an additional imitation loss that encourages the outputs to mimic the teacher's predictions. Previous work has shown that self-distillation can improve model accuracy (Zhang et al., 2022; Pham et al., 2022), and its self-imitation perspective makes it a natural asset for DRL transfer.

### 2.2 Accelerating DRL

Two directions have emerged for accelerating training and inference in DRL. The first targets system-level efficiency through software and hardware optimizations, such as parallelization and the use of accelerators like GPUs. The second approach focuses on model-level efficiency, compressing neural networks that represent policies, value functions and dynamics models.

On the system side, Weng et al. (2022) parallelizes environment simulation with a C++ backend, reducing Python overhead and enabling high-throughput sampling. We adopt this setup in our experiments as well. Pushing this further, Dalton & Frosio (2020) ports Atari to the GPU, yielding even faster parallel roll-outs. Architecturally, IMPALA (Espeholt et al., 2018) decouples acting from learning by running environments in separate processes, each with its own policy, and asynchronously aggregates experience into a shared buffer. SEED RL (Espeholt et al., 2019) refines this design by batching observations from many environments and evaluating a single policy on an accelerator throughout training.

On the model side, NN compression techniques such as quantization (Nagel et al., 2021), knowledge distillation (Hinton et al., 2015), and pruning (LeCun et al., 1989) have proven highly effective in supervised learning for improving runtime. Recent works adapt these techniques to DRL. QuaRL (Krishnan et al., 2022) quantizes policy parameters after each update from 32-bit floating point to 8-bit integers, improving throughput with minimal accuracy loss. FastAct (Zhang et al., 2023) generalizes this idea by supporting arbitrary compression schemes, while a scheduler ensures that compression remains within acceptable limits to maintain performance.

One closely related line of work is RAPID-RL (Kosta et al., 2022), which integrates early exit networks into DQN. It estimates confidence by checking whether an exit's Q-value exceeds a fixed fraction of the maximum Q-value, employs layer-wise training, and reports faster inference on Atari. Our approach differs in three key aspects: (i) we target general actor–critic methods rather than DQN, requiring early exits only for the actor and permitting more flexible critic architectures, (ii) we introduce a novel resource-aware early exit criterion and train it jointly with all exits, and (iii) whereas RAPID-RL primarily reduces deployment-time inference but incurs training overhead by evaluating all exits, our method accelerates both training and inference.

## 3 Preliminaries

Reinforcement learning (RL) problems are commonly formulated as Markov decision processs (MDPs). An MDP consists of an agent interacting with an environment, where the agent follows a policy $\pi(a \mid s)$ that determines the next action given a state $s$. At each time step $t$, the agent chooses an action $a_t$ that is executed in the environment, which, in response, returns the next state $s_{t+1} \sim P(s_{t+1} \mid s_t, a_t)$ according to the transition probability function $P$. Additionally, the agent receives a reward $r_t = R(s_t, a_t) \in \mathbb{R}$, where $R$ is the reward function. This reward is a scalar value that describes the desirability of the given state and the chosen action. The cumulative sum of rewards, known as the return, is defined as $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ where $\gamma \in [0, 1]$ is the dis-

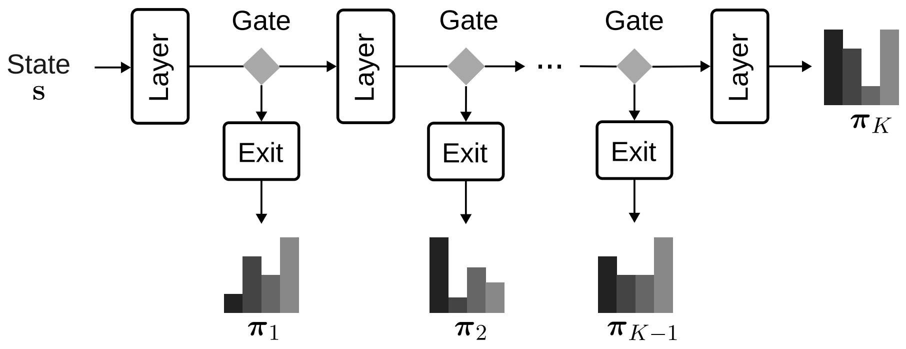


Figure 1: Example of an actor represented as an early exit neural network (ENN). The input state is processed layer-by-layer until a gate is reached. Based on a learned rule, a gate decides whether to terminate early or proceed with the computation. Each exit produces an action distribution for the current input state.

count factor that determines the importance of future rewards. We define the state-value function $V_\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s]$ to calculate the expected return following some policy $\pi$. Similarly, we define the action-value function $Q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a]$ as the expected return if first an initial action $a$ is taken, after which the policy $\pi$ is followed. Given an initial state distribution $\rho_0$, the goal in RL is to find an optimal policy $\pi^*$ that maximizes the expected return $\pi^* \in \arg\max_\pi \mathbb{E}_{s\sim\rho_0}[V_\pi(s)]$.

In deep reinforcement learning (DRL), policies and value functions are typically represented by deep neural network (NN): the policy (actor) $\pi^\theta$ with parameters $\theta$, and the action-value function (critic) $Q^\phi$ with parameters $\phi$. Actor-critic methods jointly learn both networks, where the policy typically maximizes an objective derived from the critic that is of the form $J_\pi(\theta; Q^\phi) = \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[Q^\phi(s, a)]\right]$, where $\mathcal{D}$ is a replay buffer collecting states from interactions with the environment. Many state-of-the-art algorithms, such as SAC (Haarnoja et al., 2018) and TD3 (Fujimoto et al., 2018), are off-policy actor-critic methods, meaning they learn from data $\mathcal{D}$ collected by past policies rather than requiring samples from the current policy.

## 4 METHOD

We now present our framework for integrating early exit neural network (ENN) into off-policy actor-critic methods. The approach is general and can be applied to any actor-critic method with minimal changes to the underlying architecture. We first introduce the early-exit actor architecture, then describe how exit selection is formulated as a budget-constrained resource allocation problem, and finally present the complete algorithm.

### 4.1 EARLY EXIT ACTOR

The key distinction in our approach is that we represent the actor as a deep ENN shown in Fig. 1. During the forward pass, data is propagated sequentially through the network layers. At each side branch, a gating policy decides whether to terminate the computation early. A stochastic gating rule is used instead of a deterministic one to encourage exploration during training and ensure that each exit is occasionally selected. If the gate activates, the corresponding early exit head is evaluated and its prediction is returned without subsequent layers being evaluated, thereby saving computation. Otherwise, the computation continues and the exit is not calculated. To reduce computational overhead, the gating function shares its hidden features with the actor.

We now formalize the architecture mathematically. First, we number the exits sequentially from earliest to last and denote the sub-policy at each exit by $\pi_i$ for $i = 1, \dots, K$. Additionally, each exit before the final layer has a gate policy $g_i(\cdot \mid s) = \mathrm{Bernoulli}(p_i(s))$, where $p_i$ is a learned state-dependent probability parameter; sampling 1 indicates taking the exit, while 0 means resuming.

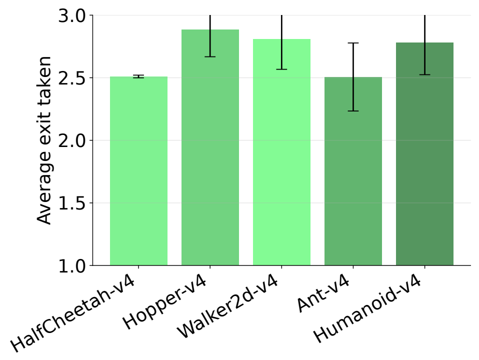


Figure 2: Average selected exit during soft actor-critic (SAC) training with greedy selection of exits based on learned Q-values. Results show the mean of the top five out of 200 agents per environment with one standard deviation. The early exit network has three branches $K = 3$. Without a resource constraint, later exits dominate.

Given a state $s$, the probability $\alpha$ that we terminate at exit $i = 1, \dots, K$ is given by

$$\alpha_i(s) \coloneqq p_i(s) \prod_{j<i} \big(1 - p_j(s)\big), \tag{1}$$

assuming that we define $p_K(s) = 1$. The resulting policy $\pi$ represented by the entire ENN is then a mixture of the exit policies

$$\pi(a \mid s) \coloneqq \sum_{i=1}^{K} \alpha_i(s)\, \pi_i(a \mid s).$$

## 4.2 Learning Budget-Aware Early Exit Actors

We now discuss how to learn the gating policies $g_i$. In the supervised learning setting, ENNs typically rely on confidence-based exit rules such as measuring the entropy of the prediction, maximum class probability or patience, a criterion that exits once at least $n$ consecutive predictions align (Xu & McAuley, 2023). These criteria are ill-suited for deep reinforcement learning (DRL), as high entropy drives exploration, which is crucial for success. Applying confidence-based methods steers behavior toward greedy action selection. Patience is also ineffective because DRL models are usually smaller than those in natural language processing (NLP) and offer much fewer exits. Such methods also introduce task-specific hyperparameters like thresholds that are difficult to tune.

In our reinforcement learning (RL) setting, each exit defines a policy $\pi_i$, and we can compare their performance directly using the expected value $V_{\pi_i}$. A natural idea is to pick the exit that maximizes the expected value in the current state, i.e., $\arg\max_i V_{\pi_i}(s)$. However, this approach tends to favor later exits, as they build upon the representations of previous layers and generally achieve higher returns. This intuition is supported by an experiment shown in Fig. 2, where later exits are selected disproportionately often, resulting in only minimal speedups. Moreover, this method provides little explicit control over the trade-off between performance and computational cost. Thus, we need to explicitly constrain the usage of later exits.

**Optimal Budget-Aware Exit Selection.** Different from approaches that rely on heuristics that are potentially hard to tune, we propose a principled approach that formulates early exit selection as a resource allocation problem. The key idea is that we maximize the expected value of the network's actions while enforcing a hard budget constraint on inference costs.

We assume that each exit policy $\pi_i$ has an associated Q-function $Q_i$. Given a state $s$, let $\mathrm{v} = [Q_1(s, a_1), \dots, Q_K(s, a_K)]^\top$ with $a_i \sim \pi_i(\cdot \mid s)$ be an unbiased value estimate for each exit, and let $\mathrm{c} = [c_1, \dots, c_K]^\top$ denote the per-exit costs, e.g., their floating point operations (FLOPs). For a given budget $b \in \mathbb{R}$, we then solve the following linear program:

$$\alpha^\star = \underset{\alpha \in \mathbb{R}^K}{\arg\max} \quad \mathrm{v}^\top \alpha \tag{2a}$$

$$\text{s.t.} \qquad \mathrm{c}^\top \alpha \leq b, \quad \alpha \geq 0, \quad \mathbf{1}^\top \alpha = 1 \tag{2b}$$

The optimal weighting vector $\alpha^\star$ denotes the optimal probability distribution over exit choices that maximizes the expected value while keeping the total cost within the budget. By rearranging Eq. 1 we obtain the optimal probabilities $p_i^\star$ for each gate. The cost definition corresponds to the resource of interest, for example FLOPs with $c_i \propto \text{FLOPs}(\pi_i)$ though other choices are also possible. For our approach we decided to use normalized FLOP counts with $c_1 = 0$ and $c_n = 1$ and scale the intermediate costs linearly, while still satisfying $c_1 \leq c_2 \leq \cdots \leq c_n$. The scalar budget $b$ specifies a limit on the usable resources. With normalized costs $b \in [0, 1]$ becomes intuitive to scale. As $b$ approaches zero, the gate favors earlier exits, while for $b$ approaching one, the gate prefers the later exits. This yields a direct and tunable trade-off between speed and performance.

Lastly, we note that the linear program in Eq. 2 admits an efficient solution. Since $K$ is small in practice, we can enumerate and evaluate the candidate extreme points quickly. For batches of states $\mathbf{s}$ with corresponding Q values, the computation parallelizes well on the GPU.

**The BEXA Training Objective.** Finally, we present the complete learning framework, which uses the optimal gate probabilities $p_i^\star$ as a supervisory signal. To keep our approach general, we assume that for a parameterized policy $\pi^\theta$ and critic $Q^\phi$ the underlying actor-critic algorithm provides an actor objective $J_{\text{actor}}(\theta; \pi^\theta, Q^\phi)$ that should be maximized with respect to the parameters $\theta$.

For each exit policy $\pi_i^\theta$, we learn a corresponding critic $Q_{\pi_i}^\phi$. Preliminary experiments indicated that maintaining a separate critic per exit substantially improved learning stability. Since the underlying method is off-policy, each critic can be learned from the same stream of data. To avoid the computational cost of $K$ separate critics, we use a single critic with shared features and $K$ heads, one per exit, which adds only minor overhead. Importantly, we do not impose an early exit structure on the critic, which can lead to significant instability during training.

The final Budgeted EXit Actor (BEXA) method trains the complete early exit actor by optimizing an actor-critic objective $J_{\text{actor}}$ and a gate loss $\mathcal{L}_{\text{gate}}$ at every exit. Thus, combined, we maximize the following objective for the actor:

$$J_{\text{BEXA}}(\theta) = \sum_{i=1}^{K} \Big( J_{\text{actor}}(\theta; \pi_i^\theta, Q_{\pi_i}^\phi) - \lambda\, \mathcal{L}_{\text{gate}}(\theta; p_i^\theta, p_i^\star) \Big).$$

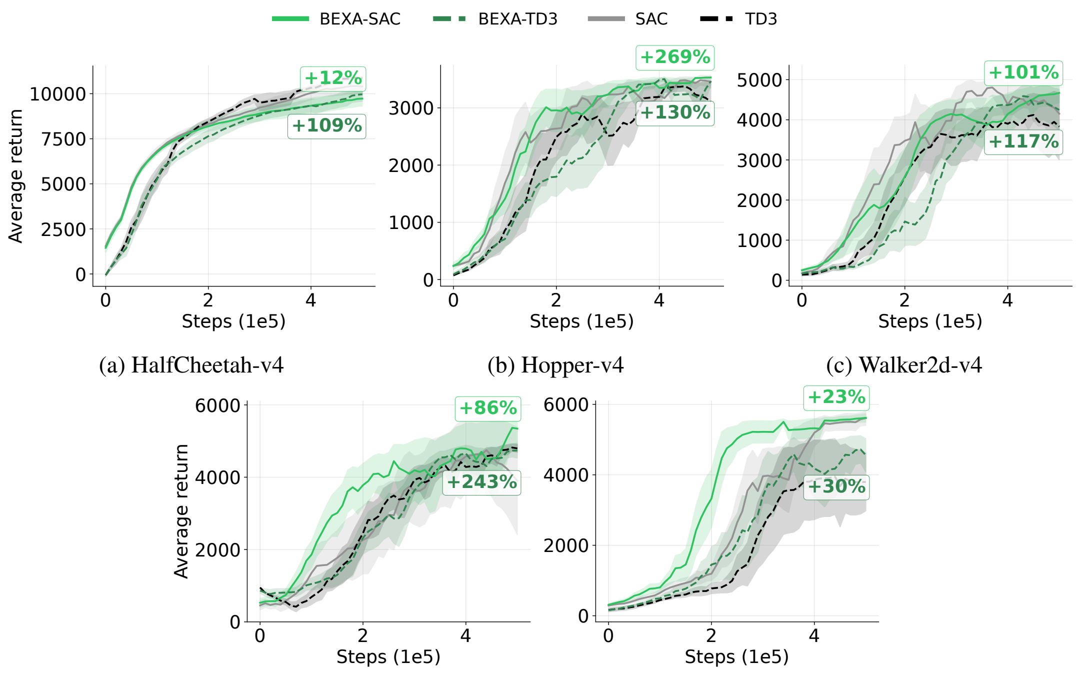
The gate loss $\mathcal{L}_{\text{gate}}$ is a binary cross entropy loss between the predicted gate probabilities $p_i^\theta$ and the probabilities $p_i^\star$ obtained by solving the linear program from Eq. 2.

An illustrative pseudocode description of combining BEXA with SAC is provided in App. B.

## 5 EXPERIMENTS

To validate our proposed approach, Budgeted EXit Actor (BEXA), and to examine the efficiency of different design choices for employing early exit neural networks (ENNs) within actor-critic methods, we conduct two large-scale experiments based on soft actor-critic (SAC) (Haarnoja et al., 2018) and twin delayed deep deterministic policy gradient (TD3) (Fujimoto et al., 2018).

**Setup and Metrics.** For both SAC and TD3, we refer to their variants with budgeted early exits as BEXA-SAC and BEXA-TD3, respectively. Experiments are conducted on five MuJoCo (Todorov et al., 2012) tasks: Ant, Humanoid, Hopper, Walker2d and HalfCheetah. We report training curves with average return and actor inference speedups measured in floating point operations (FLOPs). Actors and critics are represented by two-layer MLPs with 256 units each and ReLU activation. For BEXA variants, we place an exit after every layer, yielding $K = 3$ exits in total. For each method–environment pair we run 200 hyperparameter configurations, where specifications are equal across methods where applicable. To minimize the effect of seed variance, we re-evaluate the top



(a) HalfCheetah-v4 (b) Hopper-v4 (c) Walker2d-v4

(d) Ant-v4 (e) Humanoid-v4

Figure 3: Training curves across MuJoCo tasks. We performed one evaluation run every 10000 environment steps. Curves are smoothed for readability and shaded with 0.5 standard deviation following Fujimoto et al. (2018). Green annotations indicate the average actor FLOP speedup of BEXA variants relative to their baselines over the *entire* training.

5 runs per sweep with two additional seeds, giving us three seeds in total, and select the best by mean return over the entire training. No further tuning was performed. Before averaging the return over multiple environments, we normalize the return per environment. This and more details are described in App. D.

**Results.** Results can be seen in Fig. 3. BEXA matches or exceeds the vanilla baselines, with notable gains on Ant and Humanoid. We hypothesize that these gains stem from a regularization effect of early exits as actor capacity is reduced. Despite introducing new hyperparameters, BEXA required no extra tuning budget relative to its baselines and the budget hyperparameter was straightforward to adapt. Using early exits can accelerate actor inference tremendously with speedups up to $+269\%$ while sampling in the environment, which accounts for a significant part of deep reinforcement learning (DRL) training. For more complex tasks such as Humanoid, the speedup diminishes as the entire network capacity is required to achieve high performance. However, using a more aggressive budget constraint can yield higher speedups, albeit at the expense of performance.

## 5.1 Comparison to Early Exit Alternatives

BEXA improves both performance and speed compared to its baseline methods. It still raises the question of how it compares to alternatives in the literature. As discussed in Related Work, direct comparison is difficult as research on early exit networks in DRL is limited, with most of the existing research being conducted in supervised learning. The approach that comes closest is that described in Kosta et al. (2022), which augments DQN (Hosu & Rebedea, 2016) with early exits, but it targets discrete action spaces, whereas SAC and TD3 use continuous ones. Other DRL acceleration methods, such as quantization and pruning, are orthogonal to our approach and can be combined with it. Benchmarking against these methods offers little insight, especially since methods such as quantization do not reduce FLOPs, but rather the type of operation.

| | Method | Actor Speedup (↑) | | Mean Return (↑) | | Best Return (↑) | |
|---|---|---|---|---|---|---|---|
| | | SAC | TD3 | SAC | TD3 | SAC | TD3 |
| Actor Inference | Backbone | 1.0× | 1.0× | $36.2 \pm 5$ | $33.6 \pm 6$ | $64.1 \pm 16$ | $78.7 \pm 11$ |
| | Ensemble | 1.0× | 1.0× | $26.8 \pm 6$ | $32.7 \pm 9$ | $57.7 \pm 22$ | $73.2 \pm 24$ |
| Exit Training | Imitate | 1.18× | 1.14× | $29.2 \pm 7$ | $34.0 \pm 10$ | $65.9 \pm 23$ | $71.7 \pm 22$ |
| Gate Training | Advantage | 1.164× | 1.179× | $58.3 \pm 12$ | $40.5 \pm 12$ | $99.2 \pm 14$ | $79.6 \pm 28$ |
| | Softmax | 1.11× | 1.12× | $50.3 \pm 16$ | $\mathbf{49.3 \pm 11}$ | $98.7 \pm 18$ | $\mathbf{88.4 \pm 21}$ |
| Train Strategy | Stepwise | **1.48×** | **1.83×** | $27.8 \pm 4$ | $16.5 \pm 4$ | $40.4 \pm 6$ | $42.4 \pm 23$ |
| | BEXA | 1.3× | 1.35× | $\mathbf{62.8 \pm 17}$ | $38.7 \pm 12$ | $\mathbf{101.2 \pm 13}$ | $72.2 \pm 31$ |

Table 1: Evaluation of alternative components for BEXA on MuJoCo using SAC and TD3. We report normalized returns, averaged over tasks and 3 seeds, with error bars indicating one standard deviation.

Instead, we propose baselines derived from early exit architectures in supervised learning, adapted to DRL. To our knowledge, these baselines have not been studied in DRL, though they have been effective elsewhere. We compare them in terms of performance, speed-up, and tuning effort.

**Actor Inference.** During sampling in the environment, we already employ the early exits of our actor to achieve speedups during training. Two alternative inference schemes are also worth considering: (i) always use the final (backbone) exit, which often achieves the best performance and (ii) form an ensemble over all exits as in Sun et al. (2021) leveraging the fact that each branch solves the same task. However, both require full actor inference and thus miss out on acceleration.

**Exit Training.** Instead of using the same loss for every head, we adapt another strategy inspired by self-distillation (Zhang et al., 2019). We train only the final exit (the backbone) with the standard objective and train all earlier exits to imitate its action distribution via an auxiliary imitation loss. This reduces critic complexity, as only one critic is needed for the backbone, but introduces a loss-scale imbalance between the normal loss and the imitation loss, which requires additional hyperparameters.

**Gate Training.** The exit criterion critically affects performance, as it has to reliably pick the best exit while balancing performance and speed for each state. As data sampling and learning are tightly coupled, wrong exiting can lead to catastrophic updates. Common heuristics from literature, such as maximum class probability, entropy thresholds and patience are ill-suited as previously discussed due to exploration and smaller model sizes. We consider:

1. *Advantage over backbone*: Taking the exits that have higher value over the backbone. This is similar to the strategy of taking the exit with maximum Q-value Kosta et al. (2022), but prefers earlier exits.
2. *Softmax over Q-values*: Instead of taking a maximum, we take a softmax over the distribution of Q-values per exit. A temperature hyperparameter controls greediness. This softmax defines the target decision distribution, which we map to gate probabilities via Eq. 1.

Importantly, in App. C we show that our optimal budget-aware exit selection approach allows for direct and intuitive control in the number of FLOPs by selecting an according hard budget constraint. This is significantly harder to achieve with the strategies mentioned above as ablations.

**Training Strategy.** We train all exits and gates simultaneously under a unified objective. Early exit architectures also allow for alternative training schemes. Following Kosta et al. (2022), we also evaluate a stepwise procedure that sequentially trains each exit branch while freezing the rest of the network, starting with the earliest exit until the final one.

**Setup.** For a fair comparison, all methods were given the same hyperparameter search budget. To better observe the effects of individual components, we drastically reduce network capacity to $4 - 16$ hidden units per layer. The best configurations are re-run to obtain three seeds per setting. Returns are normalized for each environment and then averaged across tasks. We compare speedups of actors in terms of FLOPs during the whole training. See Table 1 for results.

**Results.** Using alternative actor inference yields no benefit: performance is similar for TD3 and worse for SAC, and it provides no speedup during training unlike the usage of early exits. The imitation-based training objective also underperforms. BEXA, which trains all heads using the underlying DRL loss, consistently achieves higher returns. Gate-training results are mixed. As expected, for TD3 we observe higher returns as the greedier gating favors later exits, but at the cost of reducing speedup and making the performance–efficiency trade-off difficult. For SAC, BEXA improves both return and speed, suggesting that tighter budget constraints can also act as a form of regularization, boosting performance as well. Lastly, we observe that stepwise training performs poorly. It over-optimized for speedup at the expense of return, and training time increases drastically due to additional gradient steps per iteration. Finally, TD3 and SAC diverge substantially at very low actor capacity, we attribute this to a much narrower hyperparameter region.

## 6 Conclusion

We introduced Budgeted EXit Actor (BEXA), a generic method for off-policy actor-critic methods that uses early exits in the actor to reduce the required number of computations under explicit budget constraints. To guarantee that the budget constraints are satisfied, we reformulate the exit selection as a resource allocation problem, which can be efficiently solved using linear programming. BEXA is straightforward to tune and matches or even outperforms vanilla baselines and adapted early exit alternatives from the literature across a range of tasks.

**Limitations.** BEXA inherits some limitations common to early exit architectures. Training time can increase because all exits must be optimized. To circumvent this, asynchronous training architectures could be used to amortize such costs by decoupling sampling from learning. Furthermore, dynamic branching makes efficient parallelization on GPUs challenging, a problem that affects the broader early exit community, not just deep reinforcement learning (DRL).

**Future Work.** Despite these limitations, BEXA is widely applicable and can be used alongside other acceleration techniques, such as pruning, quantization, and distillation. Future work includes plans to integrate BEXA with additional reinforcement learning (RL) paradigms e.g. model-based RL and scaling to large neural network architectures like ResNets or Transformers (Farebrother et al., 2024), where the additional floating point operations (FLOPs) required by the gating mechanism will be negligible small. In spirit with Sutton's "Bitter Lesson", our aim is to provide general and efficient methods that leverage computation rather than task-specific heuristics, providing a practical foundation for faster and stronger DRL agents.

## References

Konstantin Berestizshevsky and Guy Even. Dynamically sacrificing accuracy for reduced computation: Cascaded inference based on softmax confidence. In Igor V. Tetko, Vera Kurková, Pavel Karpov, and Fabian J. Theis (eds.), *Artificial Neural Networks and Machine Learning - ICANN 2019: Deep Learning - 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17-19, 2019, Proceedings, Part II*, volume 11728 of *Lecture Notes in Computer Science*, pp. 306–320. Springer, 2019. doi: 10.1007/978-3-030-30484-3_26. URL https://doi.org/10.1007/978-3-030-30484-3_26.

Steven Dalton and Iuri Frosio. Accelerating reinforcement learning through GPU atari emulation. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/e4d78a6b4d93e1d79241f7b282fa3413-Abstract.html.

Edanur Demir and Emre Akbas. Early-exit convolutional neural networks. *CoRR*, abs/2409.05336, 2024. doi: 10.48550/ARXIV.2409.05336. URL `https://doi.org/10.48550/arXiv.2409.05336`.

Theresa Eimer, Marius Lindauer, and Roberta Raileanu. Hyperparameters in reinforcement learning and how to tune them. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 9104–9149. PMLR, 2023. URL `https://proceedings.mlr.press/v202/eimer23a.html`.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018. URL `http://arxiv.org/abs/1802.01561`.

Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. SEED RL: scalable and efficient deep-rl with accelerated central inference. *CoRR*, abs/1910.06591, 2019. URL `http://arxiv.org/abs/1910.06591`.

Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, Aviral Kumar, and Rishabh Agarwal. Stop Regressing: Training Value Functions via Classification for Scalable Deep RL, March 2024. URL `http://arxiv.org/abs/2403.03950`. arXiv:2403.03950 [cs, stat].

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1582–1591. PMLR, 2018. URL `http://proceedings.mlr.press/v80/fujimoto18a.html`.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL `http://arxiv.org/abs/1801.01290`.

Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *CoRR*, abs/2102.04906, 2021. URL `https://arxiv.org/abs/2102.04906`.

Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(11):7436–7456, 2022. doi: 10.1109/TPAMI.2021.3117837. URL `https://doi.org/10.1109/TPAMI.2021.3117837`.

Chris Hettinger, Tanner Christensen, Ben Ehlert, Jeffrey Humpherys, Tyler Jarvis, and Sean Wade. Forward thinking: Building and training neural networks one layer at a time. *CoRR*, abs/1706.02480, 2017. URL `http://arxiv.org/abs/1706.02480`.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL `http://arxiv.org/abs/1503.02531`.

Ionel-Alexandru Hosu and Traian Rebedea. Playing atari games with deep reinforcement learning and human checkpoint replay. *CoRR*, abs/1607.05077, 2016. URL `http://arxiv.org/abs/1607.05077`.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. Multi-scale dense networks for resource efficient image classification. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=Hk2aImxAb`.

Adarsh Kumar Kosta, Malik Aqeel Anwar, Priyadarshini Panda, Arijit Raychowdhury, and Kaushik Roy. RAPID-RL: A reconfigurable architecture with preemptive-exits for efficient deep-reinforcement learning. In *2022 International Conference on Robotics and Automation, ICRA 2022, Philadelphia, PA, USA, May 23-27, 2022*, pp. 7492–7498. IEEE, 2022. doi: 10.1109/ICRA46639.2022.9812320. URL `https://doi.org/10.1109/ICRA46639.2022.9812320`.

Srivatsan Krishnan, Max Lam, Sharad Chitlangia, Zishen Wan, Gabriel Barth-Maron, Aleksandra Faust, and Vijay Janapa Reddi. Quarl: Quantization for fast and environmentally sustainable reinforcement learning. *Trans. Mach. Learn. Res.*, 2022, 2022. URL `https://openreview.net/forum?id=xwWsiFmUEs`.

Stefanos Laskaridis, Alexandros Kouris, and Nicholas D. Lane. Adaptive inference through early-exit networks: Design, challenges and directions. *CoRR*, abs/2106.05022, 2021. URL `https://arxiv.org/abs/2106.05022`.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In David S. Touretzky (ed.), *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pp. 598–605. Morgan Kaufmann, 1989. URL `http://papers.nips.cc/paper/250-optimal-brain-damage`.

Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *CoRR*, abs/2106.08295, 2021. URL `https://arxiv.org/abs/2106.08295`.

Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Milos, and Marek Cygan. Bigger, regularized, optimistic: scaling for compute and sample efficient continuous control. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL `http://papers.nips.cc/paper_files/paper/2024/hash/cd3b5d2ed967e906af24b33d6a356cac-Abstract-Conference.html`.

Johan S. Obando-Ceron, Ghada Sokar, Timon Willi, Clare Lyle, Jesse Farebrother, Jakob Nicolaus Foerster, Gintare Karolina Dziugaite, Doina Precup, and Pablo Samuel Castro. Mixtures of experts unlock parameter scaling for deep RL. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=X9VMhfFxwn`.

Haseena Rahmath P, Vishal Srivastava, Kuldeep Chaurasia, Roberto Gonçalves Pacheco, and Rodrigo S. Couto. Early-exit deep neural network - A comprehensive survey. *ACM Comput. Surv.*, 57(3):75:1–75:37, 2025. doi: 10.1145/3698767. URL `https://doi.org/10.1145/3698767`.

Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. Conditional deep learning for energy-efficient and enhanced pattern recognition. In Luca Fanucci and Jürgen Teich (eds.), *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, pp. 475–480. IEEE, 2016. URL `https://ieeexplore.ieee.org/document/7459357/`.

Minh Pham, Minsu Cho, Ameya Joshi, and Chinmay Hegde. Revisiting self-distillation. *CoRR*, abs/2206.08491, 2022. doi: 10.48550/ARXIV.2206.08491. URL `https://doi.org/10.48550/arXiv.2206.08491`.

Simone Scardapane, Danilo Comminiello, Michele Scarpiniti, Enzo Baccarelli, and Aurelio Uncini. Differentiable branching in deep networks for fast inference. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*, pp. 4167–4171. IEEE, 2020a. doi: 10.1109/ICASSP40776.2020.9054209. URL `https://doi.org/10.1109/ICASSP40776.2020.9054209`.

Simone Scardapane, Michele Scarpiniti, Enzo Baccarelli, and Aurelio Uncini. Why should we add early exits to neural networks? *CoRR*, abs/2004.12814, 2020b. URL `https://arxiv.org/abs/2004.12814`.

Tianxiang Sun, Yunhua Zhou, Xiangyang Liu, Xinyu Zhang, Hao Jiang, Zhao Cao, Xuanjing Huang, and Xipeng Qiu. Early exiting with ensemble internal classifiers. *CoRR*, abs/2105.13792, 2021. URL `https://arxiv.org/abs/2105.13792`.

Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *23rd International Conference on Pattern Recognition, ICPR 2016, Cancún, Mexico, December 4-8, 2016*, pp. 2464–2469. IEEE, 2016. doi: 10.1109/ICPR.2016.7900006. URL `https://doi.org/10.1109/ICPR.2016.7900006`.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109. URL `https://doi.org/10.1109/IROS.2012.6386109`.

Abhishek Vashist, Sharan Vidash Vidya Shanmugham, Amlan Ganguly, and Sai Manoj P. D. DQN based exit selection in multi-exit deep neural networks for applications targeting situation awareness. In *IEEE International Conference on Consumer Electronics, ICCE 2022, Las Vegas, NV, USA, January 7-9, 2022*, pp. 1–6. IEEE, 2022. doi: 10.1109/ICCE53296.2022.9730182. URL `https://doi.org/10.1109/ICCE53296.2022.9730182`.

Jue Wang, Ke Chen, Gang Chen, Lidan Shou, and Julian J. McAuley. Skipbert: Efficient inference with shallow layer skipping. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pp. 7287–7301. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.ACL-LONG.503. URL `https://doi.org/10.18653/v1/2022.acl-long.503`.

Jiayi Weng, Min Lin, Shengyi Huang, Bo Liu, Denys Makoviichuk, Viktor Makoviychuk, Zichen Liu, Yufan Song, Ting Luo, Yukun Jiang, Zhongwen Xu, and Shuicheng Yan. Envpool: A highly parallel reinforcement learning environment execution engine. *CoRR*, abs/2206.10558, 2022. doi: 10.48550/ARXIV.2206.10558. URL `https://doi.org/10.48550/arXiv.2206.10558`.

Canwen Xu and Julian J. McAuley. A survey on dynamic neural networks for natural language processing. In Andreas Vlachos and Isabelle Augenstein (eds.), *Findings of the Association for Computational Linguistics: EACL 2023, Dubrovnik, Croatia, May 2-6, 2023*, pp. 2325–2336. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EACL.180. URL `https://doi.org/10.18653/v1/2023.findings-eacl.180`.

Hongjie Zhang, Haoming Ma, and Zhenyu Chen. Fastact: A lightweight actor compression framework for fast policy learning. In *International Joint Conference on Neural Networks, IJCNN 2023, Gold Coast, Australia, June 18-23, 2023*, pp. 1–8. IEEE, 2023. doi: 10.1109/IJCNN54540.2023.10191108. URL `https://doi.org/10.1109/IJCNN54540.2023.10191108`.

Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. *CoRR*, abs/1905.08094, 2019. URL `http://arxiv.org/abs/1905.08094`.

Linfeng Zhang, Chenglong Bao, and Kaisheng Ma. Self-distillation: Towards efficient and compact neural networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(8):4388–4403, 2022. doi: 10.1109/TPAMI.2021.3067100. URL `https://doi.org/10.1109/TPAMI.2021.3067100`.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian J. McAuley, Ke Xu, and Furu Wei. BERT loses patience: Fast and robust inference with early exit. *CoRR*, abs/2006.04152, 2020. URL `https://arxiv.org/abs/2006.04152`.

Wei Zhu. Leebert: Learned early exit for BERT with cross-level optimization. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the*

*Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pp. 2968–2980. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.ACL-LONG.231. URL https://doi.org/10.18653/v1/2021.acl-long.231.

## A Use of Large Language Models (LLMs)

In accordance with ICLR authorship guidelines, we disclose our use of LLMs. GitHub Copilot and ChatGPT were used to provide coding assistance, especially for plotting scripts, and for language editing of the paper. They were also used to identify related work and compare alternative design ideas. All methodological choices, experiments, and analyses were conducted by the authors.

## B Pseudocode

In Alg. 1 we provide pseudocode for Budgeted EXit Actor (BEXA) using soft actor-critic (SAC) as an example. As stated before, BEXA is agnostic with respect to the underlying actor-critic method, which we denote as the base in the algorithm description. The critic updates shown here correspond to those used in SAC.

**Algorithm 1** Budgeted EXit Actor (BEXA)

```
Require: Off-policy base (e.g. SAC or TD3); budget $b$; early-exit actor with exits $i = 1, \dots, K$;
    sub-policies $\pi_i(\cdot \mid s)$; gates $g_i \sim \mathrm{Bernoulli}(p_i^\theta(s))$; critics $Q_i^\phi(s,a)$
 1: Initialize replay buffer $\mathcal{D}$, parameters $\theta, \phi$
 2: for environment step $t = 1, 2, \dots$ do                                ▷ Act with early exits
 3:     Observe $s_t$
 4:     for $i = 1..K$ do
 5:         Compute $p_i^\theta(s_t)$ and sample $g_i \sim \mathrm{Bernoulli}(p_i^\theta(s_t))$
 6:         if $g_i = 1$ then
 7:             $a_t \sim \pi_i(\cdot \mid s_t)$; break
 8:     Step environment, observe $(r_t, s_{t+1}, d_t)$
 9:     Store $(s_t, a_t, r_t, s_{t+1}, d_t)$ in $\mathcal{D}$
10:     for update step $u = 1, \dots, U$ do                                 ▷ Learn from replay
11:         Sample minibatch $\mathcal{B} \subset \mathcal{D}$
12:         (1) Critic update (base-agnostic). For each exit $i = 1..K$:
             Compute a TD target $y_i^{(\text{base})}$ per the chosen off-policy base, e.g. for SAC:
             $$y_i^{(\text{SAC})} = r + \gamma(1-d)\,\mathbb{E}_{a' \sim \pi_i(\cdot|s')}\big[\min_{m \in \{1,2\}} Q_{i,m}^{\bar{\phi}_m}(s',a') - \lambda \log \pi_i(a' \mid s')\big].$$
             Then update $\phi$ by a gradient step on $\frac{1}{|\mathcal{B}|}\sum(Q_i^\phi(s,a) - y_i^{(\text{base})})^2$.
13:         (2) Linear program for exit mixture.
             $$\alpha^\star = \arg\max_{\alpha \in \mathbb{R}^K} v^\top \alpha \quad \text{s.t.} \quad c^\top \alpha \le b, \quad \alpha \ge 0, \quad \mathbf{1}^\top \alpha = 1$$
14:         (3) Map mixture to target gate probabilities.
             Using Eq. 1 to compute $p^\star$ recursively from $\alpha^\star$:
             $$p_1^\star(s) = \alpha_1^\star(s), \qquad p_i^\star(s) = \frac{\alpha_i^\star(s)}{\prod_{j<i}(1 - p_j^\star(s))} \quad \text{for } i = 2, \dots, K.$$
15:         (4) Actor update (BEXA objective).
16:         $\theta \leftarrow \theta + \eta_\pi \nabla_\theta \frac{1}{|\mathcal{B}|}\sum_{s \in \mathcal{B}} J_{\text{BEXA}}(\theta; s, p^\star)$
```

## C Effect of the Budget on Computational Cost and Return

Here, we investigate how we can control the numbers of required floating point operations (FLOPs) using our resource allocation formulation. In Fig. 4 we see how the expected FLOPs linear scale with the normalized budget. Furthermore, Fig. 5 highlights that the performance increases with the allowed budget.

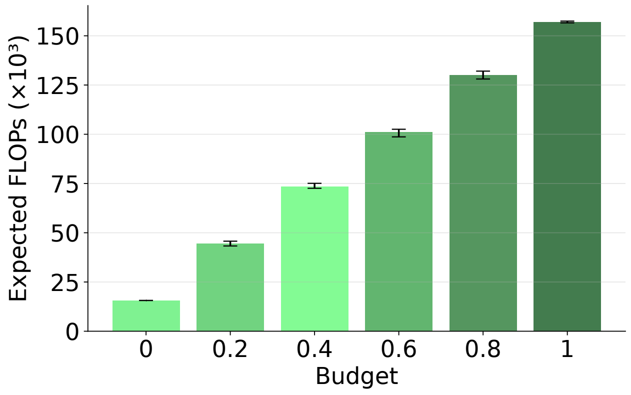


Figure 4: Average FLOPs for the actor in relation of budget $b$ when using `BEXA-SAC`. Evaluated on the Halfcheetah-v4 environment using $\sim$ 70 runs per bar. One standard deviation is plotted. This shows that budget regulates flops explicitly and in a intuitive way.

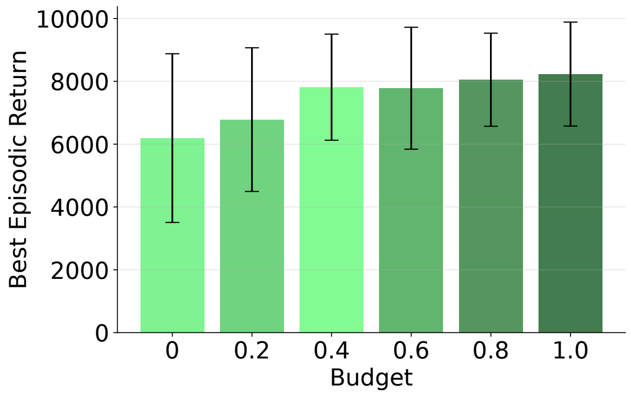


Figure 5: Best Return reported for the actor in relation of budget $b$ when using `BEXA-SAC`. Evaluated on the Halfcheetah-v4 environment using $\sim$ 70 runs per bar. One standard deviation is plotted. Giving more budget allows for higher return.

## D Hyperparameters

To follow best practices (Eimer et al., 2023), we list all relevant hyperparameters and search spaces used in the experiments. For tuning the hyperparameters we used random search. For continuous hyperparameters, we used q-log-uniform, which samples logarithmically and rounds to discrete multiples of a step $q$.

In Tab. 2 and Tab. 3 we highlight the search spaces used for Fig. 3. For the ablation studies presented in Tab. 1, we used the search spaces in Tab. 4 and Tab. 5. Furthermore, to facilitate comparison of performance across environments, we normalize the return when aggregating results, see Tab. 6

| **Hyperparameter** | **Values / Range** |
|---|---|
| batch_size | 256 |
| learning_starts | 5000 |
| policy_frequency | 2 |
| autotune | True |
| gamma | 0.99 |
| tau | q-log-uniform (min: 1e−3, max: 1e−2, q: 1e−3) |
| policy_lr | q-log-uniform (min: 1e−4, max: 7e−4, q: 1e−4) |
| q_lr | q-log-uniform (min: 3e−4, max: 1e−3, q: 1e−4) |
| gate_loss_scale | q-log-uniform (min: 1e−3, max: 1e−1, q: 1e−3) |
| budget | [0.0, 0.2, 0.4, 0.6, 0.8, 1.0] |
| actor_inference | early_exit |
| critic_kind | multi_head |
| actor_training | all_exits |
| gate_training | budget |
| training_scheme | jointly |
| total_timesteps | 500000 |

Table 2: Hyperparameter configuration used for comparison of SAC and BEXA-SAC.

| **Hyperparameter** | **Values / Range** |
|---|---|
| batch_size | 256 |
| learning_starts | 25000 |
| policy_frequency | 2 |
| gamma | 0.99 |
| tau | q-log-uniform (min: 1e−3, max: 1e−2, q: 1e−3) |
| lr | q-log-uniform (min: 1e−4, max: 1e−3, q: 1e−4) |
| policy_noise | [0.1, 0.2, 0.3, 0.4] |
| exploration_noise | [0.1, 0.2, 0.3] |
| noise_clip | [0.1, 0.2, 0.3] |
| gate_loss_scale | q-log-uniform (min: 1e−3, max: 1e−1, q: 1e−3) |
| budget | [0.0, 0.2, 0.4, 0.6, 0.8, 1.0] |
| actor_inference | early_exit |
| critic_kind | multi_head |
| actor_training | all_exits |
| gate_training | budget |
| training_scheme | jointly |
| total_timesteps | 500000 |

Table 3: Hyperparameter configuration used for comparison of TD3 and BEXA-TD3.

| Hyperparameter | Values / Range |
|---|---|
| batch_size | 256 |
| hidden_size | [4, 8, 16] |
| learning_starts | 5000 |
| policy_frequency | 2 |
| autotune | True |
| gamma | 0.99 |
| tau | q-log-uniform (min: 1e−3, max: 1e−2, q: 1e−3) |
| policy_lr | q-log-uniform (min: 1e−4, max: 7e−4, q: 1e−4) |
| q_lr | q-log-uniform (min: 3e−4, max: 1e−3, q: 1e−4) |
| imitate_loss_scale | q-log-uniform (min: 1e−2, max: 4e−1, q: 1e−2) |
| gate_loss_scale | q-log-uniform (min: 1e−3, max: 1e−1, q: 1e−3) |
| gate_loss_freq | [1, 2] |
| budget | [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9] |
| gate_softmax_tmp | q-log-uniform (min: 1e−1, max: 2.0, q: 1e−1) |
| actor_inference | [early_exit, backbone, ensemble] |
| actor_training | [imitate, all_exits] |
| gate_training | [budget, adv, softmax] |
| training_scheme | [stepwise, jointly] |
| total_timesteps | 500000 |

Table 4: Sweep configuration for BEXA-SAC and alternative ablation components.

| Hyperparameter | Values / Range |
|---|---|
| batch_size | 256 |
| hidden_size | [4, 8, 16] |
| learning_starts | 25000 |
| policy_frequency | 2 |
| gamma | 0.99 |
| tau | q-log-uniform (min: 1e−3, max: 1e−2, q: 1e−3) |
| lr | q-log-uniform (min: 1e−4, max: 1e−3, q: 1e−4) |
| policy_noise | [0.1, 0.2, 0.3, 0.4] |
| exploration_noise | [0.1, 0.2, 0.3] |
| noise_clip | [0.1, 0.2, 0.3] |
| imitate_loss_scale | q-log-uniform (min: 1e−2, max: 4e−1, q: 1e−2) |
| gate_loss_scale | q-log-uniform (min: 1e−3, max: 1e−1, q: 1e−3) |
| gate_loss_freq | [1, 2] |
| budget | [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9] |
| gate_softmax_tmp | q-log-uniform (min: 1e−1, max: 2.0, q: 1e−1) |
| kl_eps | q-log-uniform (min: 1e−1, max: 2.0, q: 1e−1) |
| actor_inference | [early_exit, backbone, ensemble] |
| actor_training | [imitate, all_exits] |
| gate_training | [budget, adv, softmax] |
| training_scheme | [stepwise, jointly] |
| total_timesteps | 500000 |

Table 5: Sweep configuration for BEXA-TD3 and alternative ablation components.

| Environment | Normalization (return) |
|---|---|
| HalfCheetah-v4 | 60.0 |
| Walker2d-v4 | 30.0 |
| Hopper-v4 | 30.0 |
| Humanoid-v4 | 50.0 |
| Ant-v4 | 40.0 |

Table 6: Normalization constants used to scale returns for MuJoCo tasks.