LISA: Learning Interpretable Skill Abstractions from Language

Divyansh Garg* **Skanda Vaidyanath*** **Kuno Kim**
Jiaming Song **Stefano Ermon**
Stanford University
{divgarg, svaidyan, khkim, tsong, ermon} @stanford.edu

Abstract

Learning policies that effectively utilize language instructions in complex, multi-task environments is an important problem in imitation learning. While it is possible to condition on the entire language instruction directly, such an approach could suffer from generalization issues. To encode complex instructions into skills that can generalize to unseen instructions, we propose *Learning Interpretable Skill Abstractions (LISA)*, a hierarchical imitation learning framework that can learn diverse, interpretable skills from language-conditioned demonstrations. LISA uses vector quantization to learn discrete skill codes that are highly correlated with language instructions and the behavior of the learned policy. In navigation and robotic manipulation environments, LISA outperforms a strong non-hierarchical baseline in the low data regime and is able to compose learned skills to solve tasks containing unseen long-range instructions. Our method demonstrates a more natural way to condition on language in sequential decision-making problems and achieve interpretable and controllable behavior with the learned skills.

1 Introduction

Intelligent machines should be able to solve a variety of complex, long-horizon tasks in an environment and generalize to novel scenarios. In the sequential decision-making paradigm, provided expert demonstrations, an agent can learn to perform these tasks via multi-task imitation learning (IL). As humans, it is desirable to specify tasks to an agent using a convenient, yet expressive modality and the agent should solve the task by taking actions in the environment. There are several ways for humans to specify tasks to an agent, such as task IDs, goal images, and goal demonstrations. However, these specifications tend to be ambiguous, require significant human effort, and can be cumbersome to curate and provide at test time. One of the most natural and versatile ways for humans to specify tasks is via natural language.

The goal of language-conditioned IL is to solve tasks in an environment given language-conditioned trajectories at training time and a natural language instruction at test time. This becomes challenging when the task involves completing several sub-tasks sequen-

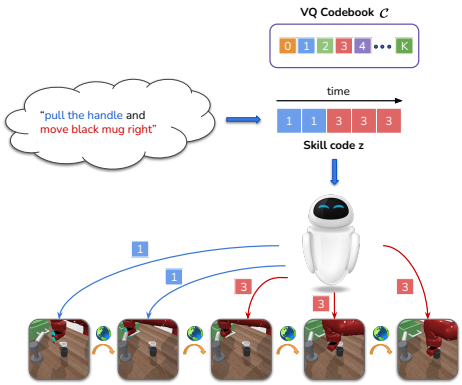

Figure 1: **Overview of LISA.** Given a language instruction, our method learns discrete skill abstractions z, picked from a codebook $\mathcal{C}$. The policy conditioned on the skill code learns to execute distinct behaviors and solve different sub-goals. See GIF.

*Equal contribution.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

tially, like the example shown in Figure 1. A crucial step towards solving this problem is exploiting the inherent hierarchical structure of natural language. For example, given the task specification "pull the handle and move black mug right", we can split it into two independent *skills*, *i.e.* "pull the handle" and "move black mug right". If we are able to decompose the problem of solving these complex tasks into learning skills, we can compose these skills to generalize to unseen tasks in the future. This is especially useful in the low-data regime, since *we may not see all possible tasks given the limited dataset, but may see all the constituent sub-tasks.* One of the main goals of language conditioned IL is to utilise language effectively so that we can learn skills as the building blocks of complex behaviours.

Utilising language effectively to learn skills is a non-trivial problem and raises several challenges. (i) The process of learning skills from language-conditioned trajectories is unsupervised as we may not have knowledge about which parts of the trajectory corresponds to each skill. (ii) We need to ensure that the learned skills are useful, *i.e.* encode behavior that can be composed to solve new tasks. (iii) We would like the learned skills to be interpretable by humans, both in terms of the language and the behaviours they encode. There are several benefits of interpretability. For example, it allows us to understand which skills our model is good at and which skills it struggles with. In safety critical settings such as robotic surgery or autonomous driving, knowing what each skill does allows us to pick and choose which skills we want to run at test time. It also provides a visual window into a neural network policy which is extremely desirable [54]. There have been prior works such as [38, 47, 13] that have failed to address these challenges and condition on language in a monolithic fashion without learning skills. As a result, they tend to perform poorly on long-horizon composition tasks such as the one in Figure 1.

To this end, we propose **L**earning **I**nterpretable **S**kill **A**bstractions from language **(LISA), a hierarchical imitation learning framework that can learn interpretable skills from language-conditioned offline demonstrations.** LISA uses a two-level architecture – a skill predictor that predicts quantized skill codes and a policy that uses these skill codes to predict actions. The discrete skill codes learned from language are interpretable (see Figure 4) and can be composed to solve long-range tasks. Using skill quantization maximizes code reuse and enforces a bottleneck to pass information from the language to the policy, enabling unsupervised learning of interpretable skills. We perform experiments on grid world navigation and robotic manipulation tasks and show that our hierarchical method can outperform a strong non-hierarchical baseline in the low-data regime. We analyse these skills qualitatively and quantitatively and find them to be highly correlated to language and behaviour. Finally, using these skills to perform long-range composition tasks on a robotic manipulation environment results in performance that is nearly *2x better* than the non-hierarchical version.

Concretely, our contributions are as follows:

- We introduce LISA, a novel hierarchical imitation framework conditioned on language to decompose complex tasks into skills.
- We demonstrate the effectiveness of our approach in the low-data regime where its crucial to break down complex tasks to generalize well.
- We show our method performs well in long-range composition tasks where we may need to perform multiple skills sequentially.
- We also show that the learned skills are highly correlated to language and behaviour and can easily be interpreted by humans.

2 Related Work

2.1 Imitation Learning

Imitation learning (IL) has a long history, with early works using behavioral cloning [41–43] to learn policies via supervised learning on expert demonstration data. Recent methods have shown significant improvements via learning reward functions [21] or Q-functions [18] from expert data to mimic expert behavior. Nevertheless, these works typically consider a single task. An important problem here is multi-task IL, where the imitator is trained to mimic behavior on a variety of training tasks with the goal of generalizing the learned behaviors to test tasks. A crucial variable in the multi-task IL set-up is *how the task is specified*, e.g vectorized representations of goal states [37], task IDs [24], and single demonstrations [56, 14, 16, 57]. In contrast, we focus on a multi-task IL setup

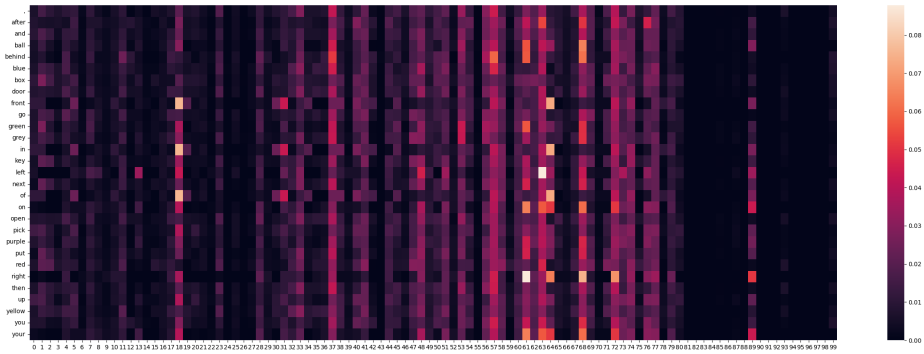

Figure 2: **LISA Skill Heat map.** We show the corresponding Word Freq. for each learnt skill code from 0-99 on BabyAI BossLevel task. (zoom in for best view)

with task-specification through language, one of the most natural and versatile ways for humans to communicate desired goals and intents.

2.2 Language Grounding

Several prior works have attempted to ground language with tasks or use language as a source of instructions for learning tasks with varying degrees of success [32, 55, 4, 39, 5]. [27] is a good reference for works combining language with sequential-decision making.

But apart from a few exceptions, most algorithms in this area use the language instruction in a monolithic fashion and are designed to work for simple goals that requires the agent to demonstrate a single skill. [40, 9, 20, 6] or tasks where each constituent sub-goal has to be explicitly specified [10, 46, 34, 3, 52, 50, 17, 35, 31]. Some recent works have shown success on using play data [28] or *pseudo-expert* data such as LOReL [38] and CLIPORT [47]. LOReL and CLIPORT are not hierarchical techniques. [28] can be interpreted as a hierarchical technique that generates latent sub-goals as a function of goal images, language instructions and task IDs but the skills learned by LISA are purely a function of language and states alone and do not require goal images or task IDs. [23, 22] and [49] are some examples of works that use a two-level architecture for language conditioned tasks but neither of these methods learn skills that are interpretable.

2.3 Latent-models and Hierarchical Learning

Past works have attempted to learn policies conditioned on latent variables and some of them can be interpreted as hierarchical techniques. For example, [15] learns skills using latent variables that visit different parts of the environment's state space. [45] improved on this by learning skills that were more easily predictable using a dynamics model. But these fall more under the category of skill discovery than hierarchical techniques since the skill code is fixed for the entire trajectory, as is the case with [15]. [29] and [26] are other works that use a latent-variable approach to IL. But these approaches don't necessarily learn a latent variable with the intention of breaking down complex tasks into skills. With LISA, we sample several skills per trajectory with the clear intention of each skill corresponding to completing a sub-task for the whole trajectory. Also, none of the methods mentioned here condition on language.

There has been some work on hierarchical frameworks for RL to learn high-level action abstractions, called options [51], such as [25, 58, 36] but these works are not goal-conditioned. Unlike LISA, these works don't use language and the options might lack diversity and not correspond to any concrete or interpretable skills. Furthermore, none have used the VQ technique to learn options and often suffer from training instabilities.

3 Approach

The key idea of LISA is to learn quantized skill representations that are informative of both language and behaviors, which allows us to break down high-level instructions, specified via language, into discrete, interpretable and composable codes (see Fig. 2, Fig. 6 and Fig. 8 for visualizations). These codes enable learning explainable and controllable behaviour, as shown in Fig. 1 and Fig. 4.

Section 3.1 describes the problem formulation, an overview of our framework, and presents our language-conditioned model. Section 3.2 provides details on the training approach.

3.1 Language-conditioned Skill Learning

3.1.1 Problem Setup

We consider general multi-task environments, represented as a task-augmented Markov decision process (MDP) with a family of different tasks $\mathcal{T}$. A task $\mathcal{T}_i$ may be composed of other tasks in $\mathcal{T}$ and encode multiple sub-goals. For example, in a navigation environment, a task could be composed of two or more sub-tasks - "pick up ball", "open door" - in any hierarchical order. $\mathcal{S}, \mathcal{A}$ represent state and action spaces. We assume that each full task has a *single* natural language description $l \in L$, where L represents the space of language instructions. Any sub-goals for the task are encoded within this single language instruction.

We assume access to an offline dataset $\mathcal{D}$ of trajectories obtained from an optimal policy for a variety of tasks in an environment with only their language description available. Each trajectory $\tau^i = (l^i, \{(s_1^i, a_1^i), (s_2^i, a_2^i), ..., (s_T^i, a_T^i)\})$ consists of the language description and the observations $s_t^i \in \mathcal{S}$, actions $a_t^i \in \mathcal{A}$ taken over T timesteps. The trajectories are not labeled with any rewards.

Our aim is to predict the expert actions a_t, given a language instruction and past observations.

Note that each trajectory in the training dataset can comprise of any number of sub-tasks. For example, we could have a trajectory to "open a door" and another to "pick up a ball and close the door" in the training data. With LISA we aim to solve the task "open a door and pick up the ball" at test time even though we haven't seen this task at training time. In a trajectory with multiple sub-tasks, the training dataset **does not** give us information about where one sub-task ends and where another one begins.

LISA must learn how to identify and stitch together these sub-tasks learned during training, in order to solve a new language instruction such as the one shown in Fig. 1 at test time.

3.1.2 Hierarchical Skill Abstractions

We visualize the working of LISA in Figure 3. Our framework consists of two modules: a skill predictor $f : L \times \mathcal{S} \rightarrow \mathcal{C}$ and a policy $\pi : \mathcal{S} \times \mathcal{C} \rightarrow \mathcal{A}$. Here, $\mathcal{C} = \{z^1, \ldots, z^K\}$ is a learnable codebook of K quantized skill codes.

Our key idea is to break learning behavior from language in two stages: 1) Learn discrete codes z, representing skills, from the full-language instruction to decompose the task into smaller sub-goals 2) Learn a policy π conditioned only on these discrete codes. In LISA, both stages are trained end-to-end.

Given an input $\tau = (l, \{s_t, a_t\}_{t=1}^T)$, the skill predictor f predicts a skill code at a timestep t as $\tilde{z} = f(l, (s_t, s_{t-1}, ...))$. These codes are discretized using a vector quantization operation $\mathbf{q}(\cdot)$ that maps a code $\tilde{z}$ to its closest codebook entry $z = \mathbf{q}(\tilde{z})$. The quantization operation $\mathbf{q}(\cdot)$ helps in learning discrete codes and acts as a bottleneck on passing language information. We detail its operation in Sec. 3.2.

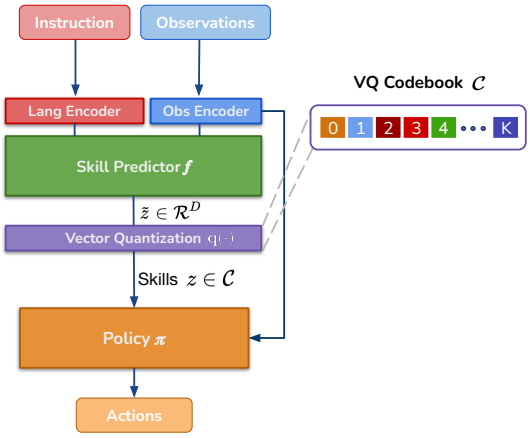

Figure 3: **LISA Architecture**: The skill predictor f gets the language instruction and a sequence of observations as the input, processed through individual encoders. It predicts quantized skill codes z using a learnable cookbook $\mathcal{C}$, that encodes different sub-goals, and passes them to the policy π. LISA is trained end-to-end.

The chosen skill code z, is persisted for H timesteps where H is called the horizon. More details on how we chose the horizon and ablations studies on the choice of H can be found in appendix sections D and F.1. After H timesteps, the skill predictor is invoked again to predict a new skill. This enforces the skill to act as a temporal abstraction, i.e. options [51]. The policy π predicts the action a_t at each timestep t conditioned on the state and a single skill code z that is active at that timestep. For π to correctly predict the original actions, it needs to use the language information encoded in the skill codes.

Algorithm 1 Training LISA

Input: Dataset $\mathcal{D}$ of language-paired trajectories
Input: Num skills K and horizon H
Initialize skill predictor f_ϕ, policy π_θ
Vector Quantization op $\mathbf{q}(\cdot)$
while *not converged* **do**
  Sample $\tau = (l, \{s_0, s_1, s_2 ... s_T\}, \{a_0, a_1, a_2 ... a_T\})$
  Initialize $S = \{s_0\}$ ▷ List of seen states
  for $k = 0..\left\lfloor \frac{T}{H} \right\rfloor$ **do** ▷ Sample a skill every H steps
    $z \leftarrow \mathbf{q}(f_\phi(l, S))$
    for *step* $t = 1..H$ **do**
      $a_{kH+t} \leftarrow \pi_\theta(z, S[: -H])$ ▷ Use recent H steps
      $S \leftarrow S \cup \{s_{kH+t}\}$ ▷ Append last state
    end for
    Train f_ϕ, π_θ using objective $\mathcal{L}_{\text{LISA}}$
  end for
end while

LISA learns quantized skill codes in a codebook instead of continuous embeddings as this encourages reusing and composing these codes together to pass information from the language input to the actual behavior. Our learnt discrete skill codes adds interpretability and controllability to the policy's behavior.

3.2 Training LISA

Learning Discrete Skills. LISA uses Vector Quantization (VQ), inspired from [53]. It is a natural and widely-used method to map an input signal to a low-dimensional discrete learnt representation. VQ learns a codebook $\mathcal{C} \in \left\{z^1, \ldots, z^K\right\}$ of K embedding vectors. Given an embedding $\tilde{z}$ from the skill predictor f, it maps the embedding to the closest vector in the codebook:

$$z = \mathbf{q}(\tilde{z}) =: \underset{z^k \in \mathcal{C}}{\arg\min} \|\tilde{z} - z^k\|_2$$

with the codebook vectors updated to be the moving average of the embeddings z closest to them. This can be classically seen as learning K cluster centers via k-means [19].

Backpropagation through the non-differentiable quantization operation is achieved by a straight-through gradient estimator, which simply copies the gradients from the decoder to the encoder, such that the model and codebook can be trained end-to-end.

VQ enforces each learnt skill z to lie in $\mathcal{C}$, which can be thought as learning K prototypes or cluster centers for the language embeddings using the seen states. This acts as a bottleneck that efficiently decomposes a language instruction into sub-parts encoded as discrete skills.

LISA Objective. LISA is trained end-to-end using an objective $\mathcal{L}_{\text{LISA}} = \mathcal{L}_{\text{BC}} + \lambda \mathcal{L}_{\text{VQ}}$, where $\mathcal{L}_{\text{BC}}$ is the behavior-cloning loss on the policy π_θ, λ is the VQ loss weight and $\mathcal{L}_{\text{VQ}}$ is the vector quantization loss on the skill predictor f_ϕ given as:

$$\mathcal{L}_{\text{VQ}}(f) = \mathbb{E}_\tau[\|\text{sg}\left[\mathbf{q}(\tilde{z})\right] - \tilde{z}\|_2^2] \tag{1}$$

with $\tilde{z} = f_\phi(l, (s_t, s_{t-1}, ..))$.

sg $[\cdot]$ denotes the stop-gradient operation. $\mathcal{L}_{\text{VQ}}$ is also called *commitment loss*. It minimizes the conditional entropy of the skill predictor embeddings given the codebook vectors, making the embeddings stick to a single codebook vector.

The codebook vectors are learnt using an exponential moving average update, same as [53].

Avoiding language reconstruction. LISA avoids auxiliary losses for language reconstruction and it's not obvious why the skill codes are properly encoding language. It's known that given a signal X and a code Z, reconstructing the signal $\tilde{X} = f(Z)$ using cross-entropy loss amounts to maximizing a lower bound to the Mutual Information (MI) $I(X, Z)$ between X and Z [1, 7]. In our case, we can write the MI between the skill codes and language using entropies as: $I(z, l) = H(z) - H(z \mid l)$,

Figure 4: **Behavior with fixed LISA options.** We show the word clouds and the behavior of the policy obtained by using a fixed skill code $z = 14$ for an entire episode. We find that this code encodes the skill "closing the drawer", as indicated by the word cloud. The policy executes this skill with a high degree of success when conditioned on this code for the entire trajectory, across different environment initializations and seeds.

whereas methods that attempt to reconstruct language apply the following decomposition: $I(z, l) = H(l) - H(l \mid z)$ (where $H(l)$, the entropy of language, is independent from LISA's skill encoder).

Thus we can avoid language reconstruction via cross-entropy loss by maximizing $I(z, l)$ directly. In LISA, $\mathcal{L}_{\text{vq}} = -H(z \mid l)$, and we don't observe a need to place a constraint on $H(z)$ as the codes are diverse, needing to encode enough information to correctly predict the masked actions.[1]

As a result, LISA can maximize the MI between the learnt skills and languages without auxiliary losses and enforcing only $\mathcal{L}_{\text{vq}}$ on the skill codes. We empirically estimate the MI and find that our experiments confirm this in Sec 4.6.

3.2.1 LISA Implementation

LISA can be be implemented using different network architectures, such as Transformers or MLPs.

In our experiments, we use Transformer architectures with LISA, but we find that out method is effective even with simple architectures choices such as MLPs, as shown in the appendix section F.5. Even when using Transformers for both the skill predictor and the policy network, our compute requirement is comparable to the non-hierarchical Flat Transformer policy as we can get away with using fewer layers in each module.

Language Encoder. We use a pre-trained DistilBERT [44] encoder to generate language embeddings from the text instruction. We fine-tune the language encoder end-to-end and use the full language embedding for each word token, and not a pooled representation of the whole text.

Observation Encoder. For image observations, we use convolution layers to generate embeddings. For simple state representations, we use MLPs.

Skill Predictor. The skill predictor network f is implemented as a small Causal Transformer network that takes in the language embeddings and the observation embeddings at each time step. The language embeddings are concatenated at the beginning of the observation embeddings before being fed into the skill predictor. The network applies a causal mask hiding the future observations.

Policy Network. Our policy network π, also implemented as a small Causal Transformer inspired by Decison Transformer (DT) [11]. However, unlike DT, our policy is not conditioned on any reward signal, but on the skill code. The sequence length of π is the horizon H of the skills which is much smaller compared to the length of the full trajectory.

Flat Baseline. Our flat baseline is implemented similar to LISA, but without a skill predictor network. The policy here is a Causal Transformer that directly takes the language instruction and past observations as inputs to predict the policy. We found this baseline to be in-efficient at handling long-range language instructions, needing sequence lengths of 1000 on complex environments such as BabyAI-BossLevel in our experiments.

Since LISA has two transformers as opposed to just one in the flat baseline we ensured that the baseline and our method had a similar number of total parameters. To this end, the flat baseline uses a 2-layer Transformer network, and LISA's skill predictor and policy use a 1-layer Transformer network each. We also ensured that the embedding dimension and the number of heads in each layer were exactly the same in both LISA and the flat baseline. Details of this are provided in appendix sections D.1 and D.2 respectively. In fact, one could argue that LISA has less representation power because the policy transformer can only attend to the last H steps while the flat baseline can attend to the entire trajectory which is what makes it an extremely strong baseline. The flat baseline also

[1] In experiments, we tried enforcing a constraint on $H(z)$ by using extra InfoNCE loss term without success.

uses the same pre-trained DistillBERT text encoder model as LISA for dealing with natural language input.

4 Experiments

In this section, we evaluate LISA on grid-world navigation and robotic manipulation tasks. We compare the performance of LISA with a strong non-hierarchical baseline in the low-data regime. We then analyse our learnt skill abstractions in detail – what they represent, how we can interpret them and how they improve performance on downstream composition tasks.

4.1 Datasets

Several language-conditioned datasets have been curated as of late. [46, 48, 13, 38, 33, 2, 10, 12] are some examples. Nevertheless, a lot of these datasets focus on complex-state representations and navigation in 3D environments, making them challenging to train on and qualitatively analyze our skills as shown in Fig. 4. We found BabyAI, a grid-world navigation environment and LOReL, a robotic manipulation environment as two diverse test beds that were very different from each other and conducive for hierarchical skill learning as well as detailed qualitative and quantitative analysis of our learned skills and we use them for our experiments.

BabyAI Dataset. The BabyAI dataset [13] contains 19 levels of increasing difficulty where each level is set in a grid world and an agent sees a partially observed ego-centric view in a square of size 7x7. The agent must learn to perform various tasks of arbitrary difficulty such as moving objects between rooms, opening or closing doors, etc. all with a partially observed state and a language instruction. The language instructions for easy levels are quite simple but get exponentially more challenging for harder levels and contain several skills that the agent must complete in sequence (examples in appendix section C.1). The dataset provides 1 million expert trajectories for each of the 19 levels, but we use $0.1 - 10\%$ of these trajectories to train our models. We evaluate our policy on 100 different instructions from the gym environment for each level, which very likely contains unseen environments and language instructions given the limited data we use for training. More details about this dataset can be found in the appendix and in the BabyAI paper.

LOReL Sawyer Dataset. This dataset [38] consists of *pseudo-expert* trajectories (play data) collected from a replay buffer of a random RL policy and has been labeled with post-hoc crowd-sourced language instructions. Hence, the trajectories complete the language instruction provided but may not necessarily be optimal. Play data is inexpensive to collect [30] in the real world and it is important for algorithms to be robust to such datasets as well. However, due to the randomness in the trajectories, this makes the dataset extremely difficult to use in a behavior cloning (BC) setting. Despite this, we are able to achieve good performance on this benchmark and are able to learn some very useful skills. The LOReL Sawyer dataset contains 50k trajectories of length 20 on a simulated environment with a Sawyer robot. We evaluate on the same set of 6 tasks that the original paper does for our results in Table 1: *close drawer, open drawer, turn faucet right, turn faucet left, move black mug right, move white mug down*. We use two different settings - with robot state space observations and partially-observed image observations. More details can be found in the appendix section C.2 and in the LOReL paper.

4.2 Baselines

Original. These refer to the baselines from the original paper for each dataset. For BabyAI, we trained their non-hierarchical RNN based method on different number of trajectories. Similarly, on LOReL we compare with the performance of language-conditioned BC. The original LOReL method uses a planning algorithm on a learned reward function to get around the sub-optimal nature of the trajectories. We found the BC baseline as a more fair comparison, as LISA is trained using BC as well. Nonetheless, we compare with the original LOReL planner in Section 4.7 for composition tasks. LOReL results in Table 1 refer to the performance on the 6 *seen* instructions in the LOReL evaluation dataset, same as ones reported in the original paper.

Flat Baseline. We implement a non-hierarchical baseline using Transformers, the details of which are in section 3.2.1.

We optimized a language-conditioned BC model following the details in the appendix of the LOReL paper to the best of our abilities but could not get better performance.

Table 1: **Imitation Results:** We show our success rates (in %) compared to the original method and a flat non-hierarchical baseline on each dataset. LISA outperforms all other methods in the low-data regime, and reaches similar performance as the number of demonstrations increases. Best method shown in **bold**.

Task	Num Demos	Original	Flat Baseline	LISA
BabyAI GoToSeq	$1k$	33.3 ± 1.3	49.3 ± 0.7	$\mathbf{59.4 \pm 0.9}$
BabyAI GoToSeq	$10k$	40.4 ± 1.2	62.1 ± 1.2	$\mathbf{65.4 \pm 1.6}$
BabyAI GoToSeq	$100k$	47.1 ± 1.1	74.1 ± 2.3	$\mathbf{77.2 \pm 1.7}$
BabyAI SynthSeq	$1k$	12.9 ± 1.2	42.3 ± 1.3	$\mathbf{46.3 \pm 1.2}$
BabyAI SynthSeq	$10k$	32.6 ± 2.5	52.1 ± 0.5	$\mathbf{53.3 \pm 0.7}$
BabyAI SynthSeq	$100k$	40.4 ± 3.3	$\mathbf{64.2 \pm 1.3}$	61.2 ± 0.6
BabyAI BossLevel	$1k$	20.7 ± 4.6	44.5 ± 3.3	$\mathbf{49.1 \pm 2.4}$
BabyAI BossLevel	$10k$	28.9 ± 1.3	$\mathbf{60.1 \pm 5.5}$	58 ± 4.1
BabyAI BossLevel	$100k$	45.3 ± 0.9	$\mathbf{72.0 \pm 4.2}$	69.8 ± 3.1
LOReL - States (fully obs.)	$50k$	6 ± 1.2[1]	33.3 ± 5.6	$\mathbf{66.7 \pm 5.2}$
LOReL - Images (partial obs.)	$50k$	29.5 ± 0.07	15 ± 3.4	$\mathbf{40 \pm 2.0}$

4.3 How does the performance of LISA compare with non-hierarchical baselines in a low-data regime?

We consider three levels from the BabyAI environment and the LOReL Sawyer environment. From the BabyAI environment, we consider the GoToSeq, SynthSeq and BossLevel tasks since they are challenging and require performing several sub-tasks one after the other. Since these levels contain instructions that are compositional in nature, when we train on limited data, the algorithm must be able to learn the skills which form these complex instructions to generalize well to unseen instructions at test time. Our results are given in Table 1. We train the models on a random sample of 1k, 10k and 100k trajectories on the BabyAI dataset and 50k trajectories on the LOReL dataset. We use more data from the LOReL dataset because of the sub-optimal nature of the trajectories. On all the environments, our method is competitive to or outperforms the strong non-hierarchical decision transformer baseline. The gap grows larger as we reduce the number of trajectories we train on, indicating that our method is able to leverage the common sub-task structures better and glean more information from limited data. As expected, with larger amounts of training data, it becomes hard to beat the flat baseline since the Transformer sees more compositions during training and can generalize better at test time [8]. As mentioned above, we evaluate on the same 6 *seen* instructions the original LOReL paper did. We also evaluated the performance on varying the language instructions, similar to the original paper with results in appendix section E.

We were pleasantly surprised that LISA is 2x better than the flat baseline on LOReL tasks, reaching 40% success rate despite the sub-optimal nature of the data. One explanation for this is that the discrete skill codes are able to capture *different ways of doing the same task*, thereby allowing LISA to *learn an implicit multi-modal policy*. This is not possible with the flat version as it has no way to compartmentalize these noisy trajectories, and perhaps tends to overfit on this noisy data, leading to performance degradation.

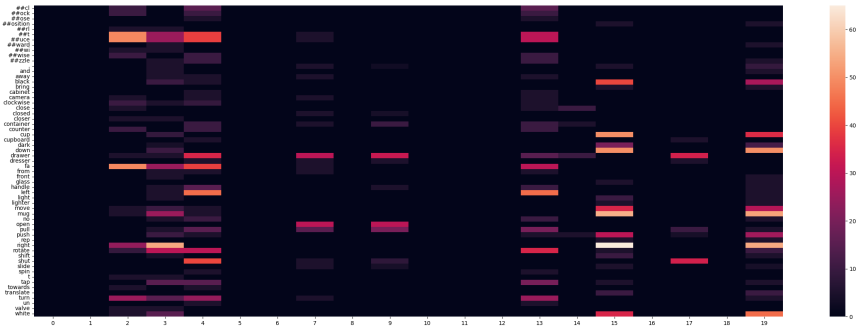

Figure 5: **LISA Skill Heat map on LOReL.** The sparsity and the bright spots show that specific skills correspond to specific language tokens. (zoom in for best view)

4.4 What skills does LISA learn? Are they diverse?

To answer this question, we analyse the skills produced by LISA and the language tokens corresponding to each skill. We plot a heat map in Figure 5 corresponding to the correlation between the language tokens and skill codes. Here, we plot the map corresponding to the LOReL dataset. From the figure, we can see that certain skill codes correspond very strongly to certain language tokens and by extension, tasks. We also see the sparse nature of the heat maps which indicates that each skill corresponds to distinct language tokens. We also plot word clouds corresponding to four different options in the LOReL environment in figure 6 and we notice that different options are triggered by different language tokens. From the figure, it is clear that the skill on the top left corner corresponds to *close the drawer* and the skill on the top right corresponds to *turn faucet left*. Similar word clouds and heat maps for the BabyAI environments are in the appendix section B.3.

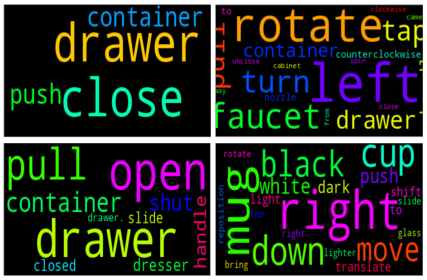

Figure 6: **Word clouds on LOReL**: We show the most correlated words for 4 different learnt skill codes on LOReL. We can see that the codes represent interpretable and distinguishable skills. For e.g, the code on the top left corresponds to closing the drawer. (note that container is a synonym for drawer in the LOReL dataset)

4.5 Do the skills learned by LISA correspond to interpretable behavior?

We have seen that the different skills correspond to different language tokens, but do the policies conditioned on these skills behave according to the language tokens? To understand this, we fix the skill code for the entire trajectory and run the policy i.e. we are shutting off the skill predictor and always predicting the same skill for the entire trajectory. As we can see from the word cloud and the corresponding trajectory in Figure 4, the behaviour for skill code 14 is exactly what we can infer from the language tokens in the word cloud – *close the drawer*. More such images and trajectories can be found in the appendix section B.5.

4.6 Why do the skills learned by LISA have such a strong correlation to language?

As mentioned in section 3.2, the *commitment loss* from VQ acts as a way to increase the MI between the language and the skill codes during training. This allows the codes to be highly correlated with language without any reconstruction losses. To analyze this, we plot the MI between the options and the language during training on the BabyAI BossLevel with 1k trajectories and the plot can be seen in Figure 7. The plots show the MI increasing over training for a wide range of settings as we vary the number of skills and the horizon. In the ablation studies below, we report the success rate corresponding to each of these curves and we notice that there's almost a direct correlation with increasing MI and task performance. This is very encouraging since it clearly shows that the skills are encoding language and that directly impacts the performance of the behavior cloning policy.

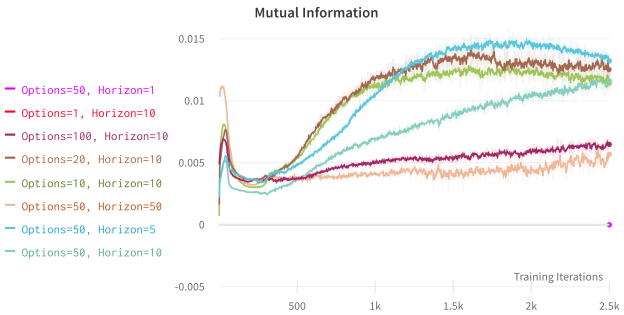

Figure 7: **MI between language and skill codes:** We show the Mutual Information over training iterations for various settings of LISA on the BabyAI BossLevel environment

4.7 Can we use the learned skills to perform new composition tasks?

To test our composition performance, we evaluate on LOReL composition tasks using images in Table 2. To this end, we handcraft 15 **unseen** composition instructions. We have listed these instructions in the Appendix Table 5 with one such example *"pull the handle and move black mug down"*. We ran 10 different runs of each instruction across 3 different seeds. As we can see, our performance is nearly 2x that of the non-hierarchical baseline. We also compare with the original LOReL planner on

Table 2: **LISA Composition Results:** We show our performance on the LOReL Sawyer environment on 15 unseen instructions compared to baselines

Method	Success Rate (in %)
Flat	13.33 ± 1.25
LOReL Planner	18.18 ± 1.8
LISA (Ours)	**20.89 ± 0.63**

these composition tasks and we notice that we perform slightly better despite them having access to a reward function and a dynamics model pre-trained on 1M frames while LISA is trained from scratch. We set the max number of episode steps to 40 from the usual 20 for all the methods while performing these experiments because of the compositional nature of the tasks.

Note that results in Table 1 show compositionality performance on the BabyAI dataset as we train with 0.1%-10% of the data. When we evaluate on the gym environment generating any possible language instruction from the BabyAI grammar, we may come across several unseen instructions at test time. To give a sense of the % of unseen instructions for BabyAI when we evaluate on the gym environment, we take the different BabyAI environments and report the % of unseen language instructions seen at test time for different training data regimes in Table 3. For each statistic, we sample 10,000 random instructions from the environment and check how many are unseen in the training dataset used, repeated over 3 different seeds.

Table 3: **BabyAI:** % of unseen instructions at test time for different training regimes.

Environment	1k trajs	10k trajs	100k trajs
GoToSeq	76%	63.8%	48%
SynthSeq	76.3%	66%	50.7%
BossLevel	77.3%	66.9%	51.3%

4.8 Is LISA just doing k-means clustering on the language and state embeddings? If not, how would LISA compare to this simple baseline?

In LISA, the VQ approach can be seen as taking the concatenated language-state inputs and projecting it into a learned embedding space. VQ here simply learns K embedding vectors that act as K cluster centers for the projected input vectors in this embedding space, and allows for differentiability, enabling learning through backpropagation. This is also similar to propotypical methods used for few-shot learning and allows for deep differentiation clustering, giving an intuition of why it works.

To compare against k-means, we construct a simple unsupervised learning baseline that clusters trajectories using k-means. Specifically, in the BabyAI BossLevel environment using 1k training trajectories, we take all concatenated language-state vectors for all trajectories in the dataset and cluster them using k-means and use the assigned cluster centers as the skill codes. We then learn a policy using these skill codes to measure their efficacy and found that LISA has a performance of $\mathbf{49.1 \pm 2.4}\%$ and k-means has a performance of $20.2 \pm 5.2\%$ (over 3 seeds).

Thus, we see that using the simple k-means skills is insufficient to learn a good policy to solve the BabyAI BossLevel task, as the skills are not representative enough of the language instructions. A reason for this is that language and state vectors lie in different embedding spaces, and K-means based on euclidean distance is not optimal on the concatenated vectors.

5 Limitations and Future Work

We present LISA, a hierarchical imitation learning framework that can be used to learn interpretable skill abstractions from language-conditioned expert demonstrations. We showed that the skills are diverse and can be used to solve long-range language tasks and that our method outperforms a strong non-hierarchical baseline in the low-data regime.

However, there are several limitations to LISA and plenty of scope for future work. One limitation of LISA is that there are several hyperparameters to tune that may affect performance like the number of options and the horizon for each option. It certainly helps to have a good idea of the task to decide these hyperparameters even though the ablations show that the method is fairly robust to these choices. Its also useful to learn the horizon for each skill by learning a termination condition and we leave this for future work.

Although our method has been evaluated on the language-conditioned imitation learning setting, its not difficult to modify this method to make it work for image goals or demos, and in the RL setting as well. Its interesting to see if the vector quantization trick can be used to learn goal-conditioned skills in a more general framework.

Acknowledgments and Disclosure of Funding

This research was supported in part by NSF (#1651565, #1522054, #1733686), ONR (N00014-19-1-2145), AFOSR (FA9550-19-1-0024), FLI and Samsung.

References

[1] Felix Agakov and David Barber. Variational information maximization for neural coding. In Nikhil Ranjan Pal, Nik Kasabov, Rajani K. Mudi, Srimanta Pal, and Swapan Kumar Parui, editors, *Neural Information Processing*, pages 543–548, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-30499-9. 5

[2] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 7

[3] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments, 2018. 3

[4] Dilip Arumugam, Siddharth Karamcheti, Nakul Gopalan, Lawson Wong, and Stefanie Tellex. Accurately and efficiently interpreting human-robot instructions of varying granularities. *Robotics: Science and Systems XIII*, Jul 2017. doi: 10.15607/rss.2017.xiii.056. URL http://dx.doi.org/10.15607/RSS.2017.XIII.056. 3

[5] Dilip Arumugam, Siddharth Karamcheti, Nakul Gopalan, Edward C. Williams, Mina Rhee, Lawson L. Wong, and Stefanie Tellex. Grounding natural language instructions to semantic goal representations for abstraction and generalization. *Auton. Robots*, 43(2):449–468, feb 2019. ISSN 0929-5593. doi: 10.1007/s10514-018-9792-8. URL https://doi.org/10.1007/s10514-018-9792-8. 3

[6] Valts Blukis, Nataly Brukhim, Andrew Bennett, Ross A. Knepper, and Yoav Artzi. Following high-level navigation instructions on a simulated quadcopter with imitation learning, 2018. 3

[7] Malik Boudiaf, Jérôme Rony, Imtiaz Masud Ziko, Eric Granger, Marco Pedersoli, Pablo Piantanida, and Ismail Ben Ayed. A unifying mutual information view of metric learning: cross-entropy vs. pairwise losses. In *European Conference on Computer Vision*, pages 548–564. Springer, 2020. 5

[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL https://arxiv.org/abs/2005.14165. 8

[9] Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding, 2018. 3

[10] Howard Chen, Alane Suhr, Dipendra Misra, Noah Snavely, and Yoav Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments, 2020. 3, 7

[11] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling, 2021. 6

[12] Valerie Chen, Abhinav Gupta, and Kenneth Marino. Ask your humans: Using human instructions to improve generalization in reinforcement learning, 2021. 7

[13] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning, 2019. 2, 7, 19

[14] Yan Duan, Marcin Andrychowicz, Bradly Stadie, Openai Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-Shot imitation learning. In I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1087–1098. Curran Associates, Inc., 2017. 2

[15] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function, 2018. 3

[16] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning, 2017. 2

[17] Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. Speaker-follower models for vision-and-language navigation, 2018. 3

[18] Divyansh Garg, Shuvam Chakraborty, Chris Cundy, Jiaming Song, and Stefano Ermon. Iq-learn: Inverse soft-q learning for imitation. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=Aeo-xqtb5p. 2

[19] R.M. Gray and D.L. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 44(6):2325–2383, 1998. doi: 10.1109/18.720541. 5

[20] Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Marian Czarnecki, Max Jaderberg, Denis Teplyashin, Marcus Wainwright, Chris Apps, Demis Hassabis, and Phil Blunsom. Grounded language learning in a simulated 3d world, 2017. 3

[21] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning, 2016. 2

[22] Hengyuan Hu, Denis Yarats, Qucheng Gong, Yuandong Tian, and Mike Lewis. Hierarchical decision making by generating and following natural language instructions, 2019. 3

[23] Yiding Jiang, Shixiang Gu, Kevin Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning, 2019. 3

[24] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale, 2021. 2

[25] Alexander C. Li, Carlos Florensa, Ignasi Clavera, and Pieter Abbeel. Sub-policy adaptation for hierarchical reinforcement learning, 2020. 3

[26] Yunzhu Li, Jiaming Song, and Stefano Ermon. Infogail: Interpretable imitation learning from visual demonstrations, 2017. 3

[27] Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language, 2019. 3

[28] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. *Robotics: Science and Systems*, 2021. URL https://arxiv.org/abs/2005.07648. 3

[29] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. *Conference on Robot Learning (CoRL)*, 2019. URL https://arxiv.org/abs/1903.01973. 3

[30] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. *Conference on Robot Learning (CoRL)*, 2019. URL https://arxiv.org/abs/1903.01973. 7

[31] Chih-Yao Ma, Jiasen Lu, Zuxuan Wu, Ghassan AlRegib, Zsolt Kira, Richard Socher, and Caiming Xiong. Self-monitoring navigation agent via auxiliary progress estimation, 2019. 3

[32] Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, AAAI'06, page 1475–1482. AAAI Press, 2006. ISBN 9781577352815. 3

[33] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin - a benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *arXiv preprint arXiv:2112.03227*, 2021. 7

[34] Dipendra Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning, 2017. 3

[35] Dipendra Misra, Andrew Bennett, Valts Blukis, Eyvind Niklasson, Max Shatkhin, and Yoav Artzi. Mapping instructions to actions in 3d environments with visual goal prediction, 2019. 3

[36] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning, 2018. 3

[37] Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals, 2018. 2

[38] Suraj Nair, Eric Mitchell, Kevin Chen, Brian Ichter, Silvio Savarese, and Chelsea Finn. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation, 2021. 2, 3, 7, 20

[39] Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning, 2017. 3

[40] Chris Paxton, Yonatan Bisk, Jesse Thomason, Arunkumar Byravan, and Dieter Fox. Prospection: Interpretable plans from language by predicting the future, 2019. 3

[41] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991. ISSN 0899-7667. 2

[42] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010.

[43] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011. 2

[44] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020. 6

[45] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills, 2020. 3

[46] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks, 2020. 3, 7

[47] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation, 2021. 2, 3

[48] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning, 2021. 7

[49] Simon Stepputtis, Joseph Campbell, Mariano Phielipp, Stefan Lee, Chitta Baral, and Heni Ben Amor. Language-conditioned imitation learning for robot manipulation tasks, 2020. 3

[50] Alane Suhr and Yoav Artzi. Situated mapping of sequential instructions to actions with single-step reward observation, 2018. 3

[51] Richard Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. 1999. 3, 4

[52] Hao Tan, Licheng Yu, and Mohit Bansal. Learning to navigate unseen environments: Back translation with environmental dropout, 2019. 3

[53] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning, 2018. 5

[54] Jesse Vig. A multiscale visualization of attention in the transformer model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 37–42, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-3007. URL https://www.aclweb.org/anthology/P19-3007. 2

[55] Sida I. Wang, Percy Liang, and Christopher D. Manning. Learning language games through interaction, 2016. 3

[56] Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3795–3802. IEEE, 2018. 2

[57] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot imitation from observing humans via domain-adaptive meta-learning, 2018. 2

[58] Jesse Zhang, Haonan Yu, and Wei Xu. Hierarchical reinforcement learning by discovering intrinsic options, 2021. 3

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See section 5
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See appendix section A
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We will provide a zip file of code with the supplementary material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See appendix section D
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Tables 1 and 2 are both over 3 seeds.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] This information is in the Training details section of section 5
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] The datasets we use are not collected with the help of humans
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] The datasets we use are not collected with the help of humans
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

A Broader Societal Impact

We introduce a new method for language-conditioned imitation learning to perform complex navigation and manipulation tasks. Our intention is for this algorithm to be used in a real-world setting where humans can provide natural language instructions to robots that can carry them out. However, we must ensure that the language commands that we provide to these agents must be well aligned with the objectives of humans and must ensure that we are aware of what actions the agent could take given said command.

B More visualizations

B.1 Generating heat maps and word clouds

To generate heat maps and word clouds, for each evaluation instruction, we run the model and record all the skill codes used in the trajectory generated. We now tokenize the instruction and for each skill code used in the trajectory, record *all* the tokens from the language instruction. Once we have this mapping from skills to tokens, we can plot heat maps and word clouds. This is the best we can do since we don't know exactly which tokens in the instruction correspond to the skills chosen. Therefore, these plots can tend to be a little noisy but we still see some clear patterns. Especially in BabyAI, since the vocabulary is small, we see that several skills correspond to the same tokens because many instructions contain the same tokens. But in LOReL because each task uses almost completely different words, we can see a very sparse heat map with clear correlations.

For reader viewability and aiding the interpretability on the LISA skills, we show below the unnormalized heatmaps showing the skill-word correlations, the column normalized heatmaps showing word frequencies for each skill as well as the row normalized showing the skill frequencies for each word.

B.2 BabyAI

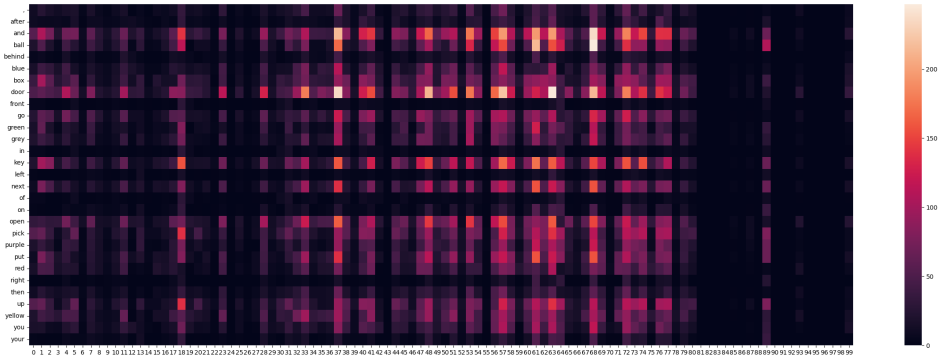

Figure 8: Skill Heat map on BabyAI BossLevel

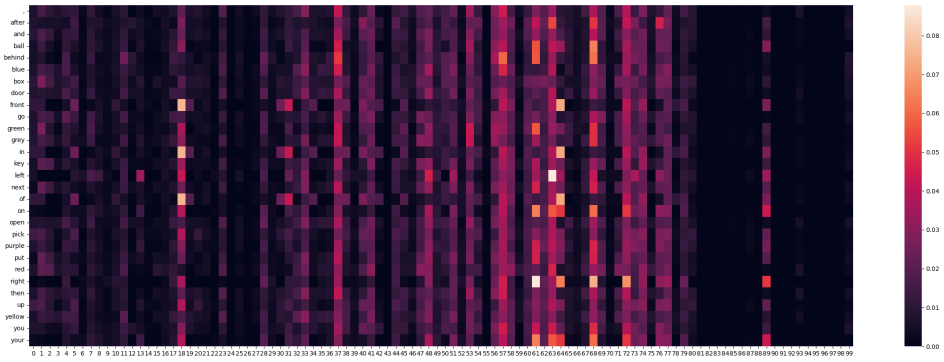

Figure 9: Word Freq. for each skill on BabyAI BossLevel (column normalized)

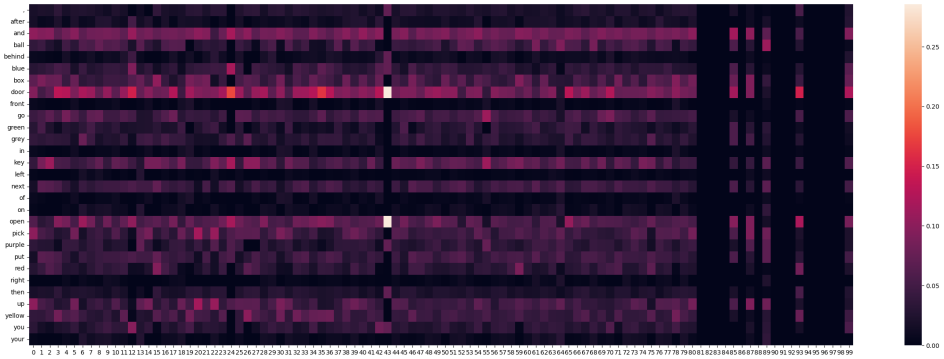

Figure 10: Skill Freq. for each word on BabyAI BossLevel (row normalized)

B.3 WordClouds

Due to the small vocabulary in BabyAI environment, its hard to generate clean word clouds, nevertheless, we hope they help with interpreting LISA skills.

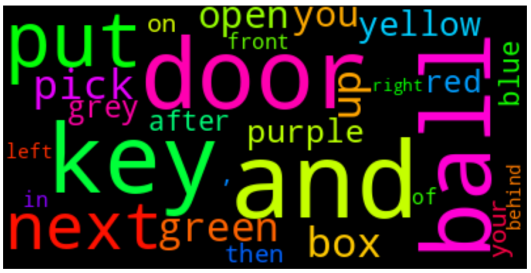

Figure 11: Word Cloud on BabyAI BossLevel for $z = 1$

Figure 12: Word Cloud on BabyAI BossLevel for $z = 13$

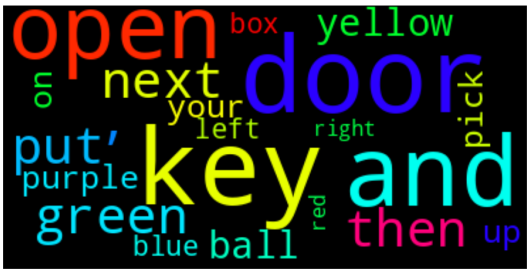

Figure 13: Word Cloud on BabyAI BossLevel for $z = 37$

B.4 LOReL Sawyer

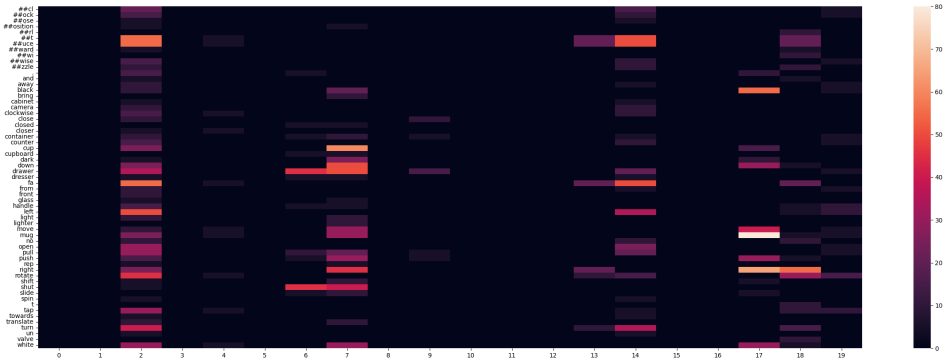

Figure 14: Skill Heat map on LOReL Sawyer

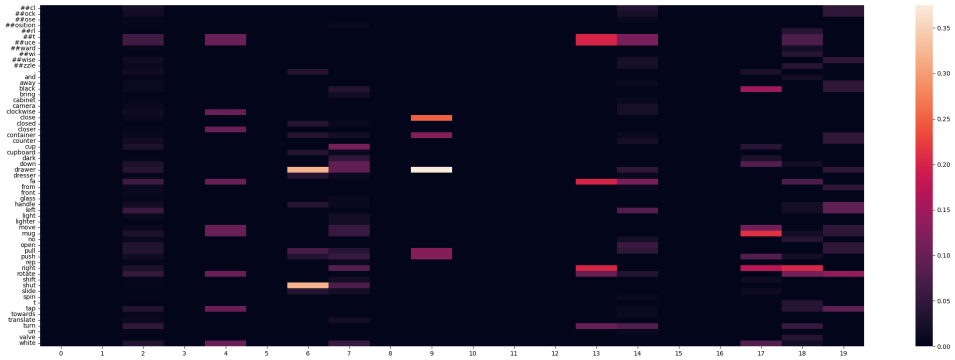

Figure 15: Word Freq. for each skill on LOReL Sawyer (column normalized)

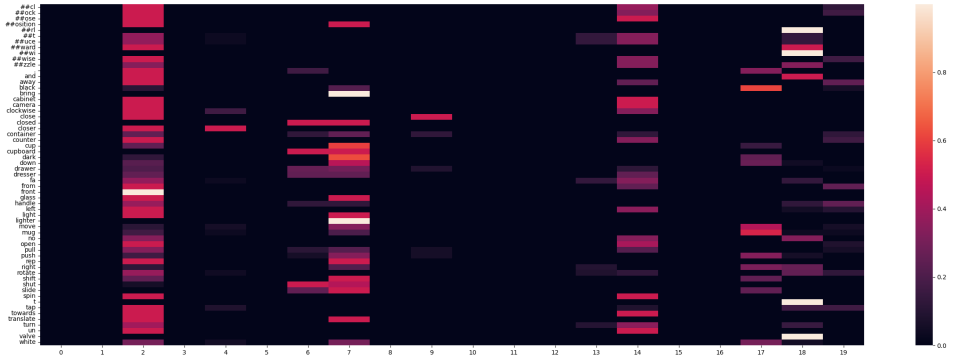

Figure 16: Skill Freq. for each word on LOReL Sawyer (row normalized)

B.5 Behavior with fixed skills

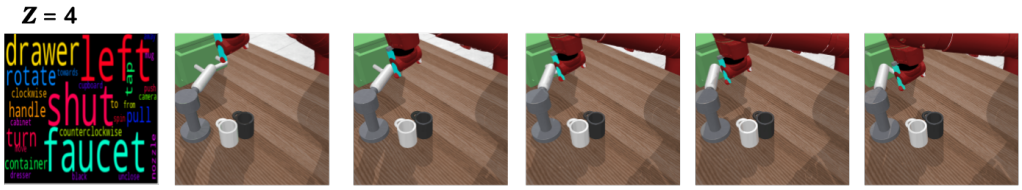

Figure 17: Behavior and language corresponding to skill code 4: “turn faucet left”

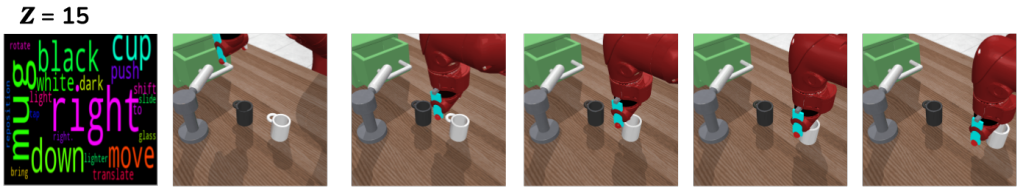

Figure 18: Behavior and language corresponding to skill code 15: “move white mug right”

Figure 19: LOReL Composition task: “close drawer and turn faucet left”

C Datasets

C.1 BabyAI Dataset

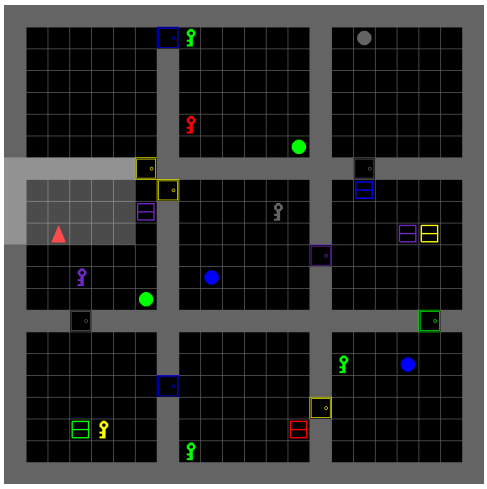

Figure 20: BabyAI BossLevel

The BabyAI dataset [13] contains 19 levels of increasing difficulty where each level is set in a grid world where an agent has a partially observed state of a square of side 7 around it. The agent must learn to perform various tasks of arbitrary difficulty such as moving objects between rooms, opening doors or closing them etc. all with a partially observed state and a language instruction.
Each level comes with 1 million language conditioned trajectories, and we use a small subset of these for our training. We evaluate our model on the environment provided with each level that generates a new language instruction and grid randomly.
We have provided details about the levels we evaluated on below. More details can be found in the original paper.

C.1.1 GoToSeq

Sequencing of go-to-object commands.
Example command: *“go to a box and go to the purple door, then go to the grey door”*
Demo length: 72.7 ± 52.2

C.1.2 SynthSeq

Example command: *"put a purple key next to the yellow key and put a purple ball next to the red box on your left after you put a blue key behind you next to a grey door"*
Demo length: 81.8 ± 61.3

C.1.3 BossLevel

Example command: *"pick up a key and pick up a purple key, then open a door and pick up the yellow ball"*
Demo length: 84.3 ± 64.5

C.2 LOReL Sawyer Dataset

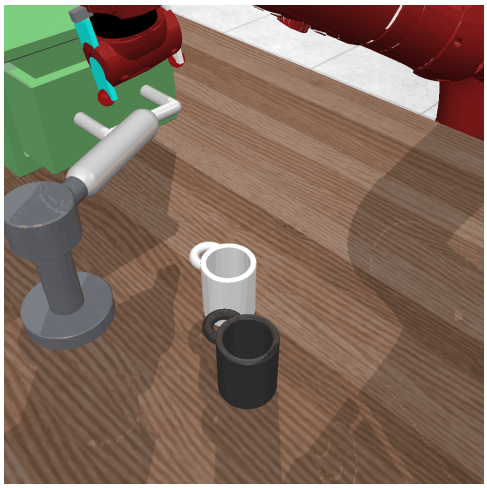

Figure 21: LOReL Sawyer Environment

This dataset [38] consists of *pseudo-expert* trajectories collected from a RL buffer of a a random policy and has been labeled with post-hoc crowd-sourced language instructions. Therefore, the trajectories complete the language instruction provided but may not necessarily be optimal. The Sawyer dataset contains 50k language conditioned trajectories on a simulated environment with a Sawyer robot of demo length 20.
We evaluate on the same set of instructions the original paper does for 1, which can be found in the appendix of the original paper. These consist of the following 6 tasks and rephrasals of these tasks where they change only the noun, only the verb, both noun and verb and rewrite the entire task (human provided). This comes to a total of 77 instructions for all 6 tasks combined. An example is shown below and the full list of instructions can be found in the original paper.

1. Close drawer
2. Open drawer
3. Turn faucet left
4. Turn faucet right
5. Move black mug right
6. Move white mug down

Table 4: LOReL Example rephrasals for the instruction *"close drawer"*

Seen	Unseen Verb	Unseen Noun	Unseen Verb + Noun	Human Provided
close drawer	shut drawer	close container	shut container	push the drawer shut

For the composition instructions, we took these evaluation instructions from the original paper and combined them to form 12 new composition instructions as shown below.

Table 5: LOReL Composition tasks

Instructions
open cabinet and move black mug right
pull the handle and move black mug down
shift white mug right
shift black mug down
shut drawer and turn faucet right
close cabinet and turn faucet left
turn faucet left and shift white mug down
rotate faucet right and close drawer
move white mug down and rotate faucet left
open drawer and turn faucet counterclockwise
slide the drawer closed and then shift white mug down
rotate faucet left and move white mug down
shift white mug down and shift black mug right
turn faucet right and open cabinet
move black mug right and turn faucet clockwise

We included the instructions *"move white mug right"* and *"move black mug down"* as composition tasks here in the hope that we may have skills corresponding to colors like black and white or directions like right and down that can be composed to form these instructions but we did not observe such behaviour.

D Training details

We plan to release our code on acceptance. Here we include all hyper-parameters we used. We implement our models in PyTorch. Our original flat baseline implementation borrows from Decision Transformer codebase which uses GPT2 to learn sequential behavior. However, we decided to start from scratch in order to implement LISA to make our code modular and easily support hierarchy. We use 1 layer Transformer networks for both the skill predictor and the policy network in our experiments for the main paper. We tried using large number of layers but found them to be too computationally expensive without significant performance improvements. In BabyAI and LOReL results we train all models for three seeds.

For BossLevel environment we use 50 skill codes, for other environments we used the settings detailed in the table below:

D.1 LISA

Table 6: **LISA Hyperparameters**

Hyperparameter	BabyAI	LORL
Transformer Layers	1	1
Transformer Embedding Dim	128	128
Transformer Heads	4	4
Skill Code Dim	16	16
Number of Skills	20	20
Dropout	0.1	0.1
Batch Size	128	128
Policy Learning Rate	$1e-4$	$1e-4$
Skill Predictor Learning Rate	$1e-5$	$1e-5$
Language Model Learning Rate	$1e-6$	$1e-6$
VQ Loss Weight	0.25	0.25
Horizon	10	10
VQ EMA Update	0.99	0.99
Optimizer	Adam	Adam

D.2 Baselines

Table 7: **Flat Baseline Hyperparameters**

Hyperparameter	BabyAI	LORL
Transformer Layers	2	2
Transformer Embedding Dim	128	128
Transformer Heads	4	4
Dropout	0.1	0.1
Batch Size	128	128
Policy Learning Rate	$1e-4$	$1e-4$
Language Model Learning Rate	$1e-6$	$1e-6$
Optimizer	Adam	Adam

Hyperparameter choices. The hyperparameters were chosen approximately by us based on our estimate of the number of skills in the dataset and the complexity of the dataset. For example, we use 20 skill codes for LOReL experiments while we use 50 skill codes for the BabyAI BossLevel experiments as mentioned above. These hyperparameter choices are by no means exhaustive nor optimal as our ablation study in section F.1 suggests that our choice of horizon is perhaps sub-optimal (we used H=10 for our experiments in Table 1 but appendix section F.1 suggests that H=5 is better). The ablation study also suggests that the performance is fairly stable for a reasonable range of hyperparameter choices removing the burden from the practitioner.

Baseline Implementations. For the original baseline for BabyAI, we used the code from the original repository. For the LOReL baseline, we used the numbers from the paper for LOReL Images. For LOReL States BC baseline, we implemented it based on the appendix section of the paper. We ran the LOReL planner from the original repository for the composition instructions.

D.3 Ablations

As mentioned in the paper, all our ablations were performed on BabyAI BossLevel with 1k trajectories over a single seed for the sake of time. Unless otherwise specified, we use the following settings. We use a 1-layer, 4-head transformer for both the policy and the skill predictor. We use 50 options and a horizon of 10. We use a batch size of 128 and train for 2500 iterations. We use a learning rate of 1e-6 for the language model and 1e-4 for the other parameters of the model. We use 2500 warm-up steps for the DT policy. Training was done on Titan RTX GPUs.

D.4 MI Calculation

We calculate mutual information between the language instructions (L) and the skill codes (z) by writing $MI(L, z) = H(L) - H(L|z)$. Our procedure for this is very simple and uses $\sim$ 10 lines of code. Specifically, we first calculate $H(L|z)$ by assuming $p(L|z)$ to be gaussian $N(\mu, I)$ with unit variance, centered at the codebook vector z. We can calculate μ by finding the distance between the codebook vectors and the language vectors in the latent embedding space. We can similarly calculate the $H(L)$ by taking the expectation of $H(L|z)$ over all discrete codebook vectors.

```
def MI(option_codes, lang_state_embeds):
    """Calculate entropy of options over each batch
    option_codes: [N, D]
    lang_state_embeds: [B, D]
    """
    with torch.no_grad():
        N, D = option_codes.shape
        lang_state_embeds = lang_state_embeds.reshape(-1, 1, D)

        embed = option_codes.t()
        flatten = rearrange(lang_state_embeds, '... d -> (...) d')

        distance = -(
            flatten.pow(2).sum(1, keepdim=True)
            - 2 * flatten @ embed
            + embed.pow(2).sum(0, keepdim=True)
        )
        cond_probs = torch.softmax(distance / 2, dim=1)

        # get marginal probabilities
        probs = cond_probs.mean(dim=0)
        entropy = (-torch.log2(probs) * probs).sum()

        # calculate conditional entropy with language
        # sum over options, and then take expectation over language
        cond_entropy = (-torch.log2(cond_probs) * cond_probs).sum(1).mean(0)
        return entropy - cond_entropy
```

E Detailed LOReL Sawyer results

We provide details results on the LOReL evaluation instructions below for LISA and the flat baseline in the same format as the original paper. The results are averaged over 10 runs. The time horizon used was 20 steps.

Table 8: **Task-wise success rates (in %) on LOReL Sawyer.**

Task	Flat	LISA
close drawer	10	**100**
open drawer	**60**	20
turn faucet left	0	0
turn faucet right	0	**30**
move black mug right	20	**60**
move white mug down	0	**30**

Table 9: **Rephrasal-wise success rates (in %) on LOReL Sawyer.**

Rephrasal Type	Flat	LISA
seen	15	**40**
unseen noun	13.33	**33.33**
unseen verb	28.33	**30**
unseen noun+verb	6.7	**20**
human	26.98	**27.35**

F More experiments

F.1 Ablation Studies

For the sake of time, all our ablations were performed with a 1-layer, 4-head transformer for the skill predictor and for the policy. All our ablations are on the BabyAI-BossLevel environment with 1k expert trajectories.

Our first experiment varies the horizon of the skills. The table below shows the results on BabyAI BossLevel for 4 different values of the horizon. We see that the method is fairly robust to the different choices of horizon unless we choose a very small horizon. For this case, we notice that a horizon of 5 performs best, but this could vary with different tasks.

Table 10: **Ablation on horizon.** We fixed the number of options to be 50 for these experiments

Horizon	1	5	10	50
Success Rate (in %)	32	**52**	47	47

We also tried varying the number of skills the skill-predictor can choose from and found that this hyperparameter is fairly robust as well unless we choose an extremely high or low value. We suspect using more skills worsens performance because it leads to a harder optimization problem and the options don't clearly correspond to specific language skills.

Table 11: **Ablation on number of options.** We fixed the horizon to be 10 for these experiments

Number of Options	10	20	50	100
Success Rate (in %)	**47**	**47**	**47**	43

F.2 Can we leverage the interpretability of the skills produced by LISA for manual planning?

Since our skills are so distinct and interpretable, its tempting to try and manually plan over the skills based on the language tokens they encode. In the LOReL environment, using the same composition tasks as section above, we observe the word clouds of options and simply plan by running the fixed option code corresponding to a task for a certain horizon and then switch to the next option corresponding to the next task. This means we are using a manual (human) skill predictor as opposed to our trained skill predictor. While this doesn't work as well because skills are a function of both language and trajectory and we can only interpret the language part as humans, it still shows how interpretable our skills are as humans can simply observe the language tokens and plan over them to complete tasks. We show a successful example and a failure case below for the instruction *"close drawer and turn faucet right"* in figure 22. We first observe that the two skills we want to compose are $Z = 14$ and $Z = 2$ as shown by the word clouds. We then run skill 14 for 20 steps and skill 2 for 20 steps. In the failure case, the agent closes the drawer but then pulls it open again when trying to turn the faucet to the right.

F.3 Do the skills learned transfer effectively to similar tasks?

We want to ask the question whether we can use the skills learned on one task as a initialization point for a similar task or even freeze the learned skills for the new task. To this end, we set up experiments where we trained LISA on the BabyAI GoTo task with 1k trajectories and tried to transfer the learned options to the

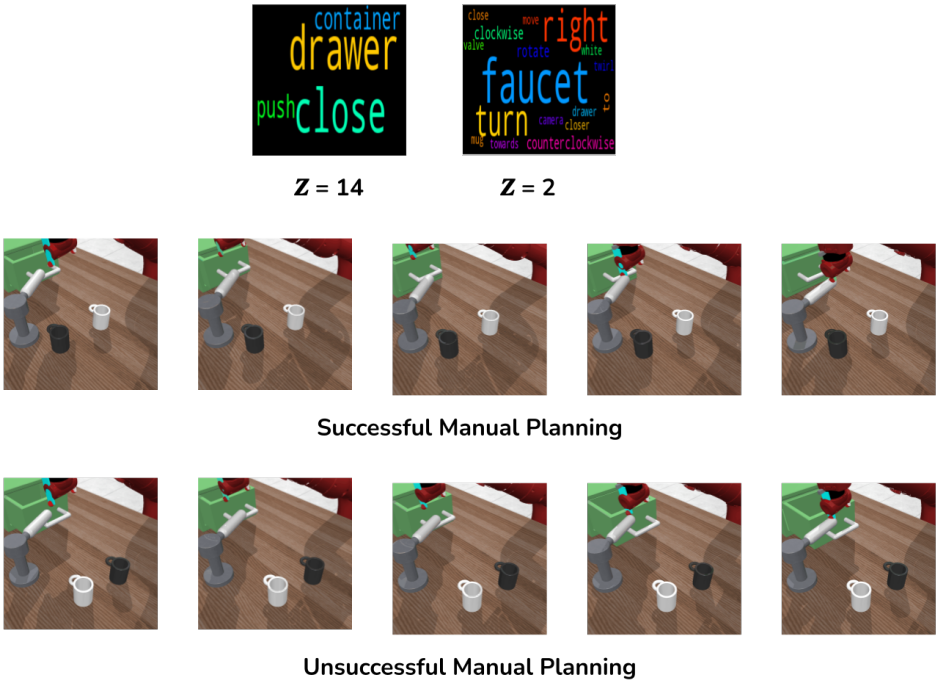

Figure 22: We show a successful manual planning and unsuccessful manual planning example for the instruction "close the drawer and turn the faucet right"

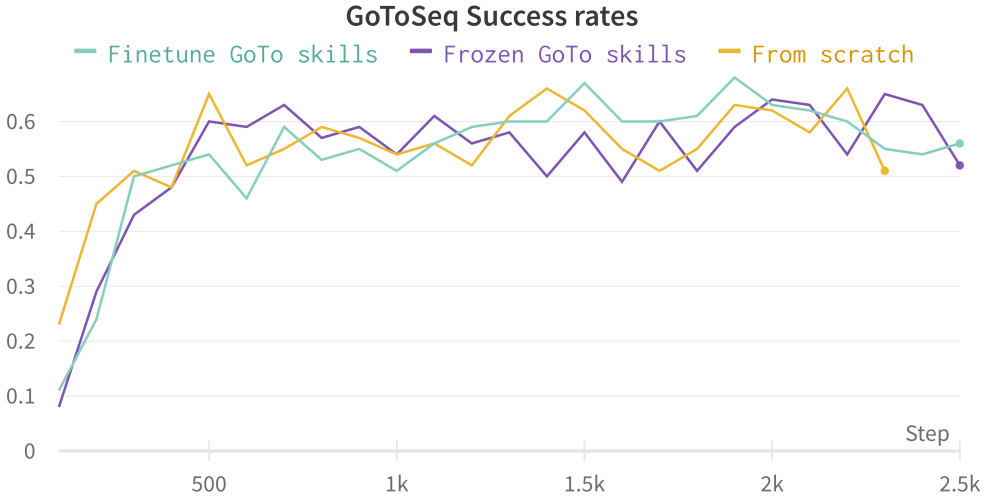

Figure 23: Transferring skills on the BabyAI GoToSeq environment

GoToSeq task with 1k trajectories. Similarly we trained LISA on GoToSeq with 1k trajectories and tried to transfer to BossLevel with 1k trajectories. The results are shown in figures 23 and 24 respectively.

As we can see from the GoToSeq experiment in figure 23, there is no major difference between the three methods. We notice that we can achieve good performance even by holding the learned option codes from GoTo frozen. This is because the skills in GoTo and GoToSeq are very similar except that GoToSeq composes these skills as tasks. We also notice that finetuning doesn't make a big difference – once again probably because the skills are similar for both environments.
In the BossLevel case in 24, however, we do notice that the frozen skills perform slightly worse than the other two methods. This is because the BossLevel contains more skills than those from GoToSeq. We also notice that the performance of finetuning and starting from scratch is nearly the same. This could be because the

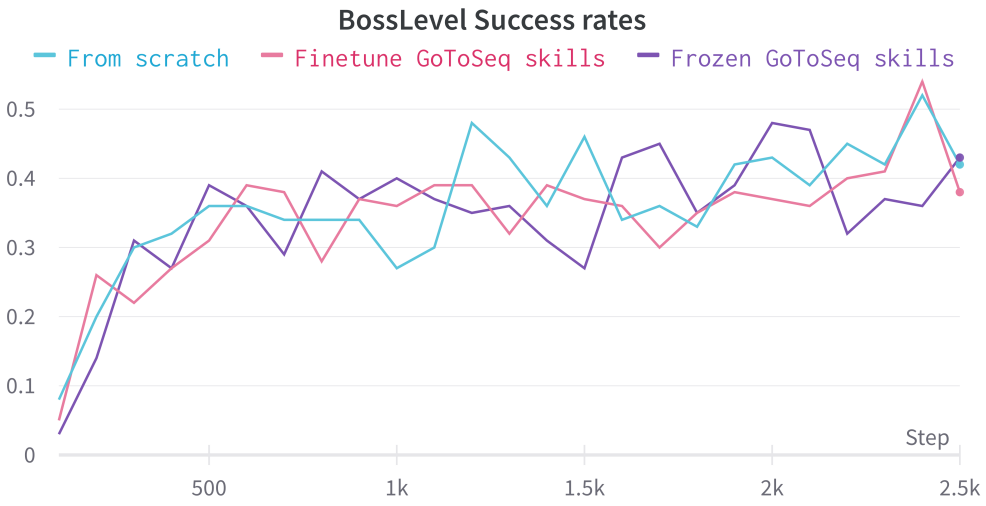

Figure 24: Transferring skills on the BabyAI BossLevel environment

meta-controller needs to adapt to use the new skills in the BossLevel environment anyway and there is no benefit from loading learned options here.

F.4 Are the skill codes in LISA chosen because of environment affordances and not language?

A natural hypothesis for LISA choosing relevant skill codes for each task could be that they are chosen because of the environment affordances and not because of the language instructions. For example, LISA chooses the code corresponding to opening a drawer because the drawer is closed in the environment or because the robot arm is close to it when initialized. We systematically show that this is not the case. We initialize the LOReL environment with the drawer wide open and test all possible combinations of words meaning the same as “drawer" and “close".

- We issue the following language commands: “close drawer”, “close container”, “push drawer” and “push container”. LISA is unable to solve the task “push container” but for the other three tasks, over 20 different environment seeds, LISA generates the skill code 14 (as shown in Figure 4), all 20 times for each instruction.
- We then issue the commands containing the object “drawer" with actions meaning different than “close": “rotate drawer”, “rotate container”, “move container down” and “move drawer down”, “open drawer", “open container", “pull drawer", “pull container". We observe that none of these instructions used skill code 14 even once out of 20 different runs of each instruction.
- Next, we ran the commands containing the object different from “drawer" and actions same as “close": “close mug”, “close faucet”, “push mug” and “push faucet” and none of these instructions use the skill code 14 even once out of 20 different runs of each instruction.

These experiments indicate that the skill code 14 clearly is generated by LISA only when the “close drawer" instruction or a synonym is given in the language instruction. This clearly highlights that the codes are highly correlated to language and are not just being chosen based on the affordances of the environment.

F.5 State-based skill-predictor

We have already spoken in section 3.2 about the fact that our method using two transformers doesn’t necessarily mean its more compute-heavy than the non-hierarchical counterpart. But to test whether we really need two transformers, we perform an ablation study that replaces the skill predictor to be just a state-based selector MLP as opposed to a trajectory-based transformer. Our results show that the performance is only slightly worse when using a state-based skill predictor in this case, but once again this may not be the case in more complex environments. However, we notice that the skills collapsed in this case and the model tends to use fewer skills than normal as shown in figure 25. This is expected because the skill predictor is now predicting skills with much less information.

F.6 Continuous skill codes

We also compare to the non-quantized counterpart where we learn skills from a continuous distribution as opposed to a categorical distribution. We expect this to perform better because the skill predictor has access to a larger number of skill codes to choose from and this is what we observe in table 13. However, this comes

Table 12: **Comparing state-based MLP skill predictor vs trajectory-based transformer skill predictor.** We fixed the number of options to be 50 and horizon as 10 for these experiments

Skill Predictor Architecture	Success Rate
State-based MLP	46%
Trajectory-based Transformer	**47%**

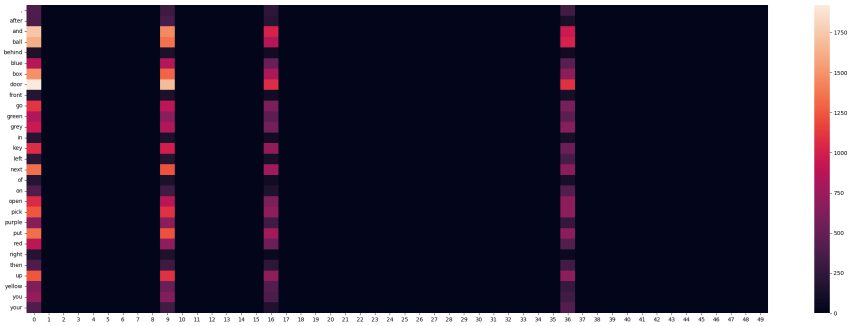

Figure 25: State-based skill predictor heat map shows that the model tends to use fewer options compared to figure 8

at the price of interpretability and its harder to interpret and choose continuous skill codes than discrete codes. We also observe that on the LOReL with states environment, using discrete codes performs better than using continuous codes (table 14). This could be because learning a multi-modal policy with discrete skills is an easier optimization problem than learning one with continuous skills (see the end of section 4.3).

Table 13: **Ablation on Quantization on BabyAI BossLevel.** We fix number of options to 50 and horizon to 10

Skill codes	Success Rate (in %)
Continuous	**51**
Discrete	47

Table 14: **Ablation on Quantization on LOReL with states on seen tasks.** We fix number of options to 20 and horizon to 10

Skill codes	Success Rate (in %)
Continuous	60.0
Discrete	**66.7**