Appendix of EC^2 : Emergent Communication for Embodied Control

Yao Mu[1], Shunyu Yao [2], Mingyu Ding [1], Ping Luo[1], Chuang Gan[3,4]
[1]The University of Hong Kong [2] Princeton University [3]UMass Amherst, [4]MIT-IBM Watson AI Lab
ymu, myding, pluo@cs.hku.hk shunyuy@princeton.edu ganchuang@csail.mit.edu

A. Implementation Details of EC^2

A.1. Emergent Language Generation

The maximum length of emergent language is set as 10, and the length of the input video is set as 20. For the videos whose length is larger than 20, we select a sub-sampled video that is randomly sub-sampled from the original video, and the first and last frames of the video remain in the sub-sampled video. We encode each frame into a 512 dimension vector by ResNet-50 [1] and send the sequence of embedding into the transformer under GPT [2] framework, as shown in Listing 1, the transformer generates the logit of emergent language tokens step by step.

```
def EC_Generation(idx, max_new_tokens, temperature):
    ec_voc=[]
    for index in range(max_new_tokens):
        idx_cond = idx
        logits = gpt_forward(idx_cond)
        logits = logits[:, -1, :] / temperature
        idx_nexts = F.softmax(logits, dim=-1)
        idx_next = ori_map(idx_nexts)
        idx_next = torch.unsqueeze(idx_next, 1)
        idx = torch.cat((idx, idx_next), dim=1)
        ec_voc.append(idx_nexts)
    EC = torch.stack(ec_voc,1)
    return EC
```

Listing 1. PyTorch-style pseudo-code for EC language generation.

A.2. Pre-training by Masked trajectory Completion

We conduct a trajectory completion task to pre-train the language model without action labels. Firstly, we randomly sample a sequence of observations $\hat{o} = \{o_1, o_2, \ldots, o_N\}$ stored in the dataset and map it into latent trajectory τ_{whole} by encoder g_θ which is implanted with ResNet-50 [1] (same as the encoder used in Emergent language generation). Same as the video length used in emergent language generation, in this part, the length is also set as 20. Then we crop a random segment τ_{seg} from the whole latent trajectory τ_{whole}. The remained trajectory is denoted as τ_{rem}, and we set the length of τ_{rem} is 10. The language model f_ϕ takes τ_{rem} as input and uses either emergent language e generated by the speaker or natural language l as a prompt to predict the cropped segment τ_{seg}.

A.3. Downstream policy learning for embodied control tasks

We take the emergent language or natural language as prompt and the current trajectory τ_{cur} as input which contains the current state and 9 historical states (total length is set as 10). The pre-trained language model outputs the encoded embodied representation m_t. We map m_t into action space by task-specific MLP layers [3], which is called the downstream policy network. The downstream policy network is a 2-layer MLP [3] with hidden sizes [256,256] preceded by a batch normalization [4] layer. The policy network takes the concatenation of visual embedding and proprioceptive data from the environment as input, and produces an action as output. To train the policy network, a learning rate of 0.001 is used, along with a batch size of 32 for 20,000 steps training.

A.4. Detailed model architectures

We utilize a GPT-like neural network as shown in Figure 1, to serve as the foundational model for both the speaker and listener in EC generation, as well as for the pretext task. The architecture consists of eight self-attention layers, each with 16 heads and a 512 embedding size, resulting in a total of 25.81 million parameters.

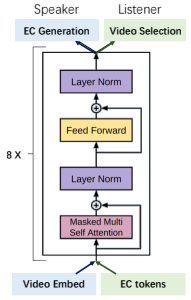

Figure 1. Detailed model architecture.

B. Additional Results

To make sure R3M and EC^2 receive fair finetuning data, we conducted additional experiments by finetuning the pre-trained R3M model on the LoReL dataset [5] and re-evaluating its performance on the Franka Kitchen task. From Table 1, we observe the performance of R3M improves slightly by finetuning on LoReL dataset [5], indicating that, though the usage of the robotic dataset helps, the main improvement comes from the representation learning

method. Our EC^2 consistently performs better than finetuned R3M.

Demos	Instruction	EC ^{2} (ours)	*R3M-Finetune*	R3M
10	Lang	$\mathbf{48.4} \pm 2.5\%$	44.6 ± 2.3	$41.8 \pm 2.5\%$
	Video	$\mathbf{53.6} \pm 2.5\%$	47.3 ± 2.0	$45.7 \pm 2.4\%$
25	Lang	$\mathbf{59.8} \pm 2.5\%$	57.6 ± 1.8	$56.0 \pm 2.3\%$
	Video	$\mathbf{63.2} \pm 2.6\ \%$	60.7 ± 2.5	$58.7 \pm 2.0\%$

Table 1. Performance comparison with finetuned R3M.

C. Details of Evaluation Environments

Franka Kitchen. The goal of the Franka Kitchen [6] is to interact with the various objects in order to reach the desired state configuration. The agent can change the position of the kettle, flip the light switch, open and close the microwave and cabinet doors, or slide the other cabinet door. The Franka Kitchen environments used in our paper are modified from the original environment as the same as R3M [7]. In order to evaluate the robustness of the learned policy, we introduced further randomization to the original scene by varying the position of the kitchen between episodes. This made the task more challenging in terms of both perception and control. In Franka Kitchen, we learn the tasks of sliding the right door open, opening the left door, turning on the light, turning the stovetop knob, and opening the microwave. We additionally provide the language instruction for each task. We also provide demonstration videos for downstream policy learning, which is generated by training a state-based agent with model-free RL [8]. The state-based trajectories are then replayed and rendered with image observations. The horizon for all Franka tasks is 50 steps, and the few-shot policy learning experiments use either 5, 10, or 25 demos. All tasks in the Franka Kitchen environment come with both proprioceptive data from the arm joints and gripper positions, as well as visual observations. The proprioceptive data is concatenated with the latent features encoded from the visual observations.

MetaWorld. MetaWorld [9] is a simulated benchmark that includes a shared tabletop environment with a Sawyer arm, designed to evaluate the robot's ability to execute various manipulation tasks involving reaching, pushing, and grasping. For instance, pushing or grasping an object with a revolute joint is required for the open door task, while pushing or grasping an object with a sliding joint is needed for the open drawer task. The MetaWorld environment features several tasks, including assembling a ring onto a peg, picking and placing a block between bins, pushing a button, opening a drawer, and hammering a nail. In all tasks, the target object's position, such as a drawer, peg, or block, is randomized between episodes. Additionally, all tasks include proprioceptive data of the gripper end-effector pose and gripper open/close, and have specific language instructions and video demonstrations. Expert data generated using a heuristic policy is replayed and rendered with image observations to create the demonstration video and corresponding state-action labels. The horizon for all tasks in MetaWorld is 500 steps, and the downstream policy learning experiments use either 5, 10, or 25 demonstrations.

D. Implementation Details of baselines

D.1. BC-Z: Zero-Shot Task Generalization with Robotic Imitation Learning

BC-Z [10] embeds the language instruction w_ℓ or video instruction w_h via an encoder $q(z|w)$ into a task embedding z. For the language instruction, a pre-trained multilingual sentence encoder [11] is used as the encoder to produce a 512-dim language vector for each task. For video instruction, the input video w is randomly subsampled from the original video to be 20 frames long so that the video's first and last frame remains in the subsampled w. Moreover, a convolutional neural network is used to produce the task embedding z. Using the demonstration video w_h^i and corresponding state-action sequence labels $\{(s,a)\}^i$, BC-Z encodes the demonstration video $z^i \sim q(\cdot \mid w_h^i)$, then pass the embedding to the control layer $\pi(a|s,z^i)$, and then back-propagate gradient of the behavior cloning loss to both the policy and encoder parameters.

To achieve better semantic alignment between video embeddings and language embeddings, BC-Z introduces a *language regression* auxiliary loss to constrain the distance between language and video embeddings. This loss function encourages the video encoder to predict the embedding of the language command for the task using a cosine loss. The objective of the video encoder can be defined as follows:

$$\min \sum_{\text{task } i} \sum_{\substack{(s,a)\sim\mathcal{D}_e^i \\ w_h\sim\mathcal{D}_h^i \bigcup \mathcal{D}_e^i}} \underbrace{-\log \pi(a|s,z^i)}_{\text{behavior cloning}} + \underbrace{D_{\cos}(z_h^i, z_\ell^i)}_{\text{language regression}},$$
$$\text{where } \underbrace{z_h^i \sim q(\cdot|w_h)}_{\text{video encoder}}, \underbrace{z_\ell^i \sim q(\cdot|w_\ell^i)}_{\text{language encoder}} \tag{1}$$

where $D_{\cos}$ denotes the cosine distance.

Specifically, BC-Z implements the instruction-conditioned convolutional neural network using the FiLM-conditioned ResNet [12] to encourage the model to use information from the task embedding to infer the task. FiLM is a technique used in neural networks that enables them to modify their output by applying an affine transformation to the intermediate features of the network. This transformation involves scaling and shifting the feature statistics based on a conditioning input, which can be any relevant information that can help the network make better predictions, such as class labels, attribute values, or textual descriptions. By modulating the features in this way, FiLM

prompts the network to focus on different aspects of the input, resulting in more diverse and informative outputs. All the hyper-parameters are implemented as same as the officially published repo[1]. And both BC-Z and EC^2 were trained with the same LoRel dataset [5].

D.2. R3M: A Universal Visual Representation for Robot Manipulation

We use the visual encoder $\mathcal{F}_\phi$ pre-trained by R3M [7], which is implemented by ResNet-50 [1] in `torchvision.models`. The pre-trained R3M model is trained for 1.5 million steps with a learning rate of 1e-4, and is officially released at their website [2].

R3M performs time contrastive learning, video-language alignment, and embedding regularization to capture effective representations from demonstration videos and language descriptions. Given a batch of videos, R3M trains the visual encoder to produce a representation such that the distance between images closer in time is smaller than for images farther in time or from different videos. Specifically, we sample a batch of sequences of frames $[I_i, I_{j>i}, I_{k>j}]^{1:B}$, then minimize the InfoNCE loss [13]:

$$\mathcal{L}_{tcn} = -\sum_{b\in B} \log \frac{e^{\mathcal{S}(z_i^b, z_j^b)}}{e^{\mathcal{S}(z_i^b, z_j^b)} + e^{\mathcal{S}(z_i^b, z_k^b)} + e^{\mathcal{S}(z_i^b, z_i^{\neq b})}} \tag{2}$$

where z is the encoded representation, and $z_i^{\neq b}$ is a negative example sampled from a different video in the batch. $\mathcal{S}$ is the measure function of similarity, which is implemented as the negative L2 distance $\mathcal{S}(x_1, x_2) = -||x_1 - x_2||_2^2$. To encourage the visual encoder to capture semantically relevant features, R3M adds a video-language alignment loss and trains a model $\mathcal{G}_\theta(\mathcal{F}_\phi(I_0), \mathcal{F}_\phi(I_i), l)$ that takes in an initial image I_0, a future image I_i, language l and outputs a score corresponding to if transitioning from I_0 to I_i completes the language l. Specifically, R3M samples a video clip and paired language $[I_i, I_{j>i}, l]^{1:B}$, and then trains for this objective directly with a contrastive loss, that is:

$$\mathcal{L}_{lang} = -\sum_{b\in B} \log \frac{e^{\mathcal{G}_\theta(z_0^b, z_{j>i}^b, l^b)}}{e^{\mathcal{G}_\theta(z_0^b, z_{j>i}^b, l^b)} + e^{\mathcal{G}_\theta(z_0^b, z_i^b, l^b)} + e^{\mathcal{G}_\theta(z_0^{\neq b}, z_{j>i}^{\neq b}, l^b)}} \tag{3}$$

where $z^{\neq b}$ is a negative example sampled from a different video in the batch that does not match the language instruction l^b. Furthermore, R3M encourages sparse and compact representations to benefit embodied control, particularly in low-dimension imitation learning. R3M learns the visual encoder with a simple L1 and L2 penalty to reduce the effective dimensionality of the state space.

The total objective of R3M is the weighted sum:

$$\mathcal{L}(\phi, \theta) = \mathbb{E}_{I_{0,i,j,k}^{1:B} \sim \mathrm{D}}[\lambda_1 \mathcal{L}_{tcn} + \lambda_2 \mathcal{L}_{lang} + \lambda_3 ||\mathcal{F}_\phi(I_i)||_1 + \lambda_4 ||\mathcal{F}_\phi(I_i)||_2] \tag{4}$$

[1] https://sites.google.com/view/bc-z/home

[2] https://github.com/facebookresearch/r3m

During the evaluation of downstream embodied control tasks, for embodied control with demonstration video instruction, we use the officially released vision encoder to extract features of every frame in the video and map them into the instruction representation by one MLP layer. For embodied control with natural language instruction, we use the language encoder of the officially released pre-trained model to encode the natural language.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 3

[2] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 1

[3] Hassan Ramchoun, Youssef Ghanou, Mohamed Ettaouil, and Mohammed Amine Janati Idrissi. Multilayer perceptron: Architecture optimization and training. 2016. 1

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 1

[5] Suraj Nair, Eric Mitchell, Kevin Chen, Silvio Savarese, Chelsea Finn, et al. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation. In *Conference on Robot Learning*, pages 1303–1315. PMLR, 2022. 1, 3

[6] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019. 2

[7] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022. 2, 3

[8] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017. 2

[9] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR, 2020. 2

[10] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022. 2

[11] Yinfei Yang, Daniel Cer, Amin Ahmad, Mandy Guo, Jax Law, Noah Constant, Gustavo Hernandez Abrego, Steve Yuan, Chris Tar, Yun-Hsuan Sung, et al. Multilingual universal sentence encoder for semantic retrieval. *arXiv preprint arXiv:1907.04307*, 2019. 2

[12] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. 2

[13] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018. 3