Appendix

A Dataset and Environment Details

A.1 ALFRED

A.1.1 Dataset Details

We base our dataset and environment on the ALFRED benchmark [44]. ALFRED originally contains over 6000 full trajectories collected from an expert planner following a set of 7 high-level tasks with randomly sampled objects (e.g., *"pick an object and heat it"*). Each trajectory has three crowd-sourced annotations, resulting in around 20k distinct language-annotated trajectories. We separate these into only the primitive skill trajectories, resulting in about 141k language-annotated trajectories. Following Zhang et al. [43], we merge navigation skills (e.g., *"Walk to the bed"*) with the skill immediately following them as these navigation skills make up about half of the dataset, are always performed before another skill, and are difficult to design online RL reward functions for that work across all house floor plans given only the information in the dataset for these skills. After this processing step, the resulting dataset contains 73k language-annotated primitive skill trajectories.

A.1.2 RL Environment Details

We modified ALFRED similarly to Zhang et al. [43], Pashevich et al. [45] to make it suitable for policy learning by modifying the action space to be fully discrete, with 12 discrete action choices and 82 discrete object types.

Furthermore, we rewrote reward functions for all primitive skill types ("CoolObject", "PickupObject", "PutObject", "HeatObject", "ToggleObject", "SliceObject", "CleanObject") so that rewards can be computed independently of a reference expert trajectory. While our rewards depend on the ground truth primitive skill type, no agents are allowed access to what the underlying true primitive skill type is. All of our reward function are sparse, with 1 for a transition that completes primitive skill and 0 for all other transitions.

A.1.3 Evaluation Tasks

We generate evaluation tasks by randomly sampling 10 tasks each for 4 unseen ALFRED floor plans, resulting in 40 total tasks unseen tasks requiring anywhere from 2-8 primitive skills to complete. The tasks for each floor plan are sampled randomly from the VALID-UNSEEN ALFRED dataset collected in these plans with the specific object arrangements, and we use the high-level task language descriptions collected by humans for ALFRED as our task descriptions for language-conditioned zero-shot evaluation. See Figure 7 for a histogram of task lengths.

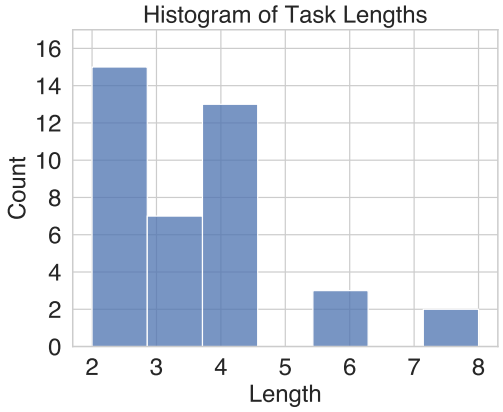


Figure 7: Task lengths regarding the number of primitive skills needed to chain together to solve the task.

Examples of common household tasks and their descriptions:
Task Steps: 1. Pick up the keys on the center table. 2. Put the keys in the box. 3. Pick up the box with keys. 4. Put the box with keys on the sofa close to the newspaper.
Task: Put the box with keys on the sofa.

Task Steps: 1. Pick up the knife from in front of the tomato. 2. Cut the lettuce on the counter. 3. Set the knife down on the counter in front of the toaster. 4. Pick up a slice of the lettuce from the counter. 5. Put the lettuce slice in the refrigerator. take the lettuce slice out of the refrigerator. 6. Set the lettuce slice on the counter in front of the toaster.
Task: Put a cooled slice of lettuce on the counter.

Task Steps: 1. Pick up the book on the table, in front of the chair. 2. Place the book on the left cushion of the couch.
Task: Put a book on the couch.

Task Steps: 1. Pick up the fork from the table. 2. Put the fork in the sink and fill the sink with water, then empty the water from the sink and remove the fork. 3. Put the fork in the drawer.
Task: Put the cleaned fork in a drawer.

Task Steps: 1. Take the box of tissues from the makeup vanity. 2. Put the tissues on the barred rack. 3. Take the box of tissues from the top of the toilet. 4. Put the tissues on the barred rack.
Task: Put the box of tissues on the barred rack.

Task Steps: 1. Pick up the glass from the sink. 2. Heat the glass in the microwave. 3. Put the glass on the wooden rack.
Task: Put a heated glass on the wooden rack.

Task Steps: 1. Pick up the box from the far side of the bed. 2. Hold the box and turn on the lamp.
Tasks: Look at the box under the lamp light.

Predict the next skill correctly by choosing from the following skills: [SKILL 1 IN LIBRARY], [SKILL 2 IN LIBRARY], ...
Task Steps: 1. [SKILL 1 EXECUTED SO FAR] 2. [SKILL 2 EXECUTED SO FAR] ... N. ____

Figure 8: Prompt for the LLM for next skill proposal Section 3.2 Text is generated after listing out all skills completed so far.

A.2 Language Model Prompts

We use two prompts when using the LLM for two different purposes. The main purpose of the LLM is to propose a distribution over next skills to chain with currently executed skills during skill bootstrapping (Section 3.2). Thus, we pass skills in the given skill library Z into the prompt and ask it to predict the next skill. We also include a fixed set of 7 in-context examples from a random sample of different tasks from the ALFRED training dataset. The prompt for bootstrapping is shown in Figure 8.

We also generate summaries (see Section 3.2 and appendix Appendix B.3) for *composite* skill annotations with the LLM. These summaries are used to label newly chained longer-horizon skills before adding them back to the skill library. We show the prompt for this in Figure 9.

B Training Implementation details and Hyperparameters

We implement IQL [41] as the base offline RL algorithm to pre-train on primitive skill data for all methods, baselines, and ablations, due to its strong offline and finetuning performance on a variety of dense and sparse reward environments.

The IQL policy is trained to maximize the following objective:

$$e^{\beta(Q(s,a)-V(s))} \log \pi(a|s),$$

which performs advantage-weighted regression [56] with an inverse temperature term β. Q and V are trained on (s, a, s', r, a') tuples from the dataset rather than sampling a policy for a' to mitigate

Instructions: give a high-level description for the following steps describing common household tasks.

Task Steps: 1. Pick up the keys on the center table. 2. Put the keys in the box. 3. Pick up the box with keys. 4. Put the box with keys on the sofa close to the newspaper.
Summary: Put the box with keys on the sofa.

Task Steps: 1. Pick up the knife from in front of the tomato. 2. Cut the lettuce on the counter. 3. Set the knife down on the counter in front of the toaster. 4. Pick up a slice of the lettuce from the counter. 5. Put the lettuce slice in the refrigerator. take the lettuce slice out of the refrigerator. 6. Set the lettuce slice on the counter in front of the toaster.
Summary: Put a cooled slice of lettuce on the counter.

Task Steps: 1. Pick up the book on the table, in front of the chair. 2. Place the book on the left cushion of the couch.
Summary: Put a book on the couch.

Task Steps: 1. Pick up the fork from the table. 2. Put the fork in the sink and fill the sink with water, then empty the water from the sink and remove the fork. 3. Put the fork in the drawer.
Summary: Put the cleaned fork in a drawer.

Task Steps: 1. Take the box of tissues from the makeup vanity. 2. Put the tissues on the barred rack. 3. Take the box of tissues from the top of the toilet. 4. Put the tissues on the barred rack.
Summary: Put the box of tissues on the barred rack.

Task Steps: 1. Pick up the glass from the sink. 2. Heat the glass in the microwave. 3. Put the glass on the wooden rack.
Summary: Put a heated glass on the wooden rack.

Task Steps: 1. Pick up the box from the far side of the bed. 2. Hold the box and turn on the lamp.
Summary: Look at the box under the lamp light.

Task Steps: 1. [SKILL 1] 2. [SKILL 2] 3. [SKILL 3] ...
Summary:

Figure 9: Prompt for the LLM to summarize completed skills into high-level *composite* annotations, following Zhang et al. [43].

issues with critic function overestimation common in offline RL. We detail shared training and implementation details below, with method-specific information and hyperparameters in the following subsections.

B.1 ALFRED Environment

We implement the same observation and action space as Zhang et al. [43]. Details are listed below.

Observation space. The observations given to agents are 300×300 RGB images. For all methods, we first preprocess these images by sending them through a frozen ResNet-18 encoder [57] pre-trained on ImageNet, resulting in a $512 \times 7 \times 7$ observation.

Action space. The agent chooses from 12 discrete low-level actions. There are 5 navigation actions: `MoveAhead`, `RotateRight`, `RotateLeft`, `LookUp`, and `LookDown` and 7 interaction actions: `Pickup`, `Put`, `Open`, `Close`, `ToggleOn`, `ToggleOff`, and `Slice`. For interaction actions the agent additionally selects one of 82 object types to interact with, as defined by Pashevich et al. [45]. In total, the action space consists of $5 + 7 * 82 = 579$ discrete action choices. For all methods, due to the large discrete action space, we perform the same action masking as Zhang et al. [43] to prevent agents from taking actions that are not possible. For example, we do not allow the agent to `Close` objects that aren't closeable or `ToggleOn` objects that can't be turned on.

Policy and critic networks. We use the transformer architecture (and hyperparameters) used by Episodic Transformers (ET) [45] for our policy and critic networks. We implement all critics (two Q functions and one V) with a shared backbone and separate output heads. Additionally, we use

LayerNorms [58] in the MLP critic output heads as recommended by Ball et al. [59]. All networks condition on tokenized representations of input language annotations.

Hyperparameters. Hyperparameters were generally selected from tuning the Oracle baseline to work as best as possible, then carried over to all other methods. Shared hyperparameters for all methods (where applicable) for pre-training on primitive skills are listed below. Any unlisted hyperparameters or implementation details are carried over from Pashevich et al. [45]:

Param	Value
Batch Size	64
# Training Epochs	150
Learning Rate	1e-4
Optimizer	AdamW
Dropout Rate	0.1
Weight Decay	0.1
Discount γ	0.97
Q Update Polyak Averaging Coefficient	0.005
Policy and Q Update Period	1 per train iter
IQL Advantage Clipping	[0, 100]
IQL Advantage Inverse Temperature β	5
IQL Quantile τ	0.8
Maximum Observation Context Length	21

When fine-tuning policies (for Oracle, CIC, and BOSS), we keep hyperparameters the same. We fine-tune one policy per floor plan (zero-shot evaluating on 10 tasks in each floor plan) in our ALFRED task set so that the aggregated results are reported over 4 runs per seed. For methods that use a skill library (BOSS, Saycan, Saycan+P), all available primitive skills across all evaluation tasks in each floor plan compose the starting skill library, resulting in anywhere from 15-40 available skills depending on the floor plan.

Additionally, when finetuning the Oracle baseline along with BOSS and its ablations, we sample old data from the offline dataset and newly collected data at equal proportions in the batch, following suggestions from [59]. We do not do this for CIC when finetuning with its unsupervised RL objective because the language embeddings from the old data are not compatible with the online collected data labeled with CIC-learned skill embeddings. Fine-tuning hyperparameters follow:

Param	Value
# Initial Rollouts	50
# Training Steps to Env Rollouts Ratio	15
ϵ in ϵ-greedy action sampling	0.05
Discrete action sampling	True
# Parallel Rollout Samplers	10

B.2 Real Robot Environment

The input observation from the environment includes environment RGB input and robot states. The RGB input consists of the third-person view RGB images from a Logitech Pro Webcam C920 cropped to 224×224×3, and wrist view images from an Intel RealSense D435. We use a pretrained R3M [60] model to get the latent representation for each view. The robot states include the robot's end-effector position, velocity, and gripper state. The end-effector position and velocity are two continuous vectors, and the gripper state is a one-hot vector, which presents OPEN, CLOSE, or NOT MOVE. We concatenate the RGB latent representations and robot states together as environment states.

The policy is language conditioned, and we use a pre-trained sentence encoder to encode the language annotation to a 384-dimensional latent vector. The pretrained sentence encoder we use is `all-MiniLM-L12-v2` from the `SentenceTransformers` package [42].

The total state input dimension is 2048 (third-person R3M) + 2048 (wrist R3M) + 15 (robot state input) + 384 (language latent representation) = 4495.

Action space. The action space of the robot encompasses the difference in the end effector position between each time step, along with discrete open and close signals for the gripper. These actions are transmitted to the robot with 10HZ and interpreted as desired joint poses using PyBullet's inverse kinematics module.

In line with [61], we adopt the Action Chunking method to train an autoregressive policy. Our policy utilizes an LSTM model to predict the next 15 actions, given the initial observation as input, denoted as $\pi(a_{t:t+15}|s_t)$. Both our Q and Value networks are recurrent as well, estimating rewards on a per-timestep basis for each action in the sequence. Similar to the policy, these networks only have access to the observation preceding the action sequence initiation.

Due to the gripper action space is discrete and imbalanced distributed in the dataset, we reweigh gripper loss inversely proportionally to the number of examples in each class.

B.3 Additional BOSS Implementation Details

Here we continue discussion of BOSS in detail. In the main text in Section 3.2 we mention that we add learned skills back to the agent's skill repertoire and then train on collected experience gathered from each rollout. Here, we detail exactly how we do that.

Labeling new composite skills. Finally, after we have finished attempting a composite skill chain, we need a natural language description for it so we can train the language-conditioned policy on this new composite skill. We ask the LLM to generate high-level task descriptions of the annotations of the two skills the agent has just attempted to chain together like proposed by Zhang et al. [43] for offline policy pre-training. Doing so will allow the agent to learn skills at a higher level of text abstraction, allowing the agent to operate on more natural evaluation task specifications. For example, humans are more likely to ask an agent to "Make coffee" than to say "Get a coffee pod. Put the coffee pod in the machine. Fill it up with water..."

We give the LLM a prompt similar to the one for generating next skills. For example, if our agent has just completed two skills: *"Pick up the spoon"*, *"Put the spoon on the counter"*, we ask the LLM to summarize "1. PICK UP THE SPOON. 2. PUT THE SPOON ON THE COUNTER.", and the LLM can generate *"put a spoon on the counter."* We denote the generated language annotation for this combined skill composed of the annotations of z^1 and z^2 as z'. We then add z' as a new composite skill to Z for the agent to possibly sample from again.

Training on new skill data After the agent has finished a rollout in the environment, it trains on the experience gathered. There are five types of data that we add to the agent's replay buffer from its rollout data:

1. The trajectory of the attempted skill chain which is collected only if the entire first skill is successfully executed (regardless if it is a primitive skill or a chain of them) since only then will another skill be used for chaining. The label for this trajectory is produced by the LLM.
2. The trajectory of the composite skill but with a label generated by concatenating the primitive skill annotations as a sequence of sentences of their language annotations. This trajectory ensures that the agent receives a description for the collected composite trajectory that specifies the exact primitive skills that make it up, in order. This is useful because the LLM-generated high-level skill description may not describe certain steps. Those steps are explicitly spelled out in this new label.
3. Trajectories for all lowest-level primitive skills executed during the rollout. These correspond to the original set of skills the policy was equipped with and will help the policy continue to finetune its ability to execute its original primitive skills.

After the rollout, we add these trajectories to the agent's replay buffer.

Other details When performing skill bootstrapping in the ALFRED environment, we set a max time limit (T in Algorithm 2) for 40 timesteps per primitive skill. For simplicity, we restrict M,

the max number of skills to chain, to be 2 during skill bootstrapping rollouts. We also restrict the second skill to be chained to only the set of primitive skills so that the agent can only learn new skill chains that are one primitive skill longer than the first sampled skill. Note that this does not restrict the agent from sampling composite skills it has learned during bootstrapping as first skills upon initialization.

B.4 CIC Implementation

For fairness in our experimental comparison, we implement CIC [46] by using its objective to train a policy pre-trained on the same dataset as BOSS; thus, the CIC agent is first initialized with a set of sensible behaviors. Since CIC operates on a fixed latent space, we modified the critic and policy architectures so that they operate on fixed-length, 768-dimensional embeddings of language inputs from the same sentence embedding model used for skill bootstrapping [42] instead of on variable length tokenized language representations.

CIC-specific hyperparameters follow:

Param	Value
CIC K-means K	12
CIC K-means avg	True
CIC Hidden Dim	1024
CIC Latent Skill Dim	768
CIC Temp	0.5
CIC Skill Projection Layer	True
# Timesteps for each skill rollout before reset	200

B.5 SayCan Implementation

We implement SayCan [12] by combining the prompt from SayCan with ours. We use the same in-context examples except but convert them to a human-robot conversation. All other details are the same, including the LLM that we use in this comparison (LLaMa-13b [49]). The Saycan prompt follows below:

> Robot: Hi there, I'm a robot operating in a house. Robot: You can ask me to do various tasks

and I'll tell you the sequence of actions I would do to accomplish your task.

Human: How would you put the box with keys on the sofa?
Robot: 1. Pick up the keys on the center table. 2. Put the keys in the box. 3. Pick up the box with keys. 4. Put the box with keys on the sofa close to the newspaper.

Human: How would you put a cooled slice of lettuce on the counter?
Robot: 1. Pick up the knife from in front of the tomato. 2. Cut the lettuce on the counter. 3. Set the knife down on the counter in front of the toaster. 4. Pick up a slice of the lettuce from the counter. 5. Put the lettuce slice in the refrigerator. take the lettuce slice out of the refrigerator. 6. Set the lettuce slice on the counter in front of the toaster.

Human: How would you put a book on the couch?
Robot: 1. Pick up the book on the table, in front of the chair. 2. Place the book on the left cushion of the couch.

Human: How would you put the cleaned fork in a drawer?
Robot: 1. Pick up the fork from the table. 2. Put the fork in the sink and fill the sink with water, then empty the water from the sink and remove the fork. 3. Put the fork in the drawer.

Human: How would you put the box of tissues on the barred rack?
Robot: 1. Take the box of tissues from the makeup vanity. 2. Put the tissues on the barred rack. 3. Take the box of tissues from the top of the toilet. 4. Put the tissues on the barred rack.

Human: How would you put a heated glass on the wooden rack?
Robot: 1. Pick up the glass from the sink. 2. Heat the glass in the microwave. 3. Put the glass on the wooden rack.

Human: How would you look at the box under the lamp light?
Robot: 1. Pick up the box from the far side of the bed. 2. Hold the box and turn on the lamp.

Predict the next skill correctly by choosing from the following skills: [SKILL 1 IN LIBRARY], [SKILL 2 IN LIBRARY], ...

Human: How would you [HIGH LEVEL TASK DESCRIPTION]?
Robot: 1. [SKILL 1 EXECUTED SO FAR] 2. [SKILL 2 EXECUTED SO FAR] ... N. ____

B.6 ProgPrompt Implementation

ProgPrompt [14] converts natural language queries to code and executes the code on a real robot. After consulting with the authors, we converted the examples in our prompt to one suitable for Progprompt by converting task descriptions into a code representation by converting spaces into underscores, e.g., "Pick up the milk" into "def pick_up_the_milk()". Then, to translate code commands into commands suitable for our pre-trained policy, we prompt ProgPrompt to output 'pick_and_place(object, object)' style code commands that we convert into two separate pick and place natural language commands in the same format as the instructions used for pre-training the policy. We then execute these instructions on the real robot in sequence.

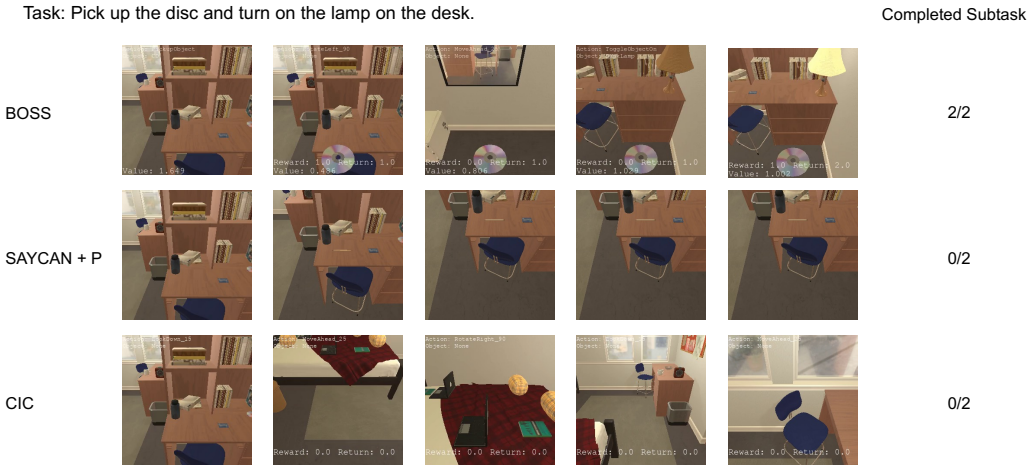
C Additional Results

C.1 Real Robot Results

We evaluate on 5 tasks, detailed below, in the environment setup shown in Figure 11.

1. Clean the black bowl (length 2): (1) Pick up the black bowl, (2) put it in the sink.
2. Put the black bowl to the dish rack (length 2): (1) Pick up the black bowl, (2) put it in the dish rack.

(a) Length 3 Task Example



(b) Length 2 Task Example

Figure 10: Qualitative visualizations of zero-shot evaluation rollouts.

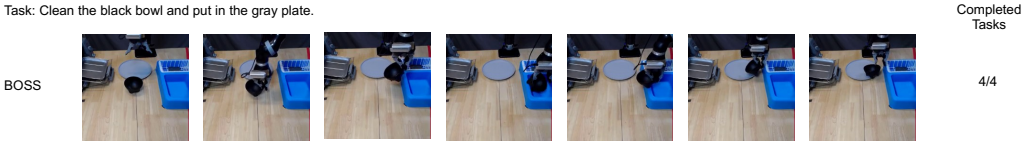


Figure 11: Example of a BOSS rollout after skill bootstrapping on task 5: "Clean the black bowl and put it in the gray plate." BOSS is able to complete all 4 tasks in this rollout after performing skill bootstrapping.

3. Put the black bowl to the gray plate (length 2): (1) Pick up the black bowl, (2) put it in the gray plate.
4. Clean the black bowl and put it in the dish rack (length 4): (1) Pick up the black bowl, (2) put it in the sink, (3) pick up the black bowl, (4) put it in the dish rack.
5. Clean the black bowl and put it in the gray plate (length 4): (2) pick up the black bowl, (2) put it in the sink, (3) pick up the black bowl, (4) put it in the plate.

We report full results in Table 4.

Task	ProgPrompt return	ProgPrompt success rate	BOSS return	BOSS success rate
1	2 ± 0	1	1.6 ± 0.8	0.8
2	0.67 ± 0.47	0	0.8 ± 0.75	0.2
3	0.67 ± 0.47	0	0.6 ± 0.49	0
4	1.20 ± 0.83	0	1.7 ± 1.1	0.1
5	2.1 ± 1.3	0.125	2.2 ± 0.98	0.2

Table 4: Full returns and success rates for real robot evaluation comparisons.

Algorithm 2 BOSS Algorithm

Require: Dataset $\mathcal{D}^L$ w/ language instruction labels, LLM, Skill Library Z, Time limit T, max chain length M

Pre-train policy $\pi(a|s)$, value function $V(s,z)$ on $\mathcal{D}^L$ with offline RL. ▷ Section 3.1
while not converged **do**
    SkillBootstrapping(V, Z, LLM, π, $\mathcal{D}^L$, M, T) ▷ Section 3.2
end while

procedure SkillBootstrapping(V, Z, LLM, π, $\mathcal{D}^L$, M, T)
    $s_1 \leftarrow$ Reset environment
    `RolloutData` $\leftarrow$ []
    $z \leftarrow$ sample from categorical distribution with parameters $\big[V(s,z_1), V(s,z_2), ..., V(s,z_{|Z|})\big]$.
    $i \leftarrow 0$
    `Success` $\leftarrow$ True
    while $i < M$ and `Success` **do**
        $i \leftarrow i + 1$
        (`Success`, τ) $\leftarrow$ Rollout $\pi(\cdot|s,z)$ in Environment for at most T steps.
        `RolloutData.append(`τ`)`
        if `Success` **then**
            $z \leftarrow$ SampleNextSkill(LLM, RolloutData, Z)
        end if
    end while
    UpdateBufferAndSkillRepertoire($\mathcal{D}^L$, RolloutData, LLM)
    Train π, V on $\mathcal{D}^L$ with offline RL.
end procedure

procedure SampleNextSkill(LLM, `RolloutData`, Z)
    `AllSkills` $\leftarrow$ extract all skill annotations from Z.
    `CompletedSkillChain` $\leftarrow$ extract executed primitive skills in current rollout from `RolloutData`.
    `Prompt` $\leftarrow$ construct prompt from `AllSkills`, `CompletedSkillChain`. ▷ Prompt in Figure 8.
    $([\hat{z}_1, ..., \hat{z}_N], [p_1, ..., p_N]) \leftarrow$ Sample N text generations from LLM(`Prompt`) with average token probabilities $p_1, ..., p_N$.
    Find closest match in Z to each of $\hat{z}_1, ..., \hat{z}_N$ in embedding space from a sentence embedding model. ▷ Embedding model: `all-mpnet-base-v2` from the SentenceTransformers package [42].
    $z \leftarrow$ sample the matches in Z from categorical distribution with parameters $p_1, ..., p_N$.
    return z
end procedure

procedure UpdateBufferAndSkillRepertoire($\mathcal{D}^L$, `RolloutData`, Z, LLM) ▷ See Appendix B.3 for details.
    $\tau_1, ..., \tau_k \leftarrow$ extract completed and attempted primitive skill trajectories from `RolloutData`.
    for τ_i in $\tau_1, ..., \tau_k$ **do**
        $\mathcal{D}^L \leftarrow \mathcal{D}^L \cup \{\tau_{i,z_i}\}$ ▷ Add primitive skill trajectory to $\mathcal{D}^L$ with primitive skill annotation z_i.
    end for
    $\tau_{1:k} \leftarrow$ concatenate all trajectories together
    $z_{LLM,1:k} \leftarrow LLM(\tau_{1:k})$ assign new skill name from LLM by asking it to write a high-level summary of annotations of $\tau_{1:k}$. ▷ See Appendix A.2 for this prompt.
    $z_{concat,1:k} \leftarrow$ "$\{z_1\}.\{z_2\}...\{z_k\}.$" ▷ Assign another label for the trajectory by concatenating sentences
    $\mathcal{D}^L \leftarrow \mathcal{D}^L \cup \{\tau_{LLM,1:k}, \tau_{concat,1:k}\}$ ▷ Add to $\mathcal{D}^L$ with annotation $z_{LLM,1:k}$ and $z_{concat,1:k}$.
    Add $z_{LLM,1:k}$ as a new skill to Z.
end procedure