## A. Continuous Multi-Embodiment Tokenizer Details

As stated in Section 4.2, we train a Residual VQ-VAE (RVQ) model to convert continuous actions from diverse robot embodiments into a shared discrete token space. In our experiments, we use 2 codebooks, each with 512 tokens, and the token vector dimension is 1024. Actions are encoded into these latent codebooks via a 4-layer MLP with 4096 hidden dimensions per layer. The decoder uses the same MLP architecture but now inputs the 1024-dimension latent and outputs the padded action. Refer to Lee et al. [40] for further details about the RVQ method. We map these 512 tokens for both of the 2 codebooks to tokens $30000 - 30512$ of the LLM vocabulary. Since these tokens are non-English tokens for all the LLMs we consider and all our tasks use English instructions, we use this same token range for all MLLM experiments. All actions are padded to be 21 dimensions during tokenization. During detokenization, the 21 dimension continuous vector is truncated from the right to fit the expected action dimension for the environment.

The RVQ tokenizer is trained with MSE loss using all datasets with continuous actions from the overall set of datasets used to train GEA described in Appendix D. The commitment loss from the RVQ is weighted by $1.0$ relative to the MSE loss. The tokenizer is trained with a per-GPU batch size of 256 across 8 H100 GPUs. The model is then trained for 15k updates using the AdamW optimizer [53] with a cosine learning rate decaying to 0 at the end of training and a linear learning rate warmup schedule for the first $10\%$ of training. At this number of updates, the validation MSE loss on unseen actions had largely plateaued at $\approx 0.0038$ averaged across all datasets and $0.002 - 0.007$ depending on the dataset. This trained RVQ tokenizer was then used to train and evaluate all models with continuous actions.

## B. SFT Training Additional Details

To limit the number of tokens per frame to be processed by the LLM, we use the "video" encoding strategy from LLaVA-OneVision. This does not use the AnyRes technique and also applies bilinear interpolation to reduce the number of tokens per image. This results in 196 tokens for each image after being processed by the visual encoder and downsampled. For the training sequence format described in Section 3.2, we use the following prompt format where per-domain strings are manually defined and substituted into each bracketed statement.

```
User:
Agent: <agent description>.
Actions:
Simulator: <simulation platform>
Camera: <camera details>
Instruction: <task instruction>

Agent:
```

When forming training batches, we randomly sample trajectories, and then randomly sample a span of the appropriate context length from that trajectory, and then use the sampled span of observation and actions as an element of the data batch. In the case of the interactive data, the model is only trained to predict the action tokens; the loss for the prompt, instruction, and image tokens is masked out.

For the details about the datasets used in this training process, see Appendix D.

## C. RL Training Additional Details

In this section, we list additional RL training details omitted from Section 4.4. The value function is a 4-layer MLP with a hidden dimension of 2048. The value function takes as input a single vector from the mean-pooled visual tokens from the visual encoder, the final activation of the LLM for the observation, and any task-specific state information that is available. For LoRA finetuning, we use a value of $r = 128, \alpha = 32$, and a dropout value of $0.1$. We use GAE return estimation with $\tau = 0.95$ and a discount factor of $\gamma = 0.999$.

We run RL with the Habitat Pick, LangR, and Procgen environments. Each GPU runs a different benchmark instance. Both Habitat Pick and Procgen are allocated to run on $50\%$ more GPUs than LangR because we found that LangR learns much faster with RL compared to the other tasks. Per benchmark variation, such as different episodes in Habitat Pick and different games in Procgen, are equally divided between the GPUs assigned to that benchmark. On each GPU, a parallelized set of 6 environments are running for batched environment experience collection. We update with 2 PPO epochs, 6 minibatches per epoch, and a clip parameter of $0.2$. We also use the AdamW optimizer for RL training but without any learning rate schedule. We now detail the RL considerations specific to each environment.

In Habitat Pick, we use the same environment details as from prior works using the same task [26, 79, 81]. The task requires the agent to pick up a target object specified by name. Specifically, the agent starts within 2 meters of the target object and faces towards the receptacle the target object is on but with random noise $\mathcal{N}(0, 1.57)$ applied to the direction of facing directly at the receptacle. The task ends in failure if the agent excessively collides with the scene, drops the object, or picks up the wrong object. The maximum number of steps per episode is 300 steps. We use the same reward as the RL-trained pick skill from Szot et al. [78] where a dense reward is provided for moving the end-

effector closer to the object, a positive bonus for picking up the object, and then negative penalties for collisions. We use the 50k training episodes from Szot et al. [79]. Note that these training episodes used for RL training are distinct from the episodes used for testing, which are in unseen house layouts.

For the LangR environment, we use the RL environment details from Szot et al. [80]. The reward function for this environment involves a sparse reward for completing the task, subgoal rewards for completing individual parts of the task, and a slack penalty to encourage completing the task faster. This environment has a maximum of 32 steps per task. In LangR, memory is important because the agent must explore to find certain objects. We therefore increased the number of visual observations in the context to 16 for the LangR RL training. We achieve this by increasing the visual encoder bilinear interpolation factor to produce only 32 visual tokens per image observation. Implementing such environment-specific policy tweaks is simpler in the RL training phase because each GPU worker runs a distinct environment.

For Procgen, we use the RL environment details from Cobbe et al. [15]. Importantly, we train over all 16 of the Procgen games at once using RL. We use the standard per-game reward functions.

## D. Dataset Details

**MetaWorld**: We use the Metaworld simulator [99] with the pre-defined MT-45 split, which consists of 45 training tabletop manipulation tasks using a Sawyer XYZ arm. Each of the 45 tasks has a language instruction associated to it, e.g. "Open the drawer", which we use in all experiments. As inputs to the generalist agent, we use RGB observations from the `corner3` camera resized to GEA resolution. We use no proprioceptive information.

The dataset consists of 500 trajectories from each of the 45 train tasks. To construct the train, validation and test datasets, we rollout the scripted expert policy 10 times for each task's starting state until the first success, to obtain sufficient data diversity. We then assign some of the starting states from the 45 training tasks to the validation set and the remaining trajectories to the training dataset.

**Atari**: We use the DQN replay dataset [2], which consists of 50 million transitions collected while training the DQN [61] algorithm on each of the 44 environments separately. Based on the findings of Multi-Game Decision Transformer [41], we construct the SFT training dataset to be the top-10% of the DQN replay data by trajectory returns, over 5 random seeds and 50 splits. We do not use 100% of the data since it is suboptimal to include low-return trajectories in the SFT dataset. However, if one would like to run offline RL on the data, they should include all of the available data.

**BabyAI**: This dataset consists of 50k trajectories split equally between each of the BabyAI tasks from Chevalier-Boisvert et al. [14]. These trajectories are collected by the shortest path expert provided in the code release for Chevalier-Boisvert et al. [14]. The observations are top-down RGB renderings of the image at $336 \times 336$ resolution.

**Procgen**: We use the training datasets across all Procgen games provided from Mediratta et al. [59]. This dataset consists of 10M observation-action transitions between all the games collected by a PPO expert policy that was individually trained on each game. These transitions amount to around 320k trajectories.

**CALVIN**: We use the standard CALVIN $ABC \rightarrow D$ dataset provided by Mees et al. [60], which are language-labeled task instructions collected by human teleoperation of the robot. This dataset consists of around 18k demonstrations. The observations are $200 \times 200$ RGB renderings from the robot head camera. Note that unlike prior work [48], we do not use the gripper camera or any robot proprioception from this dataset. The actions in the dataset are 6D end-effector control for the relative orientation and position and the gripper state.

**LangR**: For this work, we collect 150k demonstrations for each of the 150k unique training episodes defined by Szot et al. [80]. We collect this data by utilizing the RL-trained policy from [80]. This policy achieves high performance on the training set of instructions (around 98% success rate), and we collect 1 successful demonstration per training episode. The observations are $336 \times 336$ RGB images from the robot head camera. The actions are between 70 high-level skills that include picking up objects by name, navigating to receptacles, placing on receptacles by name, and opening and closing receptacles by name.

**Habitat Pick/Place**: We collect 50k demonstrations for each of these tasks via an expert policy trained with RL. We use the same setup as from the skill training in Szot et al. [78] but operate from an object class to pickup rather than the original geometric goal task specification. This expert policy was trained with the ground truth simulator state, consisting of the relative position of the target object to the robot's end-effector. The expert is trained for 100M steps until convergence. The performance of both experts on the unseen episodes are displayed in Table 2 as baselines.

For the online learning experiments from Section 6.2, we construct the training dataset as follows. We use the GEA-Base checkpoint to collect 10,000 trajectories with action sampling to allow for suboptimal trajectories to be added to the data. The rollouts policy has a success rate of about 50%, meaning that half of the trajectories can be used for SFT from successful trajectories, and all of them can be used for offline RL. When training the SFT policy, we restrict the loss to be optimized only over successful trajectories, as SFT cannot learn from suboptimal data. In addition

to observations and actions, we also log the reward values which are required for offline RL.

**Habitat Nav**: We collect 13k shortest path demonstrations of an agent navigating to receptacles by name in the house. The navigation is performed by an oracle shortest path agent.

**Maniskill**: These datasets are from the released imitation learning datasets from Gu et al. [22]. They are generated via a motion planning algorithm. We only utilize the RGB 3rd-person RGB camera. We generate data for the "StackCube", "PegInsertionSide", "PlugCharger", "PushCube", and "PickCube" tasks. The dataset from Gu et al. [22] has 1k demonstrations for each of these tasks.

**Android Control**: This is the dataset from Li et al. [47] consisting of 14k human demonstrations of using apps to accomplish UI control tasks. Each trajectory has a unique language instruction. The data spans 833 apps. The original images are $1080 \times 1920$, corresponding to the phone screen size. Since we do not use any AnyRes techniques in the MLLM visual encoder, we resize the images to be square before inputting them into the MLLM visual encoder. Using AnyRes with image crops to account for these image aspect ratios could improve the performance of GEA. Actions are generally represented as the name of the action type (like "tap", "scroll", or "input text") followed by the argument for that action where applicable. For tap action arguments, we discretize the original $1080 \times 1920$ screen into $50 \times 50$ patches. A tap action argument is represented as two integers representing the horizontal and vertical patch coordinates where the tap occurred on the screen. All these actions are represented as textual tokens and are separate from the continuous action tokens.

**OpenX**: The Open X-Embodiment dataset consists of numerous individual datasets spanning different robots. We include the following individual datasets from OpenX: Austin Buds dataset, Austin Sailor dataset, Austin Sirius dataset, Berkeley Cable Routing, CMU Stretch, DLR EDAN Shared Control, Fractal, IAMLab CMU Pickup Insert, Jaco Play, Kuka, UCSD Kitchen Dataset, UTAustin Mutex, Dobbe, FMB, RoboSet and Spoc.

**Vision language instruction data**: We use the datasets described in Section 5.

When sampling batches for any SFT training, we weight each dataset defined in Table 1 equally, except OpenX is upsampled 2x, Procgen 2x, and all the VQA data is upsampled 3x.

## E. Evaluation Details

Overall, we use the standard evaluation settings per environment as defined by the prior work. We detail the evaluation settings for each environment below:

**MetaWorld**: We evaluate the generalization performance of our agent on Metaworld unseen starting states over 5 episodes each, totaling to 450 evaluation trajectories in total. We use the simulator's notion of success to define the success rate.

**Atari**: We evaluate the in-distribution performance of our model on 44 Atari games, by using the same setup as the DQN replay dataset, which in turn relies on the Dopamine [12] framework. Specifically, we use the `{GameID}NoFramskip-v4` version of the ALE simulator [6]. We conduct 10 rollouts over all 44 Atari games and average their respective human-normalized scores. The human-normalized scores follow the same protocol as [41, 69]:

$$\text{score}_{\text{normalized}}(s) = \frac{|s - \text{score}_{\text{random}}|}{\text{score}_{\text{human}} - \text{score}_{\text{random}}}$$

Through our experiments, we have found that GEA-Base performs better according to the human-normalized average score than GEA (40.3 vs 32.71). The result is not surprising, as we do not perform any Atari online RL finetuning on the GEA-Base model, and hence, performance can degrade at the expense of much better performance on Habitat Pick and Procgen. To solve this issue, one should co-train on all tasks that support fast online simulation. However, since Atari is structurally similar to Procgen, and Procgen supports procedural level generation to test generalization, we choose not to perform online RL finetuning on Atari.

**BabyAI**: We evaluate each task from Chevalier-Boisvert et al. [14] and evaluate over 100 random episodes for each of the 17 tasks, resulting in 1700 total episodes for the numbers reported in Table 2. Each episode has a different environment state and new language instruction.

**Procgen**: We follow the test evaluation setting Mediratta et al. [59] which uses the "easy" mode setting of the game. We evaluate 50 episodes for each of the 16 games. Like Atari, the performance is evaluated in Procgen via the min-max normalized per-game scores from a random policy and the a PPO expert policy using the reported scores from Cobbe et al. [15].

**CALVIN**: We use the $ABC \rightarrow D$ evaluation setting. This means that during our evaluation, the table background and the language instructions are unseen. We report results over the full 1k evaluation episodes defined by Mees et al. [60].

**LangR**: We follow the standard evaluation settings from Szot et al. [78]. Like the original work, our numbers are reported over the 9 unseen language instruction splits. We evaluate in 100 episodes for each of the 9 evaluation splits. Each test episode is also in an unseen house layout.

**Habitat Pick/Place/Nav**: We follow the evaluation split and settings from Szot et al. [78] and evaluate the agent for 500 episodes in unseen house layouts. This version of the task where the agent has to operate from a language instruction of which object to pick is a harder version of the orig-

inal geometric goal task and has been employed by prior works [26, 81].

**Maniskill**: We use the standard evaluation settings from Gu et al. [22]. Aligning with the generated dataset for Maniskill, we evaluate in the "StackCube", "PegInsertionSide", "PlugCharger", "PushCube", and "PickCube" tasks. Each of the 5 tasks are evaluated for 100 episodes each.

**Android Control**: We evaluate on the full 1,540 test episodes from Li et al. [47]. We measure the per-step success rate, meaning the percent of the time the agent predicts the right action based on the ground truth test episode. This is distinct from the episode level success rate which requires the agent to predict every action in the trajectory correctly. We report the success rate under the more challenging setting of following high-level instructions only, without per-step low-level instructions. We consider an action as predicted correctly if it is within 5 of the horizontal and vertical patches defined in the dataset generation from Appendix D. Any entered text is considered correct if the correct text is contained in the entered text or the entered text is contained in the correct text.

# F. Further Experimental Details

## F.1. Additional Baseline Details

For each benchmark, to the best of our knowledge, we report the method from prior work with the highest performance. In this section, we add more details about these baselines from prior works in Table 2 and any differences in the evaluation settings and method assumptions from GEA. We source methods from prior work that train a single policy over all tasks in a benchmark. We consider methods that only train on tasks from a single benchmark as "specialist" and methods that train across multiple benchmarks as "generalist". Note that this means we do not compare against methods that train policies on individual tasks from the benchmark. For example, in Procgen, we do not compare to the performance of Cobbe et al. [15] since a separate policy is trained for each of the 16 tasks. Instead, we only compare to methods that train a single policy across multiple tasks.

**Meta-World**: Many prior works report performance on the Meta-World benchmark [24], but fewer works train and evaluate over the full set of 45 tasks. Gato [69] trains over all the Meta-World tasks and reports an average performance of $87.0\%$ success rate. [1] Unlike GEA, Gato also takes as input the proprioceptive state in Meta-World. Reed et al. [69] also does not clarify if the Meta-World evaluation is performed over unseen state configurations or using the same states seen during training. GEA is evaluated on unseen state configurations. The GEA Meta-World numbers are reported in the same setting as Szot et al. [81] with the same inputs.

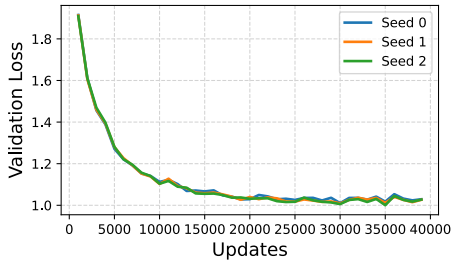
**CALVIN:** To the best of our knowledge, there are no generalist agents that report the performance on CALVIN in addition to other benchmarks. RoboFlamingo [48] also adapts an MLLM for control by finetuning it with supervised learning. We include this specialist agent since it also leverages finetuning MLLMs, to demonstrate the gains from scaling to a generalist model with GEA. Compared to GEA, RoboFlamingo uses the gripper camera, image augmentations during training and a longer context length. 3D Diffuser Actor [35] is the state-of-the-art specialist system for CALVIN $ABC \rightarrow D$ setting and narrowly outperforms GEA. However, as mentioned in the main text, 3D Diffuser Actor also assumes input to 3D pointclouds features. This method uses the head and gripper RGBD cameras to extra pointclouds of the scene. These pointclouds are then converted into a 3D feature cloud using a pretrained CLIP model. This method also simplifies the problem by compressing longer sequences of actions into end-effector keyposes.

**Maniskill:** We compare to the numbers using RGBD in Table 2 and 3 of Gu et al. [22]. Unlike these numbers, GEA uses only RGB inputs and no depth images. These results train a single policy per-task which is technically narrower than our definition of "Specialist Agent" which requires training one policy on all tasks from the benchmark. However, we still include these baselines to situate our Maniskill results. GEA outperforms doing imitation learning with the exact same demonstrations as from Table 2 of Gu et al. [22]. The superior results of $47.8\%$ success are from Table 3 of Gu et al. [22], which train with DAPG [67] and PPO. GEA does not train with RL in this task. Hansen et al. [25] reports higher success rates in Maniskill2 tasks, but uses the ground truth state information instead of visual observations and trains a single policy per individual Maniskill2.

**Habitat Pick:** Other methods that report performance on the version of the Habitat Pick task that requires picking from the object name typically achieve low success rates [26, 81, 96]. We thus also compare to the expert policy that was used to generate the Habitat Pick training dataset as described in Appendix D. Note that this expert policy was not trained in the evaluation scenes and thus also must generalize to unseen scenes. This expert policy has performance on par with pick skills trained in Habitat that operates from the much stronger assumption of a geometric goal input [78]. The specialist numbers from [81] also finetune a MLLM with imitation learning and are evaluated in the exact same setting as GEA.

**Habitat Place:** We follow the same evaluation criteria as Habitat Pick and compare to the expert policy trained with RL that was used to generate the expert demonstrations as described in Appendix D.

[1] We reference the Gato numbers from Table 8 of the TMLR paper version: https://openreview.net/pdf?id=1ikK0kHjvj




(a) Validation Loss (GEA-Base-500m).

| | HabPick | Procgen | CALVIN | BabyAI | Meta-World | Android Control |
|---|---|---|---|---|---|---|
| Seed 0 | 46.5 | **29.0** | **44.5** | **82.6** | 82.7 | 48.4 |
| Seed 1 | 49.0 | 25.4 | 42.5 | 80.0 | 86.7 | **50.1** |
| Seed 2 | **53.0** | 27.3 | 44.5 | 82.4 | **88.4** | 49.2 |
| Combined | 49.5 $\pm$ 3.3 | 27.2 $\pm$ 1.8 | 43.8 $\pm$ 1.2 | 81.7 $\pm$ 1.5 | 85.9 $\pm$ 3.0 | 49.2 $\pm$ 0.8 |

(b) Evaluation success rates (GEA-Base-500m).

Figure 6. Variance in training jobs and evaluation for GEA. We train three different random seeds for GEA-Base-500m. The left shows the validation loss during SFT is similar between each random seed. The right shows the resulting online evaluation for these three random seeds across six of the benchmarks, along with the combined averages and standard deviation per benchmark. While the standard deviation is low, it is still a couple of percent on some benchmarks despite the validation loss being very similar.

**Procgen:** We compare against the BC test numbers from Figure 2 of Mediratta et al. [59], which use the same datasets as our setup. While Gato [69] also reports numbers in Procgen, we are not able to compare to these numbers because Gato reports performance relative to unknown score of the data collection policy. To the best of our knowledge the score of the data collection policy is not released. Thus, it is unclear how the Gato Procgen performance is normalized according to the standard Procgen normalization scores [15] rendering a direct comparison impossible.

**Atari:** For a generalist system, we compare with Gato [69] which as Table 8 shows of Reed et al. [69] shows, achieves 30.9 normalized score. Multi-Game Decision Transformers [41] achieves 85 normalized score in the same scoring setting. This method was trained with offline RL based on conditioning the training on the reward-to-go [13].

**Habitat Nav:** We compare against the success rate of the navigation policy from Szot et al. [78]. This policy is trained with RL on the same training set of episodes and evaluated on the same testing set of episodes as GEA. However, this policy operates on the geometric goal specification of the receptacle rather than the receptacle name. Additionally, this policy also takes an egomotion sensor as input, whereas GEA does not, simplifying the problem.

**BabyAI:** We compare against Gato [69] which as Table 8 in Reed et al. [69] shows, achieves 93.2 normalized score. While Reed et al. [69] does not clarify this detail, presumably the normalization is with respect to a perfect expert policy, so the normalized score is equal to the success rate. Gato trains with far more data than GEA with 4.61M episodes.

**AndroidControl:** We compare against using a Set-of-Mark prompting with GPT-4o. The Set-of-Mark was implemented using the Ferret-UI model [49] for UI element detection to generate the marks with GPT-4o to determine the action. GEA and this baseline are evaluated under the same success criteria. We do not compare to the numbers from the original AndroidControl paper [47] since it uses different evaluation criteria from ours described in Appendix E and the code for the evaluation in Li et al. [47] is not publically released.

**LangR:** We compare against the state-of-the-art numbers from Szot et al. [81]. This method was trained with RL over the same set of training episodes as GEA.

### F.2. Ablation Analysis Setting

For the analysis experiments, we used a reduced subset of the total dataset. Specifically, we use the Meta-World, CALVIN, Habitat Pick, BabyAI, Procgen, and AndroidControl datasets. Datasets from the other domains are excluded.

When training methods in the analysis setting, we keep all the same settings as from the main GEA experiments with the hyperparameters described in Appendix B. However, we reduce the number of updates to 40k and use a global batch size of 256 across 2 nodes of 8 H100 GPUs each. All results in the analysis section are reported in the LLaVA-OneVision-500m setting unless specified otherwise.

## G. Further Results

### G.1. Benchmark Per-Task Success Rate

We breakdown the performance of the GEA model reported in Table 2 per individual task for the benchmarks of Meta-World (Table 5), CALVIN (Table 6), Procgen (Table 7), Maniskill (Table 8), LangR (Table 9), and BabyAI (Table 10).

### G.2. Habitat Pick RL Finetuning

Complimenting the results demonstrating the value of online learning from Figure 4, in this section, we compare the RL sample efficiency of GEA-Base versus the base MLLM.

Figure 7 shows that doing RL from the GEA-Base model is far more sample efficient and converges to much higher performance than doing RL on the MLLM model. The GEA-Base model is trained with SFT on demonstrations from the Habitat Pick task, so it is expected that its performance will start higher. However, the MLLM model is never able to make up for the performance gap, even with continued RL finetuning.

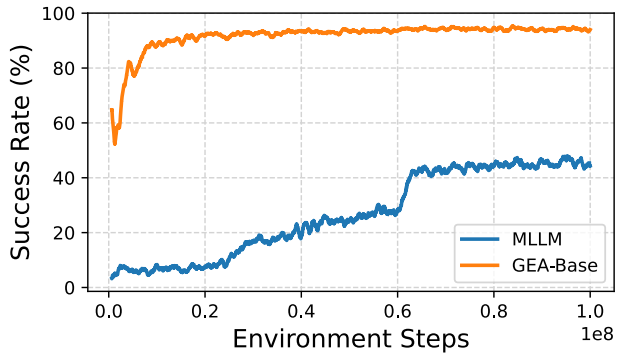



Figure 7. Success rate on the Habitat Pick task of GEA-Base and the base MLLM when finetuning with online RL. Displayed are success rates on the training dataset used in the RL process.

### G.3. Model Variance Analysis

In this section, we analyze the sensitivity of training GEA-Base-500m with different random seeds. Specifically, we vary the random seed used to initialize the model, all aspects of the algorithm, and dataset sampling. We then train the GEA-Base-500m model in the analysis setting from Appendix F.2. The results in Figure 6 demonstrate that while the training and validation curves are very similar. The online evaluation performance of GEA-Base-500m does have some variance per random seed within a couple of percentage points.

### G.4. Domain Transfer Analysis

In Figure 8 we study how performance transfers between domains by training the GEA-Base-500m model on every possible pair of datasets from BabyAI, CALVIN, Habitat Pick, Android Control and Procgen. We train the model in the same analysis setting as from Appendix F.2. We compare the performance of the dataset pair to only training on one of the datasets (the same as the "Domain Specific" results from Table 4). The results in Figure 8, with more blue colors for negative transfer and more red colors for positive transfer, show that some domains such as Procgen and CALVIN enormously benefit from data in other domains. Android Control slightly benefits from Procgen, another discrete control task. On the other hand, Habitat Pick and BabyAI have negative transfer from a variety of tasks. Despite this negative transfer, Table 4 still demonstrates that training on all the data across all these domains improves the performance.

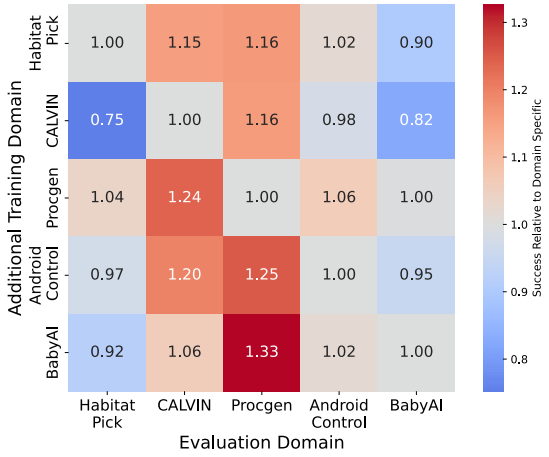



Figure 8. Each square represents the success rate of GEA-Base-500m trained with the datasets from the two domains indicated by the column and row name and evaluated on the domain indicated by the column. Success rates are scaled by training on only data from that domain, meaning each column is normalized relative to the diagonal. A more blue color means negative transfer and a more red color means positive transfer.

| | GEA |
|---|---|
| assembly | 86.0 |
| basketball | 100.0 |
| button-press-topdown | 100.0 |
| button-press-topdown-wall | 100.0 |
| button-press | 100.0 |
| button-press-wall | 92.0 |
| coffee-button | 100.0 |
| coffee-pull | 100.0 |
| coffee-push | 100.0 |
| dial-turn | 100.0 |
| disassemble | 88.0 |
| door-close | 100.0 |
| door-open | 100.0 |
| drawer-close | 100.0 |
| drawer-open | 100.0 |
| faucet-open | 100.0 |
| faucet-close | 100.0 |
| hammer | 100.0 |
| handle-press-side | 100.0 |
| handle-press | 100.0 |
| handle-pull-side | 80.0 |
| handle-pull | 100.0 |
| lever-pull | 80.0 |
| peg-insert-side | 100.0 |
| peg-unplug-side | 98.0 |
| pick-out-of-hole | 60.0 |
| pick-place | 92.0 |
| pick-place-wall | 82.0 |
| plate-slide | 100.0 |
| plate-slide-side | 96.0 |
| plate-slide-back | 100.0 |
| plate-slide-back-side | 100.0 |
| push-back | 90.0 |
| push | 100.0 |
| push-wall | 100.0 |
| reach | 60.0 |
| reach-wall | 94.0 |
| shelf-place | 98.0 |
| soccer | 76.0 |
| stick-push | 100.0 |
| stick-pull | 98.0 |
| sweep-into | 90.0 |
| sweep | 100.0 |
| window-open | 100.0 |
| window-close | 100.0 |

Table 5. Meta-World per-task success rate breakdown.

| | GEA |
|---|---|
| turn off led | 87.0 |
| move slider left | 100.0 |
| rotate red block right | 69.0 |
| open drawer | 100.0 |
| rotate red block left | 95.5 |
| push pink block right | 100.0 |
| push blue block right | 65.2 |
| push red block left | 85.7 |
| push pink block left | 87.1 |
| push red block right | 31.0 |
| push blue block left | 88.6 |
| push into drawer | 86.7 |
| rotate pink block left | 100.0 |
| turn on lightbulb | 97.6 |
| rotate pink block right | 93.5 |
| rotate blue block right | 78.6 |
| turn off lightbulb | 97.1 |
| lift blue block table | 100.0 |
| close drawer | 100.0 |
| rotate blue block left | 95.8 |
| move slider right | 95.4 |
| turn on led | 96.4 |
| lift blue block slider | 63.3 |
| lift pink block table | 100.0 |
| lift red block slider | 73.1 |
| lift red block table | 83.3 |
| lift pink block slider | 96.0 |

Table 6. CALVIN per-task success rate breakdown. Note the tasks are not equally represented in the evaluation episodes.

| | GEA |
|---|---|
| bigfish | 43.1 |
| bossfight | 54.2 |
| caveflyer | 27.7 |
| chaser | 54.0 |
| coinrun | 66.0 |
| dodgeball | 3.4 |
| fruitbot | 84.8 |
| heist | 32.0 |
| leaper | 26.0 |
| maze | 58.0 |
| miner | 27.0 |
| climber | 46.2 |
| ninja | 62.0 |
| plunder | 4.5 |
| jumper | 50.0 |

Table 7. Procgen per-game score breakdown.

| | GEA |
|---|---|
| StackCube | 4.0 |
| PegInsertionSide | 0.0 |
| PlugCharger | 0.0 |
| PushCube | 52.0 |
| PickCube | 12.0 |

Table 8. Maniskill per-task score breakdown.

| | GEA |
|---|---|
| rephrasing | 84.0 |
| referring expressions | 16.0 |
| spatial relationships | 0.0 |
| context | 34.0 |
| irrelevant text | 86.0 |
| multiple rearrangements | 82.0 |
| novel objects | 96.0 |
| multiple objects | 0.0 |
| conditional instructions | 52.0 |

Table 9. LangR per-task score breakdown.

| | GEA |
|---|---|
| GoToRedBallGrey | 88.0 |
| GoToRedBall | 97.0 |
| GoToRedBallNoDists | 100.0 |
| GoToObj | 100.0 |
| GoToObjS4 | 100.0 |
| GoToLocalS8N7 | 93.0 |
| GoToRedBlueBall | 92.0 |
| GoToDoor | 100.0 |
| Open | 76.0 |
| OpenRedDoor | 100.0 |
| OpenDoorColor | 100.0 |
| OpenDoorLoc | 80.0 |
| OpenTwoDoors | 100.0 |
| OpenRedBlueDoors | 100.0 |
| Pickup | 41.0 |
| PickupLoc | 88.0 |
| PickupDist | 94.0 |

Table 10. BabyAI per-task score breakdown.

| | GEA |
|---|---|
| Alien | 6.1 |
| Amidar | 0.0 |
| Assault | 86.5 |
| Asterix | 2.3 |
| Atlantis | 0.0 |
| BankHeist | 8.9 |
| BattleZone | 39.2 |
| BeamRider | 3.1 |
| Boxing | 0.0 |
| Breakout | 0.0 |
| Centipede | 50.0 |
| ChopperCommand | 30.2 |
| CrazyClimber | 0.0 |
| DemonAttack | 20.8 |
| DoubleDunk | 300.0 |
| Enduro | 0.0 |
| FishingDerby | 0.0 |
| Freeway | 81.1 |
| Frostbite | 2.7 |
| Gopher | 0.0 |
| Gravitar | 0.0 |
| Hero | 0.0 |
| IceHockey | 0.0 |
| Jamesbond | 0.0 |
| Kangaroo | 38.5 |
| Krull | 373.0 |
| KungFuMaster | 0.0 |
| MsPacman | 27.9 |
| NameThisGame | 0.0 |
| Phoenix | 0.0 |
| Pong | 0.0 |
| Qbert | 0.6 |
| Riverraid | 0.0 |
| RoadRunner | 33.0 |
| Robotank | 307.2 |
| Seaquest | 0.3 |
| SpaceInvaders | 52.7 |
| StarGunner | 1.4 |
| TimePilot | 0.0 |
| UpNDown | 0.0 |
| VideoPinball | 0.0 |
| WizardOfWor | 0.0 |
| YarsRevenge | 0.0 |
| Zaxxon | 0.0 |

Table 11. Atari success rate breakdown.