SPRINT: Scalable Semantic Policy Pre-Training via Language Instruction Relabeling

Anonymous Author(s)
Affiliation
Address
email

Abstract: We propose SPRINT, a scalable offline policy pre-training approach based on natural language instructions. SPRINT pre-trains an agent's policy to execute a diverse set of semantically meaningful skills that it can leverage to learn new tasks faster. Prior work on offline pre-training required tedious manual definition of pre-training tasks or learned semantically meaningless skills via random goal-reaching. Instead, our approach leverages natural language instruction labels on offline agent experience, collected at scale (e.g., via crowd-sourcing), to define a rich set of tasks with minimal human effort. Furthermore, by using natural language to define tasks, SPRINT can use pre-trained large language models to automatically expand the initial task set. As a result, we can learn an extensive collection of new skills during pre-training by relabeling and aggregating task instructions, even across multiple training trajectories. Experiments using a realistic household simulator show that SPRINT enables learning new long-horizon household tasks substantially faster than previous pre-training approaches.

1 Introduction

When humans learn a new task, *e.g.*, how to cook a new dish, we rely on a large repertoire of previously learned *skills*, like "*chopping vegetables*" or "*boiling pasta*", that make learning more efficient. Improving learning efficiency is crucial for practical deployment of artificial agents; thus, many works in reinforcement learning (RL) aim to equip agents with a similar set of skills. To autonomously acquire such skills, recent works optimize for diverse agent behaviors [1, 2, 3], imitate short action sequences [4, 5], or reach randomly sampled goal states [6] from pre-collected experience. However, such objectives may result in the agent learning skills that are not semantically plausible in practice, *e.g.*, "*placing a knife in the microwave*". To focus pre-training on *plausible* skills, one could instead manually curate a set of pre-training tasks for the policy, but this requires tedious reward function design which does not scale. Yet, defining a large set of pre-training tasks is crucial: only a policy with a wide range of skills can accelerate learning on *many* downstream tasks. How can we define a *large* set of meaningful pre-training tasks in a scalable manner?

In this paper, we propose to leverage *natural language instructions* to define a large number of semantically meaningful tasks for policy pre-training. Using natural language has two important benefits: (1) language is a natural and expressive interface for humans to specify tasks (in contrast to, *e.g.*, numerical reward functions) as it is the primary way to communicate tasks in our everyday lives. Thus, even non-experts can define tasks easily via language instructions. (2) By specifying pre-training tasks via natural language, we can leverage the knowledge captured in large language models to automatically generate more tasks through instruction relabeling.

To combine both benefits we introduce SPRINT (**S**calable **P**re-training via **R**elabeling Language **IN**s**T**ructions), a scalable pre-training approach that equips policies with a repertoire of semantically meaningful skills (see Figure 2 for an illustration). SPRINT has three core components: (1) language-conditioned offline RL, (2) LLM-based skill aggregation and (3) cross-trajectory skill chaining. SPRINT assumes access to an offline dataset of state-action trajectories, each of which performs one or more *skills*. We assume that the data has corresponding natural language instruction labels for the performed skills, such as "*place mug in coffee machine*" or "*press brew button*".

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

Such labels can be crowd-sourced from non-expert human annotators at scale. We use these annotations as task instructions and train a policy with language-conditioned offline RL to solve them. Crucially, SPRINT uses two techniques to expand this initial task set: firstly, we use a pre-trained large language model to relabel the language instructions, thereby creating new tasks. Secondly, we chain behaviors *across* multiple trajectories from the training data; starting with a skill like "*pick up bread*" from one trajectory and ending with "*place bread on table*" from another. This allows the policy to learn semantic skills completely unseen in the training data. SPRINT trains a policy on the *combined* set of task instructions, thereby equipping the agent with a policy that can execute a wide range of semantically meaningful skills. Our experiments demonstrate that this allows for substantially more sample-efficient learning and better zero-shot execution of new downstream tasks like "*prepare breakfast*" than prior pre-training approaches. Due to space limitations, we present our method in the next section and discuss related works in appendix, Section B.

2 SPRINT: Scalable Policy Pre-Training with Language Instructions

In this work, we propose SPRINT (**S**calable **P**re-training via **R**elabeling Language **IN**s**T**ructions), an approach for pre-training a policy to solve a wide range of semantic skills in order to enable efficient finetuning on unseen tasks. SPRINT has three core technical components which we will describe in this section: (1) offline instruction-conditioned RL with crowd-sourced language instructions, (2) language-model-based skill aggregation and (3) cross-trajectory skill chaining. But first, we will detail the data we use for pre-training and give an intuitive overview of our approach.

Following prior work on agent pre-training, we assume access to a large offline dataset $\mathcal{D}$ of agent experience [7, 4, 5, 6, 8, 9]. Such data can be collected at scale, *e.g.*, from prior RL runs or via teleoperation. We further assume that the data contains natural language descriptions of the skills performed by the agent, which can be collected at scale by non-experts by annotating sequences from our offline dataset $\mathcal{D}$ *in hindsight*. Given a randomly sampled sequence τ from $\mathcal{D}$, annotators can label sub-trajectories $\tau_1 = [s_0, a_0, s_1, \dots], \dots$ with free-form language descriptions $z_1, z_2, \dots$ of the tasks performed in the respective sub-trajectories, i.e.:

$$\tau_{\bar{z}} = [\tau_{z_1}, \tau_{z_2}] = [\underbrace{[(s_0^{z_1}, a_0^{z_1}, z_1), ..., (s_T^{z_1}, a_T^{z_1}, z_1)]}_{\text{“Put a mug in the coffee machine”}}, \underbrace{[(s_0^{z_2}, a_0^{z_2}, z_2), ...]}_{\text{“Push the brew button”}}]. \tag{1}$$

This results in a dataset $\mathcal{D}^L$ of diverse trajectories with language descriptions (see Fig. 3, left). Our approach leverages skill descriptions in $\mathcal{D}^L$ as instructions for the policy during pre-training: each description defines a task and the policy is rewarded for successfully executing the instruction. Intuitively, the more diverse the set of language instructions during pre-training, the more semantic skills the policy will learn and the more downstream tasks can it finetune on efficiently. Thus, SPRINT introduces two approaches for scalably increasing the diversity of the pre-training instructions. Firstly, SPRINT leverages pre-trained language models to aggregate the annotated instructions. Secondly, we propose an approach for cross-trajectory chaining that allows us to combine behaviors across multiple trajectories to generate completely unseen instructions. Both approaches expand the set of pre-training tasks (Fig. 3), leading to more effective pre-training.

2.1 Instruction-Conditioned Offline RL

To pre-train our policy with the natural language instruction dataset $\mathcal{D}^L$, we condition our policy $\pi(a|s, z)$ on *language instructions* z from $\mathcal{D}^L$ and provide a sparse reward of 1 to the agent for reaching the end-state s_T of the respective sub-trajectory. We can optimize the policy and critic $Q(s, a, z)$ on this reward function using any offline RL algorithm. We use Implicit Q-Learning [10].

2.2 Language-Model-Based Instruction Aggregation

Large language models (LLMs), trained on massive corpora of internet text data, have been shown to be effective at performing a variety of tasks – from question answering to program synthesis – when prompted with relevant text [11, 12, 13, 14]. Here we use LLMs to *aggregate*, *i.e.*, paraphrase, the existing language annotations in $\mathcal{D}^L$ (see Fig. 3, middle, for an illustration). Given a trajectory that contains multiple sub-trajectories, we can aggregate adjacent sub-trajectories into a longer trajectory and relabel its language annotation with a summary of the individual instructions generated by the

LLM, thereby generating a new *higher-level* pre-training task that encompasses instructions from multiple sub-trajectories.We use a simple summarization prompt to instruct the language model (see Section C for the full prompt used). Like before, the reward for this new aggregated sub-trajectory is 1 at the last transition and 0 otherwise. Taking Eq. 1 as an example, we prompt the language model to summarize the two skills (z_1 : "*Put a mug in the coffee machine*," z_2 : "*Push the brew button*"), resulting in a new semantic annotation $\hat{z}_{1:2}$ describing both skills (*e.g.*, "*Make coffee*"). We then add the new trajectory back to our dataset $\mathcal{D}^L$. Using this technique, we generate new language annotations for all $\binom{N}{2}$ tuples of consecutive sub-trajectories in our dataset. In practice, this allows us to increase the number of pre-training task instructions by 2.5x without additional human effort.

2.3 Cross-Trajectory Chaining

Agents trained with offline RL can combine behaviors from multiple trajectories via value propagation, *i.e.*, "stitch" them [15]. In our case of *instruction-conditioned* offline RL, enabling such stitching behavior requires special care. Due to the different language instruction conditionings for the critic $Q(s, a, z_A)$ and $Q(s, a, z_B)$, values do not naturally propagate from trajectory (B) back to trajectory (A). Instead, we must actively add "chaining examples" to our training dataset [6]. For this, we randomly sample a sub-trajectory $\tau_{z_A} = [s_{0:t}, a_{0:t}, z_A]$ from the training dataset $\mathcal{D}^L$, *e.g.*, part of trajectory (A), and *replace* its language instruction with the instruction z_B *from another trajectory* τ_{z_B}, *e.g.*, "*put mug in coffee machine*." Note that we do not need to sample the full trajectory (A). Instead, s_t can be any state from trajectory (A) from which we try to execute skill (B). Crucially, we *cannot* use the same reward function as before: since the last state s_t of the sampled sub-trajectory, *e.g.*, a mug in the sink, does not solve the instruction z_B, "*put mug in coffee machine*," we cannot set its reward to 1. Which reward should we use instead?

Let's recap that when using temporal-difference (TD) learning [16], Q functions for the sparse 0-1 reward intuitively represent a value that is proportional to the probability of reaching the goal at time T [17, 6]: $Q^\pi(s_t, a_t, z) = \mathbb{E}\left[\sum_{t'=t} \gamma^{t'} R(s_{t'}, a_{t'}, z)\right] \propto P^\pi(s_T = g_z | s_t, a_t)$. Similarly, we want $Q(s, a, z_B)$ to represent the probability of reaching the goal g_{z_B}, the mug being in the coffee machine, from state s. Thus, we need to set the reward of the last state s_t in the sampled sub-trajectory to the probability of reaching the goal g_{z_B} from s_t, *i.e.*, $Q(s_t, a_t, z_B)$:

$$R(s, a, z_B) = \begin{cases} Q(s, a, z_B), & \text{for } s = s_t \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Finally, we can apply the skill aggregation approach from Section 2.2 in the cross-trajectory case. Instead of sampling two sub-trajectories from the same training trajectory as in skill aggregation, we can now sample τ_{z_A} and τ_{z_B} from *different* training trajectories and chain them together. The aggregate instruction $\hat{z}$ implies that the agent first finishes skill (A) and then finishes skill (B), *e.g.*, "*clean the coffee mug and place it in the coffee machine*." Thus, following the logic above, we need to set the reward for the final state in trajectory (B), s_{T_B}, to 1 since it solves the task $\hat{z}$, and the reward in the final state of trajectory (A), s_{T_A}, to the probability of solving skill (B) from there:

$$R(s, a, \hat{z}) = \begin{cases} 1, & \text{for } s = s_{T_B} \\ Q(s, a, z_B), & \text{for } s = s_{T_A} \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

To generate the aggregate instruction $\hat{z}$ we can use the LLM summarization from Section 2.2. Yet, in practice we found the resulting summaries are often meaningless, since randomly paired instructions *from different trajectories* can rarely be summarized meaningfully. Instead, we saw better performance by simply concatenating the natural language instructions from τ_A and τ_B.

3 Experiments

ALFRED Benchmark. We choose the ALFRED benchmark [18] for evaluation (see Figure 4), since it allows us to test our pre-trained agents on *semantically meaningful*, long-horizon tasks in a realistic household setting. The observation space in ALFRED consists of 300×300 egocentric RGB observations and the action space consists of 12 discrete action choices (e.g. *turn left*, *look up*, *pick up object*), along with 82 discrete object types [19]. For more details, see Section E.

Evaluation Tasks. We create a set of 100 unseen long-horizon evaluation instructions (*EVAL_100*), by sampling sequences of 1 to 7 instructions from the ALFRED training dataset and asking human annotators to provide an instruction that summarizes the sequence (see appendix, Section E.2 for examples). Additionally, we create a set of 20 evaluation commands that test the agent's chaining capabilities (*EVAL_CHAIN*): we withhold trajectories and instruction summaries of sequences with 7 to 8 instructions from the ALFRED dataset, making sure that the respective skill chains are *not* present in any single training trajectory. Thus, the agent needs to learn to chain behaviors *across multiple trajectories* to solve these tasks. Finally, to test environment generalization capabilities, we also create a set of 10 tasks in household floor plans not seen in any trajectory in the training dataset (*EVAL_UNSEEN*), consisting in equal parts of 1 to 5-instruction sequences.

Comparisons. We compare SPRINT against the most common prior policy pre-training approaches: behavioral cloning and offline goal-conditioned RL. Specifically, we compare to the following prior works: **Language-conditioned BC (L-BC)** [20, 21]: Behavior cloning (BC) conditioned on the individual ALFRED language instructions. **Actionable Models (AM)** [6]: Goal-conditioned offline RL with randomly sampled goal observations from the ALFRED training set. For more details, see Section D. All results reported are means and standard deviations computed over 3 seeds.

3.1 Zero-Shot and Finetuning Results

Zero-Shot Evaluation. We first test the effectiveness of SPRINT's pre-training by analyzing zero-shot performance across the 100 tasks in the *EVAL_100* evaluation set. We report results in Figure 5 (left). Our approach, SPRINT, achieves 3x higher zero-shot task performance than prior pre-training approaches AM and L-BC. In Figure 5, right, we find that L-BC achieves good performance on short-horizon tasks. However, on long-horizon tasks, SPRINT achieves much higher returns, since it can leverage the language-model to automatically generate longer-horizon pre-training tasks.

To evaluate the pre-training approaches' ability to perform *unseen* skill sequences, we evaluate zero-shot performance on the *EVAL_CHAIN* task set in Figure 5. Unsurprisingly, both L-BC and AM largely fail to generate returns on these long-horizon tasks. In contrast, SPRINT can solve unseen long-horizon tasks with up to 8 subtasks (see Figure 6 for an example trajectory, appendix Section F.5 for more qualitative comparisons). This shows that, with the help of cross-trajectory chaining, SPRINT is able to go beyond the skills in the training data and execute unseen tasks. We present ablation studies examining specific components of our method in appendix, Section F.4.

Finetuning. To test downstream task performance, we finetune the pre-trained agents on the *EVAL_UNSEEN* task set in unseen household floor plans. We report finetuning results in Figure 1, with qualitative examples in appendix, Section F.5. Our approach, SPRINT, is able to achieve 2x higher downstream task return than the best prior work. Specifically, we find L-BC converges quickly to a low return as it is not pre-trained for longer horizon tasks. Meanwhile, AM struggles to transfer the learned skills to the new environment, possibly since the goal states from the new environment are unseen. In contrast, our method's pre-training with language conditioning allows for effective transfer even to unseen environments since the semantics of the tasks transfer well: the language description "*place cup in coffee machine*" transfers to many environments while the goal image for the same task might look very different. Thus, pre-training with semantic language instructions can enable better transfer for learning tasks in new environments than pre-training to reach goal states.

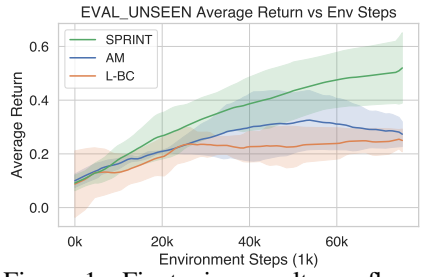


Figure 1: Finetuning results on floor-plans not seen during pre-training.

4 Discussion, Limitations, and Future Work

In this paper, we presented SPRINT, an approach for scalable agent pre-training with offline instruction-conditioned RL. We demonstrated how we can leverage easy-to-collect natural language instructions on offline agent experience for effective pre-training by using pre-trained large language models and cross-trajectory skill chaining to automatically expand the set of pre-training tasks. In our experimental on the ALFRED benchmark [18] we demonstrated that SPRINT allows for substantially more efficient finetuning on downstream tasks in unseen environments than prior works.

References

[1] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function. In *ICLR*, 2019.

[2] A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman. Dynamics-aware unsupervised discovery of skills. *ICLR*, 2020.

[3] R. Mendonca, O. Rybkin, K. Daniilidis, D. Hafner, and D. Pathak. Discovering and achieving goals via world models, 2021.

[4] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet. Learning latent plans from play. In *CoRL*, 2020.

[5] K. Pertsch, Y. Lee, and J. J. Lim. Accelerating reinforcement learning with learned skill priors. In *CoRL*, 2020.

[6] Y. Chebotar, K. Hausman, Y. Lu, T. Xiao, D. Kalashnikov, J. Varley, A. Irpan, B. Eysenbach, R. C. Julian, C. Finn, and S. Levine. Actionable models: Unsupervised offline reinforcement learning of robotic skills. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1518–1528. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/chebotar21a.html.

[7] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *CoRL*, 2019.

[8] F. Ebert, Y. Yang, K. Schmeckpeper, B. Bucher, G. Georgakis, K. Daniilidis, C. Finn, and S. Levine. Bridge data: Boosting generalization of robotic skills with cross-domain datasets. In *RSS*, 2022.

[9] K. Pertsch, Y. Lee, Y. Wu, and J. J. Lim. Demonstration-guided reinforcement learning with learned skills. In *CoRL*, 2021.

[10] I. Kostrikov, A. Nair, and S. Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=68n2s9ZJWF8.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL https://arxiv.org/abs/1810.04805.

[12] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

[13] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer. Opt: Open pre-trained transformer language models, 2022. URL https://arxiv.org/abs/2205.01068.

[14] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel. Palm: Scaling language modeling with pathways, 2022. URL https://arxiv.org/abs/2204.02311.

[15] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[16] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[17] B. Eysenbach, T. Zhang, R. Salakhutdinov, and S. Levine. Contrastive learning as goal-conditioned reinforcement learning, 2022. URL https://arxiv.org/abs/2206.07568.

[18] M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. URL https://arxiv.org/abs/1912.01734.

[19] A. Pashevich, C. Schmid, and C. Sun. Episodic Transformer for Vision-and-Language Navigation. In *ICCV*, 2021.

[20] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. BC-z: Zero-shot task generalization with robotic imitation learning. In *5th Annual Conference on Robot Learning*, 2021. URL https://openreview.net/forum?id=8kbp23tSGYv.

[21] C. Lynch and P. Sermanet. Language conditioned imitation learning over unstructured data. *Robotics: Science and Systems*, 2021. URL https://arxiv.org/abs/2005.07648.

[22] J. Andreas, D. Klein, and S. Levine. Learning with latent language. In *North American Chapter of the Association for Computational Linguistics*, 2017.

[23] K. Narasimhan, T. Kulkarni, and R. Barzilay. Language understanding for text-based games using deep reinforcement learning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics. doi:10.18653/v1/D15-1001. URL https://aclanthology.org/D15-1001.

[24] H. Küttler, N. Nardelli, A. H. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel. The nethack learning environment, 2020. URL https://arxiv.org/abs/2006.13760.

[25] S. R. Branavan, H. Chen, L. Zettlemoyer, and R. Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 82–90, 2009.

[26] J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, 2017.

[27] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022.

[28] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022.

[29] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, N. Brown, T. Jackson, L. Luu, S. Levine, K. Hausman, and B. Ichter. Inner monologue: Embodied reasoning through planning with language models. In *arXiv preprint arXiv:2207.05608*, 2022.

[30] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta. R3m: A universal visual representation for robot manipulation, 2022. URL https://arxiv.org/abs/2203.12601.

[31] L. Fan, G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D.-A. Huang, Y. Zhu, and A. Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *arXiv preprint arXiv: Arxiv-2206.08853*, 2022.

[32] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 2, pages 1398–1403. IEEE, 2002.

[33] E. Theodorou, J. Buchli, and S. Schaal. Reinforcement learning of motor skills in high dimensions: A path integral approach. In *2010 IEEE International Conference on Robotics and Automation*, pages 2397–2403. IEEE, 2010.

[34] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, et al. Deep q-learning from demonstrations. In *Association for the Advancement of Artificial Intelligence*, 2018.

[35] X. B. Peng, A. Kumar, G. Zhang, and S. Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning, 2019. URL https://arxiv.org/abs/1910.00177.

[36] A. Singh, A. Yu, J. Yang, J. Zhang, A. Kumar, and S. Levine. Cog: Connecting new skills to past experience with offline reinforcement learning. *Conference on Robot Learning*, november 2020.

[37] A. Nair, M. Dalal, A. Gupta, and S. Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[38] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. RL ^{2}: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[39] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *ICML*, 2017.

[40] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *ICML*, 2019.

[41] T. Nam, S.-H. Sun, K. Pertsch, S. J. Hwang, and J. J. Lim. Skill-based meta-reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2022.

[42] J. Achiam, H. Edwards, D. Amodei, and P. Abbeel. Variational option discovery algorithms. *arXiv*, 2018.

[43] A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv*, abs/1907.01657, 2019.

[44] A. Ajay, A. Kumar, P. Agrawal, S. Levine, and O. Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. *arXiv preprint arXiv:2010.13611*, 2020.

[45] A. Singh, H. Liu, G. Zhou, A. Yu, N. Rhinehart, and S. Levine. Parrot: Data-driven behavioral priors for reinforcement learning. *ICLR*, 2021.

[46] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[47] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi:10.1109/CVPR.2009.5206848.

[48] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL https://arxiv.org/abs/1908.10084.

[49] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. Courville. FiLM: Visual Reasoning with a General Conditioning Layer. In *Association for the Advancement of Artificial Intelligence*, 2018.

Appendix

Table of Contents

A SPRINT Pseudocode **11**

B Related Work **11**

C Large Language Model Prompt **12**

D Baselines and Implementation **13**
- D.1 Language-conditioned Behavior CLoning 14
- D.2 Actionable Models (AM) 15
- D.3 SPRINT 15

E Dataset and Environment Details **15**
- E.1 Dataset Details 15
- E.2 Evaluation Tasks 17

F Extended Experiments, Results, and Analysis **20**
- F.1 LLM Summary Examples 20
- F.2 Comparing Summaries from Different LLMs 22
- F.3 Step by Step Zero-shot evaluations 22
- F.4 Ablation Studies 23
- F.5 Qualitative Comparison Results 24

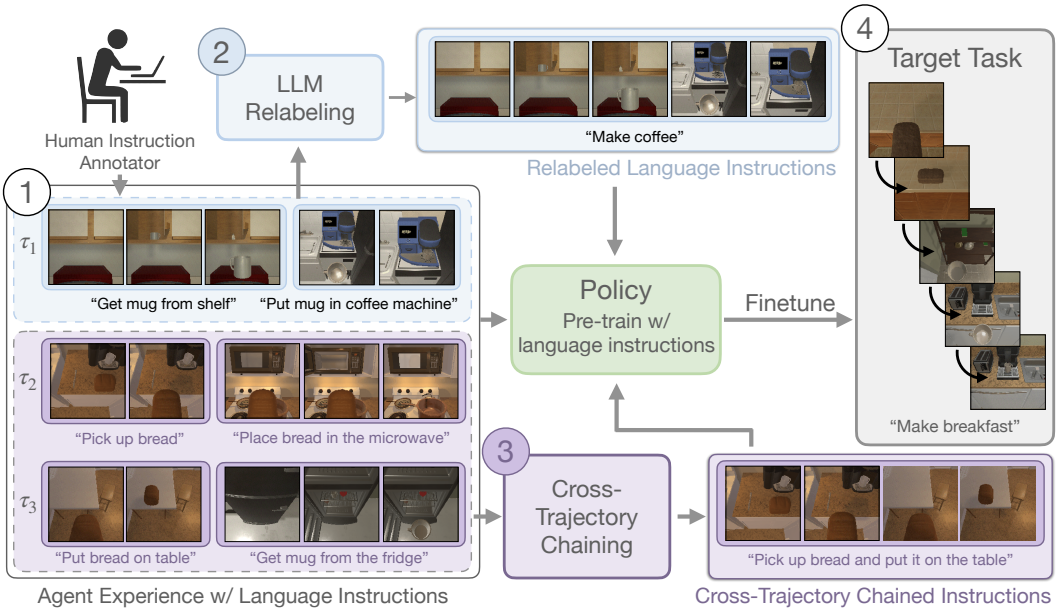


Figure 2: We propose SPRINT, a scalable approach for policy pre-training with semantic skills. We assume access to an offline dataset of agent experience with natural language instruction labels of the performed skills, *e.g.*, provided by human annotators (1). We use the instructions to pre-train a semantic skill policy via instruction-conditioned offline RL. To increase pre-training task diversity, we automatically generate new instructions via (2) language-model-based instruction relabeling, and (3) cross-trajectory skill chaining. We demonstrate that an agent pre-trained with SPRINT can leverage the diverse set of learned semantic skills to finetune efficiently on unseen target tasks (4).

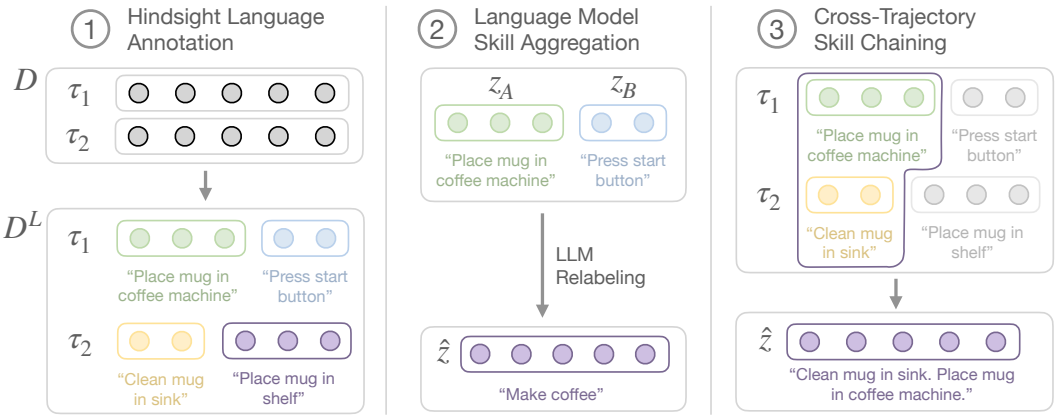


Figure 3: Overview of our approach SPRINT. **Left**: Hindsight annotation of agent experience with language instructions is a scalable approach for defining pre-training tasks. We pre-train via instruction-conditioned offline RL with a sparse goal-reaching reward (Section 2.1). **Middle**: We expand the set of pre-training tasks by aggregating language instructions with an LLM and adding the relabeled trajectories back into the pre-training dataset (Section 2.2). **Right**: We perform cross-trajectory chaining of skills to enable pre-training of skills that are unseen in the offline agent experience (Section 2.3).

Figure 4: ALFRED. Figure drawn from Shridhar et al. [18], with permission.

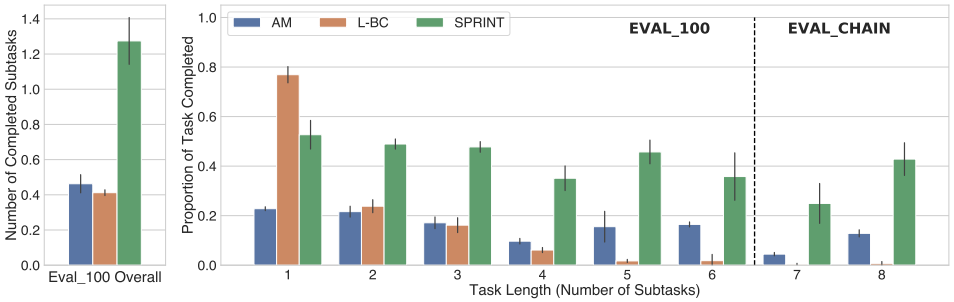


Figure 5: Zero shot evaluation performance on the *EVAL_100* and *EVAL_CHAIN* task sets. The left figure plots the overall number of completed subtasks on *EVAL_100*, with our method in green. The right figure plots the proportion of task completed (number of subtasks successfully executed divided by task length), split by length, in both task sets. See appendix, Table 2 for exact results.

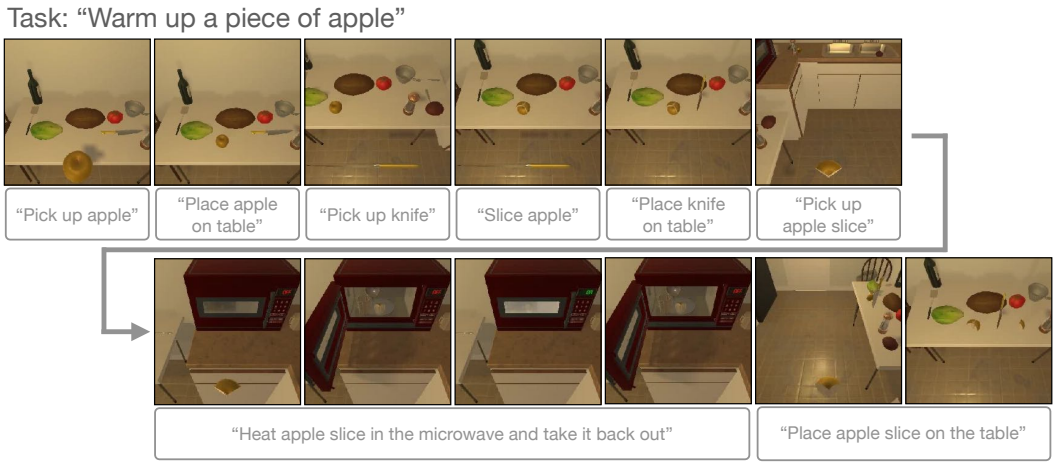


Figure 6: Example task execution of our pre-trained SPRINT agent for the task "*Warm up a piece of apple*". Successful execution of this task requires solving 8 subtasks in sequence and a total of 50 steps. This exact sequence of subtasks was never observed in the training data. SPRINT leverages cross-trajectory stitching and LLM aggregation to learn how to execute unseen tasks.

A SPRINT Pseudocode

Algorithm 1 SPRINT Algorithm

Require: Dataset $\mathcal{D}^L$ w/ language instruction labels, LLM

AggregateSkills($\mathcal{D}^L$, LLM) ▷ Automated LLM skill relabeling (Sec. 2.2)
while not converged **do**
  $\tau_z \leftarrow \mathcal{D}^L$: Sample an annotated skill (sub-)trajectory
  Train offline RL on τ_z

  $\tau_z^{\text{chain}} \leftarrow$CrossChainSkills($\mathcal{D}^L$) ▷ Cross-trajectory skill chaining (Sec. 2.3)
  Train offline RL on τ_z^{chain}

  $\tau_{\text{agg}_1}, \tau_{\text{agg}_2} \leftarrow$ CrossAggregateSkills($\mathcal{D}^L$, LLM) ▷ Cross-traj. aggregation (Sec. 2.3)
  Train offline RL on $\tau_{\text{agg}_1}, \tau_{\text{agg}_2}$
end while

procedure AggregateSkills($\mathcal{D}^L$, LLM)
  for composite trajectory $\tau_{\bar{z}}$ in $\mathcal{D}^L$ **do**
    Separate $\tau_{\bar{z}}$ into language annotated sub-trajectories $[\tau_{z_1}, ..., \tau_{z_N}]$
    for all adjacent sub-trajectories $\left[\tau_{z_i} ... \tau_{z_j}\right]$ **do**
      Assign name from LLM: LLM$(z_i ... z_j) = \hat{z}_{i:j}$
      $\tau_{\hat{z}_{i:j}} \leftarrow$ Concat $\left[\tau_{z_i}, ..., \tau_{z_j}\right]$ and relabel with $\hat{z}_{i:j}$
      $T \leftarrow$ length of $\tau_{\hat{z}_{i:j}}$
      $R(s_T, a_T, \hat{z}_{i:j}) = 1$ ▷ Label last reward with 1 (Eq. **??**).
      $\mathcal{D}^L = \mathcal{D}^L \cup \left\{\tau_{\hat{z}_{i:j}}\right\}$
    end for
  end for
end procedure

procedure CrossChainSkills($\mathcal{D}^L$)
  Sample random (sub)trajectories $\tau_{z_1}, \tau_{z_2} \sim \mathcal{D}^L$
  Sample random endpoint j in τ_{z_1}
  $\tau_{z_2}^{\text{chain}} \leftarrow [(s_0, a_0, 0, z_2), ..., (s_j, a_j, Q^\pi(s_j, a_j, z_2), z_2)]$ ▷ Relabel reward w/ Eq. 2.
  return $\tau_{z_2}^{\text{chain}}$
end procedure

procedure CrossAggregateSkills($\mathcal{D}^L$, LLM)
  Sample random (sub)trajectories $\tau_{z_1}, \tau_{z_2} \sim \mathcal{D}^L$ with lengths T_1, T_2
  Assign new name : $\hat{z} =$ "$\{z_1\}.\{z_2\}$"
  $\tau_{\text{agg}_1} \leftarrow [(s_0, a_0, 0, \hat{z}), ..., (s_{T_1}, a_{T_1}, Q^\pi(s_T, a_T | z_2), \hat{z})]$ ▷ Relabel reward w/ Eq. 3
  $\tau_{\text{agg}_2} \leftarrow [(s_0, a_0, 0, \hat{z}), ..., (s_{T_2}, a_{T_2}, 1, \hat{z})]$ ▷ Relabel reward w/ Eq. 3.
  return $\tau_{\text{agg}_1}, \tau_{\text{agg}_2}$
end procedure

B Related Work

Language in RL. There is a long-standing interest in leveraging natural language during behavior learning, *e.g.*, to structure agent's internal representations [22], to learn to interact with text-based games [23, 24], or guide long-horizon task learning via recipe-like plans [25, 26]. The recent progress in training large, general-purpose language models has enabled approaches that directly generate and execute such plans solely from a high-level task description [27, 28, 29]. Others have shown how language in combination with vision inputs can be used to learn state representations [30] or reward functions [31]. Finally, there is a rich body of work that explores using language as an intuitive agent interface for humans: given a free-form natural language command they aim to train language-conditioned policies that can execute the instruction in the environment [21]. In contrast, our work aims to leverage language for task-agnostic *agent pre-training*. Instead of training a policy to execute human language commands, we aim to equip an agent with a set of semantic skills that improves the efficiency of reinforcement learning on downstream tasks.

Pre-training Policies for RL. Developing policy pre-training approaches for faster downstream learning has been investigated for many years [32, 33, 34]. Recent advances in offline reinforcement learning [15] enabled approaches that can pre-train agents offline and effectively finetune them on online tasks [35, 36, 37, 10]. However, these approaches require the pre-training data to have target-task reward annotations and the resulting pre-trained policies are trained only to solve the target task. In contrast, meta-RL approaches pre-train on a range of tasks and allow fast adaptation to *unseen* downstream tasks [38, 39, 40, 41], yet require tedious manual definition of pre-training tasks by experts, making them less scalable. To avoid manual task design, other works have explored unsupervised pre-training approaches based on behavior diversification [42, 1, 43] or extraction of behavior priors from offline agent experience [5, 44, 45]. Closest to ours is the approach of Chebotar et al. [6], which performs unsupervised pre-training by learning to reach randomly sampled states from offline agent experience. Yet, unsupervised pre-training approaches learn semantically meaningless skills which, as we demonstrate in Section 3, lead to worse transfer to downstream tasks. In contrast, we introduce a *scalable* pre-training approach based on natural language instructions that equips agents with *semantically meaningful* skills and allows for efficient transfer to unseen tasks.

C Large Language Model Prompt

We list the full large language model summarization prompt in Fig. 7. The examples in the prompt are fixed for all summarization queries. These examples are selected from the ALFRED validation dataset (which is not otherwise used in our work) at random: We spell out the primitive skill annotations in the "Task Steps:" part of each prompt example. Then, the "Summary" for each of these is the high-level, human-written annotation for that trajectory from ALFRED. We repeatedly sampled these trajectories until each example mentioned a different object to prevent biasing the LLM towards certain types of objects.

We note that the "Look at the vase under the light" example is important to make the LLM give reasonable summaries for similar tasks in ALFRED where the agent picks something up and turns on a light. This is because most of the human labels for turning on the lamp do not mention the object in the previous step, making it difficult for the LLM to realize that the task has to do with looking at the held object under a lamp.

Instructions: summarize the following ordered steps describing common household tasks.

Task Steps: 1: Pick up the smaller knife on the counter to the left of the stove. 2: Slice the tomato with the smaller knife. 3: Put the knife in the sink. 4: Pick up a slice of tomato from the countertop. 5: Heat up the slice of tomato in the microwave, removing it afterwards.
Summary: Microwave the tomato slice after slicing it with the smaller knife on the counter.

Task Steps: 1: Pick up the vase. 2: Turn on the lamp.
Summary: Look at the vase under the light.

Task Steps: 1: Grab the pencil off of the desk. 2: Put the pencil in the bowl. 3: Grab the container off of the desk. 4: Put the container down at the back of the desk.
Summary: Put a bowl with a pencil in it on the desk.

Task Steps: 1: Pick up the bar of soap from the back of the toilet. 2: Put the bar of soap in to the sink, turn on the faucet to rinse off the soap, pick up the soap out of the sink. 3: Put the soap in the cabinet under the sink and on the left.
Summary: Put a rinsed bar of soap in the cabinet under the sink.

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

Figure 7: The full prompt that we use for summarization. Following the suggestions of Ahn et al. [28] for prompt design, we explicitly number each step. The LLM completion task begins after "Summary:". For brevity, we omit the new line characters between all numbered steps.

D Baselines and Implementation

We implement IQL [10] as the base offline RL algorithm for all goal-conditioned offline RL pre-training baselines and ablations due to its strong offline and finetuning performance on a variety of dense and sparse reward environments. At a high level, IQL trains on in-distribution (s, a, s', r, a') tuples from the dataset rather than sampling a policy for a' to ensure that the Q and value functions represent accurate estimated returns constrained to actions in the dataset. The value function is trained with an expectile regression loss controlled by a hyperparameter τ, where $\tau = 0.5$ results in standard mean squared error loss and $\tau \to 1$ approximates the max operator, resulting in a more optimistic value function that can better "stitch" together trajectories to obtain distant reward in sparse reward settings. The IQL policy is trained to maximize the following objective:

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

which performs advantage-weighted regression [35] with an inverse temperature term β. In practice, the exponential advantage term is limited to a maximum value to avoid numerical overflow issues. We detail shared training and implementation details below, with method-specific information and hyperparameters in the following subsections.

Observation space. The state space of the ALFRED environment consists of 300×300 RGB images. Following the baseline method in ALFRED [18], we preprocess these images by sending them through a frozen ResNet-18 encoder [46] pretrained on ImageNet [47]. This results in a $512 \times 7 \times 7$ feature map that we use as the observation input to all networks. Furthermore, as ALFRED is a partially observable, egocentric navigation environment, we concatenate the last 5 frames as the full observation, resulting in an observation that is of the shape $(512 * 5) \times 7 \times 7$.

Action space. The agent chooses from 12 discrete low-level actions. There are 5 navigation actions: `MoveAhead`, `RotateRight`, `RotateLeft`, `LookUp`, and `LookDown` and 7 interaction actions: `Pickup`, `Put`, `Open`, `Close`, `ToggleOn`, `ToggleOff`, and `Slice`. For interaction actions the agent additionally selects one of 82 object types to interact with, as defined by Pashevich et al. [19]. In total, the action space consists of $5 + 7 * 82 = 579$ discrete action choices. Note that this action space definition is different from the action space in Shridhar et al. [18], which used a pixel-wise mask output to determine the object to interact with. In contrast to Shridhar et al. [18] we aim to train agents with *reinforcement learning* instead of imitation learning and found the discrete action parametrization more amenable to RL training than dense mask outputs. For all methods, due to the large discrete action space, we perform some basic action masking to prevent agents from taking actions that are not possible. For example, we do not allow the agent to `Close` objects that aren't closeable nor can they `ToggleOn` objects that can't be turned on.

Language embedding. We embed language instructions with the `all-mpnet-base-v2` pre-trained sentence embedding language model from the SentenceTransformers python package [48]. This produces a 768-dimensional language embedding which is used as input for language-conditioned policy and critic functions, as detailed below.

Policy and critic networks. We train a discrete policies with two output heads of size 12 and 82 for the action and interaction object outputs respectively. Critic networks are conditioned on both the observation and the discrete action output of the policy. In all policy and critic networks, we process the ResNet feature observation inputs with 4 convolutional layers. In networks with language input, we flatten the output of the convolutional layers and concatenate the observation features with the 768-dim language embedding, before passing the concatenated image-language features through a series of fully connected layers. Additionally, we use FiLM [49] to condition convolutional layers on the language embeddings.

Pre-training hyperparameters. A hyperparameter search was performed first on the language-conditioned BC-baseline to optimize for training accuracy. These hyperparameters were carried over to the IQL implementation, and another search for IQL-specific hyperpameters were performed on a baseline IQL policy conditioned on semantic instructions. With these parameters fixed, we performed one more hyperparameter search specific to Actionable Models but for the final implementation of SPRINT we re-used the same hyperparameters and only selected SPRINT-specific parameters heuristically. Hyperparameters for each method are detailed in a separate table. Shared hyperparameters for all methods (where applicable) are listed below:

Param	Value
Batch Size	1024
Learning Rate	2e-3
Optimizer	AdamW
Dropout Rate	0.2
Weight Decay	0.05
Discount γ	0.98
Q Update Polyak Averaging Coefficient	0.005
Q-Network Discrete Action Embedding Size	48
Q-Network Discrete Object Selection Action Embedding Size	24
Policy and Q Update Period	Once every training iteration
Batch Norm	True
Nonlinearity	ReLU
IQL Advantage Clipping	[0, 100]
IQL Advantage Inverse Temperature β	10
IQL Quantile τ	0.9

Finetuning details and hyperparameters. We perform finetuning experiments for Language-conditioned BC, Actionable Models, and SPRINT. For all models, we finetune the model on only newly collected task data by running online IQL (without any of the chaining or aggregation steps). To implement finetuning for SPRINT and AM, we condition the policy on a language instruction or goal image from the target task respectively and then run IQL with online data collection. For L-BC, we first pre-train a language-conditioned Q-function with IQL on the pre-training dataset and then finetune policy and Q-function with online IQL. Each method is finetuned on every task in the *EVAL_UNSEEN* task set individually; that is, we pre-train once and then finetune ten times, once for every task in the task set. We then average returns over all tasks, then report metrics averaged over all random seeds. For each task, we define a maximum rollout time horizon of 2 timesteps per environment action required by the expert planner.

When not specified, finetuning parameters are identical to pre-training parameters. Finetuning hyperparameters are specified below:

Param	Value
Dropout Rate	0
# Initial Rollouts	50
Training to Env Step Ratio	0.5
ϵ in ϵ-greedy action sampling	0.25: annealed down to 0.05
# Parallel Rollout Samplers	4

D.1 Language-conditioned Behavior CLoning

Our language-conditioned behavior cloning (L-BC) comparison method is inspired by and replicates BC-Zero [20] and LangLfP [21]. BC-Zero performs FiLM-conditioned semantic imitation learning [49] and both BC-Zero and LangLfP have an additional image/video-language alignment objective. In BC-Zero, their video alignment objective aligns language embeddings with videos of humans performing tasks related to those the BC-Zero robot agent trains on. LangLfP's image-language alignment objective allows their policy to accept both image and natural language goals as input due to only having a subset of their data labeled with hindsight language labels. As we don't have human videos of these tasks and our entire dataset is labeled with language labels, we do not add a video or image alignment objective.

Therefore, we implement L-BC by using the same architecture as described above with just a single policy network that trains to maximize the log-likelihood of actions in the dataset. As our entire dataset consists of expert trajectories, this baseline ideally learns optimal actions for the instructions.

Hyperparameters for the L-BC baseline are identical to the shared parameters above, where applicable.

D.2 Actionable Models (AM)

Actionable Models [6] pretrains a goal-conditioned Q function conditioned on randomly sampled image goals and also performs a goal-chaining procedure very similar to our semantic skill chaining procedure. We implement AM by modifying the base IQL policy and critic networks to take in image goals instead of natural language embeddings as goals. These goals are provided in the same way as the observations, i.e., as a concatenated stack of 5 frames (the last 5 frames in the trajectory) processed by a frozen ResNet-18. Therefore, goals are the same shape as observations: $(512 * 5) \times 7 \times 7$.

To allow for fair comparison between our approach and AM, we implement AM with the same powerful offline RL algorithm, IQL [10], used in our method. IQL ensures that the policy does not choose out-of-distribution actions by using advantage-weighted regression on in-distribution actions for policy extraction. With this, we found the conservative auxiliary loss AM adds to push down Q-values for out-of-distribution actions to be unnecessary and even hurtful to its overall performance, so we omit this additional loss term.

Finally, after consulting the authors of AM, we tried varying maximum trajectory lengths when sampling random goals. We found that allowing random goals to be sampled from anywhere within a trajectory resulted in the best zero-shot evaluation performance for AM, so our numbers are reported with this implementation detail.

D.3 SPRINT

The implementation details of SPRINT follow from the discussion about implementing IQL at the top of this section. The key differences are in (1) language model skill aggregation and (2) cross-trajectory skill chaining, detailed below.

LLM Skill Aggregation. We perform LLM skill aggregation fully offline by iterating through every ALFRED trajectory and aggregating sequences of adjacent primitive skill sub-trajectories. Assuming a trajectory with N primitive skills, we select all $\binom{N}{2}$ pairs of start and end skills and aggregate all instructions from start to end with the LLM. With 73k original language-annotated trajectories, this procedure allows us to generate an additional 110k aggregated trajectories. We then add these trajectories to the original dataset and train on the entire set.

Cross-trajectory skill chaining. We perform cross-trajectory skill chaining in-batch. Instead of sampling a second trajectory to perform chaining on, we simply permute the batch indicies to generate a set of randomly sampled second trajectories. Then, we perform a second loss function update, in addition to the original update on the sampled trajectories, with equal loss weighting, to apply the skill-chaining update.

SPRINT-specific hyperparameters follow:

Param	Value
Large Language Model for Relabeling	OPT-13B [13]
LLM Token Filtering Top-p[1]	0.9
LLM Token Sampling Temperature	0.8

[1] At each token generation step, only the highest probability tokens with total probability mass that add up to the top-p are considered.

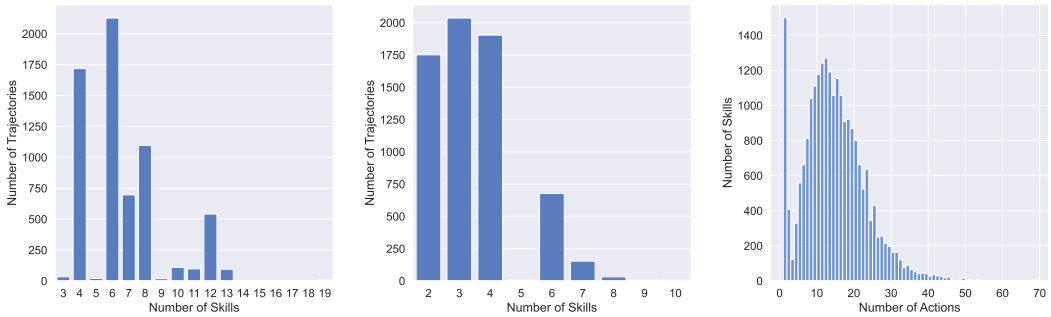
E Dataset and Environment Details

E.1 Dataset Details

For training and evaluation we leverage the ALFRED benchmark and dataset [18]. The ALFRED training dataset contains ∼6.6k trajectories collected by an optimal planner following a set of 7 high-level tasks with randomly sampled objects (*e.g.*, pick up an object and heat it). Each trajectory has at least three crowd-sourced sets of language instruction annotations. Each trajectory consists of

a sequence of 3-19 individually annotated skills (see Figure 8, left). This results in a total of 141k language-annotated skill trajectories.

However, nearly half of the language instructions in the ALFRED dataset are navigation skill instructions like "turn left, then look up and walk to the counter on the right". To get a more balanced skill annotation dataset, we merge all navigation skills with the skill that immediately follows them, using only the annotation of the next skill. After this processing step, the resulting dataset contains 73k language-annotated primitive skill trajectories. After we merge the navigation skills, the average number of skills in each trajectory is 3.5 skills per trajectory (Figure 8, middle), and the average number of actions in each skill is 14.3 (Figure 8, right).



(a) Skills per trajectory in the original ALFRED dataset.

(b) Skills per trajectory in the merged dataset.

(c) Actions per skill in the merged dataset.

Figure 8: **Left**: distribution of the number of skills in each trajectory in the original ALFRED dataset. **Middle**: distribution of skills per trajectory in the "merged" dataset with merged navigation skills. **Right**: distribution of number of actions per skill in the "merged" dataset.

E.2 Evaluation Tasks

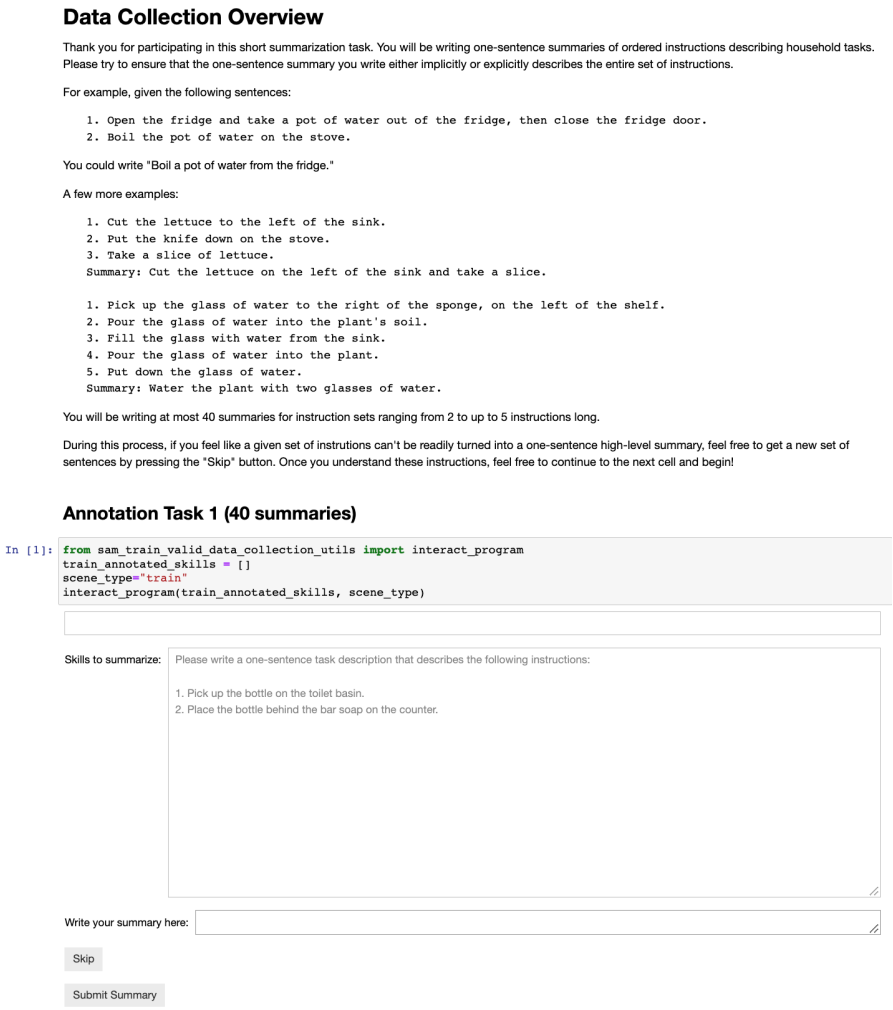


Figure 9: Data collection jupyter notebook page.

We evaluate agents through zero-shot policy evaluation and finetuning on three sets of evaluation tasks in the ALFRED environment: (1) *EVAL_100* to measure the ability of pre-trained agents to execute semantically meaningful instructions at varied levels of abstraction, (2) *EVAL_CHAIN* to measure the ability of agents to chain behaviors across multiple trajectories to solve long tasks, and (3) *EVAL_UNSEEN* to evaluate generalization performance when finetuning to unseen household floor plans.

Collecting evaluation task data. The ALFRED dataset provides high-level language annotations for each of the trajectories in the dataset. We could use these annotations as unseen task-instructions to evaluate our agents. However, we found that the different skills is not equally distributed across trajectories of different skill lengths, *e.g.*, most 2-skill trajectories perform pick-and-place tasks while tasks involving heating skills only appear in length 7+ trajectories. To allow evaluation with a less biased skill distribution, we create the *EVAL_100* task set by randomly choosing a trajectory from the ALFRED dataset *and then randomly sampling a subsequence of skills* from this trajectory. To obtain a high-level language instruction that summarizes this new subsequence, we crowd-source labels from human annotators. For labeling, each annotator is presented with a remotely hosted Jupyter notebook interface (see Fig. 9). Whenever we by chance sample a full ALFRED trajectory for annotation, we directly used the existing high-level annotation from the ALFRED dataset. We

annotate 80 trajectories with human annotators and combine them with 20 randomly sampled single-skill trajectories, resulting in a total of 100 evaluation tasks (see Figure 10 for example instructions).

For *EVAL_CHAIN*, we randomly sampled 20 full trajectories from the ALFRED dataset that had sequences of 7 or 8 skills (10 of length 7, 10 of length 8) and removed these trajectories from the post-LLM aggregated training dataset. We did not remove any of the LLM-aggregated trajectories made up of subsequences of skills within that trajectory. This allows AM and SPRINT to perform skill chaining to solve these tasks by ensuring that there were valid sequences of skills to chain together to be able to solve these removed tasks. For example, assume a (shortened for clarity) sampled skill sequence is "pick up apple," then "put apple in microwave", then "slice the apple." Then, either Actionable Models or SPRINT can chain together the sub-trajectory associated with "pick up apple" then "put apple in microwave" with the "slice the apple" sub-trajectory to solve this task. These trajectories all had annotations from ALFRED annotators, so we used those annotations directly (see Figure 12 for example instructions).

Finally, for *EVAL_UNSEEN*, we collected a set of 10 full-length trajectories from the ALFRED "valid-unseen" dataset consisting of validation tasks in unseen floor plans. We collected 2 of each length from 1 through 5 for a total of 10 tasks by sampling random full-length trajectories from this dataset, with the exception of length 1 tasks (we just sample random skills to create length 1 tasks). As these are full trajectories, they already have human annotations from ALFRED, which we directly use as the task description (see Figure 11 for example instructions).

We list additional details about the tasks in each evaluation set in Table 1.

Table 1: Evaluation Task Specifics. Note that the "number of env actions per task" corresponds to the number of environment actions the ALFRED expert planner required to complete that task.

	EVAL_100	*EVAL_CHAIN*	*EVAL_Unseen*
Number of Tasks	100	20	10
Task Lengths (# primitive skills)	[1, 2, 3, 4, 5, 6, 7]	[7, 8]	[1, 2, 3, 4, 5]
Min Number of Env Actions per Task	1	34	2
Avg Number of Env Actions per Task	39.1	60.9	46.6
Max Number of Env Actions per Task	113	104	124

Finally, we display 5 randomly sampled tasks, along with their human annotations, from each of our task sets in Figures 10, 11, and 12.

Online finetuning environment setup. During online-finetuning we initialize the agent in the same house floor plan as the trajectory the task was extracted from to ensure executability. During finetuning, we give each episode a time horizon of 2x the number of environment actions needed by the expert planner to solve the task. We give sparse rewards for each skill solved by the agent during the episode. Therefore for length 1 tasks, the agent can only be rewarded once before the episode ends, while for length 5 tasks, the episode terminates on the fifth reward signal. We give a reward of $\frac{1}{\text{num total skills}}$ for each skill the agent successfully executes so that the return sums to 1. We found that this helped to finetune all comparison methods more stably, possibly due to the fact that giving larger rewards (*e.g.*, 1 for each skill) results in out-of-distribution critic values (when compared to pre-training) that de-stabilize online reinforcement learning.

Skills to Summarize: 1: Grab the knife on the counter. 2: Place the knife in the sink then turn the faucet on so water fills the sink. Turn the faucet off and pick up the knife again. 3: Place the knife on the table to the left of the wooden bowl.
Annotator Summary: Wash the knife from the counter, put in on the table.

Skills to Summarize: 1: Pick up the blue book closest to your and the phone from the bed. 2: Turn on the lamp to take a look at the book in the light.
Annotator Summary: Examine the book by the light of a lamp.

Skills to Summarize: 1: Pick up yellow candle on counter. 2: Open cabinet, put candle in cabinet, close cabinet 3: Pick up yellow candle from toilet.
Annotator Summary: Move the candle from the sink to the cabinet under the sink, close it and and then pick the candle from the top of the toilet in front of you.

Skills to Summarize: 1: Pick the pot on the left side up from the stove. 2: Set the bowl and knife on the table next to the tomato.
Annotator Summary: Put the bowl with the knife in it next to the tomato.

Skills to Summarize: 1: Pick up the pen that's in front of you that's under the mug. 2: Put the pencil in the mug that was above it. 3: Pick up the mug with the pencil in it.
Annotator Summary: Put the pen into the mug and pick up the mug.

Figure 10: Randomly sampled, human language instruction annotations from the *EVAL_100* task set.

Skills to Summarize: 1: Pick up the lettuce on the counter. 2: Chill the lettuce in the fridge. 3: Put the chilled lettuce on the counter, in front of the bread.
Annotator Summary: Put chilled lettuce on the counter.

Skills to Summarize: 1: Pick up an egg from off of the kitchen counter. 2: Open the fridge, put the egg in to chill for a few seconds and then take it back out. 3: Place the cold egg in the sink.
Annotator Summary: Chill an egg and put it in the sink.

Skills to Summarize: 1: Pick up the butter knife off of the right side of the kitchen island. 2: Put the knife handle down in the frying pan that is on the front left burner of the stove. 3: Pick up the frying pan with the knife in it off of the stove. 4: Put the frying pan with the knife in it into the sink basin to the right of the potato.
Annotator Summary: Put a frying pan with a knife in it into the sink.

Skills to Summarize: 1: Take the pencil from the desk. 2: Put the pencil on the desk.
Annotator Summary: Take the pencil from the desk, put it on the other side of the desk.

Skills to Summarize: 1: Pick up the left pillow on the chair. 2: Put the pillow on the sofa right of the newspaper. 3: Pick up the pillow on the chair. 4: Put the pillow on the sofa left of the newspaper.
Annotator Summary: Place two pillows on a sofa.

Figure 11: Randomly sampled, human language instruction annotations from the *EVAL_UNSEEN* task set.

F Extended Experiments, Results, and Analysis

Table 2: *EVAL_100* and *EVAL_Chain* eval dataset per-length and overall skill completion rates. See Section 3 for experiment setup.

		AM	L-BC	SPRINT
EVAL_100	Number of Completed Subtasks Overall	0.46 ± 0.05	0.41 ± 0.02	$\mathbf{1.27 \pm 0.13}$
	Length 1 Progress	0.23 ± 0.01	$\mathbf{0.77 \pm 0.04}$	0.53 ± 0.07
	Length 2 Progress	0.22 ± 0.03	0.24 ± 0.03	$\mathbf{0.49 \pm 0.03}$
	Length 3 Progress	0.17 ± 0.03	0.16 ± 0.04	$\mathbf{0.48 \pm 0.03}$
	Length 4 Progress	0.10 ± 0.01	0.06 ± 0.01	$\mathbf{0.35 \pm 0.06}$
	Length 5 Progress	0.16 ± 0.08	0.02 ± 0.01	$\mathbf{0.46 \pm 0.06}$
	Length 6 Progress	0.16 ± 0.01	0.02 ± 0.03	$\mathbf{0.36 \pm 0.12}$
	Length 7 Progress	0.00 ± 0.00	0.00 ± 0.01	$\mathbf{0.01 \pm 0.02}$
EVAL_CHAIN	Number of Completed Subtasks Overall	0.67 ± 0.09	0.04 ± 0.05	$\mathbf{2.59 \pm 0.66}$
	Length 7 Progress	0.04 ± 0.01	0.00 ± 0.00	$\mathbf{0.25 \pm 0.10}$
	Length 8 Progress	0.13 ± 0.02	0.01 ± 0.01	$\mathbf{0.43 \pm 0.08}$

Here, we present additional results complementary to the experiments in the main paper in Section 3. We present and analyze LLM annotation examples in Section F.1, zero-shot evaluations with step-by-step task instructions in Section F.3, and an extended ablation analysis in Section **??**.

F.1 LLM Summary Examples

We randomly sample 12 OPT-13B task summaries produced while performing skill aggregation (explained in Section 2.2) using the prompt in Fig. 7 and display them in Fig. 13. Because we sample the tokens sequentially during text generation, some of the summaries mostly describe one or a few of the skills but in a non-sensical manner (*e.g.*, LLM(*"Pick up the plastic bowl on the left. Turn on the lamp."*) = *"Look at the plastic bowl on the left"*). The summaries also likely can be improved by including more examples in the prompt. However, after analyzing many more examples, we see that OPT-13B generally provides fitting high-level summaries for most sequences by skipping over implied sub-tasks (although it sometimes also skips over important sub-tasks, likely due to the prompt). For example, it summarizes the task *"Pick up the plaid pillow that is on the left end of the couch. Place the pillow on the ottoman"* as *"Place a plaid pillow on the ottoman,"* in which picking up the plaid pillow is implied. It also often demonstrates object relation understanding, *e.g.*, by stating *"Place a plaid pillow"* when only the first sub-task in the sequence mentions that the pillow is plaid.

Skills to Summarize: 1: Pick up the plaid pillow that is on the left end of the couch. 2: Place the pillow on the ottoman.
LLM-generated Summary: Place a plaid pillow on the ottoman.

Skills to Summarize: 1: Pick up the piece of potato furthest to the right. 2: Open the microwave, place the potato inside, close the door, microwave for two seconds, then remove the potato. 3: Place the cooked potato piece to the right of the salt shaker.
LLM-generated Summary: Place a cooked potato piece on the table.

Skills to Summarize: 1: Put the spoon in the pot on the left. 2: Pick up the pot with the spoon.
LLM-generated Summary: Put the pot with the spoon on the left.

Skills to Summarize: 1: Open the microwave above the stove and slice the apple inside with the knife. 2: Place the knife standing up on the blue dish. 3: Remove a slice of apple from the microwave. 4: Place the apple slice on the blue plate.
LLM-generated Summary: Put an apple slice on the blue plate.

Skills to Summarize: 1: Pick up the butter knife that is next to the fork. 2: Open the refrigerator. 3: Open the refrigerator and place the knife on the bottom shelf in the refrigerator then close the door.
LLM-generated Summary: Open the refrigerator and place the butter knife on the bottom shelf in the refrigerator then close the door.

Skills to Summarize: 1: Open the fridge and put the knife inside. 2: Take a piece of the apple out of the microwave.
LLM-generated Summary: Take a slice of apple out of the microwave.

Skills to Summarize: 1: Pick up the remote that is on the shelf. 2: Put the remote on the couch.
LLM-generated Summary: Put the remote on the couch.

Skills to Summarize: 1: Pick up the dirty green towel to the right of the bar of soap on the vanity. 2: Put the towel into the sink, rinse it for a few seconds and pick it back up. 3: Open the far left bottom drawer to the left of the sink, put the rinsed green towel inside and close the drawer.
LLM-generated Summary: Put the towel in the drawer.

Skills to Summarize: 1: Pick up the plastic bowl on the left. 2: Turn on the lamp.
LLM-generated Summary: Look at the plastic bowl on the left.

Skills to Summarize: 1: Slice up the fruit that is in the sink. 2: Place the knife in the second row of the fridge. 3: Grab a slice of the fruit. 4: Place the fruit slice on the back right pan. 5: Grab the pan that has the fruit in it.
LLM-generated Summary: Put a slice of fruit on a pan in the fridge

Skills to Summarize: 1: Pick up the pot on the front right burner. 2: Chill the pot in the fridge then get it back out and close the door.
LLM-generated Summary: Put a pot on the stove, put it in the fridge, get it out, close the door.

Skills to Summarize: 1: Pick up the knife on the counter. 2: Cut the potato in the sink into slices. 3: Put the knife in the sink with the potato. 4. Pick up a slice of potato from in the sink. 5. Warm the potato slice in the microwave and then remove it.
LLM-generated Summary: Warm the potato slice in the microwave.

Figure 13: LLM Summary examples. We randomly sample 12 language combinations generated by the OPT-13B text generation model [13] and display them without cherry-picking specific examples. The LLM often creates a high level description by skipping implied steps (like humans do), *e.g.*, "Put the remote on the couch" implies the need to pick up the remote.

F.2 Comparing Summaries from Different LLMs

In this section, we justify the use of OPT-13b as our summarization model by comparing its summaries of three different skill sequences to OPT-350m (350 million parameters) and OPT-1.3b (1.3 billion parameters). The summaries and sequences are displayed in Fig. 14. Although these are 3 randomly sampled examples, they highlight differences between the models that hold true across many examples that we examined.

As the language model size increases, it understands more of the relationships between sub-tasks in each sequence, therefore generating better summaries. An example of this is the second example in Fig. 14, which highlights a sequence in which both OPT-1.3b and OPT-13b understand the high-level task of putting a warmed plate in the cabinet, as opposed to OPT-350m which essentially repeats the second sub-task in the sequence. Yet sometimes, only OPT-13b produces a reasonable summary. In the last example, OPT-13b produces a summary that implies that an agent must wash the potato ("Put a *rinsed* potato on the counter..."); the other two models' summaries only mention the last step of putting the potato down.

Skills to summarize: 1. Pick the clock up from the desk. 2. Turn the lamp on.
OPT-350m Summary: Look at the clock.
OPT-1.3b Summary: Turn on the lamp.
OPT-13b Summary: Look at the clock.

Skills to summarize: 1. Take the plate from the cabinet to the left of the stove. 2. Warm the plate in the microwave, above the stove. 3. Put the plate in the cabinet.
OPT-350m Summary: Place the plate in the microwave.
OPT-1.3b Summary: Put a warmed plate in the cabinet.
OPT-13b Summary: Put a warmed plate in the cabinet.

Skills to summarize: 1. Open the microwave, take out the potato and shut the door. 2. Put the potato in the sink, turn the water on and off again. Take the potato out of the sink. 3. Place the potato on the counter, to the right of the sink.
OPT-350m Summary: Put the potato on the counter, to the left of the sink.
OPT-1.3b Summary: Place the potato on the counter, to the right of the sink.
OPT-13b Summary: Put a rinsed potato on the counter to the right of the sink.

Figure 14: Comparison of OPT-350m, 1.3b, and 13b summaries on 3 randomly sampled sequences from our dataset. In general, as the model size increases, the summary becomes better. However, there are some sequences all three on which all 3 models do not generate good summaries for, such as the first sequence in these examples.

F.3 Step by Step Zero-shot evaluations

Humans may sometimes prefer to give more detailed instructions, *e.g.*, if they do not trust a household robot to successfully execute a high-level instruction. In that case, they are likely to give detailed, step-by-step instructions instead. In this section, we demonstrate the effectiveness of our pre-training strategy in enabling execution of longer-horizon tasks when specified by such step-by-step instructions. To generate these step-by-step instructions, we combine all subtask instructions into one paragraph to condition agents on instead of the high-level human annotation. For example, a task with 2 subtasks, "Pick up the knife" and "Slice the potato," is given the simple task annotation "Pick up the knife. Slice the potato."

We compare SPRINT and IL on both *EVAL_100* and *EVAL_CHAIN* tasks with the combined, step-by-step task annotations in Table 3. L-BC and SPRINT both perform similarly on the step-by-step *EVAL_100* tasks compared to with the original annotations, while both perform better on *EVAL_CHAIN* tasks thatn with the original annotations. This is likely due to how *EVAL_CHAIN*

tasks are on average much longer (7 or 8 tasks). Therefore, instructions spelling out exactly what to do allow agents to more easily understand what and in which order subtasks should be performed.

Table 3: Step-by-step evaluation number of completed sub-tasks (number of completed sub-tasks).

	L-BC	SPRINT
EVAL_100 Completed Subtasks	0.46 ± 0.01	$\mathbf{1.24 \pm 0.19}$
EVAL_CHAIN Completed Subtasks	0.22 ± 0.03	$\mathbf{3.11 \pm 0.08}$

Table 5.

F.4 Ablation Studies

We verify the effectiveness of the components of our approach, with the following ablations:

- **SPRINT w/o chain**: removes cross-trajectory chaining (Section 2.3), instead trains only on within-trajectory human-provided and LLM-aggregated tasks
- **SPRINT w/o LLM-agg**: additionally removes the LLM aggregation (Section 2.2), thus trains offline RL agent only on the human-provided task annotations.

We report zero-shot evaluation results in Table 4. The results show the importance of the different components of our approach. Without cross-trajectory chaining, performance drops by ∼20-30%. Interestingly, we find that performance also degrades on the *EVAL_100* task set, indicating that cross-trajectory chaining can improve performance even on long-horizon tasks in the training set.

Table 4: *EVAL_100* and *EVAL_CHAIN* returns.

	EVAL_100	*EVAL_CHAIN*
SPRINT (ours)	$\mathbf{1.27 \pm 0.13}$	$\mathbf{2.59 \pm 0.66}$
SPRINT w/o chain	0.91 ± 0.03	2.04 ± 0.04
SPRINT w/o LLM-agg	0.38 ± 0.05	0.10 ± 0.04

We observe an even larger performance loss when removing the language model aggregation of pre-training tasks: the resulting agent is solely trained on the shorter horizon human task annotations, and thus struggles to solve long-horizon evaluation tasks, similar to the L-BC approach in Section **??**.

Now, we present further ablations through questions and experimental answers regarding specific design choices of our method.

How much does the LLM contribute to skill aggregation? To answer this question, we compare using SPRINT with LLM aggregation on adjacent sub-trajectory sequences but no chaining (**SPRINT w/o chain**) to SPRINT with skill aggregation, but where the skills are relabeled by naïvely concatenating the sentences together (**SPRINT w/o chain, w/ concat-agg**). Across both task sets, SPRINT w/o chain, w/ concat-agg is outperformed by SPRINT w/o chain, especially in *EVAL_CHAIN*. This signifies that using the LLM helps with understanding very-long horizon, high-level semantic instructions as the LLM generates relevant task summaries for consecutive skills in a trajectory.

What if we relabel with the LLM during chaining? We examine also using the LLM to label skills during cross-trajectory skill aggregation (**SPRINT w/ LLM-chain**), rather than concatenating the skill annotations together (**SPRINT**). Overall, SPRINT w/ LLM-chain performs slightly worse in average return and success rates. When analyzing the summaries generated by the LLM, we found that randomly paired instructions can rarely be summarized meaningfully, thereby resulting in noisy and sometimes meaningless instructions. Therefore, we implemented SPRINT w/ LLM-chain by only utilizing the top 1% (top 10 candidates with our batch size of 1024) of in-batch chaining candidates (ranked by the LLM's prediction of what the next skill should be). Even so, the performance is not better than just simply concatenating the instructions, as even these top 1% candidates still have a high chance of not being sensible sentences to summarize.

Table 5: *EVAL_100* Ablation Returns and Success Rates

	EVAL_100 Average Return	*EVAL_CHAIN* Average Return
SPRINT (ours)	**1.27 ± 0.13**	**2.59 ± 0.66**
SPRINT w/ LLM-chain	1.15 ± 0.01	2.46 ± 0.21
SPRINT w/o chain	0.91 ± 0.03	2.04 ± 0.04
SPRINT w/o chain, w/ concat-agg	0.77 ± 0.06	0.67 ± 0.20
SPRINT w/o LLM-agg	0.38 ± 0.05	0.10 ± 0.04

F.5 Qualitative Comparison Results

Zero-shot evaluation. We compare SPRINT, AM, and L-BC zero-shot evaluation results on long *EVAL_CHAIN* tasks in Fig. 15. In general, SPRINT is able to make substantially more progress on *EVAL_CHAIN* tasks as it leverages the large language model to generate longer-horizon, semantically meaningful pre-training tasks and performs cross-trajectory chaining to learn to chain its existing dataset tasks. In the visualized examples, SPRINT is able to understand and successfully execute many of the sub-tasks implied but not directly stated by the natural language task instruction. L-BC makes very little progress on these tasks, not even understanding what the first sub-task to complete should be as the task annotation is out of distribution from what it saw while training. Finally, AM is able to make some progress on some of these tasks due to its long-horizon goal pre-training objective. However, this is less effective than our language-conditioned pre-training in such zero-shot evaluations.

Finetuning. We finetune SPRINT, AM, and L-BC on *EVAL_UNSEEN* tasks, in household floorplans that were never seen while training, and visualize qualitative policy rollout examples *after finetuning* in Fig. 16. In general, SPRINT is able to finetune to longer-horizon tasks while AM and L-BC both struggle with making progress on longer-horizon tasks despite receiving rewards for every completed sub-task. SPRINT's ability to complete more sub-tasks on many of the longer-horizon tasks is demonstrated in Figure 16a, while a case in which both SPRINT and AM make partial progress throughout finetuning is demonstrated in Figure 16b. We believe that AM has more trouble finetuning on these tasks than SPRINT because the task specification for AM (goal images) is out of distribution; pre-training on *semantic* tasks with SPRINT allows agents to more easily learn longer-horizon behaviors as the task specifications may still be in-distribution of the pre-training tasks that LLM skill-aggregation and skill chaining produce.

Skills to Summarize: 1: Pick up the knife in front of the lettuce. 2: Slice the apple in the sink with the knife. 3: Place the knife into the sink. 4: Pick up the sliced apple from the sink. 5: Place the apple slice into the pot on the stove. 6: Pick up the pot from the stove. 7: Pick up the pot from the stove.
Annotator Summary: Slice an apple for the pot on the stove and put the pot on the counter to the right of the door.

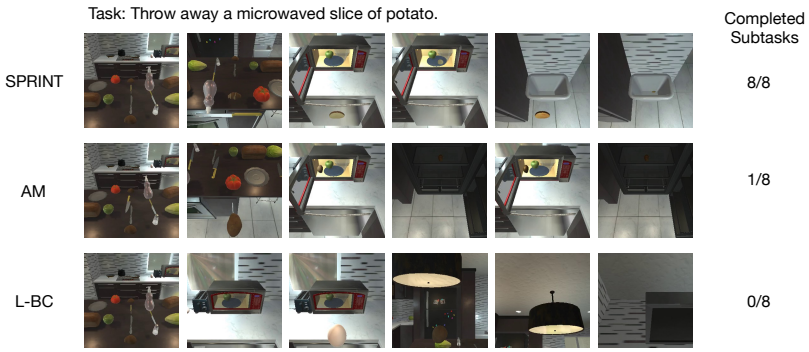
Skills to Summarize: 1: Take the apple from the counter in front of you. 2: Place the apple in the sink in front of you. 3: Take the knife by the sink in front of you. 4: Cut the apple in the sink in front of you. 5: Place the knife in the sink in front of you. 6: Take an apple slice from the sink in front of you. 7: Heat the apple in the microwave, take it out and close the microwave. 8: Place the apple slice in the sink in front of you.
Annotator Summary: Place a warm apple slice in the sink.

Skills to Summarize: 1: Pick up the loaf of bread. 2: Put the bread on the counter above the spatula. 3: Pick up the knife that's above and to the right of the loaf of bread. 4: Cut the top half of the loaf of bread into slices. 5: Put the knife on the edge of the counter in front of you horizontally. 6: Pick up a slice of bread from the middle of the loaf. 7: Cook the bread in the microwave then take it out and close the microwave door. 8: Throw the cooked slice of bread away.
Annotator Summary: Put a microwaved slice of bread in the oven.

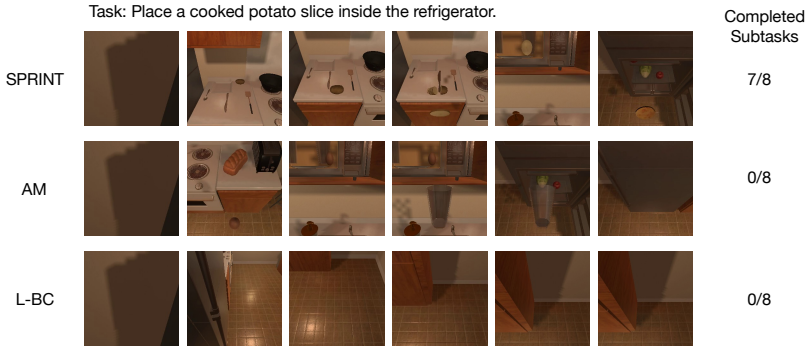
Skills to Summarize: 1: Pick the knife up from off of the table. 2: Open the microwave, slice the potato, and close the microwave. 3: Open the microwave, place the knife inside of it, and close the microwave. 4: Open the microwave, pick up the potato slice inside, close the microwave. 5: Place the potato slice in the pan on the stove. 6: Pick up the pan from the stove. 7: Open the refrigerator, place the pan inside, and close the refrigerator.
Annotator Summary: Move the pan from the stove top to inside the black refrigerator.

Skills to Summarize: 1: Pick up the red tomato on the counter to the right of the stove. 2: Put the tomato onto the island below the butter knife. 3: Pick up the butter knife off of the kitchen island. 4: Slice up the tomato on the kitchen island. 5: Place the butter knife onto the island to the right of the sliced tomato. 6: Pick up a tomato slice off of the kitchen island. 7: Open the fridge and put the tomato slice on the bottom shelf, then close the door, after a couple seconds open the fridge and remove the tomato slice then close the door. 8: Open the microwave door and place the tomato slice inside the microwave in front of the egg.
Annotator Summary: Put a chilled tomato slice into the microwave.

Figure 12: Randomly sampled, human language instruction annotations from the *EVAL_CHAIN* task set.



(a) SPRINT successfully solves this task, while AM fails to slice the potato and repetitively iterates between putting the potato in the fridge and microwave. L-BC fails to even pick up the potato, as the task annotation does not directly describe picking up a potato.

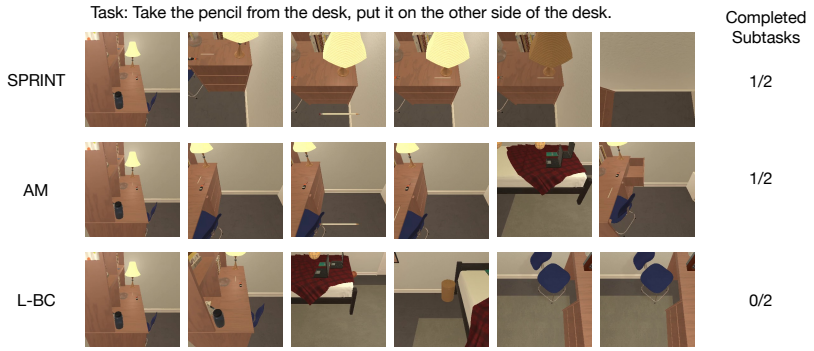


(b) SPRINT nearly solves this task, while AM picks up an egg instead of a potato. L-BC picks up random objects not related to the annotation.

(c) SPRINT completes the entire task. AM picks up the tomato but fails to put it down onto the counter and slice it. L-BC aimlessly wanders and picks up random objects.

Figure 15: Visualizations of zero-shot policy rollouts on three tasks in the *EVAL_CHAIN* task set.

(a) SPRINT picks up and places one of the pillows on the sofa, and picks up the second but does not manage to place the second on the sofa, thus completing 3/4 subtasks. AM and L-BC both learn to pick up a pillow but never learned to place it in the correct spot.



(b) SPRINT and AM both learn to pick up a pencil from the desk, although neither manage to put the pencil down in the correct place "on the other side of the desk." Meanwhile, L-BC never picks up the pencil.

Figure 16: Visualizations of policy rollouts on two tasks in the *EVAL_UNSEEN* task set, after finetuning each method. These floor plans were originally unseen to all agents until finetuning.