# Supporting Information for

## Having "multiple selves" helps learning agents explore and adapt in complex changing worlds

**Zack Dulberg, Rachit Dubey, Isabel M. Berwian, Jonathan Cohen**

**Corresponding Author: Zack Dulberg.**
**E-mail: zdulberg@princeton.edu**

**This PDF file includes:**



**Other supporting materials for this manuscript include the following:**

Movies S1 to S4

## Supporting Information Text

**Hyperparameter optimization.** We performed our initial hyper-parameter search to optimize the monolithic model in an environment with 4 resources that were fixed in the 4 corners of our grid-world. We first adjusted hyperparameters by hand, in order to learn that the learning rate $\alpha$ and the discount factor $\gamma$ were the most important factors when it came to overall performance. We then performed a grid-search over values $\alpha \in [1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-1]$ and $\gamma \in [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99]$. We found that the best setting was $\alpha = 1e-3, \gamma = 0.5$ as in Figure S1. These values were then used for all models and experiments. Other hyper-parameters were adjusted by hand but stayed near typical values, and can be seen in Table S1 for the case of 4 resources. In order to maintain matched number of parameters for greater number of resources / modules, we added the following number of hidden layer units to the baseline 1024 units of the monolithic model: 0,130,250,360,470 for 4,5,6,7,8 modules respectively.

**Example trajectories.** We visualize randomly selected example trajectories in Figure S2 for three different environmental set-ups - 4 stationary resources, 4 non-stationary resources, and 8 non-stationary resources. All environments are non-trivial; agents moving randomly deplete their stats monotonically. Modular agent examples display how all stats eventually reach set-point in all environment set-ups. Monolithic agent examples struggle to varying degrees.

**Drive Parameters.** Different drive surface parameters represent different reward function properties. For example, in the monolithic case, the greater the coefficients $(n, m)$, the larger the relative size of multi-dimensional drive reductions relative to unidimensional ones, which become equal when $n = m$, and reverse in importance when $(n, m) < 1$. As well, with $n > m > 1$, the drive gradient is greater farther from set-point. With $n < m < 1$, the drive gradient is greater closer to set-point. Therefore, we investigated an expanded set of drive parameters which increased the generalizability of our results while offering additional insights into how the shape of the reward function affected the behavior of our agents. We tested parameters (n,m) = (4,2), (3,2), (2,2), (1,1), and (0.8,0.9). (Note that (2,2) and (1,1) are degenerate with respect to the modular model, but we included both anyway).

Figure S3 (left) replicates Figure 2d (main text, stationary environment) for the new parameter settings, with corresponding drive surfaces for reference (2D for monolithic and 1D for modular). The modular agent maintained a very similar trajectory in all settings (achieved homeostasis early in training and remained near set-point with low variance). The overshoot-undershoot pattern in the monolithic agent was preserved, but diminished as $(n, m)$ decreased in a way that worsened performance (i.e. for $(n, m) = (0.8, 0.9)$, the monolithic stats had higher variance and had trouble reaching set-point). Therefore, when the drive surface did not favour multi-dimensional drive reductions, or even favored unidimensional reductions, the monolithic agent did not become equivalent to the modular one.

As well, when the drive gradient was steeper near set-point ($m = 0.8$, $n = 0.9$), it appears that caused instability (since both positive and negative rewards would be higher), and less drive towards set-point when farther away from it, causing higher stat variance and worse homeostatic performance overall in the monolithic agent. Lastly, Figure S3 (right) generalizes our results from Figure 3d (main text, non-stationary environment) over these new drive parameters for two extremes of exploration schedules. Again, the modular agent exhibits better performance across the board, and is relatively insensitive to drive parameters, while the monolithic agent is largely sensitive to the drive parameters, and significantly improves with additional exploration.

**Exploring the intrinsic exploration effect.** We investigated further the intrinsic exploration that emerged in the modular models. First, in a simplified case in which agents had only 2 stats, we examined the statistics of the actions selected by each of the two modules. Figure S4 shows that a module could expect to take its preferred action approximately 55% of the time, and that the chance of taking each of the three other (non-preferred) actions ranged approximately between 10% and 20%. This supports our claim that, from the perspective of each module, exploratory (non-preferred) actions were selected approximately randomly in expectation.

Next, we outline one scenario in which this source of exploration was particularly helpful, and may help explain why slow-moving environments were more difficult for the monolithic agent. Occasionally, agents could get stuck in local minima solutions, in which stats steadily declined for long periods of time. Despite its difficulty, one benefit of the non-stationary environment was that a needed resource could randomly appear in front of the agent, helping it solve this problem. An example is shown for the monolithic agent in Figure S5a (top). The purple 'x' indicates a resource moving into view, so the agent could begin to replenish a stat that was stuck in decline. This helps explain why faster changing environments benefited our agents (i.e., by providing more opportunities to escape these local minima).

As discussed, modular agents also benefit from an intrinsic source of exploration, shown in Figure S5a, (bottom), in which the red dots indicate when another module 'took the wheel,' helping a module find the resource it needed along the way, when its own maximum Q-value was too low to drive the relevant action. Having this additional source of exploration helps explain why the modular agent was less sensitive to the slowly changing resources; Figure S5b shows that a greater proportion of recoveries (defined as a stat declining for at least 200 steps in a row before its resource is found) for the monolithic agent were due to resource location changes, meaning the monolithic agent would be more sensitive to sparsely changing environments (less opportunities for recoveries). Indeed, Figure S5c shows a parametric relationship such that less dependence on resource changes for getting unstuck was associated with better performance for both agents.

Recent work has shown that training a simulated robot with a global reward also decreased its capacity to escape local minima compared with individual limbs learning from local rewards (1). Our results regarding homeostasis, where modularity

is with respect to different internal drives, rather than different effectors, indicate that this type of exploration benefit might be a general principle that applies to a variety of important control problems.

**Elaborated results.** We show average internal stat trajectories for all annealing schedules (a more granular look at Figure 3d in the main text). Figure S6 and Figure S7 show shows these trajectories for a selected range of number of stats and rates of resource movement respectively.

**Control experiments.** Since the modular agent updated each module on each step in the environment, while the monolithic only had a single network to update on each step, a modular agent with $N$ modules sampled $N$ times more batches from memory overall. One important control experiment therefore involved increasing the number of gradient updates taken per step in the environment for the monolithic model. We show this control in the case of 4 stationary resources, in which the monolithic agent performed 4 gradient update steps (using different sampled batches) at one quarter the learning rate per step in the environment. Our results were unchanged, as seen in Figure S8.

Next, we wondered whether the greatest-mass Q-learning decision process (2) we used was uniqely responsible for our results, or whether other decision processes would display similar results. Therefore, we replaced the decision process in the modular model with the following; the variance over each module's action values was calculated as $W_i = \sigma(Q_i(s, a))$ and the module with the highest $W$ was selected to make the decision on that time-step (3). The idea here was that decisions matter most to the module with the widest range of action values (rather than modules for which all actions mattered about the same amount). The same pattern of results emerged as seen in Figure S9, suggesting that modularity, rather than a particular decision process, was the main driver of our reported effects.

In a similar fashion, we tested whether our results were unique to the particular $\epsilon$-greedy exploration versus some other form of action selection. We therefore repeated simulations in non-stationary and scaled-up environments using SoftMax exploration. That is, actions were selected according to probabilities $P(a_i|s) = \frac{\beta e^{Q(s,a_i)}}{\sum_j \beta e^{Q(s,a_j)}}$, and the inverse temperature parameter $\beta$ was either annealed from 0.2 to 20 on the same schedules as we annealed $\epsilon$ originally, or held constant at different values. Figure S10 displays these results; the monolithic agent remains strongly exploration-dependent, and struggles in difficult environments compared to the modular agent, further supporting the generality of our main findings. Finally, displayed at the bottom of Figure S10, we explored to what extent the performance of the monolithic agent might benefit from a more advanced form of uncertainty-based exploration. We therefore used step-by-step TD-error as a proxy for uncertainty, and adaptively annealed the epsilon value for $\epsilon$-greedy exploration so it was proportional (equal) to the TD-error. This did improve performance of the monolithic agent, in a similar fashion to using 10000 initial annealing steps, bringing it closer to, but remaining inferior to, the modular agent (which did not need adaptive annealing, and is shown for comparison in the plot for additional runs of the 1 annealing step schedule).

Next, in order to enhance the opportunity of the monolithic model to make use of attention, we ran a monolithic agent that used a vision transformer rather than an MLP as its backbone. To do this, we used 1x1 patches (i.e. each scalar element of our input vector was passed into the ViT separately to produce a learned embedding, which then were concatenated with a learned positional embedding to be passed to transformer layers). Other than this, we used an out-of-the-box vision transformer (4), trained to output action values and trained identically to our previous models. We used embedding dimension of 64, 3 transformer layers, 6 transformer heads, and output MLP hidden layer dimension of 512 units. This agent still could not outperform the modular agent (Figure S11).

A last note is that the particular reward scaling used, as well as attentional masking, were not crucial for our results in the stationary environment, as a similar pattern of results was observed without masking and whether we used un-normalized rewards, or bounded reward values between -1 and 1 by passing them through a tanh nonlinearity (Figure S12**a** and Figure S12**b**).

**Effect of attentional masking for changing resource locations.** Our solution to the increased difficulty of moving resources is outlined in Figure S12. Panels **a** and **b** show a replication of our results with 4 stationary resource locations. Figure S12**c** shows how this pattern of results degrades for 4 moving resources; both modular and monolithic agents have significant difficulty achieving homeostasis. Our hypothesis was that the modular agent could take advantage of its modularity by learning to ignore irrelevant features (i.e. if many resources were moving around, learning to ignore all but the resource it cares about could help it learn). Based on this, we added a learned attentional mask (described in Methods) on each module, as well on the monolithic model for a fair comparison (Figure S12**d**). As predicted, Figure S12**e** strongly enhanced the capabilities of the modular model, and surprisingly also improved the performance of the monolithic model as well (possibly because it added some additional noise into the system given the L1-regularization loss on the learned mask). Figure S12**f** shows the actual learned values of the attentional masks for the modular agent, showing that each module learns to put weights only on the environmental features that are relevant to it.

**Videos.** We include sample videos of the last 300 training time-steps for a typical modular and monolithic agent in a stationary and a non-stationary environment. Both agents tend to learn an appropriate policy in the stationary environment, while a typical monolithic agent still struggles by the end of training in the non-stationary environment. All videos display agents trained using the 1 annealing step exploration schedule.

**Table S1. Parameter settings for environment and models assuming 4 resources**

| Model | Monolithic | Modular | Environment | |
|---|---|---|---|---|
| Trainable parameters | 1.09e6 | 1.09e6 | # of resources $N$ | 4 |
| MLP hidden layer units | 1024 | 500 | HRRL exponents $(n, m)$ | $(4, 2)$ |
| Learning rate | 1e-3 | 1e-3 | Stat set-points $h_i^*$ | $5$ |
| Discount factor $\gamma$ | 0.5 | 0.5 | Initial stat levels $h_{i,t=0}$ | $0$ |
| Memory buffer capacity | 30k | 30k | Resource locations $\mu_x, \mu_y$ | variable |
| Target network update frequency | 200 | 200 | Resource covariance $\Sigma$ | $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ |
| Batch size | 512 | 512 | Stat depletion per step | 0.01 |
| Initial $\epsilon$ | 1 | 1 | $R_{\text{thresh}}$ | $95\%$ |
| Final $\epsilon$ | 0.01 | 0.01 | mask sparsity weight $\beta$ | $0.0001$ |

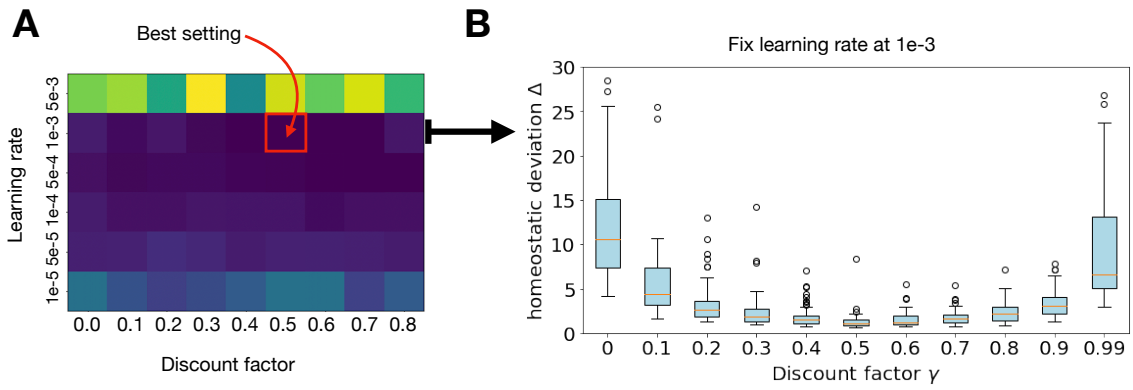


**Fig. S1.** Hyper-parameter search for monolithic model. **a**: Grid-search over learning rates and discount factor. Brightness corresponds to loss; the lowest loss parameter setting of learning rate $\alpha = 1e-3$, discount factor $\gamma = 0.5$ is indicated by the red square. **b**: A sweep of discount factors at $\alpha = 1e-3$ for $N = 30$ models to provide a sense of performance as a function of this parameter. Boxplots display inter-quartile range and outliers

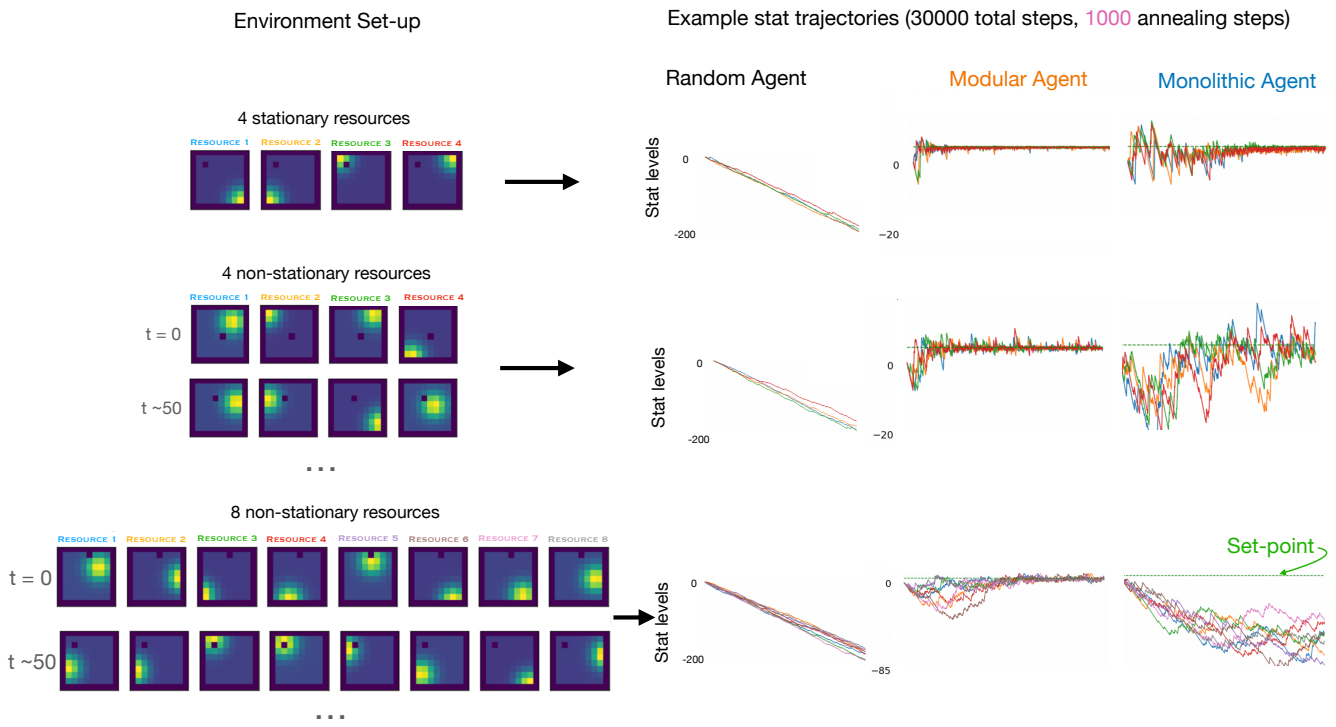


**Fig. S2.** Example learning trajectories for various environment settings. On the left, visual depictions of 3 environment settings are shown; top: 4 resources fixed in 4 corners of grid, middle: 4 resources that eached jumped to a random location according to a poisson process with a rate $\lambda = 0.02$, bottom: 8 resources that eached jumped to a random location according to a poisson process with a rate $\lambda = 0.02$. At this rate, resources are expected to change positions every 50 time-steps on average. On the right, an example trajectory of internal stats are shown for 30000 training steps with 1000 epsiolon-annealing steps for a random agent, a modular agent, and a monolithic agent. Separate colours indicate separate internal stats. The set-point of 5 is inidicated by the green dotted line. The modular agent has consistent performance and tends to achieve set-point for all stats. The monolithic agent struggles as task difficulty increases.

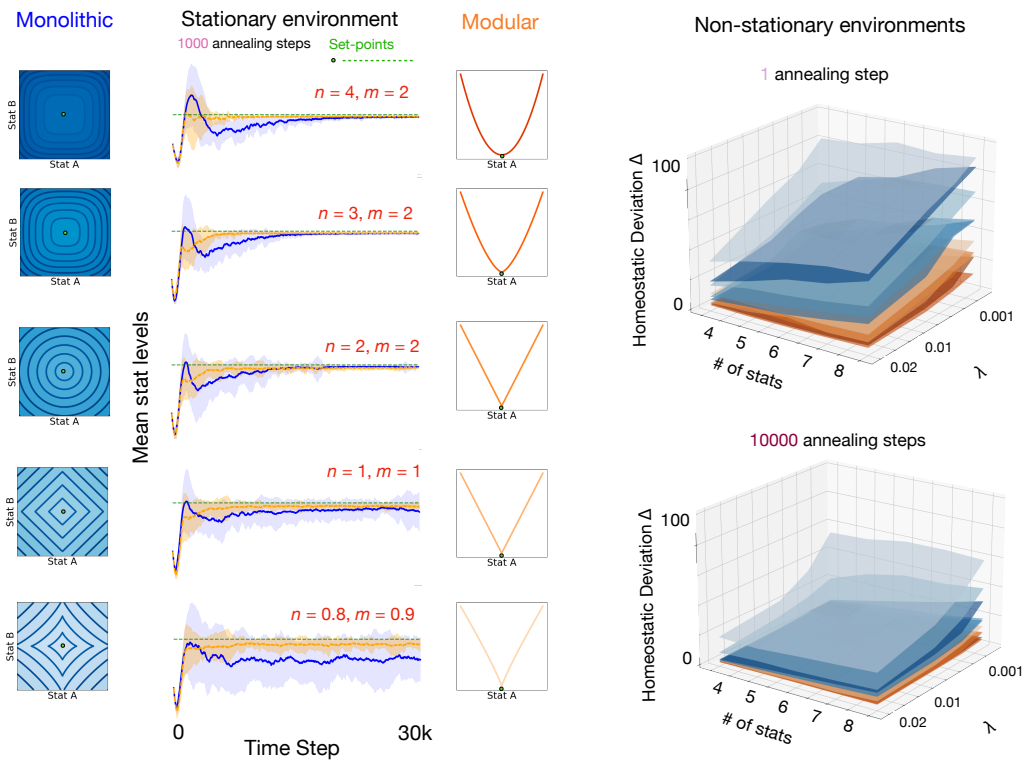


**Fig. S3.** Varying homeostatic drive surface parameters. Left: Replication of Figure 2d in the main text (stationary environment, 1000 annealing steps) over additional settings of drive parameters $n$ and $m$ (red text). Time course of average stat levels during training displayed for monolithic (blue) and modular (orange) agents (N = 30 each; shading shows standard deviation at each time-point, and green dotted line represents set-point of 5 used for all four stats). Corresponding drive surfaces (blue/orange color shading reflecting different $(n, m)$ parameters) are displayed for reference (2D for monolithic, shown to the left of the plots; and 1D for modular, shown to the right). Right: Replication of low and high exploration conditions from Figure 3d (main text, non-stationary environment) over additional settings of drive parameters $n$ and $m$, averaged over N=60 agents (blue/orange color shading reflecting the same $(n, m)$ parameters as the corresponding drive surface schematics on the left). The left and right horizontal axes of each plot represent number of stats and rate of resource location change $(\lambda)$ respectively, and the vertical axis displays homeostatic performance. The modular agent (orange surfaces) exhibit superior performance across all conditions and parameter settings.

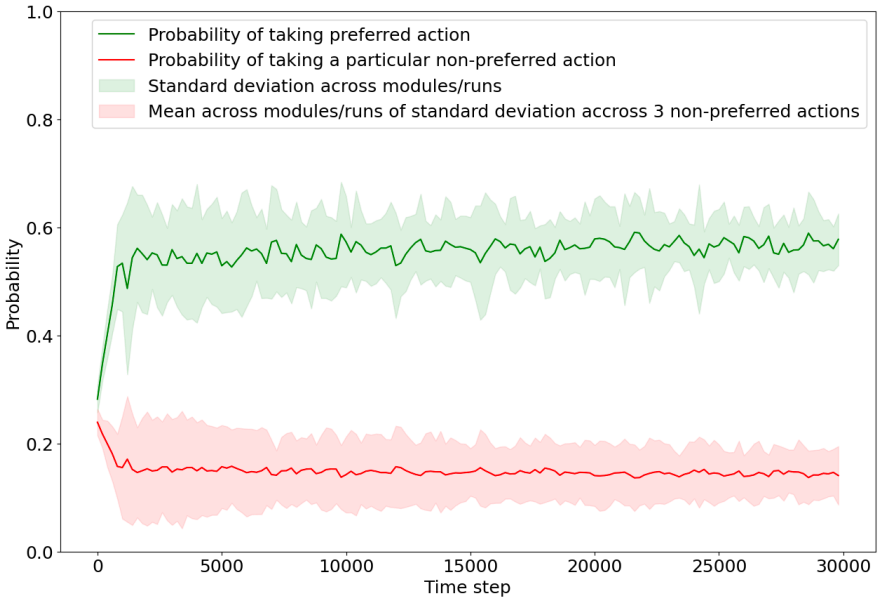


**Fig. S4.** Quantifying action probabilities in the modular agent. We used a simplified non-stationary environment (2 stats, 1000 annealing steps, $\lambda = 0.01$) to calculate the probability of a module taking its preferred action (green) or any particular non-preferred action (red). The red shading represents the average standard deviation across non-preferred actions, showing that non-preferred actions were selected approximately randomly in expectation (i.e. they all had a similar probability of being selected, conditioned on the preferred action of the agent).

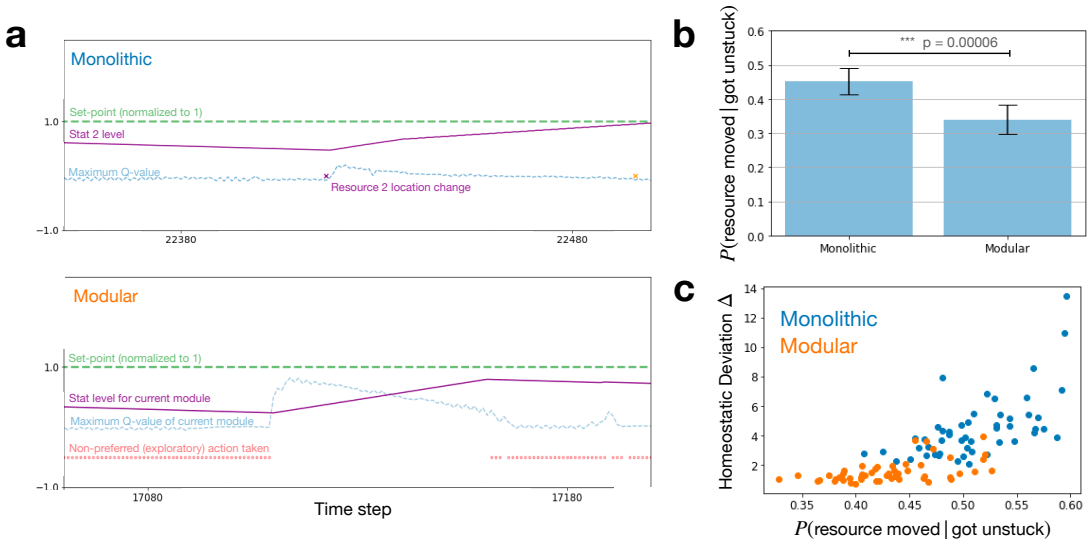


**Fig. S5.** Escaping local minima in changing environments. **a**: Top: Example scenario in which a resource location change (purple x) helps a monolithic agent replenish stat 2 (purple line), that was continually depleting due to an unchanging maximum Q-value (blue dotted line). Bottom: An analogous scenario for a modular agent, but here, non-preferred actions taken by a different module (red dots) help the agent find the resource it needs. **b**: The probability that a resource moved location within 10 steps before an agent 'got unstuck' (defined as a particular stat being replenished after a period of at least 200 continuous steps of decline). Error bars reflect mean and standard deviation for N=60 agents, p-value calculated with two-sample t-test. The modular agent was more likely to have got unstuck without any preceding resource location. **c**: The probabilities from panel **b** correlated with homeostatic performance for both agents; the more an agent relied on resource changes to escape local minima, the worse the overall performance.

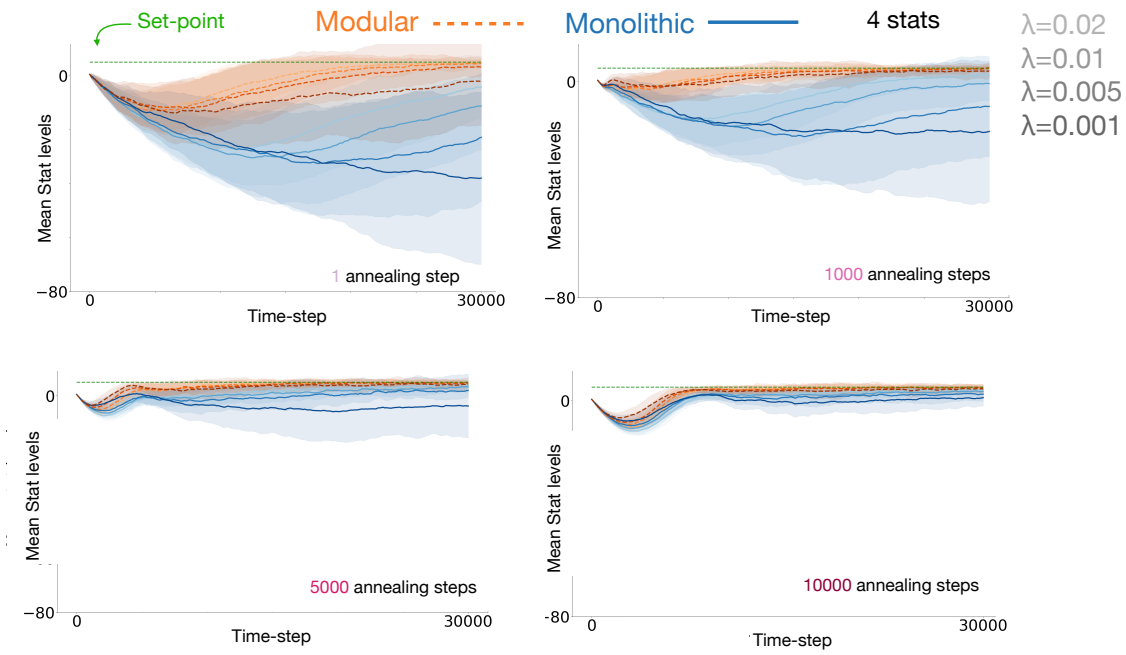


**Fig. S6.** Modular (orange) and monolithic (blue) internal stat trajectories for different amounts of epsilon annealing, with 4 internal stats. Darker colors indicate decreasing Poisson rate of resource location changing (i.e. resources move locations more slowly). Homeostasis is worse for both agents at slower change rates, but the effect is significantly greater for the monolithic agent - the worst case modular agent is better than the best-case monolithic agent in all cases. Shading represents standard deviation across N=100 models.

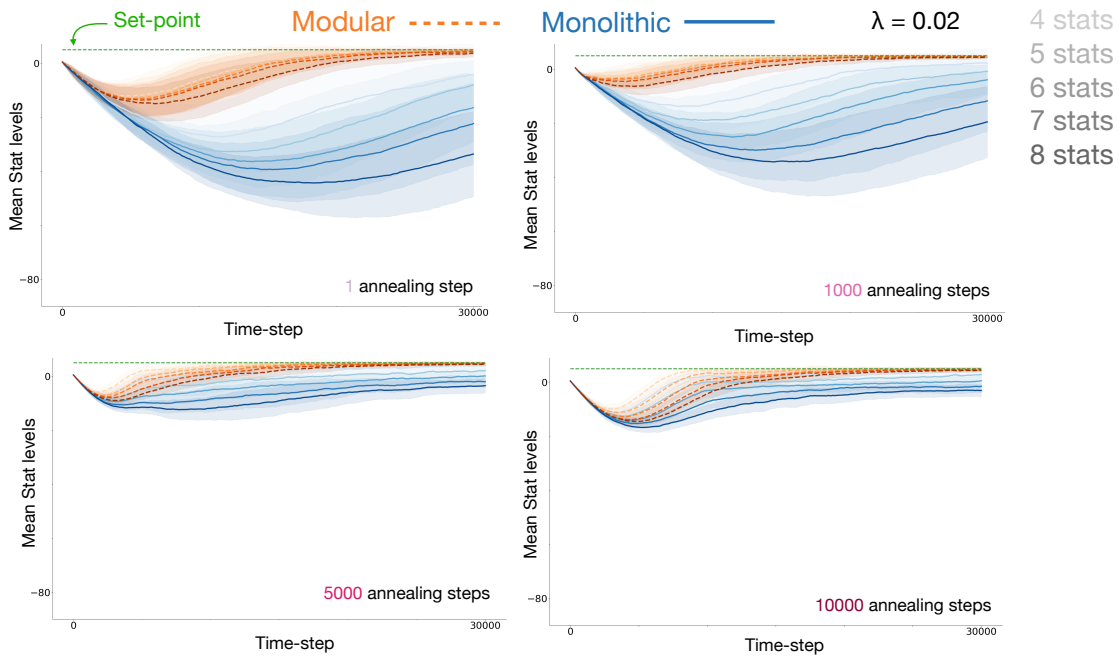


**Fig. S7.** Modular (orange) and monolithic (blue) internal stat trajectories for different amounts of epsilon annealing, resource locations changing at Poisson rate of $\lambda = 0.02$. Darker colours indicate increasing number of needs. Homeostasis is worse for both agents with more internal stats, but the effect is significantly greater for the monolithic agent - the worst case modular agent is better than the best-case monolithic agent in all cases. Shading represents standard deviation accross N=100 models.

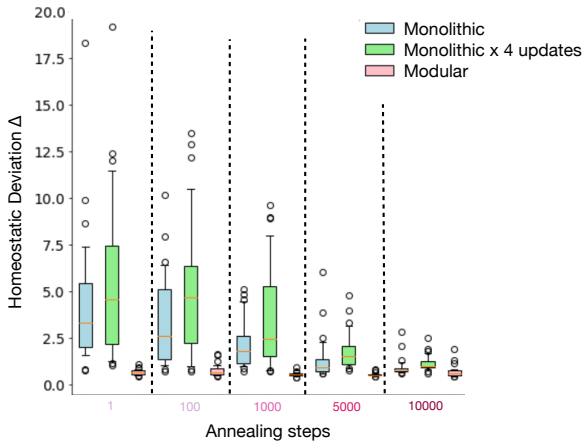


**Fig. S8.** Comparison of modular (pink), monolithic (light blue) and monolithic with 4 gradient updates per step (green) models in a stationary environment where resources are fixed in the four corners of the grid-world. Homeostatic deviation is calculated as in main text (lower is better). The modular agent is relatively unaffected by the exploration annealing schedule, whereas both monolithic models require significant exploration to have comparable performance. Boxplots display inter-quartile range and outliers for N=30 models.

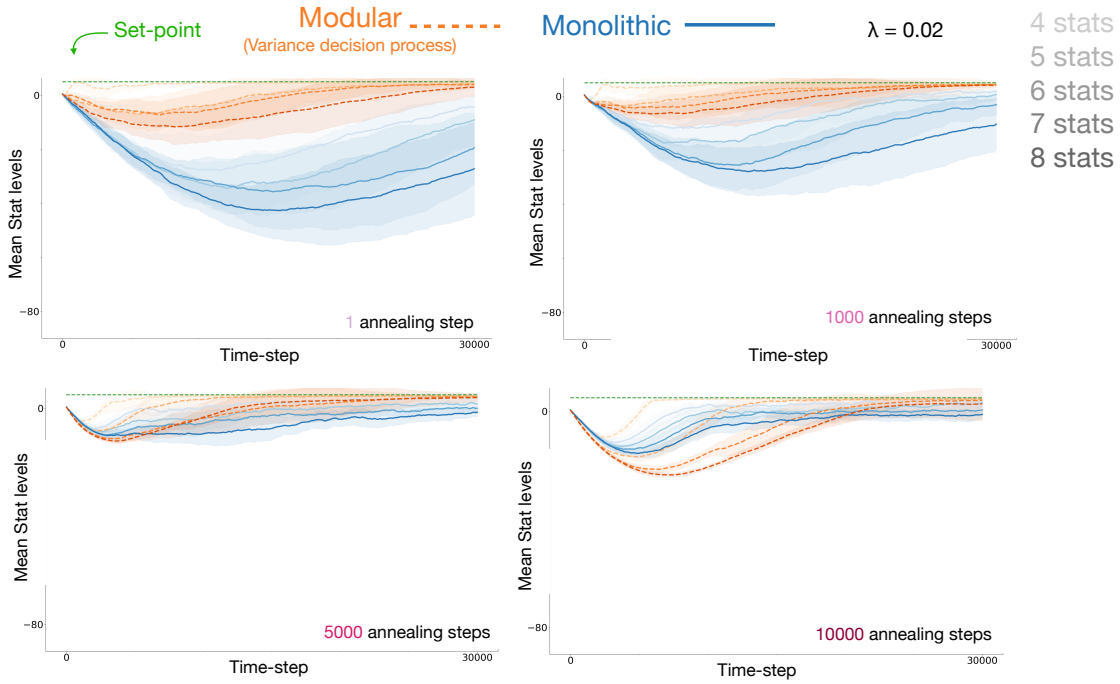


**Fig. S9.** Modular agent using the variance decision process (orange) and monolithic (blue) internal stat trajectories for different amounts of epsilon annealing, resource locations changing at Poisson rate of $\lambda = 0.02$. Darker colours indicate increasing number of needs. Homeostasis is worse for both agents with more internal stats, but the effect is again significantly greater for the monolithic agent - the worst case modular agent is better than the best-case monolithic agent in all cases. Shading represents standard deviation across N=30 models.

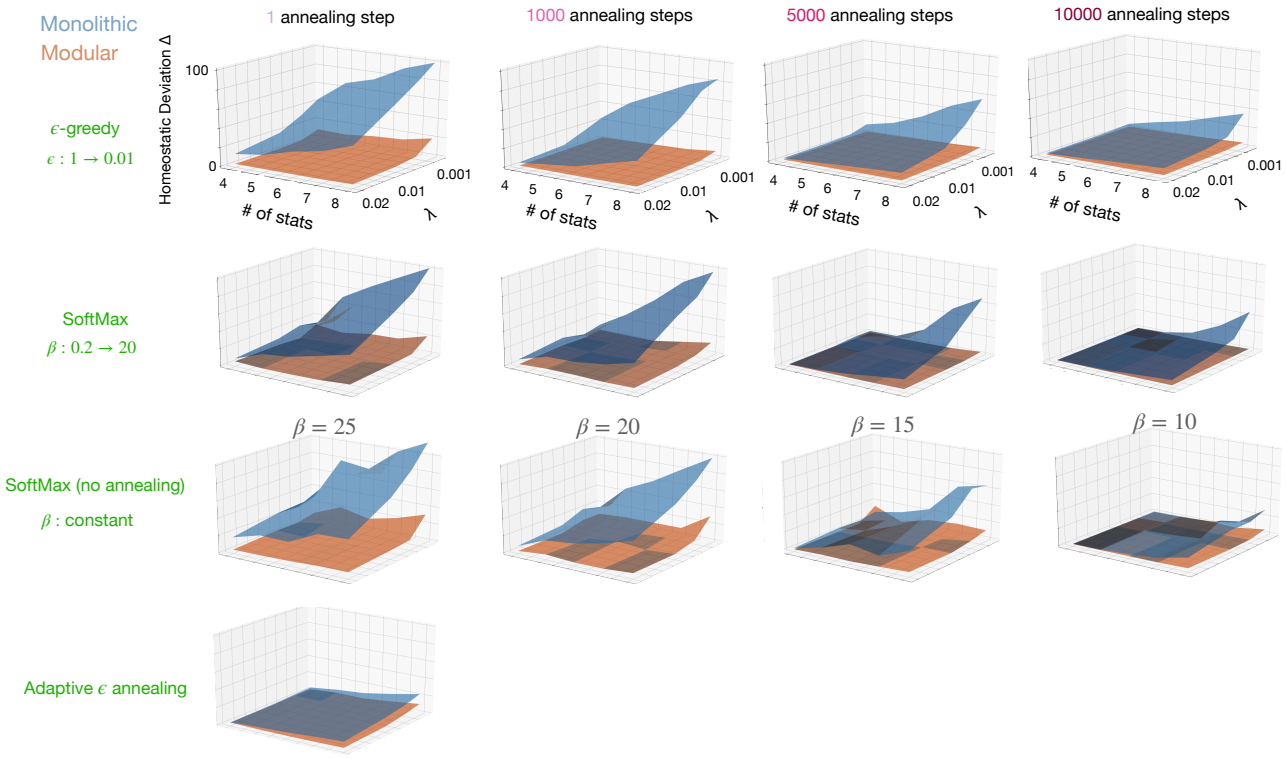


**Fig. S10.** Replication of results using SoftMax exploration. Top row: original results in non-stationary environment (from Fig. 3d). The left and right horizontal axes of each plot represent number of stats and rate of resource location change ($\lambda$) respectively, and the vertical axis displays homeostatic performance. Subsequent rows share axis labels. Second row: Corresponding results using SoftMax exploration with inverse temperature $\beta$ annealed from 0.2 to 20 on the same schedules as originally used for $\epsilon$-greedy. Third row: Results using SoftMax exploration but with constant $\beta$ throughout training of 25, 20, 15, or 10. Bottom: Monolithic agent uses adaptive annealing, such that $\epsilon$ in $\epsilon$-greedy exploration is the temporal difference error the agent experiences at each time step (whereas modular agent again only uses 1 annealing step of standard $\epsilon$-greedy).

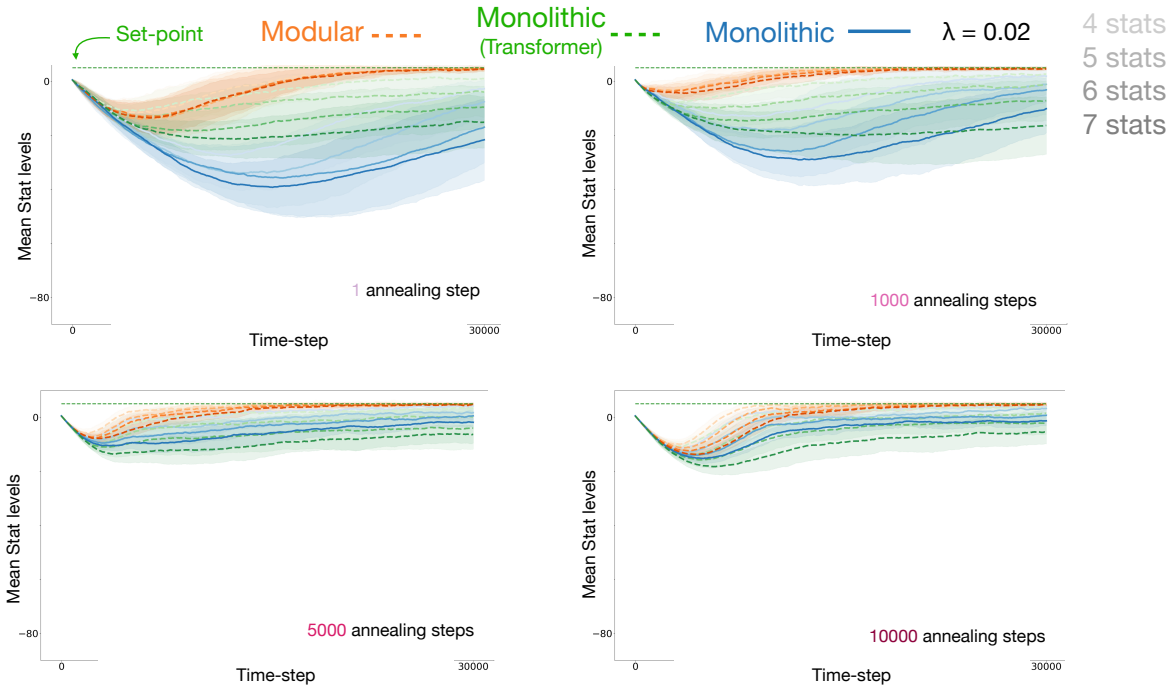


**Fig. S11.** Modular agent (orange), monolithic agent (blue) and monolithic agent using vision transformer (green) internal stat trajectories for different amounts of epsilon annealing, resource locations changing at Poisson rate of $\lambda = 0.02$. Darker colours indicate increasing number of needs. The modular agent is still the best performing model. Shading represents standard deviation across N=30 models.

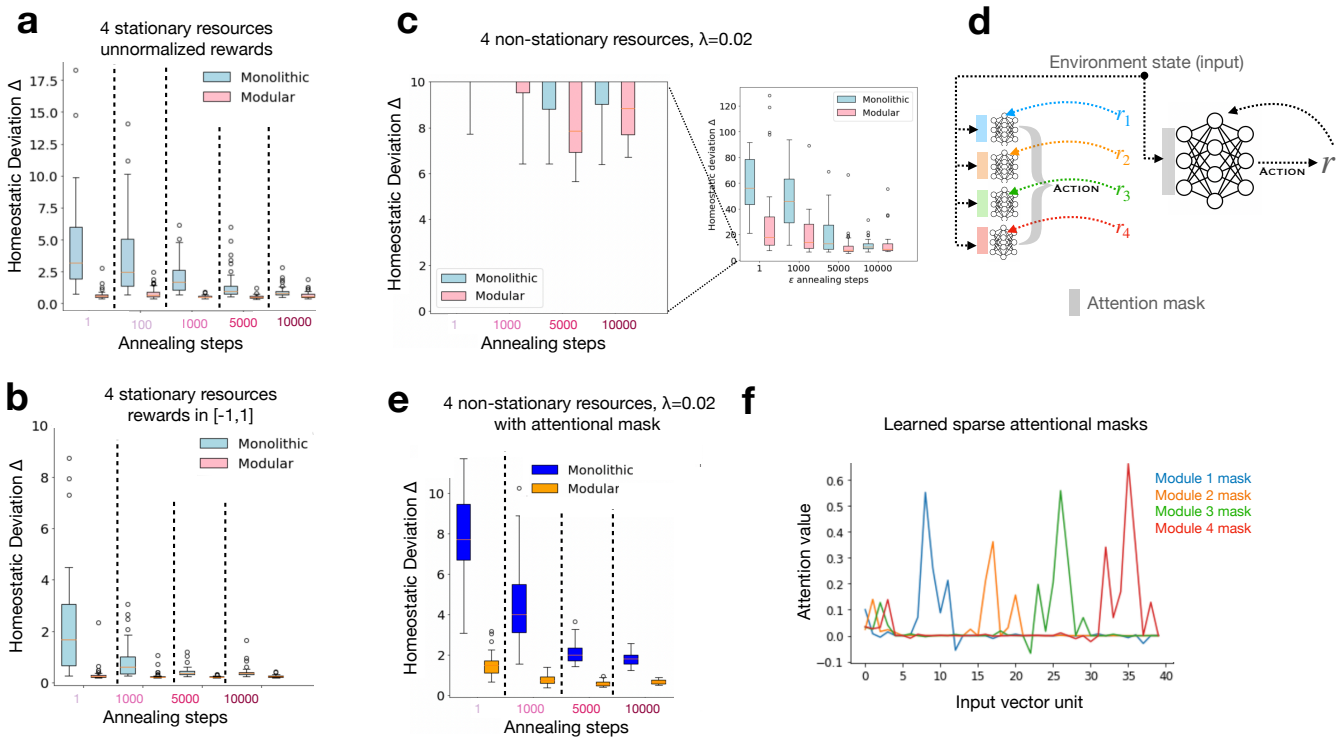


**Fig. S12.** Effect of learning an attentional mask. Lightblue/pink color schemes are used for results without attention, blue/orange scheme indicates the addition of attention. **a**: replication of results in stationary environment using unnormalized rewards (N=30). **b**: Original stationary environment results reported in main text. **c**: Using the same settings as in **b**, the degraded performance of monolithic and modular models for non-stationary resources locations. Inset shows the complete y-axis scale. **d**: Learned attention masks were vectors the same size as the state input vectors, that were element-wise multiplied with the inputs before they were passed into the DQN networks. **e**: Results as reported in the main text with attention masking. **f**: Learned values of the 40-element attention masks for the modular agent after training. The first 4 elements of the input vector are 4 internal stats, and the following 4 sets of 9 elements are the 3x3 egocentric views of the agent of each resource map in order. It can be seen that the attention values for module 1, for example, are highest on internal stat 1 and on resource map 1, and fall to 0 elsewhere.

**Movie S1. Monolithic agent in stationary environment, final 300 training steps. Top: Location of resources (yellow) and agent (moving pixel). Middle: State-value (i.e. maximum Q-value) for each agent (or sub-agent) calculated at each grid location. Bottom: Internal stat levels over time.**

**Movie S2. Modular agent in stationary environment, final 300 training steps. Top: Location of resources (yellow) and agent (moving pixel). Middle: State-value (i.e. maximum Q-value) for each agent (or sub-agent) calculated at each grid location. Bottom: Internal stat levels over time.**

**Movie S3. Monolithic agent in non-stationary environment, final 300 training steps. Top: Location of resources (yellow) and agent (moving pixel). Middle: State-value (i.e. maximum Q-value) for each agent (or sub-agent) calculated at each grid location. Bottom: Internal stat levels over time.**

**Movie S4. Modular agent in non-stationary environment, final 300 training steps. Top: Location of resources (yellow) and agent (moving pixel). Middle: State-value (i.e. maximum Q-value) for each agent (or sub-agent) calculated at each grid location. Bottom: Internal stat levels over time.**

## References


1. M Schilling, A Melnik, FW Ohl, HJ Ritter, B Hammer, Decentralized control and local information for robust and adaptive decentralized deep reinforcement learning. *Neural Networks* **144**, 699–725 (2021).
2. SJ Russell, A Zimdars, Q-decomposition for reinforcement learning agents in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. pp. 656–663 (2003).
3. R Zhang, Z Song, D Ballard, Global policy construction in modular reinforcement learning in *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 29, (2015).
4. A Dosovitskiy, et al., An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).