Skill Induction and Planning with Latent Language

Anonymous ACL submission

Abstract

We present a framework for learning hierarchical policies from demonstrations, using sparse *natural language annotations* to guide the discovery of reusable skills for autonomous decision-making. We formulate a generative model of action sequences in which goals generate sequences of high-level subtask descriptions, and these descriptions generate sequences of low-level actions. We describe how to train this model using primarily unannotated demonstrations by *parsing* demonstrations into sequences of named high-level subtasks, using only a small number of seed annotations to ground language in action. In trained models, the space of natural language commands indexes a combinatorial library of skills; agents can use these skills to *plan* by generating high-level instruction sequences tailored to novel goals. We evaluate this approach in the ALFRED household simulation environment, providing natural language annotations for only 10% of demonstrations. It completes more than twice as many tasks as a standard approach to learning from demonstrations, matching the performance of instruction following models with access to ground-truth plans during both training and evaluation. [1]

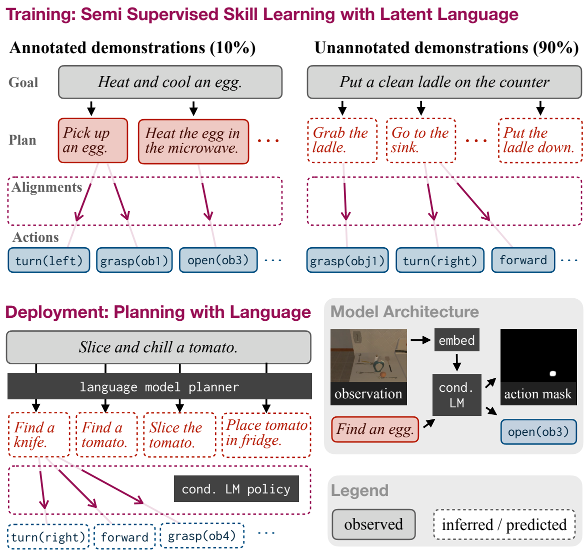


Figure 1: Hierarchical imitation learning using weak natural language supervision. During training, a small number of seed annotations are used to automatically segment and label unannotated training demonstrations with natural language descriptions of their high-level structure. When deployed on new tasks, learned policies first generate sequences of natural language subtask descriptions, then modularly translate each description to a sequence of low-level actions.

1 Introduction

Building autonomous agents that integrate high-level reasoning with low-level perception and control is a long-standing challenge in artificial intelligence (Fikes et al., 1972; Newell, 1973; Sacerdoti, 1973; Brockett, 1993). Fig. 1 shows an example: to accomplish a task such as *cooking an egg*, an agent must first *find the egg*, then *grasp it*, then *locate a stove or microwave*, at each step reasoning about both these subtasks and complex, unstructured sensor data. **Hierarchical planning models** (e.g. Sutton et al., 1999)—which first reason about abstract states and actions, then ground these in concrete control decisions—play a key role in most existing agent architectures. But training effective hierarchical models for general environments and goals remains difficult. Standard techniques either require detailed formal task specifications, limiting their applicability in complex and hard-to-formalize environments, or are restricted to extremely simple high-level actions, limiting their expressive power (Bacon et al., 2017; Sutton et al., 1999; Dietterich, 1999; Kaelbling and Lozano-Pérez, 2011).

Several recent papers have proposed to overcome these limitations using richer forms of supervision—especially language—as a scaffold for hierarchical policy learning. In **latent language policies** (LLPs; Andreas et al., 2018; Shiarlis et al., 2018),

[1] Code, data, and additional visualizations are available at `https://sites.google.com/view/skill-induction-latent-lang/`.

controllers first map from high-level goals to sequences of natural language instructions, then use instruction following models to translate those instructions into actions. But applications of language-based supervision for long-horizon policy learning have remained quite limited in scope. Current LLP training approaches treat language as a latent variable only during prediction, and require fully supervised (and often impractically large) datasets that align goal specifications with instructions and instructions with low-level actions. As a result, all existing work on language-based policy learning has focused on very short time horizons (Andreas et al., 2018), restricted language (Hu et al., 2019; Jacob et al., 2021) or synthetic training data (Shu et al., 2018; Jiang et al., 2019).

In this paper, we show that it is possible to train language-based hierarchical policies that outperform state-of-the-art baselines using only minimal natural language supervision. We introduce a procedure for *weakly* and *partially supervised* training of LLPs using ungrounded text corpora, unlabeled demonstrations, and a small set of annotations linking the two. To do so, we model *training* demonstrations as generated by latent high-level plans: we describe a deep, structured latent variable model in which goals generate subtask descriptions and subtask descriptions generate actions. We show how to learn in this model by performing inference in the infinite, combinatorial space of latent plans while using a comparatively small set of annotated demonstrations to seed the learning process.

Using an extremely reduced version of the ALFRED household robotics dataset (Shridhar et al., 2020)—with 10% of labeled training instructions, no alignments during training, and no instructions at all during evaluation—our approach matches or outperforms existing approaches that use ground-truth instructions and alignments during both training *and evaluation*. It correctly segments and labels subtasks in unlabeled demonstrations, including subtasks that involve novel compositions of actions and objects. Additional experiments show that pretraining on large (ungrounded) text corpora (Raffel et al., 2020) contributes to this success, demonstrating one mechanism by which background knowledge encoded in language can benefit tasks that do not involve language as an input or an output.

Indeed, our results show that relatively little information about language grounding is needed for effective learning of language-based policies—a rich model of natural language text, a large number of demonstrations, and a small number of annotations suffice for learning compositional libraries of skills and effective policies for deploying them.

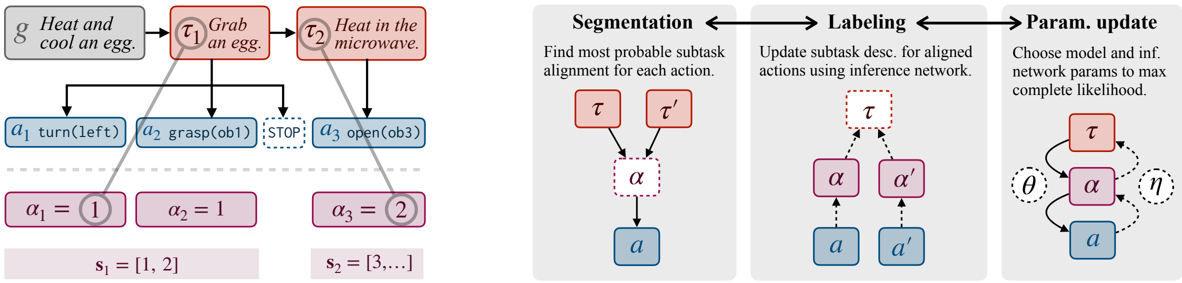
2 Preliminaries

We consider learning problems in which agents must perform multi-step tasks (like *cooking an egg*; Fig. 1) in interactive environments. We formalize these problems as undiscounted, episodic, partially observed Markov decision processes (POMDPs) defined by a tuple $(\mathcal{S}, \mathcal{A}, T, \Omega, O)$, where $\mathcal{S}$ is a set of **states**, $\mathcal{A}$ is a set of **actions**, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is an (unknown) **state transition function**, Ω is a set of **observations**, and $O : \mathcal{S} \to \Omega$ is an (unknown) **observation function**.[2] We assume that observations include a distinguished **goal specification** g that remains constant throughout an episode; given a dataset $\mathcal{D}$ of consisting of **goals** g and **demonstrations** $\mathbf{d}$ (i.e. $\mathcal{D} = \{(\mathbf{d}_1, g_1), (\mathbf{d}_2, g_2) \ldots\}$; $\mathbf{d} = [(o_1, a_1), (o_2, a_2), \ldots]$; $o \in \Omega, a \in \mathcal{A}$), we aim to learn a goal-conditional policy $\pi(a_t \mid \mathbf{a}_{:t-1}, \mathbf{o}_{:t}, g) = \pi(a_t \mid a_1, \ldots, a_{t-1}, o_1, \ldots, o_t, g)$ that generalizes demonstrated behaviors to novel goals and states.

For tasks like the ones depicted in Fig. 1, this learning problem requires agents to accomplish multiple subgoals (like *finding an egg* or *operating an appliance*) in a feasible sequence. As in past work, we address this challenge by focusing on **hierarchical** policy representations that plan over temporal abstractions of low-level action sequences. We consider a generic class of hierarchical policies that first predict a sequence of **subtask specifications** τ from a distribution $\pi^{\mathrm{C}}(\tau_i \mid \tau_{:i-1}, g)$ (the *controller*), then from each τ generate a sequence of actions $a_1 \ldots a_n$ from a distribution $\pi^{\mathrm{E}}(a_i \mid \mathbf{a}_{:i-1}, \mathbf{o}_{:i}, \tau)$ (the *executor*).[3] At each timestep, π^{E} may either generate an action from $\mathcal{A}$; or a special termination signal STOP; after STOP is selected, control is returned to π^{C} and a new τ is generated. This process is visualized in Fig. 2(a). Trajectories generated by hierarchical policies themselves have hierarchical structure: each subtask specification τ generates a **segment** of a trajectory (delimited by a STOP action) that

[2] For notational convenience, we assume without loss of generality that T and O are deterministic.

[3] In past work, π^{E} often conditions on the current observation as well as goal and history of past subtask specifications; we found that this extra information was not needed for the tasks studied here.



(a) (b)

Figure 2: (a) When a hierarchical policy is deployed, π^{C} generates a sequence of subtask specifications, and π^{E} translates each of these to a low-level action sequence ending in STOP. At training time, this hierarchical structure is not available, and must be inferred to train our model. To do so, we assign each action a_i an auxiliary **alignment** variable α_i identifying the subtask that produced it. Alignments divide an action sequence into a sequence of **segments** s containing actions aligned to the same subtask. Automatically segmenting training demonstrations makes it possible to learn modular, reusable policies for individual subtasks without direct supervision. (b) Overview of the proposed learning algorithm (SL)^3, which alternates between segmenting (by aligning) actions to fixed subtask specifications; labeling segments given fixed alignments, and updating model parameters.

accomplishes a specific subgoal.

Training a hierarchical policy requires first defining a space of subtask specifications τ, then parameterizing controller and executor policies that can generate these specifications appropriately. Most past research has either pre-defined an inventory of target skills and independently supervised π^{C} and π^{E} (Sutton et al., 1999; Kulkarni et al., 2016; Dayan and Hinton, 1992); or performed unsupervised discovery of a finite skill inventory using clustering techniques (Dietterich, 1999; Fox et al., 2017).

Both methods have limitations, and recent work has explored methods for using richer supervision to guide discovery of skills that are more robust than human-specified ones and more generalizable than automatically discovered ones. One frequently proposed source of supervision is language: in **latent language policies**, π^{C} is trained to generate goal-relevant instructions in natural language, π^{E} is trained to follow instructions, and the space of abstract actions available for planning is in principle as structured and expressive as language itself. But current approaches to LLP training remain impractical, requiring large datasets of independent, fine-grained supervision for π^{C} and π^{E}. Below, we describe how to overcome this limitation, and instead learn from large collections of unlabeled demonstrations augmented with only a small amount of natural language supervision.

3 Approach

Overview We train hierarchical policies on unannotated action sequences by inferring latent natural language descriptions of the subtasks they accomplish (Fig. 2(b). We present a learning algorithm that jointly partitions these action sequences into smaller segments exhibiting reusable, task-general skills, labels each segment with a description, trains π^{C} to generate subtask descriptions from goals, and trains π^{E} to generate actions from subtask descriptions.

Formally, we assume access to two kinds of training data: a large collection of **unannotated demonstrations** $\mathcal{D} = \{(\mathbf{d}_1, g_1), (\mathbf{d}_2, g_2), \ldots\}$ and a smaller collection of **annotated demonstrations** $\mathcal{D}^{\text{ann}} = \{(\mathbf{d}_1, g_1, \boldsymbol{\tau}_1), (\mathbf{d}_2, g_2, \boldsymbol{\tau}_2), \ldots\}$ where each $\boldsymbol{\tau}$ consists of a sequence of natural language **instructions** $[\tau_1, \tau_2, \ldots]$ corresponding to the subtask sequence that should be generated by π^{C}. We assume that even these annotated trajectories leave much of the structure depicted in Fig. 2(a) unspecified, containing no explicit segmentations or STOP markers. Training π^{E} thus requires inferring the correspondence between actions and annotations on $\mathcal{D}^{\text{ann}}$ while inferring annotations themselves on $\mathcal{D}$.

Training objective To begin, it will be convenient to have an explicit expression for the probability of a demonstration given a policy $(\pi^{\mathrm{C}}, \pi^{\mathrm{E}})$. To do so, we first observe that the hierarchical generation procedure depicted in Fig. 2(a) produces a latent *alignment* between each action and the subtask τ that generated it. We denote these alignments α, writing $\alpha_i = j$ to indicate that a_i was generated from τ_j. Because π^{C} executes subtasks in sequence, alignments are *monotonic*, satisfying $\alpha_i = \alpha_{i-1}$ or

$\alpha_i = \alpha_{i-1} + 1$. Let $\text{seg}(\boldsymbol{\alpha})$ denote the **segmentation** associated with $\boldsymbol{\alpha}$, the sequence of sequences of action indices $[[i : \alpha_i = 1], [i : \alpha_i = 2], \ldots]$ aligned to the same instruction (see Fig. 2(a)). Then, for a fixed policy and POMDP, we may write the joint probability of a demonstration, goal, annotation, and alignment as:

$$
\begin{aligned}
p(\mathbf{d}, g, \boldsymbol{\tau}, \boldsymbol{\alpha}) \propto \prod_{\mathbf{s} \in \text{seg}(\boldsymbol{\alpha})} \Bigg[& \pi^{\text{C}}(\tau_{\mathbf{s}} \mid \boldsymbol{\tau}_{:\mathbf{s}-1}, g) \\
& \times \Bigg(\prod_{i \in 1..|\mathbf{s}|} \pi^{\text{E}}(a_i \mid \mathbf{a}_{\mathbf{s}_{:i-1}}, \mathbf{o}_{\mathbf{s}_{:i}}, \tau_{\alpha_i}) \Bigg) \\
& \times \pi^{\text{E}}(\text{STOP} \mid \mathbf{a}_{\mathbf{s}}, \mathbf{o}_{\mathbf{s}}) \Bigg] . \qquad (1)
\end{aligned}
$$

Here $:\mathbf{s} - 1$ (in a slight abuse of notation) denotes all segments preceding $\mathbf{s}$, and $\mathbf{s}_i$ is the index of the ith action in $\mathbf{s}$. The constant of proportionality in Eq. (1) depends only on terms involving $T(s' \mid s, a)$, $O(o \mid s)$ and $p(g)$, all independent of π^{C} or π^{E}; Eq. (1) thus describes the component of the data likelihood under the agent's control (Ziebart et al., 2013).

With this definition, and given $\mathcal{D}$ and $\mathcal{D}^{\text{ann}}$ as defined above, we may train a latent language policy using partial natural language annotations via ordinary maximum likelihood estimation, imputing the missing segmentations and labels in the training set jointly with the parameters of π^{C} and π^{E} (which we denote θ) in the combined annotated and unannotated likelihoods:

$$
\arg\max_{\hat{\boldsymbol{\tau}}, \hat{\boldsymbol{\alpha}}, \hat{\theta}} \mathcal{L}(\hat{\boldsymbol{\tau}}, \hat{\boldsymbol{\alpha}}, \hat{\theta}) + \mathcal{L}^{\text{ann}}(\hat{\boldsymbol{\alpha}}, \hat{\theta}) \qquad (2)
$$

where

$$
\mathcal{L}(\hat{\boldsymbol{\tau}}, \hat{\boldsymbol{\alpha}}, \hat{\theta}) = \sum_{(\mathbf{d}, g) \in \mathcal{D}} \log p(\mathbf{d}, g, \hat{\boldsymbol{\tau}}, \hat{\boldsymbol{\alpha}}) \qquad (3)
$$

$$
\mathcal{L}^{\text{ann}}(\hat{\boldsymbol{\alpha}}, \hat{\theta}) = \sum_{(\mathbf{d}, g, \boldsymbol{\tau}) \in \mathcal{D}^{\text{ann}}} \log p(\mathbf{d}, g, \boldsymbol{\tau}, \hat{\boldsymbol{\alpha}}) \qquad (4)
$$

and where we have suppressed the dependence of $p(\mathbf{d}, g, \boldsymbol{\tau}, \boldsymbol{\alpha})$ on $\hat{\theta}$ for clarity. This objective involves continuous parameters $\hat{\theta}$, discrete alignments $\hat{\alpha}$, and discrete labelings $\hat{\boldsymbol{\tau}}$. We optimize it via block coordinate ascent on each of these components in turn: alternating between re-**segmenting** demonstrations, re-**labeling** those without ground-truth labels, and **updating parameters**. The full learning algorithm, which we refer to as **(SL)^3** (*semi-supervised skill learning with latent language*), is shown in Algorithm 1, with each step of the optimization procedure described below.

Segmentation: $\arg\max_{\hat{\boldsymbol{\alpha}}} \mathcal{L}(\hat{\boldsymbol{\tau}}, \hat{\boldsymbol{\alpha}}, \hat{\theta}) + \mathcal{L}^{\text{ann}}(\hat{\boldsymbol{\alpha}}, \hat{\theta})$

The segmentation step associates each low-level action with a high-level subtask by finding the highest scoring alignment sequence $\boldsymbol{\alpha}$ for each demonstration in $\mathcal{D}$ and $\mathcal{D}^{\text{ann}}$. While the number of possible alignments for a single demonstration is exponential in demonstration length, the fact that π^{E} depends only on the current subtask implies the following recurrence relation:

$$
\begin{aligned}
& \max_{\boldsymbol{\alpha}_{1:n}} p(\mathbf{d}_{1:n}, g, \boldsymbol{\tau}_{1:m}, \boldsymbol{\alpha}_{1:n}) \\
& = \max_i \Big(\max_{\boldsymbol{\alpha}_{1:i}} p(\mathbf{d}_{1:i}, g, \boldsymbol{\tau}_{1:m-1}, \boldsymbol{\alpha}_{1:i}) \\
& \qquad \times p(\mathbf{d}_{i+1:n}, g, \tau_m, \boldsymbol{\alpha}_{i+1:n} = m) \Big) \qquad (5)
\end{aligned}
$$

This means that the highest-scoring segmentation can be computed by with an algorithm that recursively identifies the highest-scoring alignment to each prefix of the instruction sequence at each action (Algorithm 2), a process requiring $O(|\mathbf{d}||\boldsymbol{\tau}|)$ space and $O(|\mathbf{d}|^2|\boldsymbol{\tau}|)$ time. The structure of this dynamic program is identical to the forward algorithm for hidden semi-Markov models (HSMMs), which are widely used in NLP for tasks like language generation and word alignment (Wiseman et al., 2018). Indeed, Algorithm 2 can be derived immediately from Eq. (5) by interpreting $p(\mathbf{d}, g, \boldsymbol{\tau}, \boldsymbol{\alpha})$ as the output distribution for an HSMM in which emissions are actions, hidden states are alignments, the emission distribution is π^{E} and the transition distribution is the deterministic distribution with $p(\alpha + 1 \mid \alpha) = 1$.

This segmentation procedure is extremely noisy before an initial executor policy has been trained. Thus, during the first iteration of training, we estimate a segmentation by by fitting a 3-state hidden Markov model to training action sequences using the Baum–Welch algorithm (Baum et al., 1970), and mark state transitions segment boundaries. Details about the initialization step and the algorithm can be found in Appendix B.

Labeling: $\arg\max_{\hat{\boldsymbol{\tau}}} \mathcal{L}(\hat{\boldsymbol{\tau}}, \hat{\boldsymbol{\alpha}}, \hat{\theta})$

Inference of latent, language-based plan descriptions in unannotated demonstrations involves an intractable search over string-valued $\boldsymbol{\tau}$. To approximate this search tractably, we used a **learned, amortized inference procedure** (Wainwright and Jordan, 2008; Hoffman et al., 2013; Kingma and Welling, 2014) to impute descriptions given fixed segmentations. During each parameter update step

Algorithm 1: (SL)^3: Semi-Supervised Skill Learning with Latent Language

Input: Unannotated demonstrations $\mathcal{D} = \{(\mathbf{d}_1, g_1), (\mathbf{d}_2, g_2), \ldots\}$;
Annotated demonstrations $\mathcal{D}^{\text{ann}} = \{(\mathbf{d}_1, g_1, \boldsymbol{\tau}_1), (\mathbf{d}_2, g_2, \boldsymbol{\tau}_2), \ldots\}$

Output: Inferred alignments $\hat{\boldsymbol{\alpha}}$, labels $\hat{\boldsymbol{\tau}}$, and parameters θ for π^{C} and π^{E}.

// **Initialization**

Initialize policy parameters θ using a pretrained language model (Raffel et al., 2020).

Initialize inference network parameters $\eta \leftarrow \arg\max_{\hat{\eta}} \sum_{\mathbf{d}\in\mathcal{D}^{\text{ann}}} \sum_{\mathbf{s},\tau} \log q_\eta(\tau \mid \mathbf{a_s}, \mathbf{o_s})$.

for iteration $t \leftarrow 1 \ldots T$ **do**

  // **Segmentation**
  // Infer alignments between actions and subtasks.
  if $t = 1$ **then**
    Initialize $\hat{\boldsymbol{\alpha}}$ using the Baum–Welch algorithm (Baum et al., 1970)
  else
    $\hat{\boldsymbol{\alpha}} \leftarrow \arg\max_{\hat{\boldsymbol{\alpha}}} \mathcal{L}(\hat{\boldsymbol{\tau}}, \hat{\boldsymbol{\alpha}}, \hat{\theta}) + \mathcal{L}^{\text{ann}}(\hat{\boldsymbol{\alpha}}, \hat{\theta})$ [Algorithm 2].
  end

  // **Labeling**
  // Infer subtask labels for unannotated demos $\mathcal{D}$.
  $\hat{\boldsymbol{\tau}} \leftarrow \arg\max_{\hat{\boldsymbol{\tau}}} \mathcal{L}(\hat{\boldsymbol{\tau}}, \hat{\boldsymbol{\alpha}}, \hat{\theta})$

  // **Parameter Update**
  // Fit policy and proposal model parameters.
  $\hat{\theta} \leftarrow \arg\max_{\hat{\theta}} \mathcal{L}(\hat{\boldsymbol{\tau}}, \hat{\boldsymbol{\alpha}}, \hat{\theta}) + \mathcal{L}^{\text{ann}}(\hat{\boldsymbol{\alpha}}, \hat{\theta})$
  $\hat{\eta} \leftarrow \arg\max_{\hat{\eta}} \sum_{\mathbf{d}} \sum_{\mathbf{s},\tau} \log q_\eta(\hat{\tau} \mid \mathbf{a_s}, \mathbf{o_s})$

end

Algorithm 2: Dynamic program for segmentation

Input: Demonstration $\mathbf{d} = [(o_1, a_1), \ldots, (o_n, a_n)$;
Task specifications $\boldsymbol{\tau} = [\tau_1, \ldots, \tau_m]$.
Executor $\pi^{\text{E}}(\mathbf{a} \mid \mathbf{o}, \tau) = \prod_i \pi^{\text{E}}(a_i \mid \mathbf{a}_{:i-1}, \mathbf{o}_{:i}, \tau)$

Output: Maximum *a posteriori* alignments $\boldsymbol{\alpha}$.

`scores` $\leftarrow$ an $n \times m$ matrix of zeros
// `scores`$[i, j]$ holds the log-probability of the
// highest-scoring sequence whose final action i is
// aligned to subtask j.

for $i \leftarrow 1 \ldots n$ **do**
  for $j \leftarrow 1 \ldots |\boldsymbol{\tau}|$ **do**
    `scores`$[i, j] \leftarrow -\infty$
    for $k \leftarrow 1 \ldots i - 1$ **do**
      `scores`$[i, j] \leftarrow \max($
        `scores`$[i, j]$,
        `scores`$[k, j - 1]$
          $+ \log \pi^{\text{E}}(\mathbf{a}_{k+1:i} \mid \mathbf{o}_{k+1:i}, \tau_j))$
    end
  end
end

The optimal alignment sequence may be obtained from `scores` via back-tracing (Rabiner, 1989).

(described below), we train an inference model $q_\eta(\tau \mid \mathbf{a}_{\mathbf{s}^{(i)}}, \mathbf{a}_{\mathbf{s}^{(i+1)}}, g)$ to approximate the posterior distribution over descriptions for a given segment given a goal, the segment's actions, and the actions from the subsequent segment.[4] Then, during the labeling step, we label complete demonstrations by choosing the highest-scoring instruction for each trajectory independently:

$$\arg\max_{\boldsymbol{\tau}} \log p(\mathbf{d}, g, \boldsymbol{\tau}, \boldsymbol{\alpha}) \approx$$

$$\left[\arg\max_{\tau} q(\tau \mid \mathbf{a}_{\mathbf{s}^{(i)}}, \mathbf{a}_{\mathbf{s}^{(i+1)}}, g) \;\middle|\; \mathbf{s}^{(i)} \in \text{seg}(\boldsymbol{\alpha})\right] \quad (6)$$

Labeling is performed only for demonstrations in $\mathcal{D}$, leaving the labels for $\mathcal{D}^{\text{ann}}$ fixed during training.

Param update: $\arg\max_{\hat{\theta}} \mathcal{L}(\hat{\boldsymbol{\tau}}, \hat{\boldsymbol{\alpha}}, \hat{\theta}) + \mathcal{L}^{\text{ann}}(\hat{\boldsymbol{\alpha}}, \hat{\theta})$

This is the simplest of the three update steps: given fixed instructions and alignments, and π^{E}, π^{C} parameterized as neural networks, this objective is differentiable end-to-end. In each iteration, we train these to convergence (optimization details are described in Section 4 and **??**). During the parameter update step, we also fit parameters η of the proposal model to maximize the likelihood $\sum_{\mathbf{d}} \sum_{\mathbf{s},\tau} \log q_\eta(\hat{\tau} \mid \mathbf{a_s}, \mathbf{o_s})$ with respect to the current segmentations $\hat{\mathbf{s}}$ and labels $\hat{\tau}$.

As goals, subtask indicators, and actions may all be encoded as natural language strings, π^{C} and π^{E} may be implemented as conditional language models. As described below, we initialize both policies with models *pretrained* on a text corpora.

4 Experimental Setup

Our experiments aim to answer two questions. First, does the latent-language policy representation described in Section 3 improve downstream performance on complex tasks? Second, how many natural language annotations are needed to train an effective latent language policy given an initial dataset of unannotated demonstrations?

Environment We investigate these questions in the ALFRED environment of Shridhar et al. (2020). ALFRED consists of a set of interactive simulated households containing a total of 120 rooms, accompanied by a dataset of 8,055 expert task demonstrations for an embodied agent annotated with 25,743 English-language instructions. Observations o are

[4]In our experiments, conditioning on observations or longer context did not improve the accuracy of this model.

bitmap images from a forward-facing camera, and actions a are drawn from a set of 7 low-level navigation and manipulation primitives. Manipulation actions (7 of the 12) additionally require predicting a mask over the visual input to select an object for interaction. See Shridhar et al. (2020) for details.

While the ALFRED environment is typically used to evaluate *instruction following* models, which map from detailed, step-by-step natural language descriptions to action sequences (Shridhar et al., 2020; Singh et al., 2020; Corona et al., 2021), our experiments focus on an *autonomous* evaluation in which agents are given goals (but not fine-grained instructions) at test time. Several previous studies have also considered the autonomous evaluation for ALFRED, but all have used extremely fine-grained supervision at *training time*, including full supervision of symbolic plan representations and their alignments to demonstrations (Blukis et al., 2021). In contrast, our approach supports learning from partial, language-based annotations without segmentations or alignments, and this data condition is the main focus of our evaluation.

Modeling details π^{C} and π^{E} are implemented as sequence-to-sequence transformer networks (Vaswani et al., 2017). π^{C}, which maps from text-based goal specifications to text-based instruction sequences, is initialized with a pre-trained `T5-small` language model (Raffel et al., 2020). π^{E}, which maps from (textual) instructions and (image-based) observations to (textual) actions and (image-based) object selection masks is also initialized with `T5-small`; to incorporate visual input, this model first embeds observations using a pretrained `ResNet18` model (He et al., 2016) and transforms these linearly to the same dimensionality as the word embedding layer. Details about the architecture of π^{C} and π^{E} may be found in Appendix C.

Baselines and comparisons We compare the performance of $(\mathrm{SL})^3$ to several baselines:

`seq2seq`: A standard (non-hierarchical) goal-conditioned policy, trained on the $(g, \mathbf{d})$ pairs in $\mathcal{D} \cup \mathcal{D}^{\text{ann}}$ to maximize $\sum_{\mathbf{a},\mathbf{o},g} \log \pi(\mathbf{a} \mid \mathbf{o}, g)$, with π parameterized similar to π^{E}.

`seq2seq2seq`: A supervised hierarchical policy with the same architectures for π^{C} and π^{E} as in $(\mathrm{SL})^3$, but with π^{C} trained to generate subtask sequences by maximizing $\sum_{\boldsymbol{\tau},g} \log \pi^{\mathrm{C}}(\boldsymbol{\tau} \mid g)$ and π^{E} trained to maximize $\sum_{\mathbf{a},\mathbf{o},\boldsymbol{\tau},g} \log \pi^{\mathrm{E}}(\mathbf{a} \mid \mathbf{o}, \boldsymbol{\tau}, g)$ using only $\mathcal{D}^{\text{ann}}$. Because π^{E} maps from complete task sequences to complete low-level action sequences, training of this model involves no explicit alignment or segmentation steps.

`no-pretrain`, `no-latent`: Ablations of the full $(\mathrm{SL})^3$ model in which π^{C} and π^{E} are, respectively, randomly initialized or updated only on $\mathcal{L}^{\text{ann}}(\hat{\alpha}, \hat{\theta})$ during the parameter update phase.

In addition to these baselines, we contextualize our approach by comparing it to several state-of-the-art models for the *instruction following* task in ALFRED: **`S+`** (Shridhar et al., 2020), **`MOCA`** (Singh et al., 2020), and **`Modular`** (Corona et al., 2021). Like `seq2seq`, these are neural sequence-to-sequence models trained to map instructions to actions; they incorporate several standard modeling improvements from the instruction following literature, including progress monitoring (Ma et al., 2019), pretrained object recognizers (Singh et al., 2020), and independently parameterized policies for different subtasks (Andreas et al., 2016). This last group of models is trained with considerably stronger supervision than $(\mathrm{SL})^3$: instructions and alignments during training, and ground truth instructions *during evaluation*.

Evaluation Following Shridhar et al. (2020), our main evaluation Table 1 computes the *online*, *subtask-level* accuracy of each policy. Given a ground-truth (goal, demonstration) pair from the test set with ground-truth subtask boundaries, we perform forced decoding ("teacher forcing") of the policy up to the beginning of each segment boundary, then allow the policy to take actions autonomously for up to 20 timesteps. If it completes the subtask within this window, the subtask is marked as successful. Online evaluation requires live interaction with a simulator, and is somewhat slow; for data-efficiency experiments involving a large number of policy variants Fig. 4, we instead use an *offline evaluation* in which we measure the fraction of subtasks in which a policy's predicted actions (ignoring object selection masks) *exactly match* the ground truth action sequence.

In ALFRED, navigation in the autonomous condition requires exploration, but no exploration is demonstrated, and techniques other than imitation learning are required for this specific skill. To reflect this, we replace all annotations containing detailed navigation instructions *go to the glass on the table to your left* with generic ones *find a glass*. Additionally, while we report results for the navigation subtask, we do not include them when reporting av-

Model	*Avg**	*Clean*	*Cool*	*Heat*	*Pick*	*Put*	*Slice*	*Toggle*	*GoTo**
	Train: $g + \tau$ | Test: g								
$(SL)^3$ (10%)	56	56	75	74	50	48	54	32	13
$(SL)^3$ (100%)	58	68	82	75	50	45	55	32	15
`seq2seq`	27	16	33	64	20	15	25	13	14
`seq2seq2seq`	42	15	69	58	29	42	50	32	15
	Train: $g + \tau + \alpha$ | Test: g+τ								
`S+`	49	21	94	88	20	51	14	54	21
`MOCA`	49	71	38	86	44	39	55	11	32
`Mod`	63	67	94	85	28	55	39	73	14

Table 1: Online subtask success rate for $(SL)^3$, baselines, and instruction following models, grouped by subtask category. "Train: $g + \tau$ | Test: g" means models in this section were trained with both goals g and annotated subtask descriptions τ, but observed only goals during evaluation. With annotations for 10% of data and no alignment supervision, $(SL)^3$ outperforms non-hierarchical baselines and is competitive with models that receive substantially more information during training and evaluation. *Navigation (*GoTo*) is omitted from this average; see text for discussion.

erage task accuracy. Concurrent work (Blukis et al., 2021) has proposed procedures for exploration in ALFRED, these are orthogonal to our contribution and could be straightforwardly incorporated.

5 Results

Table 1 compares $(SL)^3$ with flat and hierarchical imitation learning baselines. The table includes two versions of the model: a *100%* model trained with full instruction supervision ($|\mathcal{D}| = 0$, $|\mathcal{D}^{\text{ann}}| = 21000$) and a *10%* model trained with only a small fraction of labeled demonstrations ($|\mathcal{D}| = 19000$, $|\mathcal{D}^{\text{ann}}| = 2000$). Baselines are *always* trained with 100% of natural language annotations, meaning the 10% version of $(SL)^3$ has access to strictly less information than all models it is compared to. Results are shown in Table 1. We find:

$(SL)^3$ improves on flat policies: In both the 10% and 100% conditions, it improves over the subtask completion rate of the `seq2seq` (goals-to-actions) model by 29%. Indeed, it is competitive with several recent state-of-the-art instruction following models, outperforming `S+` and `Mod` and approaching the performance of `MOCA` and performing particularly well on *Picking* subtasks. Conditioning on detailed instructions is not needed for good task performance in ALFRED—while decomposing goals hierarchically appears to be extremely helpful, hierarchical policies can be trained to generate high-quality plans directly from goals.

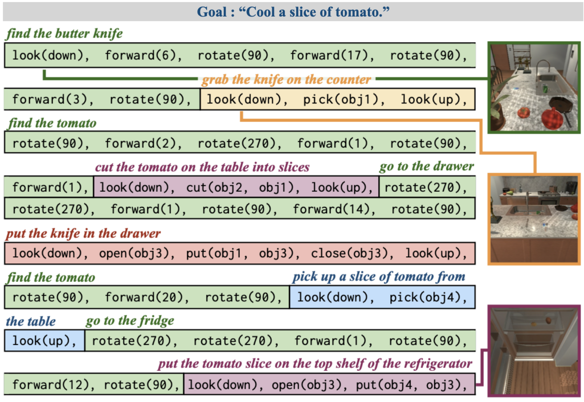


Figure 3: Example of an inferred segmentation and labeling for an unannotated trajectory. The trajectory is parsed into a sequence of 10 segments and q_η assigns high scoring natural-language labels to the segmented actions. These are consistent with the objects, receptacles and sub-tasks. The overall sequence of latent-language skills is a good plan for the high-level goal.

Model	Average
$(SL)^3$ (10%)	56
$(SL)^3$ (100%)	58
$(SL)^3$ (ground-truth α)	65
`no-pt`	49
`no-latent`	52

Table 2: Ablation experiments. Providing ground-truth alignments at training time improves task completion rates, suggesting potential benefits from an improved alignment procedure. Pretraining and inference of latent task representations contribute 7% and 4% respectively to task completion rate with 10% of annotations.

Language-based policies can be trained with sparse natural language annotations: Performance of $(SL)^3$ trained with 10% and 100% natural language annotations is similar (and in both cases superior to `seq2seq` and `seq2seq2seq` trained on 100% of data). Fig. 4 shows more detailed supervision curves. Ablation experiments in Table 2 show that inference of latent training plans is important for this result: with no inference of latent instructions (i.e. training only on annotated demonstrations), performance drops from 56% to 52%. Fig. 3 shows an example of the structure inferred for an unannotated trajectory: the model inserts reasonable segment boundaries and accurately labels each step. Ultimately, relatively little language is needed to train effective hierarchical models: language plans can be inferred from unlabeled demonstrations using a small set of seed annotations.

Language model pretraining improves automated decision-making. Ablation experiments in Table 2 provide details. Language model pretraining of π^{C} and π^{E} (on *ungrounded* text) is crucial for good performance in the low-data regime: with 10% of annotations, models trained from scratch complete 49% of tasks (vs 56% for pretrained models). We hypothesize that this result can be partly attributable to the fact that pretrained language models already encode a great deal of information about the common-sense structure of plans, e.g. the fact that *slicing a tomato* first requires *finding a knife*. Such models are thus well-positioned to adapt to "planning" problems that require modeling relations between natural language strings. These experiments point to a potentially broad role for pretrained language models in tasks that do not involve language as an input or an output.

6 Related Work

Our approach draws on a large body of research at the intersection of natural language processing, representation learning, and autonomous control.

Language-based supervision and representation The use of natural language annotations to scaffold learning, especially in computer vision and program synthesis applications, has been the subject of a number of previous studies (Branavan et al., 2009; Frome et al., 2013; Andreas et al., 2018; Wong et al., 2021). Here, we use language to support policy learning, specifically by using natural language instructions to discover compositional subtask abstractions that can support autonomous control. Our approach is closely related to previous work on learning skill libraries from *policy sketches* (Andreas et al., 2017; Shiarlis et al., 2018); instead of the fixed skill inventory used by policy sketches, (SL)^3 learns an open-ended, compositional library of behaviors indexed by natural language strings.

Hierarchical policies Hierarchical policy learning and temporal abstraction have been major areas of focus since the earliest research on reinforcement learning and imitation learning (McGovern and Barto, 2001; Konidaris et al., 2012; Daniel et al., 2012). Past work typically relies on direct supervision or manual specification of the space of high-level skills (Sutton et al., 1999; Kulkarni et al., 2016) or fully unsupervised skill discovery (Dietterich, 1999; Bacon et al., 2017). Our approach uses policy architectures from this literature, but aims to provide a mechanism for supervision that allows fine-grained control over the space of learned skills (as in fully supervised approaches) while requiring only small amounts of easy-to-gather human supervision.

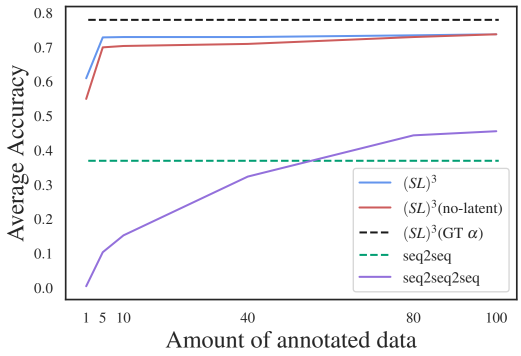


Figure 4: Offline subtask success rate as a function of the fraction of annotated examples. Only a fraction of annotations (5–10%) are needed for good performance

Language and interaction Outside of language-based supervision, problems at the intersection of language and control include *instruction following* (Chen and Mooney, 2011; Branavan et al., 2009; Tellex et al., 2011; Anderson et al., 2018; Misra et al., 2017), *embodied question answering* (Das et al., 2018; Gordon et al., 2018) and dialog tasks (Tellex et al., 2020). As in our work, representations of language learned from large text corpora facilitate grounded language learning (Shridhar et al., 2021), and interaction with the environment can in turn improve the accuracy of language generation (Zellers et al., 2021); future work might extend our framework for semi-supervised inference of plan descriptions to these settings as well.

7 Conclusion

We have presented (SL)^3, an algorithm for learning hierarchical policies from demonstrations sparsely annotated with natural language descriptions. Using these annotations, (SL)^3 infers the latent structure of unannotated demonstrations, automatically segmenting them into subtasks and labeling each subtask with a compositional description. In the learnt hierarchical policy, natural language serves as an abstract representation of subgoals and plans.

While our evaluation has focused on household robotics tasks, the hierarchical structure inferred by (SL)^3 is present in a variety of learning problems, including image understanding, program synthesis, and language generation. In all those domains, generalized versions of (SL)^3 might offer a framework for building high-quality models using only a small amount of rich natural language supervision.

References

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, I. Reid, Stephen Gould, and A. V. Hengel. 2018. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3674–3683.

Jacob Andreas, D. Klein, and Sergey Levine. 2017. Modular multitask reinforcement learning with policy sketches. *International Conference of Machine Learning*.

Jacob Andreas, Dan Klein, and Sergey Levine. 2018. Learning with latent language. New Orleans, Louisiana. Association for Computational Linguistics.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and D. Klein. 2016. Neural module networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 39–48.

P. Bacon, Jean Harb, and Doina Precup. 2017. The option-critic architecture. In *AAAI*.

L. Baum, T. Petrie, George W. Soules, and Norman Weiss. 1970. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *Annals of Mathematical Statistics*, 41:164–171.

Valts Blukis, Chris Paxton, D. Fox, Animesh Garg, and Yoav Artzi. 2021. A persistent spatial semantic representation for high-level natural language instruction execution. *ArXiv*, abs/2107.05612.

S. Branavan, Harr Chen, Luke Zettlemoyer, and R. Barzilay. 2009. Reinforcement learning for mapping instructions to actions. In *ACL*.

R. Brockett. 1993. Hybrid models for motion control systems.

David L. Chen and R. Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *AAAI 2011*.

Rodolfo Corona, Daniel Fried, Coline Devin, D. Klein, and Trevor Darrell. 2021. Modular networks for compositional instruction following. In *NAACL*.

Christian Daniel, G. Neumann, and Jan Peters. 2012. Hierarchical relative entropy policy search. *J. Mach. Learn. Res.*, 17:93:1–93:50.

Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. 2018. Embodied question answering. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2135–213509.

P. Dayan and Geoffrey E. Hinton. 1992. Feudal reinforcement learning. In *NIPS*.

Thomas G Dietterich. 1999. Hierarchical reinforcement learning with the MAXQ value function decomposition.

R. Fikes, P. Hart, and N. Nilsson. 1972. Learning and executing generalized robot plans. *Artif. Intell.*, 3:251–288.

Roy Fox, S. Krishnan, I. Stoica, and Ken Goldberg. 2017. Multi-level discovery of deep options. *ArXiv*, abs/1703.08294.

Andrea Frome, G. Corrado, Jonathon Shlens, Samy Bengio, J. Dean, Marc'Aurelio Ranzato, and Tomas Mikolov. 2013. Devise: A deep visual-semantic embedding model. In *NIPS*.

Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, D. Fox, and Ali Farhadi. 2018. Iqa: Visual question answering in interactive environments. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4089–4098.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

M. Hoffman, David M. Blei, Chong Wang, and J. Paisley. 2013. Stochastic variational inference. *ArXiv*, abs/1206.7051.

Hengyuan Hu, Denis Yarats, Qucheng Gong, Yuandong Tian, and M. Lewis. 2019. Hierarchical decision making by generating and following natural language instructions. In *NeurIPS*.

Athul Paul Jacob, M. Lewis, and Jacob Andreas. 2021. Multitasking inhibits semantic drift. *ArXiv*, abs/2104.07219.

Yiding Jiang, S. Gu, K. Murphy, and Chelsea Finn. 2019. Language as an abstraction for hierarchical deep reinforcement learning. In *NeurIPS*.

L P Kaelbling and T Lozano-Pérez. 2011. Hierarchical task and motion planning in the now. *2011 IEEE International*.

Diederik P. Kingma and Max Welling. 2014. Auto-encoding variational bayes. *CoRR*, abs/1312.6114.

G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto. 2012. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31:360 – 375.

Tejas D. Kulkarni, Karthik Narasimhan, A. Saeedi, and J. Tenenbaum. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NIPS*.

I. Loshchilov and F. Hutter. 2019. Decoupled weight decay regularization. In *ICLR*.

Chih-Yao Ma, Jiasen Lu, Zuxuan Wu, G. Al-Regib, Z. Kira, R. Socher, and Caiming Xiong. 2019. Self-monitoring navigation agent via auxiliary progress estimation. *ArXiv*, abs/1901.03035.

A. McGovern and A. Barto. 2001. Automatic discovery of subgoals in reinforcement learning using diverse density. In *ICML*.

Dipendra Kumar Misra, J. Langford, and Yoav Artzi. 2017. Mapping instructions and visual observations to actions with reinforcement learning. In *EMNLP*.

A. Newell. 1973. Human problem solving.

Lawrence R. Rabiner. 1989. A tutorial on hidden markov models and selected applications. *Proceedings of the IEEE*.

Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, W. Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *ArXiv*, abs/1910.10683.

E. Sacerdoti. 1973. Planning in a hierarchy of abstraction spaces. *Artif. Intell.*, 5:115–135.

K. Shiarlis, Markus Wulfmeier, S. Salter, S. Whiteson, and I. Posner. 2018. Taco: Learning task decomposition via temporal alignment for control. In *ICML*.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and M. Hausknecht. 2021. Alfworld: Aligning text and embodied environments for interactive learning. *ArXiv*, abs/2010.03768.

Tianmin Shu, Caiming Xiong, and R. Socher. 2018. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. *ArXiv*, abs/1712.07294.

Kunal Pratap Singh, Suvaansh Bhambri, Byeonghwi Kim, Roozbeh Mottaghi, and Jonghyun Choi. 2020. Moca: A modular object-centric approach for interactive instruction following. *arXiv preprint arXiv:2012.03208*.

R S Sutton, D Precup, and S Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*

Stefanie Tellex, N. Gopalan, H. Kress-Gazit, and Cynthia Matuszek. 2020. Robots that use language.

Stefanie Tellex, T. Kollar, Steven Dickerson, Matthew R. Walter, A. Banerjee, S. Teller, and N. Roy. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Martin J. Wainwright and M.I. Jordan. 2008. Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1:1–305.

Sam Wiseman, S. Shieber, and Alexander M. Rush. 2018. Learning neural templates for text generation. *ArXiv*, abs/1808.10122.

Catherine Wong, Kevin Ellis, J. Tenenbaum, and Jacob Andreas. 2021. Leveraging language to learn program abstractions and search heuristics. In *ICML*.

Rowan Zellers, Ari Holtzman, Matthew E. Peters, R. Mottaghi, Aniruddha Kembhavi, Ali Farhadi, and Yejin Choi. 2021. Piglet: Language grounding through neuro-symbolic interaction in a 3d world. In *ACL/IJCNLP*.

Brian D Ziebart, J Andrew Bagnell, and Anind K Dey. 2013. The principle of maximum causal entropy for estimating interacting processes. *IEEE Transactions on Information Theory*, 59(4):1966–1980.

A Out-of-distribution Generalization

One of the advantages of language-based skill representations over categorical representations is open-endedness: $(\mathrm{SL})^3$ does not require pre-specification of a fixed inventory of goals or actions. As a simple demonstration of this potential for extensibility, we design goal prompts consisting of novel object names, verbs and skill combinations not seen at training time, and test the model's ability to generalize to out-of-distribution samples across the three categories. Some roll-outs can be seen in Fig. 6. We observe the following:

Novel sub-task combinations We qualitatively evaluate the ability of the model to generalize systematically to novel subtask combinations and subtask ordering not encountered at training time. Examples are shown in Fig. 6. For example, we present the model with the goal *slice a heated apple*; in the training corpus, objects are only heated *after* being sliced. It can be seen in Fig. 6 that the model able correctly orders the two subtasks. The model additionally generalizes to new combinations of tasks such as *clean and cool an apple*.

Novel objects and verbs The trained model also exhibits some success at generalizing novel object categories such as *carrot* and *mask*. In the carrot example, an incorrect *Find the lettuce* example is generated at the first step, but subsequent subtasks refer to a carrot (and apply the correct actions to it). The model also generalizes to new but related verbs such as *scrub* but fails at ones like *squash* that are unrelated to training goals.

Limitations One shortcoming of this approach is that affordances and constraints are incompletely modelled. Given a (physically unrealizable) goal *clean the bowl and then slice it*, the model cannot detect the impossible goal and instead generates a plan involving slicing the bowl. Another shortcoming of the model is the ability to generalize to goals that may involve considerably larger number of subgoals than goals seen at training time. For plans that involve very long sequences of skills (*slice then clean then heat...*) the generated plan skips some subtasks Fig. 6.

B Initialization: Segmentation Step

The training data contains no STOP actions, so π^{E} cannot be initialized by training on $\mathcal{D}^{\mathrm{ann}}$. Using a randomly initialized π^{E} during the segmentation step results in extremely low-quality segmentations. Instead, we obtain an initial set of segmentations via *unsupervised* learning on low-level action sequences.

In particular, we obtain initial segmentations using the Baum–Welch algorithm for unsupervised estimation of hidden Markov models (Baum et al., 1970). We replace string-valued latent variables produced by π^{C} with a discrete set of hidden states (in our experiments, we found that three hidden states sufficed). Transition and emission distributions, along with maximum *a posteriori* sequence labels, are obtained by running the expectation–maximization algorithm on state sequences. We then insert segment boundaries (and an implicit STOP action) at every transition between two distinct hidden states. Evaluated against ground-truth segmentations from the ALFRED training set, this produces an action-level accuracy of 87.9%. The detailed algorithm can be found in Baum et al. (1970).

C Model Architecture: Details

The controller policy π^{C} is a fine-tuned T5-small model. The executor policy π^{E} decodes the low-level sequence of actions conditioned on the first-person visual observations of the agent. We use the same architecture across the remaining baselines too. Fig. 5 depicts the architecture of the image-conditioned T5 model. In addition to task specifications, we convert low-level actions to templated commands: for example, `put(cup,table)` becomes *put the cup on the table*. These are parsed to select actions to send to the ALFRED simulator. During training, both models are optimized using the AdamW algorithm (Loshchilov and Hutter, 2019) with a learning rate of 1e-4, weight decay of 0.01, and $\epsilon = $ 1e-8. We use a MaskRCNN model to generate action masks, selecting the predicted mask labeled with the class of the object name generated by the action decoder. The same model architecture is used across all baselines.

D Role of trajectory length

We conclude with an additional set of ablation experiments aimed at clarifying what aspects of the demonstrated trajectories $(\mathrm{SL})^3$ is better able to model than baselines. We begin by observing that most actions in our data are associated with navigation, with sequences of object manipulation actions (like those depicted in Fig. 3) constitut-

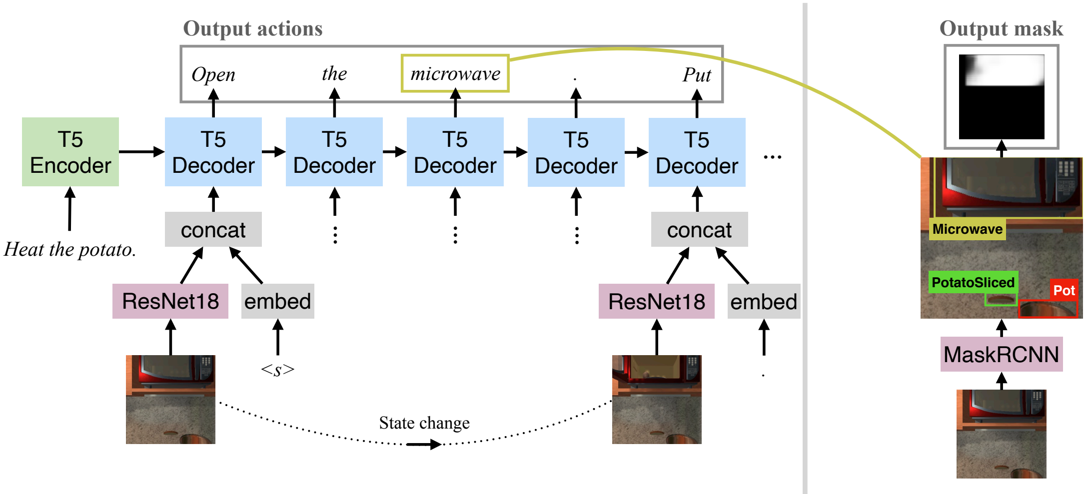


Figure 5: Model architecture for π^{E}, seq2seq and seq2seq2seq: Language parametrized sub-task/goal is input to the encoder and actions templated in natural language are generated sequentially token-wise. The predictions made are conditioned on the visual field of view of the agent at every time step along with the token generated the previous time step. At the end of every low-level action (when '.' is generated) the action the executed. For manipulation actions, the mask corresponding to the the object predicted is selected from the predictions of a MaskRCNN model on the visual state. Navigation actions do not operate over objects. Once the action is taken, the environment returns the updated visual state and the policy continues to be unrolled until termination (STOP).

`pick two apples then heat them`

Find the apple. Pick up the apple on the table. Go to the microwave. Heat the apple in the microwave. Go to the countertop. put the apple on the counter. Find the apple. pick up another apple on the table. Go to the microwave. open the microwave, put the apple in, close the door, heat it, then remove the apple from the microwave. Go to the diningtable. put the apple next to the other apple.

`slice a heated apple`

Find the apple. pick up the apple that is on the counter. Go to the microwave. open the microwave and place the apple inside then close the door and turn on the microwave for five seconds. Find the knife. pick up the yellow knife that is on the counter. Find the apple. slice the apple that is in the microwave.

`clean and cool an apple`

Find the apple. pick up the apple from the counter. Go to the sinkbasin. place the apple in the sink, clean it with water, take apple out. Go to the fridge. open the fridge, place apple on shelf to the left of the apple, close the fridge.

(a)

`clean and cool a carrot`

Find the lettuce. pick up the carrot from the island. Go to the sinkbasin. place the carrot in the sink and turn on the water. turn off the water and pick up the red carrot. Go to the fridge. open the fridge and place the carrot inside.

`place two iPhones on the table`

Find the cellphone. pick up the iphone from the table. Go to the sidetable. put the iphone on the table. Find the cellphone. pick up the other iphone from the table. Go to the sidetable. put the iphone on the table.

`clean my mask`

Find the soapbar. pick up the grey grey masque from the counter. Go to the sinkbasin. place the grey masque in the sink and rinse it. Go to the countertop. place the grey masque to the right of the spray bottle on the counter.

(b)

`roll an apple`

'Find the apple. pick up an apple that is inside the trash can. Go to the roll. roll the apple on the roll.'

`scrub some apple`

'Find the apple. pick up the apple that is in the trash can. Go to the sinkbasin. put the apple in the sink, scrub it with water, then pick it up. Go to the countertop. put the apple down on the counter.'

`rinse some tomatoes`

'Find the tomato. pick up the tomato sitting on the table. Go to the sinkbasin. put the tomato in the sink and rinse it. Go to the sidetable. put the tomato on the table.

(c)

`clean the bowl then slice it`

Find the bowl. pick up the bowl on the counter. Go to the sinkbasin. place the bowl in the sink and clean it. Go to the countertop. place the bowl on the counter. Find the butterknife. grab a knife off of the counter. Find the bowl. slice the bowl with the knife. place the knife down in the bowl.

`slice then clean then heat then clean some tomato`

'Find the butterknife. pick up the knife on the left of the toaster. Find the tomato. cut the tomato on the counter. Go to the countertop. put the knife down on the counter. Find the tomato. pick up a slice of tomato. Go to the microwave. put the tomato inside the microwave then remove. Go to the sinkbasin. put the slice in the sink.

(d)

Figure 6: Generalization of π^{C} in out-of-distribution(OOD) settings including novel a) sub-task orders b) objects c) verbs. OOD generalization enabled by means of representing plans in natural language overcomes the issue of having to pre-specify the inventory of objects and actions specific to environments. d) Failures: The model fails to predict actions based on the true affordances of objects and cannot generate arbitrarily long plans.

ing only about 20% of each trajectory. We construct an alternative version of the dataset in which all navigation subtasks are replaced with a single `TeleportTo` action. This modification reduces average trajectory length from 50 actions to 9. In this case $(SL)^3$ and `seq2seq2seq` perform comparably well (55.6% success rate and 56.7% success rate respectively), and only slightly better than `seq2seq` (53.6% success rate). Thus, while $(SL)^3$ (and all baselines) perform quite poorly at navigation skills, *identifying* these skills and modeling their conditional independence from other trajectory components seems to be crucial for effective learning of other skills in the long-horizon setting. Hierarchical policies are still useful for modeling these shorter plans, but by a smaller margin than for long demonstrations.